

Co-Optimization and Co-Learning Methods for Automated Design of Rigid and Soft Robots

by

Andrew Everett Spielberg

B.S., Cornell University (2010)

M.Eng., Cornell University (2011)

Submitted to the Department of Electrical Engineering and Computer
Science

in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2021

© Andrew Everett Spielberg, MMXXI. All rights reserved.

The author hereby grants to MIT permission to reproduce and to distribute
publicly paper and electronic copies of this thesis document in whole or in
part in any medium now known or hereafter created.

Author

Department of Electrical Engineering and Computer Science

August 27, 2021

Certified by

Daniela L. Rus

Andrew (1956) and Erna Viterbi Professor of Electrical Engineering and

Computer Science

Thesis Supervisor

Certified by

Wojciech Matusik

Professor of Electrical Engineering and Computer Science

Thesis Supervisor

Accepted by

Leslie A. Kolodziejcki

Professor of Electrical Engineering and Computer Science

Chair, Department Committee on Graduate Students

Co-Optimization and Co-Learning Methods for Automated Design of Rigid and Soft Robots

by

Andrew Everett Spielberg

Submitted to the Department of Electrical Engineering and Computer Science
on August 27, 2021, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Nature demonstrates an incredible diversity, capability, and complexity of life, with organisms that can robustly run, jump, and swim. Compared with their biological brethren, robotic “life” lacks rich dexterity or economy of motion, and their plainly simple designs indicate room for improvement. Unfortunately, a major barrier to creating similarly adept robots is the design process itself. Each aspect of robot design, including the (physical) body (*e.g.* actuation, sensing, geometry, materials) and the (cyber) brain (*e.g.* control, proprioception) is typically not integrated in a single design workflow, and a lack of fast, accurate, useful simulators leads to expensive, spiraling, hardware intensive iteration. This thesis introduces methods to marry all aspects of robot design into combined algorithms for holistic cyberphysical design. Core to this solution is co-optimization and co-learning methods which can simultaneously reason about different design domains and achieve locally optimal performance. This thesis further discusses considerations in modeling (*via* differentiable simulation) and realizability (through automated and semi-automated fabrication workflows). We describe how this entire suite of capabilities from modeling to automated fabrication can be conceptualized in a complete end-to-end “robot design stack,” providing full CAD-CAM computational design capabilities. We demonstrate these capabilities on for rigid, compliant, and soft robot design tasks, including locomotion, manipulation, and tactile sensing, and discuss the frontiers of this burgeoning field of computational robot design.

Thesis Supervisor: Daniela L. Rus

Title: Andrew (1956) and Erna Viterbi Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Wojciech Matusik

Title: Professor of Electrical Engineering and Computer Science

To my father, Craig, and the memory of my mother, Joni.

Acknowledgments

I would like to thank all of the people who made my dream of completing my PhD possible.

First and foremost, I would like to thank my advisors, whose advice and guidance has been invaluable over the last seven years. First, thank you to Daniela Rus, who believed in me and gave me an opportunity first as a research engineer in her group, then as a PhD student, and then allowing me to contribute to the Printable Programmable Machines project, which evolved into work for my PhD thesis. Thank you for teaching me how to perform research, how to communicate my research, how to think about contributions to the field, and for always encouraging me to “Keep The Gradient.” You always pushed me to think about what’s next, and push for more impressive results. I felt under your guidance that every project had the opportunity to be great. Second, I would like to thank Wojciech Matusik, whose creativity and ambitiousness inspired me every day. Thank you for pushing me to make good research happen, for teaching me to be thorough, providing me an opportunity to learn to formulate research problems and translate those into successful grant proposals, for immersing me in various teaching experiences, and most of all, for showing me that with some clever thinking and some hard work, what looks almost impossible can be made real.

I would also like to thank Pulkit Agrawal, the third member of my thesis committee, who consistently offered advice during the finalization of this thesis.

Thank you to those closest me who provided me an incredible web of support during these last seven years. First, thank you to my father Craig and my mother Joni, who sadly passed away during my PhD; I would have loved for you to be around to see this work completed. Thank you to my caring partner, Gloria Chang, who was supportive and patient during all of my late nights working on papers. Of course, I would be remiss if I did not also thank my wonderful cat Mew, who is the cutest and best cat in the world and who has been my most enduring roommate through my PhD.

I would like to give a special thank you to some of my earliest mentors during my PhD: Mehmet Dogar, Cynthia Sung, and Adriana Schulz, and especially Ross Knepper, my first hands-on academic mentor. The four of you showed me the ropes on how to navigate all of the complexities of academia, both the forest and the trees views of how to formulate

and tackle complex research problems on a broad scale and on a day-to-day basis, how to think about publishing, conferences, and careers, and for bringing joy into the early days of academic journey.

Thank you also to some other important people in my academic life. Thank you to mentors at Cornell University who helped shape my pre-PhD career and gave me invaluable advice: Kavita Bala, Andy Ruina, Bobby Kleinberg, Walker White, Johannes Gehrke, Dan Cosley, Hod Lipson, and Ashutosh Saxena. Thank you also to some of my even *earlier* mentors from High School who kept me excited down the track of math and science: Ms. McCann, Mr. Colavita, and Mr. Boyle. Finally, thank you to a few people who gave me extremely important and valuable career advice: David Martin Warsinger, Haden Hooyeon Lee, and Liane Slaughter — you did not have to give me your time, but without it, I do not believe I would be where I am today.

Internships played an important role in broadening my horizons. Thank you to Jim McCann, who took me on and mentored me to very successful research at Disney Research — Pittsburgh. Thank you also to Jim’s group, including Lea Albaugh, Chenxi Liu, and April Grow; also thank you to lab director Jessica Hodgins for the opportunity. Thank you also to some awesome other people I worked with there: Alanson Sample, Jen Mankoff, Scott Hudson; and some other helpful and inspiring people I met along the way: Stelian Coros, Braden McDorman, Kate Samson, Mo Mahler, Eric Brockmeyer, and Alice Moon. Thank you to Bernhard Thomaszewski for his mentorship during my internship at Disney Research — Zürich. Thank you also to lab director Markus Gross for the opportunity. Thank you to Bernhard’s group for fostering a wonderful research environment: Christian Schumacher, Johnas Zehnder, Raphaël Charrondiére, and co-authors Espen Knoop, Vittorio Megaro, and Moritz Bächer. I’d further like to thank Prof. Olga Sorkine-Hornung, for her support. I’d also like to thank Virag Varga and Yagiz Aksoy, who were close and inspiring friends at the lab. Finally, thank you for Vladlen Koltun and Samarth Brahmabhatt at Intel for an interesting and educational final stop on my PhD journey.

Thank you to all the other wonderful collaborators I have had over the years: Profs. Henrik I. Christensen, Eitan Grinspun, Russ Tedrake, Nora Ayanian, David I.W. Levin, Armando Solar-Lezama, Joshua B. Tenenbaum, William T. Freeman, Bo Zhu, Eftychios Sifakis,

and Robert K Katzschmann; and other researchers, many of whom are now professors in their own right: Ankur Mehta, Jeffrey I. Lipton, Josie Hughes, Changhyun Choi, Matthew Arbesfeld, Jason Strauss, Wei Zhao, Yu Robin Cheng, Luis Trueba, Jeevana P. Inala, Yewen Evan Pu, Stuart Baker, Jiancheng Liu, Yunfei Li, Mina Konaković-Luković, sebastian Wah, Kui Wu, John Z. Zhang, Lillian T. Chin, Vik Parthiban, Prof. Derek Nowrouzezahrai, Julien Valentin, Sofien Bouaziz, Cengiz Oztireli, David Matthews, and Sam Kriegman. I'd also like to give a special thank you to a few of my closest collaborators during grad school: Tao Du, Yuanming Hu, Alexander A. Amini, Allan Zhao, and Jie Xu.

Thank you to researchers whose discussions and work directly aided the work in this thesis, including Joseph DelPreto, Jacob Haip, Isaque Dutra, Michael Posa, Philipp Herholz, Alessia Marra, and Liane Makatura. Thank you also to all of the anonymous reviewers and editors who helped revise the work in this thesis throughout the years.

Thank you to Britton “Bryt” Bradley, Roger White, and Mieke Moran, for being incredibly supportive on the administrative side and utterly irreplaceable.

A special thank you to my other labmates, who served as my closest friends: Wenshou Wang, Vahid Babaei, Tae-Hyun Oh, Szu-Po Wang, Sung-Ho Bae, Subra Sundaram, Shinjiro Sueda, Pitchaya Sitthi-amorn, Petr Kellnoher, Nick Bandiera, Mélina Skouras, Masha Shugrina, Louise van den Heuvel, Kiril Vidimče, Javier Ramos, James Minor, Harrison Wang, Emma Steinhardt, Desai Chen, David Kim, Changil Kim, Yunsheng Tian, Yifei Li, Yiyue “Alyssa” Luo, Pingchuan Ma, Liang Shi, Liane Makatura, Lara Zlokapa, Beichen Li, Alexandre Kaspar, Wan Shou, Timothy Erps, Michael Foshey (and of course Buttercup Foshey), Daniel Feldman, Shuhei Miyashita, Kyle Gilpin, Jingjin Yu, Nick Wang, Guy Rosman, Stephanie Gil, Robert MacCurdy, Stephane Bonardi, Javier Alonso-Mora, Hayim Shaul, Sedat Ozer, Liam Paull, Cristian-Ioan Vasile, Harry Lang, Alyssa Pierson, Igor Gilitschenski, Byoung Kwon An, Daniel Soltero, Gavin Hall, Bianca Homberg, Pramod Kandel, Tim Zhong, Rohan Banerjee, Brian Julian, Marek Doniec, Andrew Marchese, Mikhail Volkov, Sebastian Claici, Steven Guitron, Alex Wallar, Felix Naser, Wilko Schwarting, Murad Abu-Khalaf, Hunter Hansen, Ryan Truby, Josie Hughes, Xiao Li, Ramin Hasani, James Bern, Paul Tylkin, Yutong Ban, Shuguang Li, John Romanishin, Lucas Liebenwein, Cenk Baykal, Noam Buckman, Teddy Ort, Veevee Cai, Aaron Ray, Lillian Chin, Tim Seyde,

Annan Zhang, Davd Dorhout, Thomas Balch, and Tsun-Hsuan Wang.

Thank you to the many awesome UROPs and M.Eng students I've had the opportunity to work with over the years: James Hobin, Banti Gheneti, Kelly Liu, Hanna Lee, Scott Viteri, Hanjun Li, Megan D. Levin, Tianyi Li, Christopher Krause, YeonHwan Brian Park, Phoebe Piercy, Nathaniel Lee, Lantian Chen, Sanchit Bhattia, Eric Bradford.

I would also like to thank the academic community at MIT and the research community at large. I'd specifically like to give speical thank yous to my academic advisor Prof. Daniel Jackson; other instructors I have had at MIT, including Profs. Samuel Madden, Ankur Moitra, Justin Solomon, Marjorie Liu, Phillip Tan, Neil Gershenfeld, and David S. Kong. Thank you to other supportive members of the community, including Prof. Stefanie Mueller, Prof. Phillip Isola, and CSAIL support Rachel Gordon, Adam Conner-Simons, and the TIG team.

Finally, thank you to all of my other friends and family who supported and believed in me throughout the years.

The work presented in this thesis was made possible by the following funding opportunities: NSF Grant Nos. 1240383 and 1138967, H2020-ICT- 645599), the Unity Global Graduate Fellowship, IARPA Grant 2019-19020100001, NSF 1644558, IARPA grant No. 2019-19020100001, and National Science Foundation EFRI 1830901.

Contents

1	Introduction	29
1.1	Vision: Task-Based Robot Design	30
1.2	Comparison To State of the Art	33
1.3	The Future of Robot Design	34
1.3.1	Interactive Robogami — A First Demonstration	37
1.3.2	Design Decisions	39
1.3.3	Outlook	45
1.4	Contributions and Overview	46
1.5	Related Work	48
1.5.1	Manufacturing Oriented Design	48
1.5.2	Design of Robots and Functional Mechanisms	48
1.5.3	Concurrent (Co-)Design	49
2	Background And Notation	51
2.1	The Life And Times of A Robot Simulation	52
2.2	Rigid Body Simulation	54
2.2.1	Kinematics	54
2.2.2	Dynamics	56
2.3	Soft Body Simulation	57
2.3.1	Kinematics	58
2.3.2	Dynamics	58
2.3.3	Discretization and Contact	60
2.3.4	Material Models	61

2.4	Time integration	63
2.5	Notation	65
3	Algorithms for the Co-Optimization of Rigid Robots	67
3.1	Parametric Trajectory Optimization	70
3.1.1	Robot Specification and Dynamics	71
3.1.2	Constraints	74
3.1.3	Objectives and Actuator Selection	76
3.1.4	Initialization	77
3.1.5	Optimization	78
3.2	Implementation	79
3.3	Analysis	79
3.3.1	Convergence Guarantees	80
3.4	Experiments	81
3.4.1	Hexapod	83
3.4.2	Biped	85
3.4.3	Quadruped	86
3.4.4	Quadcopter	87
3.5	Related Work	88
3.6	Outlook: Rigid Robot Co-Optimization	90
4	The (Differentiable) Material Point Method and Soft Robot Co-Optimization	93
4.1	Preliminaries	94
4.2	Forward Simulation	95
4.2.1	Boundary conditions (Contact models)	97
4.3	Material Models	97
4.3.1	Fixed Corotated	97
4.3.2	Neohookean	98
4.3.3	Incompressible materials	98
4.4	Actuation Models	99
4.5	Back-Propagation	100

4.5.1	Implementation	101
4.6	Pitfalls and Resilience	103
4.6.1	Numerical Fracture	104
4.7	Soft Robotics API	104
4.7.1	Graph Construction — Preliminaries	104
4.7.2	Choosing the Objective Function	106
4.7.3	Observers	106
4.7.4	Controllers	106
4.7.5	Actuators	107
4.7.6	Optimizers	107
4.7.7	Other Utilities	107
4.7.8	An Example	107
4.8	Applications	108
4.8.1	Inference	108
4.8.2	Control	110
4.9	Soft-Body Co-Optimization	113
4.10	Related Work	118
4.10.1	Differentiable Simulation For Soft Robots	118
4.10.2	Co-Optimization of Control and Materials	120
4.11	Outlook: Differentiable Simulation	121
4.12	Outlook: Soft Robot Co-Design	121
5	Co-Learning Algorithms For Proprioceptive Modeling Of Soft Robots	123
5.1	Co-Learning Control, Materials, and Proprioceptive Models	124
5.2	Overview and Preliminaries	126
5.3	Method	127
5.4	Results and Discussion	132
5.4.1	Problem Hyperparameters	138
5.4.2	Complete 2D Elephant Experimental Results	141
5.5	Related Work	142

5.6	Outlook: Co-Learning of Proprioceptive Models	142
6	Design and Fabrication For Learning-Based Soft Sensing	145
6.1	Co-Learning Of Task And Sensor Placement For Soft Robotics	146
6.1.1	Preliminaries	147
6.1.2	Method	149
6.1.3	Results	153
6.1.4	Applications	159
6.1.5	Derivation of the Strain Rate Tensor	159
6.1.6	Proprioception Problem Hyperparameters	161
6.1.7	Outlook	169
6.2	Sensor Fabrication - A Simple Sensing Architecture For Soft Robotics . . .	169
6.2.1	System Overview	169
6.2.2	Implementation	175
6.2.3	Results	178
6.2.4	Related Work	187
6.2.5	Outlook: Sensor Fabrication	189
7	Conclusion	191
7.1	The Future Of The Future of Robot Design	192
7.1.1	Full Co-Design Algorithms	192
7.1.2	More On The Horizon	195
7.2	Lessons Learned	197
7.3	Closing Remarks	200
A		203
A.1	Soft Robotics API Listings	203
A.2	A robot optimization in 50 lines of Python code	206
A.3	Forward Simulation and Differentiation	208
A.4	Soft Robot Problem Parameters	218

List of Figures

- 1-1 **Left:** The computational robot design stack. The first two layers provide CPU/GPU hardware and languages catered to fast, differentiable simulation (layer 3). Co-optimization algorithms (layer 4), use the gradients computed from simulation to optimize continuous design parameters. This local co-optimization scheme is used as a black-box in more sophisticated learning-based design searches (layer 5). The finalized design is compiled to a digitally fabricable format (layer 6), which is guaranteed to closely match the simulated version *via* the sim-to-real layer (7). **Right:** The proposed design stack incorporating multi-task optimization and digital fabrication, with changed layers in cyan. 35
- 1-2 A robot design that topples while walking (left) can be modified to follow a gait that only wobbles slightly (center), but changing the geometry (right) allows the robot to move much faster and more steadily. 40
- 1-3 System diagram. Users interact with geometry and gait design tools. The designed models are simulated to provide feedback to the user. The user may iterate over the design before fabricating and assembling the model. . . 41
- 1-4 User interface. Icons that link to geometry components are displayed on the left and the gait design tool is on the bottom. Users design models by dragging components into the center canvas and editing them. Performance metrics for the design are shown on the right. 41
- 1-5 Geometry components by category. Our system’s database contains 45 components: 12 bodies, 23 limbs, and 10 peripherals. 43

1-6	The electronics (servomotors, microcontroller, and battery) inside the one example robot designed and fabricated using Interactive Robogami, nicknamed the “Monkey.”	44
1-7	The system provides guidance arrows for users who want to make local geometry optimizations. The up and down arrows indicate that the user should lengthen and shorten the leg respectively. No arrow indicates that the part dimension is already at a local optimum.	44
1-8	Six robots designed and fabricated using the system. Numbers over the limbs indicate the step sequence.	45
1-9	Comparison of Monkey gait in simulation and from physical robot. The robot motion matches those from the simulations.	45
2-1	A cartoon of a 2D bipedal robot (a kinematic tree) and its corresponding graph representation. The root node, labeled 1, has 3 degrees of freedom. Links are drawn in both representations in blue, and edges in gray. The world coordinates of any link on the tree can be computed by recursively computing the world coordinates of its parents.	55
3-1	A Hexapod robot must walk forward to hit a button on the wall. a) The Hexapod’s legs are too short to reach the button. b) Lengthening the legs makes the Hexapod tall enough to reach the button c) Shortening the body allows for motions that are both feasible and require less energy.	68
3-2	A conservative mapping of the necessary output torque to individual minimum motor mass used to preserve continuity. We avoid flat regions in our model to ensure that our optimizer has meaningful gradients to work with. .	75
3-3	Example input keyframes, used for seeding the Hexapod co-optimization. .	77
3-4	A montage of our simulated Biped in motion for the flat walking task, along with the original parameterized version. The Biped is too short and heavy in its original configuration to reach the goal circle.	81

3-5	A montage of our simulated Biped in motion for the step walking task, along with the original parameterized version. The Biped is too short and heavy in its original configuration to reach the goal circle, and its links cannot clear the terrain without collision.	81
3-6	A montage of our simulated Quadruped in motion for the energy efficient task, along with the original parameterized version. The Quadruped is too short and heavy in its original configuration to reach the goal circle.	82
3-7	A montage of our simulated Quadruped in motion for the fast motion, along with the original parameterized version. The Quadruped is too short and heavy in its original configuration to reach the goal circle.	82
3-8	A montage of our simulated Squirrel in motion for the energy efficient task, along with the original parameterized version. Because of the 60 degree limit on the tail links, only one set of geometric configurations for legs and tail extents will result in a squirrel that can reach both red circles.	82
3-9	A montage of our simulated and fabricated Hexapod in motion.	85
3-10	The non-Hexapod models.	88
3-11	A surface plot of the actuation costs versus various parameterizations for the Quadcopter. Our parameterized trajectory optimization is successfully able to find the optimal parameterization at the bottom-left corner.	89
4-1	One time step of MLS-MPM. Top arrows are for forward simulation and bottom ones are for back propagation. A controller is embedded in the P2G process to generate actuation given particle configurations.	94
4-2	The backpropagation graph for a time step (which may include a final state and/or a final loss). The forward simulated system is reverse-mode autodifferentiated in order to compute gradients with respect to state, actuation, and/or controller and design parameters. A memo containing all of the relevant information for analysis is computed and maintained according to the checkpointing strategy discussed in 4.5.1.	101

4-3	Each step is, in itself, a large TensorFlow graph (from Fig. 4-2), which the TensorFlow compiler is ill-suited to optimize. For practical efficiency, we compute only the gradients with respect to individual time steps, and manually backpropagate them using <code>tf.gradients</code>	102
4-4	One full step of simulation. The full state of the robot (masses, positions, and velocities of particles and grids, particle material properties, and particle volumes deformation gradients) is fed into an observer. The observer transmits a more compact summary of the full state to the controller function, which produces an output actuation. The system is then substepped a number of user-specified times with the computed actuation, resulting in the robot state for the next time step.	105
4-5	A montage of our simulated Cable-Driven Biped in motion. Colored lines demonstrate cable placement.	108
4-6	A montage of our simulated terrain-traversing Biped in motion. Green colored actuators are expanding; red, contracting.	108
4-7	A montage of our simulated 3D Bulbasaur in motion. Green colored actuators are expanding; red, contracting.	109
4-8	Convergence of system identification for the soft finger. All material parameters converge to their initial values within around 100 epochs of SGD. The x -axis presents epochs of SGD, while the y -axis meaning depends on the trend-line observed. While the loss does not appear to change much in the absolute scale, this is a local (and ground truth) minimum. It cannot converge completely to 0 due to the noise added to the dataset.	109
4-9	Progress of the 2D Cable-Driven Biped vs. iteration and wall-time. By pulling in an alternating motion on each of the two cables, the biped can ambulate forward.	111
4-10	Progress of the 2D Biped along terrain vs. iteration and wall-time.	112
4-11	Progress of the 3D Bulbasaur vs. iteration and wall-time.	112

4-12	A montage of our simulated co-optimized Arm in motion, along with the original parameterization. Green colored actuators are expanding; red, contracting. This problem is impossible to solve without changing the design. The initial arm p[arameterization is less than half the final optimized height.	113
4-13	A montage of our simulated co-optimized Biped in motion, with the first frame showing the optimized material distribution. Green colored actuators are expanding; red, contracting.	113
4-14	A montage of our simulated co-optimized Quadruped in motion, with the first frame showing the optimized material distribution. Green colored actuators are expanding; red, contracting.	114
4-15	A montage of our simulated co-optimized Hexapod in motion, with the first frame showing the optimized material distribution. Green colored actuators are expanding; red, contracting.	114
4-16	A montage of our simulated co-optimized Octoped in motion, with the first frame showing the optimized material distribution. Green colored actuators are expanding; red, contracting.	115
4-17	Convergence of the 3D arm reaching task for co-design vs. fixed arm designs. The fixed designs take significantly longer to optimize than with co-design; it's worth noting that the final total sum-squared actuation cost of the co-optimized 3D arm is less than 75% that of the full 3D arm. Constraint violation is the norm of two constraints: distance of end-effector to goal and mean squared velocity of the particles.	115
4-18	Progress of the 2D Biped vs. iteration and wall-time.	116
4-19	Progress of the 3D Quadruped vs. iteration and wall-time.	117
4-20	Progress of the 3D Hexapod vs. iteration and wall-time.	117
4-21	Progress of the 3D Octoped vs. iteration and wall-time.	118

- 5-1 Our algorithm learns a latent representation of robot state which it uses as input for control. Above are velocity field snapshots of a soft 2D biped walker moving to the right (top), the corresponding latent representations (middle), and their reconstructions (bottom) from our algorithm. In each box, the x (left) and y (right) components of the velocity fields are shown; red indicates negative values, blue positive. 124
- 5-2 At each step of our simulation, the following procedure is performed. First, the unreduced state is fed into an observer function — centroids of a segmentation, as in [65], or, as we demonstrate in this section, an automatically learned latent space. Regardless, the observer outputs features to be processed by a parametric controller, which converts these features to actuation signals. Finally, the actuation is fed into our MPM simulator, which performs a simulation step. The entire pipeline is differentiable and therefore we can compute derivatives with respect to design variables even when executing the workflow for many steps. 126
- 5-3 **Left:** The architecture of our convolutional variational autoencoder. The autoencoder takes in 2-channel pixel grid data as input, with each channel representing the x or y velocity field at that pixel. We use five layers of strided convolutions followed by ReLU operations. This is followed by a final flattening operation which coalesces the weights into latent variables. The latent variables parameterize Gaussians, used in our variational loss ($\mathcal{L}_v$). The architecture is mirrored on the opposite side (without a final ReLU, to allow for negative outputs). The 3D version is completely analogous, but takes in 3-channel voxel grid velocity field data and applies 3D convolutions. Above, the filter sizes are specified with K , and the strides are specified with S . **Right:** At inference time, simulation data is fed into the encoder, $\mathcal{E}$, which produces a latent $[\mu, \sigma]^\dagger$ vector. The mean μ variables are then fed as inputs to the controller, $\mathcal{C}$ 129
- 5-4 A montage of our simulated optimized arm in motion. Green colored actuators are expanding; red, contracting. 132

5-5	A montage of our simulated optimized Elephant in motion. Green colored actuators are expanding; red, contracting.	132
5-6	A montage of our simulated optimized Bunny in motion. Green colored actuators are expanding; red, contracting. This is a particularly challenging problem that cannot be solved with zero error.	133
5-7	Progress of robot performance vs. optimization iteration, along with drawings of our 2D demos. Each black rectangle denotes an actuated region; in precision tasks, regions denoted with black X 's are those which must reach target locations, denoted with green circles.	133
5-8	The average reconstruction loss $\mathcal{L}_R$ per pixel for the 2D Biped scaled as measured in average pixel distance vs. stochastic gradient descent step, demonstrating that not only does our algorithm converge in objective value, but also in model learning.	134
5-9	A montage of our simulated Co-Optimized Biped in motion, with bluer regions corresponding to stiffer materials. Green colored actuators are expanding; red, contracting.	134
5-10	A montage of our simulated Co-Optimized Lanky Biped in motion, with bluer regions corresponding to stiffer materials. Green colored actuators are expanding; red, contracting.	135
5-11	A montage of our simulated Co-Optimized Stubby Biped in motion, with bluer regions corresponding to stiffer materials. Green colored actuators are expanding; red, contracting.	135
5-12	Four further robot demonstrations of we present montages.	135
5-13	A montage of our control optimized Rhino in motion. Green colored actuators are expanding; red, contracting.	136
5-14	A montage of our control optimized Hemisphere Quadruped in motion. Green colored actuators are expanding; red, contracting.	137
5-15	A montage of our control optimized Boxy Quadruped in motion. Green colored actuators are expanding; red, contracting.	137

5-16	A montage of our control optimized Hexapod in motion. Green colored actuators are expanding; red, contracting.	138
5-17	Two suboptimal clusterings for the bunny with different k values. For $k = 5$ (left) the upper arms are clustered together; for $k = 10$ (right) clusters overemphasize the importance of the body compared with the feet or arms.	138
5-18	After Episode 1	139
5-19	After Episode 10	139
5-20	Visualization of the latent space of the autoencoder (2D Biped) with 10 latent variables. Each row represents a (normalized) decoded output for a one-hot latent feature, varied from -1 to 1 . As the algorithm proceeds, the latent features become more descriptive. Formatting per box is the same as in Fig. 5-1.	139
5-21	Progress of robot performance vs. optimization iteration for the 2D Elephant example.	141
6-1	A) The foundation of our models is a Particle Sparsifying Feature Extractor (PSFE) which takes as input the full, dense sensory information (left) and extracts a global feature representation (right) from a sparse subset of the inputs. Our model simultaneously learns this representation and sparsification of the input. Since the input is an unordered point cloud, the PSFE also maintains order invariance through shared feature and point transformations as well as global pooling operations. We employ the PSFE on various complex tasks: B) Supervised regression and classification of object characteristics from grasp data. C) Learned proprioception by combining PSFE with a variational decoder network. D) Learned control policies for a soft robot.	147
6-2	Learning sparsification. A 1D sigmoid function independently parameterizes the probability of each sensor location with weight W_i . Sampled gradients evolve W_i toward $\pm\infty$, pushing the corresponding θ_i probabilities toward 0 or 1, sparsifying the candidate sensor set.	152

6-3	Results for object classification (A-D) and stiffness regression (E-G) tasks. (A) Convergence of the loss for $\tau = 5, 10, 20$ on the sparifying algorithm, with comparison to baselines and full sensorization. (B) Confusion matrices for our co-learning algorithm across different sensor budgets. (C) Evolution of sensor probabilities over training time, validating that each θ_i converges toward 0 or 1, with final sensor placements shown in (D) . (E) Prediction accuracy for the stiffness regression task, comparing the baseline (top) to our algorithm (bottom). (F) Evolution of sensor probabilities. G Optimized sensors have more task-relevant locations.	152
6-4	Exemplar frames of the soft gripper manipulating the four shape classes. . .	153
6-5	Exemplar frames of the soft gripper manipulating four disks of different stiffnesses.	153
6-6	Left: Visualization of the effects of sensor locations (large green circles) on the 2D elephant with different sensor placements (columns) over five exemplar poses (rows). Frames show the reconstruction error with all but the shown sensor turned off. Brighter colors indicate larger deviation of the reconstructed robot (blue) from ground-truth (red). Sensors lower reconstruction errors in their immediate neighborhoods the most. The rightmost column shows all five sensors turned on, yielding the best reconstructions. Right: Latent space of the 2D elephant. Columns represent different latent dimensions of the elephant, generated by one-hot latent vector activations. Each column ranges latent activations from -1.0 to 1.0 . Redder particles indicate higher speeds in x ; bluer particles indicate higher speeds in y	155
6-7	Example reconstructions from a 2D Arm proprioception co-learning example. Sensors are shown as large, reddened circles. Points near the base are most important to track as they kinematically affect everything above them. All coordinates are normalized. Green channels represent the velocity in x (higher velocity is greener); blue channels represent the velocity in y (higher velocity is bluer.	156

- 6-8 Example reconstructions from a 2D Biped proprioception co-learning example. Sensors are shown as large, reddened circles. Points on the legs are the most important to track as they have the most dynamic motion. All coordinates are normalized. Green channels represent the velocity in x (higher velocity is greener); blue channels represent the velocity in y (higher velocity is bluer). 156
- 6-9 Example reconstructions from a 2D Elephant proprioception co-learning example. Sensors are shown as large, reddened circles. Points on the legs are the most important to track as they have the most dynamic motion. Points near the base of the trunk or on the trunk are most important to track as they comprise the most dynamic portion of the robot. Only two sensors are reserved for the more static base. All coordinates are normalized. Green channels represent the velocity in x (higher velocity is greener); blue channels represent the velocity in y (higher velocity is bluer). 157
- 6-10 Example reconstructions from a 3D Arm co-learning example. Sensors are shown as large circles. All coordinates are normalized. Red channels represent the velocity in x (higher velocity is redder); green channels represent the velocity in y (higher velocity is greener); blue channels represent the velocity in z (higher velocity is bluer). 164
- 6-11 Example reconstructions from a 3D Quadruped co-learning example. Sensors are shown as large circles. All coordinates are normalized. Red channels represent the velocity in x (higher velocity is redder); green channels represent the velocity in y (higher velocity is greener); blue channels represent the velocity in z (higher velocity is bluer). 165
- 6-12 Example reconstructions from a 3D Hexapod co-learning example. Sensors are shown as large circles. All coordinates are normalized. Red channels represent the velocity in x (higher velocity is redder); green channels represent the velocity in y (higher velocity is greener); blue channels represent the velocity in z (higher velocity is bluer). 166

6-13	A montage of the 2D Biped, controlled purely from strain sensors, colored in blue.	167
6-14	Our wearable glove system. Resistive sensing and fluidic sensing combine to provide signals for strain and contact force information, respectively. These signals are fed into a pre-trained neural network architecture for inference, which is trained from captured data labeled with ground-truth knowledge. Depending on the neural network used, the output of the network provides inference for a wide variety of downstream tasks, which we demonstrate. Electrode points which are usually not visible have been colored and enlarged in the left-hand images for clarity.	170
6-15	a) Characterization of the conductive knit to applied strain in both directions; b) Characterization of the conductive knit to external and inter-glove contact forces; c) Exemplar time series of the response of the fluidic tube sensor to a fixed force and normal pressure; d) Exemplar time series of the response of the fluidic tube sensor to a fixed strain.	173
6-16	a) Recipe for the fabrication of our wearable glove; b) The layout of the sensors on the glove (top) and the fabricated glove including sensor integration; c) System architecture diagram showing the interactions between key components; d) Raw exemplar time series data from both the resistive and fluidic sensors for a variety of different hand poses.	175
6-17	a) Training, test and validation data for pose estimation and accuracy of the reconstruction across the different joint types; b) The different joints and their types; c) Different hand poses when wearing the glove and corresponding visualizations of the the reconstructed positions, along with our kinematic and biological hand model; d) Measured heart rate spectrum analysis, and comparison of our measured heart rates to those of a Fitbit.	180

6-18	a) Holding Force Experiments; b) Evolution of the loss function for training, test, and validation sets, as well as real-world performance (in terms of average error) for Holding force estimation; c) Temperature experiments; d) Evolution of the loss function for training, test, and validation sets, as well as real-world performance (in terms of average error) for Holding force estimation.	183
6-19	a) Conductability Experiments; b) Evolution of the loss function for training, test, and validation sets, as well as real-world performance (in terms of average error) for conductability estimation; c) Stiffness experiments; d) Evolution of the loss function for training, test, and validation sets, as well as real-world performance (in terms of average error) for stiffness estimation.	186
6-20	a) Our object classification dataset, with associated grasps. Evolution of the loss function for training, test, and validation sets, as well as real-world performance (in terms of correct classification probability) for a) Grasped object classification; b) Handwriting recognition classification.	188
7-1	Row 1: The rest and target poses for the Fighter character; Row 2: The cable network along with realized simulated poses; Row 3: The fabricated character reaching the desired poses.	193
7-2	Row 1: The rest pose of the gripper; Row 2: The cable network along with realized simulated poses; Row 3: The fabricated gripper reaching the desired poses in order to pick up two differently shaped objects.	201
7-3	Row 1: The rest pose of the hand, along with several desired target poses; Row 2: The cable network along with a subset of the realized simulated poses; Row 3: The fabricated hand reaching various combinations of desired poses in order create a variety of finger poses.	202

List of Tables

3.1	The motors from our selected Dynamixel motor library, ordered by increasing torque and size. All motors consume 12 V and have a top speed of approximately 2π rad / s. *Though this motor's Y extents are 0.0356, we model it as 0.0405 in order to keep the Y extents monotonically increasing.	83
3.2	Results of the simulated Hexapod optimizations for two β values. Parentheses denote standard deviations. Headers of parameter columns denote lower bound / upper bound / initial value. The task was feasible for 7 of the 10 initializations.	84
3.3	Results of the simulated Biped optimizations for the flat and step terrain. Parentheticals denote standard deviations. Headers of parameter columns denote lower bound / upper bound / initial value.	86
5.1	The mean backward progress remaining from retraining after a single optimization iteration on the 2D Biped locomotion task, with corresponding standard deviations. A negative value indicates a decrease in the distance traversed. As can be seen, the backward progress is a very small negative number, or positive in all cases, indicating that a single optimization almost completely reverses the adverse effects of retraining.	136

6.1	Average minimum reconstruction errors (lower is better) of our six robots for three sensor combinations, for our adaptive method, for the random sensor placement baseline, and where applicable, for the human baseline. For our adaptive method, minima were taken only once the training reached the clamped, fine-tuning phase. Our method always outperforms the random and human baselines and typically beats them by a significant margin; bolded results indicate where our algorithm outperforms the random baseline by at least a standard deviation on either side. An estimated “best” reconstruction error for our architecture is presented in the far right columns, computed by training with a sensor on each target particle.	162
6.2	Performance scores (lower is better) for manipulation tasks on both classification (object detection) and regression (material stiffness) with varying number of sensors. Our method significantly outperforms both random as well as human labelling baselines.	163
6.3	P-values from a paired t-test comparing our adaptive algorithm to the random baseline approach.	163
6.4	Average minimum reconstruction errors (lower is better) of our two most dynamic robots for three sensor combinations with 1% Gaussian noise added, for our adaptive method, for the random sensor placement baseline. Analysis method is the same as in the no-noise case. Our method always outperforms the random and human baselines; bolded results indicate where our algorithm outperforms the random baseline by at least a standard deviation on either side.	163
6.5	Mean and standard deviations of joint error for LSTM-based reconstruction, in degrees.	181

Chapter 1

Introduction

Nature demonstrates incredible biodiversity, filled with highly capable animals that can efficiently walk, run, fly, jump, swim, and more. Each organism is optimized to fill its niche. Core to what makes these organisms so capable is their complexity. Skeletal and integumentary systems encode topology and materiality, enabling for efficient locomotion and dexterous manipulation; peripheral nervous systems with dense, tuned arrays of neurons allow for awareness and reasoning about external stimuli; musculoskeletal systems provide catered actuation systems to the tasks those organisms expect to perform. All of this is tied together with a central nervous system, capable of proprioception, fine motor control, and task planning, allowing these creatures to solve complex tasks such as hunting, tool use, and creative problem solving.

Compared to these organic marvels, modern robots pale. Typically rigid, clumsy, and lacking robust motion, these creations are stifled by their simplicity; further, the most successful types of designs are limited to a small set of mature working paradigms that have taken decades to develop. While robots have seen tremendous progress over even the last ten years (see, *e.g.* Boston Dynamics's Atlas robot, which has progressed from harness-constrained running to untethered parkour), these robots still leave much to be desired. There are many tasks which are simple for animals that are difficult for robots; for example, robust tactile manipulation, energy efficient running, failure recovery, terrain adaptation, and higher-level task planning. If it takes ten years to progress one type of morphology (humanoids) from walking to running, must we wait another ten years before

that same robot is capable of tasks such as climbing, another ten years for higher order tasks such as furniture assembly, and another ten years for yet even more complex tasks such as disaster relief? And, what about robots in domains which require radically different morphologies, such as deep sea exploration, surgery, or microrobots for drug delivery? Will it be decades per morphology per application? Suddenly, a robot-filled future feels very far away!

But much like nature, robotics should abhor a vacuum. If we wish for modern robots to approach nature both in both in capability and diversity, with different designs filling different niches in society, then faster design cycles are needed. This thesis presents algorithmic, computational tools focused on automated robot design. The goals here are threefold. First, by shortening robot *design* cycles, we can accelerate the rate at robot engineers can iterate, leading to faster *product* cycles. Second, by leveraging more powerful computational reasoning algorithms, we can design robots that are more efficient for target tasks than fully manually designed robots. Third, by automating aspects of robot design and fabrication that previously required human-intensive labor, we can free up man-hours for the conceptual design tasks which require more intense human creativity. The end result is a new, more automated robot design process that, when coupled with human designers and engineers, will lead to more diverse and interesting robot forms which are more capable at more tasks, designed more quickly than ever before.

1.1 Vision: Task-Based Robot Design

Given a target task or set of tasks, such as “move forward as fast as possible,” “robustly grasp a target object,” or “agilely turn in place,” we desire an algorithm which automatically synthesizes candidate robot designs that are optimal for that task. Robot design spaces may be specified parametrically or nonparametrically, and these design spaces cover morphology and behavior. The ideal algorithm takes a mathematically specified task as input and provides the task-optimal fabricable robot morphology and control program from the specified design space as output.

The vision of automated robot design workflows is a far departure from modern robot

design strategies. Presently, robots are typically developed as such. Given a task or set of tasks a designer wants a robot to perform, the robot outlines a high-level view of the robot's morphology. With task and a conceptual design of morphology in tow, the designer selects a set of actuators. The actuator selection drives the scale of capabilities the robot will be able to perform, both in payload and precision. Once actuators are chosen, a topology and concrete morphology is created. The robot is then fabricated and tested, and controllers are synthesized; sensors may be added if necessary. If the robot does not perform as desired at a given stage, earlier aspects of the design may need to be revisited.

In this workflow, use of computation is typically limited. Computer-aided design software may be used, but such software will typically only allow engineers to represent geometry and sometimes explore linkage kinematics. Simulation, which may require its own software platform, is usually restricted in application. It can be used for verification once a design has been synthesized, and sometimes for generated model-based control strategies. But, simulation is rarely viewed as a first-class citizen in the design loop as a means of synthesizing the design. For that, engineer experience and intuition is typically relied on. Of course, this leads to a trap that limits robot functionality — if all future designs are based on a set of paradigms that are known to work, then we seldom explore more experimental designs with high risk but high potential. Of course, this hesitancy toward design exploration is not without merit — as described, modern workflows are hardware intensive, making them expensive in both time and human labor. Each failure is expensive.

In order to promote and facilitate the design of more creative and capable robot designs, we envision workflows in which robots are designed not in a spiraling fashion, but rather a concurrent one.

In current workflows, each domain of robot design is considered (mostly) independently. In practice, however, each aspect of the robot is interrelated. Changing actuators may require changing the size of the housing chassis; changing robot geometry will require reconsidering robot kinodynamic control; changing control schemes may inform more optimal actuator design, and so on. As each aspect of a robot's design changes, the others have to be revisited, and it can take many iterations until a final design is converged upon.

In our envisioned concurrent design (or “co-design”) workflow, each aspect of a robot's

design is considered simultaneously. Further, the robot is *co-optimized* over each aspect of its design *in silico* by algorithms which are capable of jointly reasoning about them. These algorithms should be able to more quickly find satisfactory designs which are capable of solving specified tasks, and find designs which are more optimal by finding designs where the different aspects of the robot work in harmony rather than antagonistically with each other.

This thesis considers both rigid and soft robots for static and dynamic tasks, and each of these robots has a number of different domains to consider. Each domain of a robotic cyberphysical system (roughly) falls into one of two categories: “body,” or the physical part of the robot, and “brain,” or the cyber part of the robot. For rigid robots, the body is comprised of the robot topology, geometry and inertial parameters, actuator design, and, in some cases, sensing. For soft robots, the body is comprised of shape, spatially varying materiality, actuator design, and sensor design. In both rigid and soft robots, the “brain” is responsible for control, task planning, and higher level reasoning; soft robot brains must also reason about proprioception. Hybrid robots (both partially rigid and partially soft, like vertebrates) are not considered in this thesis, but would include all aspects of both rigid and soft robots. Ultimately, robot design algorithms need to consider many domains, some static, some time-varying, some with few variables, some with many variables, and each affecting system kinematics and dynamics at different scales. It should be obvious that this sort of complexity not only makes algorithms for robot design difficult to formulate, but also extremely worthwhile.

As we will see, the types of co-optimization problems solved by the algorithms in this thesis are difficult for a number of reasons. First, since we desire certificates that the robots generated by these algorithms will perform well, they are model-based. This means that evaluative simulations must be run *in-the-loop* of optimization algorithms. Since simulations can be slow, these algorithms need to be efficient. Second, these optimization problems are highly nonlinear, and so naïve optimization strategies rarely work. To find good solutions to these complex problems, we need to formulate algorithms cleverly, or look to more complex, data-driven, learning-based searches. Finally, since we wish for the resulting robot designs to be fabricable, we must rely on realistic models that are at least fabrication-aware.

In summary, put concretely and succinctly, this thesis proposes co-optimization solutions to co-designing rigid and soft robots across various design domains. Given a target task or set of tasks, such as “run forward,” “grasp an object,” “reach a target pose,” *etc.*, we desire algorithms which can efficiently provide the best robot design to complete that task. Specifically, the proposed algorithms seek to marry the “body” (topology, materiality, actuators, sensors) and the “brain” (proprioception, control, and task-planning) in a single workflow, leading to a single pipeline for the fully automated design of rigid and soft robots. If these resulting designs are predictive, meaning they can be realized in physical reality, then this severely cuts the need for slow, spiraling iteration or expensive prototyping. **The thesis of this *thesis* is that algorithmic tools can be used to vastly accelerate the design and fabrication process of (rigid and soft) robots in a single CAD-CAM workflow by jointly reasoning about all aspects of robot design in a single physically predictive pipeline.**

1.2 Comparison To State of the Art

As described, robot design has traditionally been an experiential field, focusing on iterative fabrication-based prototyping of robots. Computational modeling and simulation have been used sparingly in the past.

In recent years, computational design and fabrication has emerged as a rapidly growing field that has grown out of the advent digital fabrication platforms and the increased computational horsepower of modern workstations. The last decade has seen a proliferation of techniques for computationally designing functional, physical objects, such as spinners [11] or self-supporting structures [36]. These capabilities had started to proliferate into the space of computational robot design, but were limited in three ways.

The first limitation had been in computational method. Most historical computational design algorithms for robots have been evolutionarily-based [129, 23]. While these algorithms demonstrated enormous promise for morphological search of robots, they were inefficient, requiring a large number of iterations or massive parallelization across CPU clusters to produce results. Further, these methods presented no guarantees of optimality.

Worse, these methods were typically ill-suited to controller design, and control optimization is thus limited in that line of work. The model-based numerical optimization and learning schemes presented in this thesis have increased efficiency and wider applicability than their evolution-based ancestors.

The second limitation had been in design domain. Most computational techniques were limited to structures that were purely kinematic in nature [27], sometimes moving as far in physical modeling as static physical analysis [130]. Obviously, this limits the types of tasks that such mechanisms can perform, and removes the need to consider the (exceptionally) complex domain of control. This thesis will deal extensively with dynamic robots (and some static analysis), and discuss the role of control in the co-optimization problem at length.

The third limitation has been in realization. In the instances where complex virtual characters have been designed [157], fabricability has often been ignored. Models have been simplified for computational convenience, focusing on physical plausibility rather than on being physically-based. All of the work here has been grounded in physically-based models, with fabricability being considered and promoted throughout.

In summary, compared with previous work, the problems we consider in this thesis extend over more domains than previous work and target more complex systems. Meanwhile, the solutions we present are often more efficient, optimal, or physically realistic (and useful for fabrication) than previous techniques. Throughout each chapter of this thesis, we will present related work and comparisons to state-of-the-art techniques in more detail.

1.3 The Future of Robot Design

This thesis presents algorithms for task-based co-optimization of fabricable rigid and soft robots over multiple different design domains. Such a problem requires reasoning about modeling and simulation, optimization-based design, fabrication, and problem specification. These are scenarios with many intricacies and complexities. In order to make reasoning about them more tractable, we turn to the ideas of abstraction that have been so successful in computer science. Rather than reason about every aspect of the robot design problem simultaneously, we formalize the capabilities needed for computational robot design into a

robot design stack. The computational stack has two core features. First, each layer of the stack is modularized, such that it only needs to know about the outputs and requirements of the layers below and above it respectively. Each layer efficiently leverages information from the layer below it in order to produce output for the layers above it. Second, when specified properly, layers of the stack with the same inputs and outputs can be modularly swapped in and out without requiring modification of other layers of the stack; however, the ability to swap in and out different instances of a layer can help improve performance with respect to a target application. We will discuss these two intricacies in more detail in an example later in this section.

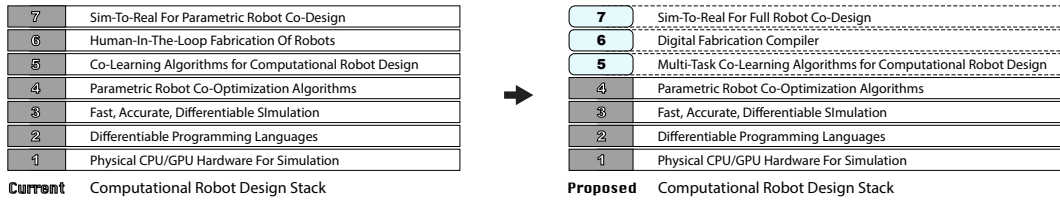


Figure 1-1: **Left:** The computational robot design stack. The first two layers provide CPU/GPU hardware and languages catered to fast, differentiable simulation (layer 3). Co-optimization algorithms (layer 4), use the gradients computed from simulation to optimize continuous design parameters. This local co-optimization scheme is used as a black-box in more sophisticated learning-based design searches (layer 5). The finalized design is compiled to a digitally fabricable format (layer 6), which is guaranteed to closely match the simulated version *via* the sim-to-real layer (7). **Right:** The proposed design stack incorporating multi-task optimization and digital fabrication, with changed layers in cyan.

The robot design stack is presented in Figure 1-1. Each layer builds upon the one below it. Layer 1 is computer hardware in which robots are design *in silico*. Layer 2 is differentiable computer programming languages such as TensorFlow, PyTorch, or DiffTaichi [1, 112, 63] which make it easy to implement the differentiable models in layer 3 (but also can provide functionality for layer 5). Layer 3 is differentiable simulation, which uses the languages presented in layer 2 in order to provide fast, inner-loop simulation that can also provide gradients. Layer 4 is gradient-based model-based co-optimization algorithms, which provide a means of co-optimizing over various (continuous) robot design domains; obviously, layer 4 requires the differentiable simulation of layer 3. It provides co-optimization techniques that are used by layer 5. Layer 5 is learning-based co-design. This layer uses data generated

during the optimization procedures in layer 4, and uses these optimization layers for updating learnable models. As output, it provides a purely virtual model of the robot that is to be fabricated. Layer 6, the fabrication layer, provides a way of turning the virtual designs from layer 5 into physical robots, and provides physical data to be used by layer 7. Layer 7 uses the data generated from the physical robot to fine-tune control and sensing models in order to achieve performance that matches the virtual simulations, overcoming the simulation and modeling gaps. As can be seen, the layers are carefully ordered such that they immediately depend on the layer immediately below and provide black-box methods for the layer above to exploit.

Figure 1-1 also shows the capability of layers of the stack to be swapped in and out, for different classes of problem. The instance of the stack on the left is presented for a problem in which the goal is to optimize a robot for a single task, would be fabricated by hand, and where the control and sensing architectures are parametric in nature. The stack on the right shows how three layers can be swapped out to change the functionality of the design stack. Layer 5, for instance, can be swapped from a single-task learning algorithm in order to create robots that are capable of multiple tasks. Layer 6 can be swapped from creating instruction for manual assembly to a file for digital fabrication. Layer 7 can be swapped from a sim-to-real algorithm that reasons about robots with parametrically specified controllers to one that works for robots with nonparametric control (*e.g.* from neural control to model-predictive control). Each layer still provides the same outputs and receives the same input, and they thus can be swapped independently of one another.

This thesis contributes the following solutions and results for layers 3-6 specified above (with layers 1, 2, and 7 being outside the scope of this thesis).

Differentiable simulation preliminaries for rigid robots will be presented in Chapter 2, while fast, accurate, fully differentiable simulation for soft robots is examined with particular attention and presented in Chapter 4. The presented solution is a GPU-accelerated, physically-based material point method algorithm with a modular API for specifying soft robot morphologies, controllers, actuators, and sensors. Soft robots are given special care in this chapter, since fast, differentiable rigid robot simulators have been covered extensively by other preceding and concurrent work [145, 32, 33, 42].

Co-optimization algorithms which use symbolic and differentiable simulation are presented in Chapters 3 and 4. Specifically, we present three algorithms for three different types of robots. First, we present methods for co-optimizing dynamic rigid robots over parametric design, open-loop control, and actuator selection. Then, we present methods for co-optimizing dynamic soft elastic robots over closed-loop control and spatially varying material stiffnesses.

Co-learning algorithms are presented in chapter 5 and 6, which can reason about aspects of robot design problems that cannot be completely optimized from gradient-based methods. This is either because the domains being optimized over are discrete, the representations nonlinear enough that they require many iterations for optimization-based convergence (and thus cannot be optimized by simulations in the loop), or the system is difficult to model in a differentiable manner. In particular, we present learning-based solutions for proprioceptive modeling of soft robots (which is iteration-hungry) and sensor design (which is both iteration hungry and, depending on the sensor inputs, difficult to differentially model). These methods leverage capabilities from prior layers. Examples of learning over discrete domains is present in other concurrent work [168].

Fabrication results are presented in chapter 3, where we present physical demonstrations of our co-optimization results by fabricating rigid dynamic robots, while an in-depth analysis of soft sensing fabrication is presented in chapter 6 following our sensor learning results.

While most of the results presented in this thesis present a single method for a particular layer of the stack, we briefly walk through one end-to-end system, titled Interactive Robogami, that relies on an implementation of each layer 3 through 6.

1.3.1 Interactive Robogami — A First Demonstration

This thesis is a step toward democratizing the design and fabrication of robots through intuitive tools and user interfaces. Suppose Alice wishes to design a legged robot for managing her yard. To do so, Alice heads to a 24 hour robot manufacturing service where she can explore the design space using an intuitive interface. Once she settles on the design, the store will fabricate the robot and its control and programming environment.

Inspired by this vision we developed a system called Interactive Robogami. Specifically, as a first demonstration of the type of systems that will exemplify the future of robot design, we present Interactive Robogami as a motivating Example. Interactive Robogami is an end-to-end CAD-CAM system; a self-contained robot design system that handles everything from ideation to conception. The system guides users through composition-based design and parametric optimization of fully fabricable robots. Users are given a collection of fabricable components; these components are parameterized in such a way that they can be composed, with component geometry automatically changing in order to make connections consistent. Users compose a robot with desired geometry, interactively altering design parameters along the way. As users resize the robot parts, consistent connections and other high-level constraints (such as leg spacing, to avoid self-collisions) are maintained. Users may also design and optimize robot locomotion sequences from parametric motion primitives; recommendations are provided to users based on the count and shape of robot legs and wheels. Real-time previews of robot motion are provided as users explore geometric and motion parameters. As users explore the space of geometric parameters (with the control scheme held fixed), the system provides recommendations on how to alter those parameters in order to improve the selected objective (users may select from nine objectives in total). Alternatively, users may opt for the system to take over, automatically optimizing the current parameterized design to its local optimum. When users are satisfied with their design, they simply press the “export” button, and a complete cyberphysical design plan — .stl files to 3D print, a wiring diagram, and C code to upload onto a microcontroller — are outputted to the user. From there, all that remains is for the robot to be 3D printed, the electronics attached, and the code uploaded to the microcontroller.

Interactive Robogami is a good case study because it implements nearly all aspects of the computational design stack. Simulation is used to preview and measure the performance of in-progress designs. Optimization algorithms are used to optimize robot body geometric parameters for tasks and aid the user in part composition and interactive resizing. A fabrication compiler is used to convert virtual designs into design files for digitally fabricating their physical counterparts, and it is demonstrated that the physical simulation is accurate and predictive. Finally, these components are all tied together in an interactive graphical user

interface that allows natural exploration and conceptual design. Only two aspects of the proposed design stack are not accurately reflected here. First, while optimization algorithms are used in order to improve robot designs and ensure fabricability, these optimization algorithms are only employed across the shape domain. As demonstrated, shape, kinematics and dynamics, and control are tightly interrelated. Thus, ideally, such a system would have the capability of co-reasoning and co-optimizing over both robot body and brain. Second, while there are no applications of data-driven design and learning in Interactive Robogami, there is a clear opportunity — automating the currently human-driven compositional design of both geometry and gait sequences through data-driven part recommendations.

Such limitations, however, only further motivate the need for design-stack like thinking. Treating the capabilities in Interactive Robogami as a stack, one could imagine introducing co-optimization algorithms that replace the geometry-only optimization algorithms currently in place. Learning algorithms could be introduced that can learn optimal robot topologies for tasks and recommend them to users. More accurate and expressive dynamic simulation and motor models could replace the current simplified model. In all such cases, swapping such capabilities changes nothing about the rest of the system, maintaining the clean abstractions by which the layers are defined. We note that Interactive Robogami presents one success story already of the stack-based mentality — a metallic robot (Fig. 1-6), waterjet out of sheet metal and folded was fabricated as a final example. The fabrication instructions were modified from the print-and-fold strategy, presenting cut patterns that can allow similar large-scale folding. These robots are made of materials that can handle more substantial loads, demonstrating the utility and versatility of stack-based thinking and system design.

1.3.2 Design Decisions

One of the main challenges in robot design is the interdependence of the geometry and motion. A typical design process for a robot starts with geometry design, followed by actuator selection, and then controller design. However, minor changes in geometry can drastically simplify actuator or control design. Take, for example, the robot in Figure 1-2. This four-legged robot was designed by a user for deliver ice cream. In this task, speed

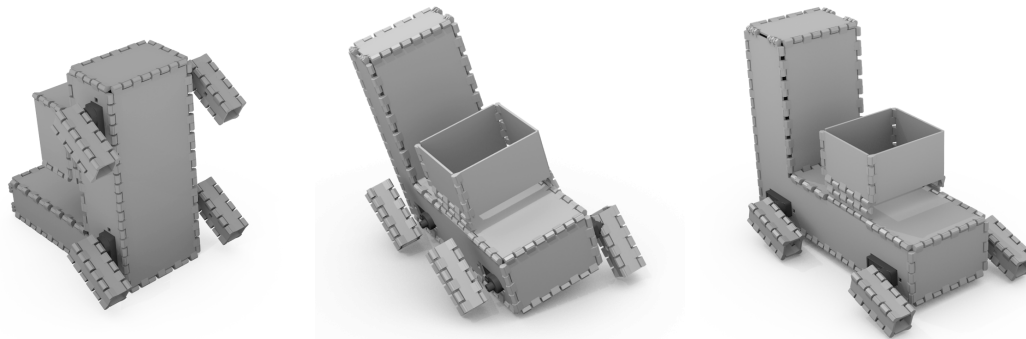


Figure 1-2: A robot design that topples while walking (left) can be modified to follow a gait that only wobbles slightly (center), but changing the geometry (right) allows the robot to move much faster and more steadily.

is essential to ensure the ice cream does not melt. In addition, the robot must be steady enough that the ice cream does not spill on the way to its destination. In general, the fastest way for a statically stable four-legged robot to move is for all the legs to move simultaneously, resulting in a scooting motion. With the user's initial design (left), though, this gait is unstable and the robot topples backwards. The user can find a gait with the left and right pairs of legs moving sequentially that allows the robot to move forward with only a slight wobble (center). However, changing the original geometry slightly allows the robot to succeed with the original gait and maintain its high speed without falling over (right), resulting in a robot that is almost 50% faster.

Interactive Robogami presents an interactive system that allows designers to explore how both geometry and motion affect robot performance, enabling them to make these types of design choices. Our system contains tools for geometry and gait design, as well as simulations for evaluating the model (Fig. 1-3). Users interact with the tool through a graphical user interface (Fig. 1-4) in which they can visualize the models they create and receive real time feedback as to how changes to the design affect the robot's performance. The system keeps track of the geometry and motion in order to output a full fabrication plan once the user is ready to build the design.

Our design choices were driven by three main objectives: 1) facilitate user creativity while maintaining a wide range of accessibility, 2) enable concurrent design of motion and geometry, 3) guarantee fabricability, and 4) provide real time feedback for ground

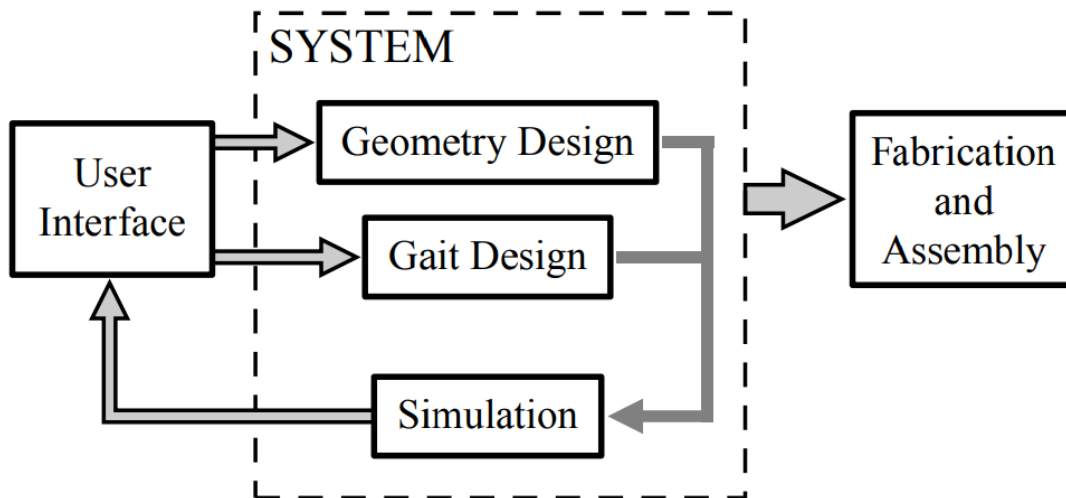


Figure 1-3: System diagram. Users interact with geometry and gait design tools. The designed models are simulated to provide feedback to the user. The user may iterate over the design before fabricating and assembling the model.

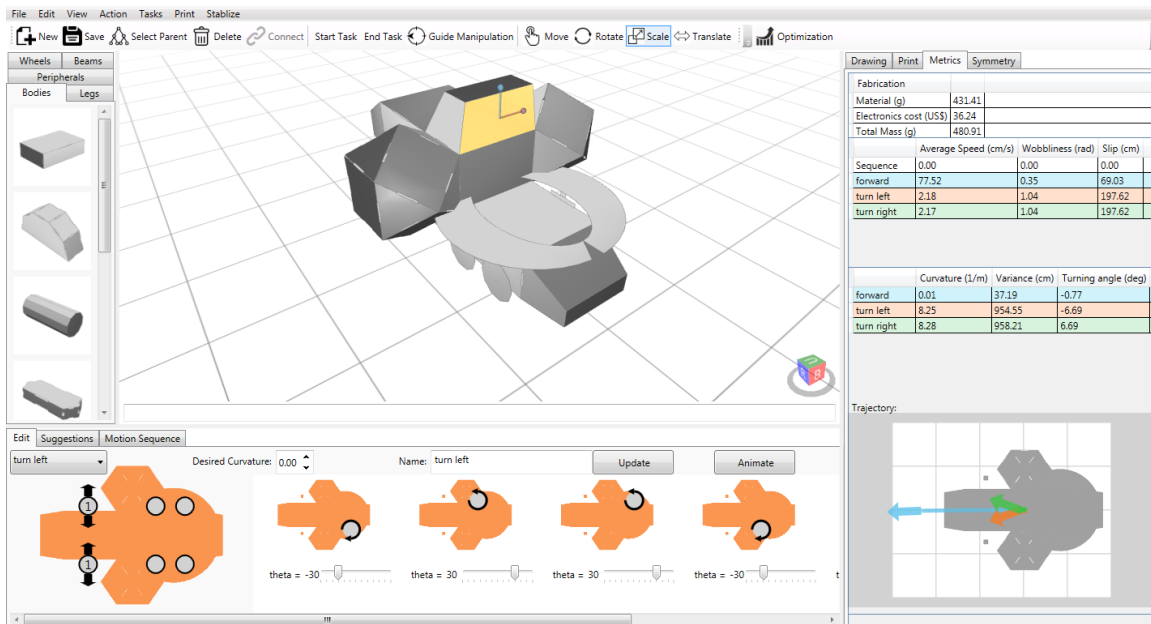


Figure 1-4: User interface. Icons that link to geometry components are displayed on the left and the gait design tool is on the bottom. Users design models by dragging components into the center canvas and editing them. Performance metrics for the design are shown on the right.

locomotion design.

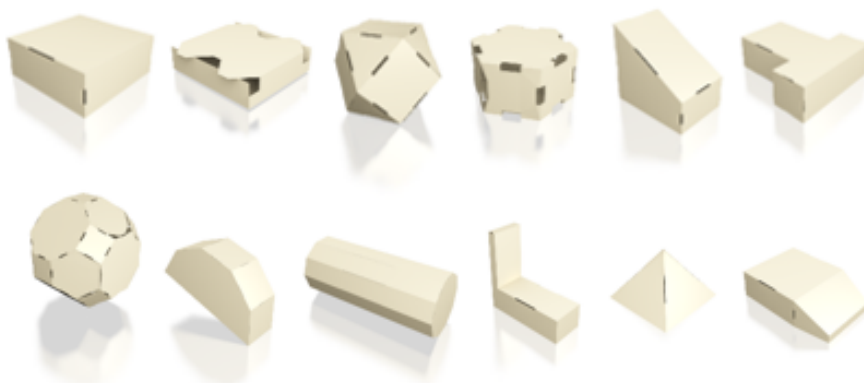
Since the design space of robots is too large to be explored interactively by casual users, we introduce a component library and an assembly-based modeling tool to allow users to combine components in the database to create new robots. This method has been shown to facilitate user creativity while maintaining a wide range of accessibility [49]. The components in the database are parameterized to allow variability while keeping the space of possible designs manageable. A linear parameterization simplifies the data representation and speeds up computation without compromising expressiveness.

In order to address the concurrent design problem, our database contains both geometric parts and motions and is supported by a grammar that dictates composition rules for both these components. To this end we drew many insights from the experience of robotics engineers. We categorized parts into bodies, legs, and peripherals based on the functionality of components in existing robot designs. The joint controllers were chosen based on the physiology literature and parameterized according to when legs were expected to make contact with the ground [6].

All the geometric components can be fabricated using our 3D print-and-fold method and the parameterization is designed so that this is preserved at any configuration. The composition tool uses the parametric representation to snap geometric components together, constraining the composed models to a fabricable set of geometries. The composition rules are used to determine the electronic components needed in the assembly and the parametric motion representation is used for automatic generation of the control software. This allows the system to automatically generate a full fabrication plan for every composed model.

Optimization and simulation tools are used to guide users in realizing a stable ground robot. The parametric representation defines the design space for a fixed topology over which optimization can occur. Nine performance and fabrication metrics are computed in real time when a user creates or modifies a robot and are displayed to the user. Users can optimize designs relative to these metrics by altering geometry, gait, or both. We chose metrics that are commonly of interest to roboticists designing ground behavior. These objectives allow inexperienced users to optimize their designs to follow trajectories, minimize fabrication cost, maximize speed, and overall make more intelligent design tradeoffs than when using

Body



Peripheral



Limb (leg)



Limb (wheel)



Figure 1-5: Geometry components by category. Our system's database contains 45 components: 12 bodies, 23 limbs, and 10 peripherals.

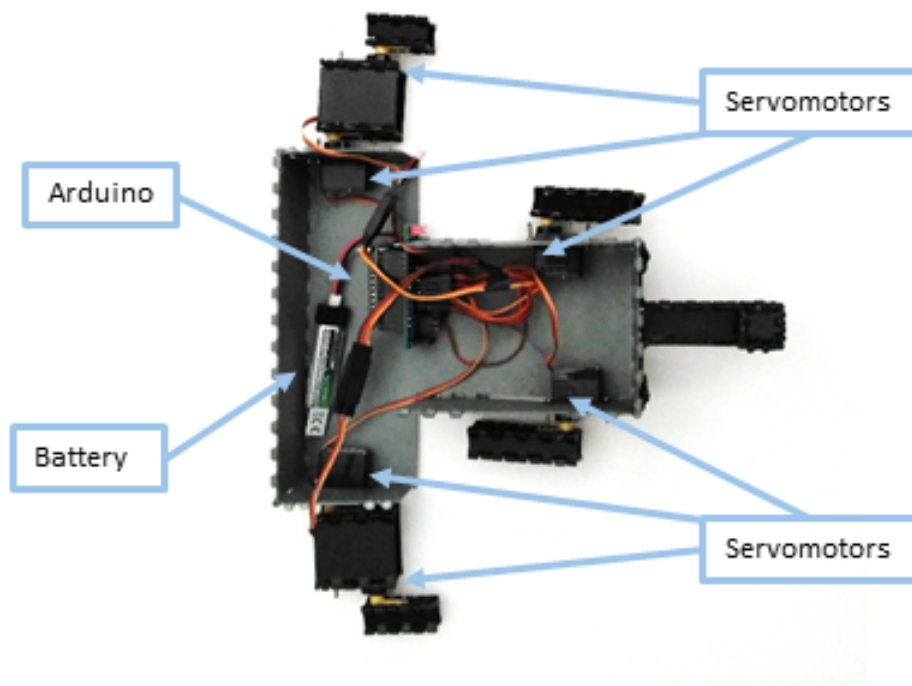


Figure 1-6: The electronics (servomotors, microcontroller, and battery) inside the one example robot designed and fabricated using Interactive Robogami, nicknamed the “Monkey.”

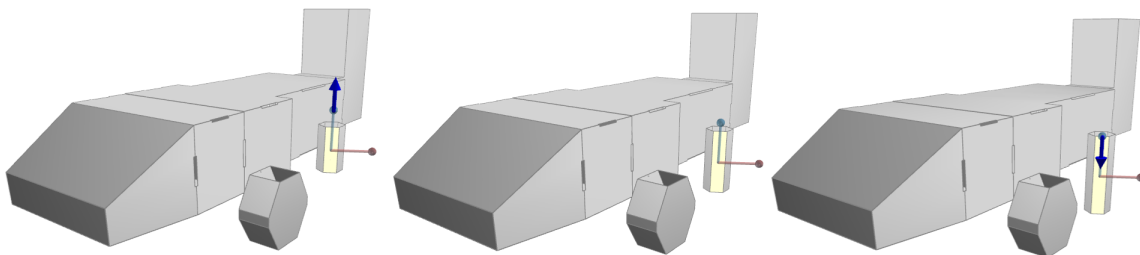


Figure 1-7: The system provides guidance arrows for users who want to make local geometry optimizations. The up and down arrows indicate that the user should lengthen and shorten the leg respectively. No arrow indicates that the part dimension is already at a local optimum.

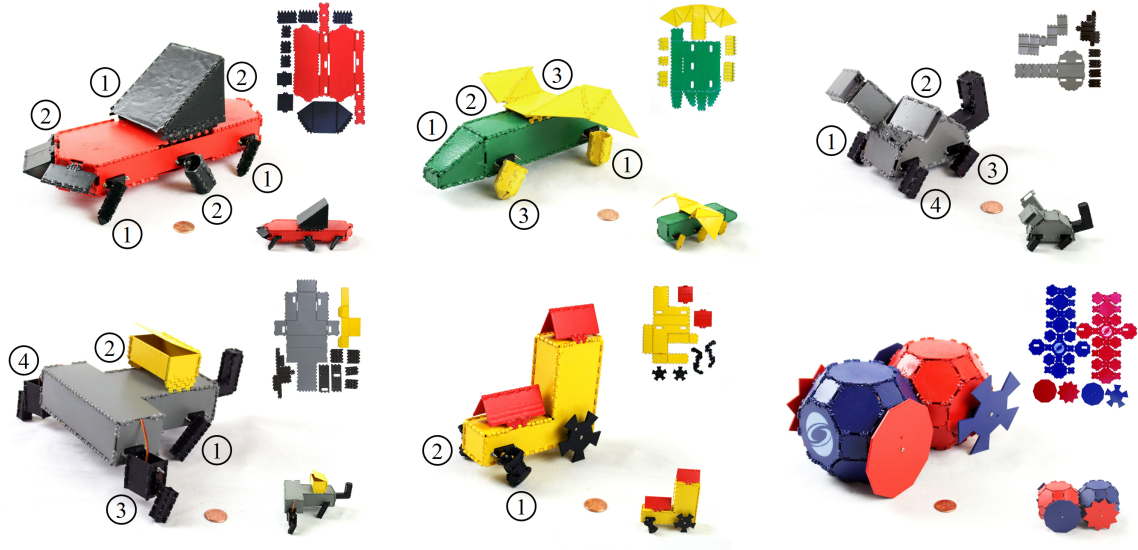


Figure 1-8: Six robots designed and fabricated using the system. Numbers over the limbs indicate the step sequence.

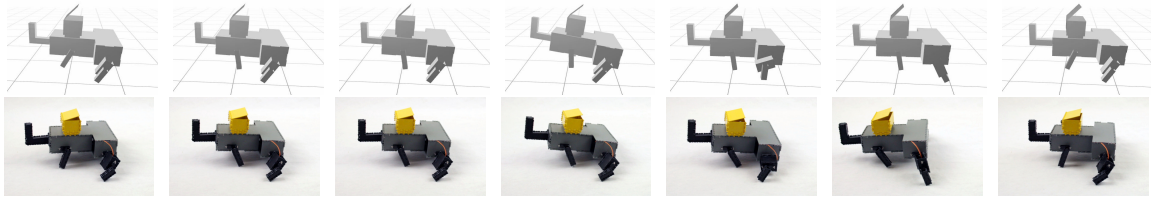


Figure 1-9: Comparison of Monkey gait in simulation and from physical robot. The robot motion matches those from the simulations.

single-metric systems often found in the literature. During a pilot study, we found that users preferred some metrics (*e.g.* speed, turning angle) and found others to be unintuitive (*forward travel*) and we incorporated this feedback into our user interface design.

1.3.3 Outlook

Interactive Robogami presents a case study and in order to help the reader appreciate how components of the robot design stack interplay. Further, we hope such a cognitive walkthrough will help the reader understand the considerations that must be taken in developing such a system, as well as the opportunities and challenges that lie ahead.

However, in the form presented, a system such as Interactive Robogami is, in fact, limited. Robots tend to look more like automata or toys than the functional robots we expect

in our future, unable to handle substantial load or move with great speed. The dynamic behavior of these robots is ultimately very limited, since the simulator used is constrained to limited types of dynamic motion. Further, while useful for human-driven design, Interactive Robogami lacks advanced algorithmic design capabilities, unable to optimize over discrete topology, control, or non-parametric geometry. Finally, the print-and-fold construction of these robots, while fast, lacks any sort of guarantees on maximum structural load, and can sometimes buckle when carrying large contents. These limitations motivate much of the work in this thesis, aiming to help designers transition from exploratory, mostly kinematic design to dynamic, task-optimal design.

1.4 Contributions and Overview

If we want to approach the complexity of organisms found in nature in modern robotics, we will need algorithms which can co-reason over all of the different “organ systems” of dynamical robots: link shape (skeletal system), spatially varying materiality (integumentary system), proprioception and sensor layout (nervous system), and actuator design (muscular system). This thesis focuses on the former three of these systems, although it will briefly touch upon the last of these. Such reasoning capability is not realized by the quasistatic and geometry-focused Interactive Robogami, though it provides a convincing vision of the types of end-user systems we would like to target with these capabilities. The remainder of this thesis will be focused on developing co-optimization and co-learning algorithms for designing each robot organ system – never in isolation, but always combined with capabilities realized by the robot brain; *e.g.* motion planning, control, and reasoning. Contrasted with Interactive Robogami, the bulk of this thesis pays special attention to dynamical robot systems, with extra attention focused on the especially dynamically rich and dimensionally complex class of soft robotics. While this thesis is focused on methods that are proven to work in simulation, some results and demonstrations in fabrication will be discussed throughout.

in this thesis, we contribute:

- A formalization of robot co-design into a robot design stack.

- A method for co-optimizing rigid robots over geometric and inertial parameters, open-loop controllers, and actuator selection rooted in trajectory optimization and powered by a symbolic differentiation system.
- A modular API for soft robot design and differentiable simulation.
- A method for co-optimizing soft robots over spatially varying materials and open- and closed-loop controllers.
- An algorithm for co-designing soft robots over control, material, and latent proprioceptive models.
- A co-learning algorithm for co-designing soft robots over task and sensor placement.
- Methods for fabricating elements useful for soft robot design, with special care paid to textile-based sensing architectures powered by machine learning models.

This thesis provides work on filling out the computational design stack. The remainder of this thesis is organized as follows. Chapter 2 provides simulation preliminaries. Chapter 3 describes work on co-optimization algorithms for rigid robots. Chapter 4 extends the type of thinking presented for rigid robots to the more highly nonlinear soft robots, co-optimizing over high-dimensional material distributions and open- and closed-loop control. This chapter also introduces a differentiable soft robot simulator based on the material point method, which will be used for most of the remaining contributions of this paper. Chapter 5 pushes our co-design algorithms past the regime of pure optimization and introduces learning-based methods to expand our capabilities. In particular, chapter 5 will focus on learning-techniques for co-designing proprioceptive capabilities in the framework of soft robot material/control co-optimization. Chapter 6 extends the work of chapter 5, focusing on learning intrinsic sensorizations that lead to meaningful proprioceptive models, which can be used for soft robot learning and control tasks. Chapter 6 will also introduce a novel fabrication method for creating soft sensor networks that could one day be optimized *via* the sensor design algorithms presented here. Finally, chapter 7 concludes this thesis by summarizing our progress toward the initial vision, describing the foreseen impact of this work, and proposing important future horizons for the field.

1.5 Related Work

Throughout this thesis, related work will accompany the corresponding topics presented in each chapter. However, we conclude this chapter by describing related work in the field of computational design workflows at large, with an emphasis on manufacturing-oriented design, computational design of robots, and co-design.

1.5.1 Manufacturing Oriented Design

Manufacturability has long been a concern in engineering design [31, 57, 74, 113, 158] and has more recently become an area of interest in the computer graphics community. Proposed fabrication-oriented systems include tools for plush toys [105], furniture [81, 123, 127, 151], clothes [152], inflatable structures [130], wire meshes [51], model airplanes [153], twisty puzzles [139], prototypes of mechanical objects [78], and architecture-scale objects [166]. Our work is similar to these efforts in the sense that our tool constrains the design space to ensure fabricability. For fabrication, we use a method that combines 3D printing with origami-inspired fabrication methods: robots are 3D printed as flat faces that are then folded into their final shapes (Deng and Chen, 2013). This method exploits origami-inspired fabrication, which generates low-cost, lightweight structures [21, 46, 47, 56, 62, 85, 98, 101, 141, 165] and combines it with the flexibility of additive manufacturing technologies to produce complex geometries and durable functional joints [8, 17, 19, 95, 147].

1.5.2 Design of Robots and Functional Mechanisms

Design generation work often considers mechanical structure and task-specific movement in isolation, with mechanisms being designed independently of actuation [97, 99, 169]. Although fabrication plans for fully functional robots were presented in [100], the designs were driven by mechanical considerations, and the electrical and software components generated in a post-processing stage. On the other side of the spectrum are studies in generating driving mechanisms that match desired input motions for toy designs [27, 147, 169], robots [97, 132], and other models [48]. The combined geometry-motion design challenge, although acknowledged, is not addressed by these systems.

1.5.3 Concurrent (Co-)Design

Concurrent design of multiple coupled subcomponents is a common challenge when developing complex systems. Proposed methods for tackling this challenge include methods for combined structure and control optimization of mechatronic systems [83, 110, 116], cooptimization of dimensions and gait for a quadrupedal robot [38], cogeneration of mechanics and actuation for printable robots [23, 89], and concurrent design of planar linkage structures and trajectories [9]. These works search for satisfactory designs under space and kinematic constraints. Our work is similar in that my robots require concurrent design of geometry and motion. However, our system handles 3D robot designs and additionally considers the stability, speed, and actuation requirements associated with a ground robot.

Chapter 2

Background And Notation

Any robot design optimization algorithm requires full or partial simulation, often as an inner loop of a step of the design optimization process. This is for two reasons. The first reason, perhaps most obviously, is to evaluate the robot’s performance, in order to test the efficacy of the design optimization procedure thus far. The second, potentially less obvious reason for simulating robots during design optimization is to compute *gradients* of a robot’s performance. Performance gradients are gradients of mathematically specified loss functions which numerically evaluate robot performance, and can be with respect to any state or design parameter. These gradients provide search directions for an optimization procedure. Such gradients are useful in finding solutions that improve objective values, and, in the case of constrained optimization, provide clues for finding a valid, *feasible* robot design with respect to specified design constraints. Simulation which can provide these gradients of objective functions and state variables with respect to design variables or other state variables is referred to as *differentiable* simulation. Most forms of robot simulation can be made differentiable, including rigid-body and soft-body elastic simulation, which are useful for rigid robots and soft robots, respectively.

In this chapter, we provide a brief introduction to fast, accurate differentiable simulation for rigid and soft robotics, which will serve as a cornerstone for the design optimization algorithms in future chapters. In particular, we will discuss two simulators: the open source dynamic rigid robot simulator Drake [145], a modified version of which is used for work on co-optimizing rigid robots, and [66], a GPU-accelerated differentiable soft robot simulator

based in the material point method used for work on co-optimizing rigid robots. Throughout, we will discuss design choices that help avoid modeling and performance pitfalls.

2.1 The Life And Times of A Robot Simulation

In this section, we briefly describe the process of a robot simulation. Though we ground some examples of this discussion in the soft robotics regime, the discussion here is in fact general to any type of robot, be it rigid, soft, hybrid, or other.

A robot simulation has a beginning, a middle, and an end. The beginning of a robot simulation is to *instantiate* the robot. This involves defining the robot’s physical design, which is typically parameterized in some way for optimization, either directly or *via* a function which generates the robot body. In the soft robot case, this includes its geometry, material parameters, actuator placement, and, if relevant, sensor placement. For rigid robots, this involves defining the robot’s topology, inertial parameters, actuator placement/design, and, if relevant, sensor placement. The middle of a robot simulation is to evolve the robot through time, according to the robot’s controllers and its dynamics. The end of a robot simulation is to *evaluate* the performance of a robot, by evaluating an objective function describing how well the robot performed. We note that although this is the traditional setup of a robot simulation, robot evaluation can also be interleaved with the middle part of robot simulation, evaluating an objective function at every time step that is accumulated for analysis. In this case, the objective is referred to as a *running* objective (cf *final* objective). Running objectives are often useful for evaluating quantities such as robot energy consumption *throughout* a simulation, while final objectives are more useful for evaluating quantities such as total robot progress *after* simulation.

The beginning, physical design aspect of robot simulation is a topic covered later in this thesis when discussing parameterizations and algorithms for co-optimizing robots, and the end, evaluation aspect of robot simulation is typically simple but will be discussed at various points throughout this thesis. However, the middle, time evolution aspect of robot simulation is typically the most complex, and is the main focus of this chapter. However, its structure is typically similar across domains. The structure is as follows (and illustrated for

soft robots in Figure 4-4).

At time step i , a robot has state $\mathbf{s}_i$. This state may be very high dimensional, especially in the case of (infinitely) high dimensional continuum soft robots.

First, in order for the robot to reason about this high dimensional state, it is summarized *via* an *observer* function. This observer function translates the potentially high-dimensional state of the robot to what is usually (but not always) a lower-dimensional state, usually from sensory input. For rigid robots, the observer function is often the identity function relying on motor encoder values (though it may also involve complex state estimation when a visual sensing of a rigid robot's environment is involved); for soft robots, it may involve processing strain or pressure sensor input. We refer to the observation state of the robot at time i as $\mathbf{z}_i$. Fig. 4-4 depicts an observer which summarizes the center of mass and center of velocity of structured regions of a soft robot.

Second, given a robot observation, a controller processes $\mathbf{z}_i$ in order to produce an actuation signal $\mathbf{u}_i$. This controller is a function, but is free to ignore the observation input as in the case of open-loop control and may be time-varying. Other common control functions include (time-varying) linear-quadratic regulators and neural networks (such as multilayer perceptron (MLP) controllers). Fig. 4-4 depicts an MLP.

Third, the actuation signal $\mathbf{u}_i$ is transmitted to the robot's actuators, where it is transformed from a cyber signal into physical actuation. For rigid robots, this could be motor torques. For soft robots powered by fluidic actuation, for example, this could be pressures on the actuator walls. Fig. 4-4 depicts a soft robot which translates actuation signals into expansion and contraction of pneumatic chambers.

Fourth and finally, the forces, torques, pressures, *etc.* produced by the actuators are fed into the time-stepping procedure, which updates robot state $\mathbf{s}_i$ to $\mathbf{s}_{i+1}$ according to actuator outputs and environment (terrain). This time stepping procedure is based on the physical dynamics of robot itself. In other words, it is the application of Newton's laws of motion. We will discuss procedures for time-stepping rigid and soft robots later in this chapter. For design reasons or physical limitations (such as maximum sensing or actuator frequency), the time-stepping may occur multiple times before emitting a new state $\mathbf{s}_{i+1}$ to be processed. This process is called "substepping."

The complex middle loop, including substepping is summarized in Fig. 4-4.

For the next two sections of this chapter, we briefly describe formulations of rigid body, and then soft body mechanics and simulation, so that they can serve as a framework for later demonstrations in robot control and co-optimization.

2.2 Rigid Body Simulation

In this section, we provide a brief overview of differentiable rigid body simulation. While not a contribution of this thesis, rigid body kinematics and differentiable rigid body simulation are needed for the rigid body co-optimization algorithms presented in Chapter 3. In this section, we provide a brief recap of rigid body kinematics, before discussing the Manipulator Equation formulation of rigid body dynamics and rigid body contact.

2.2.1 Kinematics

A rigid body in 2D can be represented with three degrees of freedom (DoF) - its location in x and y , along with its rotation in the $x - y$ plane. Given any point on the mass's body, every other point on that body can be deduced from its rigidity. In 3D, the same principle applies, with the rigid body being describable with six degrees of freedom - its location in x, y, z and its rotation in roll, pitch, and yaw. For a robot consisting of n rigid bodies in 3D, then, we naïvely need $6n$ coordinates to describe it; in 2D, we would need $3n$ coordinates.

However, most rigid robots, and all of the rigid robots we consider in this thesis, in fact, are describable as kinematic trees. A kinematic tree is a set of rigid bodies connected by joints such that no loops are formed. A kinematic tree can be put into correspondence with trees in graph theory — If each robot's link is a node, and each robot's joint is an edge connecting exactly two nodes (links), a robot is a kinematic tree if and only if its graph representation is also a tree. Directed acyclic graphs and more general graphs form linkages, which can be overconstrained. Simulating these structures is more complex and outside the scope of this thesis.

An example of a 2D kinematic tree is found in Fig. 2-1. We assume, for the purposes of this thesis, that every joint considered on rigid robots is rotational, and provides one

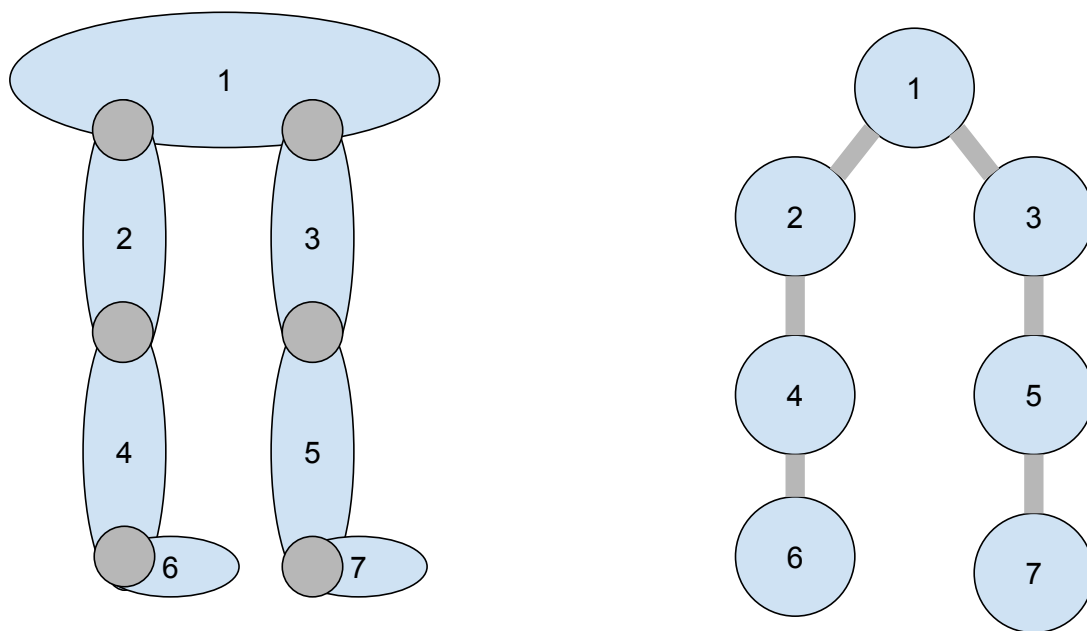


Figure 2-1: A cartoon of a 2D bipedal robot (a kinematic tree) and its corresponding graph representation. The root node, labeled 1, has 3 degrees of freedom. Links are drawn in both representations in blue, and edges in gray. The world coordinates of any link on the tree can be computed by recursively computing the world coordinates of its parents.

degree of rotational freedom. Extensions to more complex joint types are possible. Note that from only the global 3 DoF of the root node/link and the rotations of every joint, we can reconstruct the 3-DoF coordinates of every link. In order to do this, we apply the following recursive formula. If link j is a child of link i connected by a joint with current angle θ_j , the coordinates of link j are:

$$\mathbf{x}_j = \mathbf{R}_j(\theta_j)\mathbf{x}_i + \mathbf{T}_j \quad (2.1)$$

where $\mathbf{R}_j(\theta_j)$ is derived from the current rotation of link j 's parent joint and $\mathbf{T}_j$ is the offset of link j from link i in link i 's coordinate frame. Meanwhile, the rotation is simply the world rotation of link i plus θ_j . The 3D analogue has the same general formulation, but involves 3D rotations and translations for $\mathbf{R}$, $\mathbf{T}$. This representation in which we store only the rigid offset of the base link and the joint coordinates of each robot, referred to as reduced coordinates. Reduced coordinates are size $3 + (n - 1)$ in 2D, and $6 + (n - 1)$ in 3D, a much smaller number than in the unreduced case. In a fixed base robot (cf floating base robot) such as a robot arm (cf untethered quadruped), only $(n - 1)$ coordinates are needed in both 2D and 3D.

2.2.2 Dynamics

Rigid body dynamics follow from Newtonian principles; robots are special in that they are comprised not only of rigid bodies, but also joints which serve as *constraints*. These constraints force the bodies to continuously affect each other's kinematics and dynamics. This is manifested in two core ways. First, the degrees of freedom of the system are reduced, as described above — this simplifies dynamics but makes describing the “mass” of the system tricky. Second, each connected link can apply force and torque on its adjacent links *via* Coriolis forces.

Consider a kinematic tree with reduced coordinates $\mathbf{q} = [\mathbf{x}, \mathbf{v}]^T$, where $\mathbf{x}$ are the reduced positions and $\mathbf{v}$ are the corresponding velocities. The general form for dynamics of kinematic trees is described by the manipulator equations:

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} = \mathbf{J}(\mathbf{q}) - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{B}\mathbf{u} \quad (2.2)$$

Here, $\mathbf{H}$ is a reduced inertia matrix, similar to the “ m ” in $F = ma$. Meanwhile, the remaining forces can be found on the right hand side: First, environmental forces, as a function of the positional configuration of the robot, second, the Coriolis forces caused by the rotation of the joints, and finally, forces caused by actuator torques $\mathbf{u}$ at the joints.

$\mathbf{B}$ is static and simple to define, usually as a diagonal matrix of 1s or 0s depending on if there is a motor at a given joint. $\mathbf{J}$ encodes external forces, such as gravity, whose resultant torques and forces are straightforward to compute, and normal and frictional forces from environmental constraints. Computing $\mathbf{H}$ and $\mathbf{C}$, however, require their own algorithm. There are two common ways to compute forward dynamics : The Recursive Composite Rigid-Body Method, which builds all of the matrices and solves the resulting linear system directly, and the articulated body algorithm, also called Featherstone’s Algorithm, which directly computes $\ddot{\mathbf{q}}$ in linear time. Which algorithm to use depends on what information is needed for your algorithm. For a more detailed description of the computation of these matrices, please see [90] and [44].

For co-optimization of rigid bodies, it is also important to be able to derive the gradients of these systems. Given the Manipulator equation derivation, this can largely be derived by elementary Calculus. The important fact is that the gradients of all quantities are smooth and continuous outside of regions of joint singularities.

2.3 Soft Body Simulation

Unlike their rigid counterparts, soft robots cannot be described fully with a compact set of coordinates. This is true both for kinematics and dynamics. The state space of soft robots are described as a field of quantities that varies continuously in Cartesian space. In this section, we begin by describing the more complex kinematic and dynamic formulation of continuum elastic mechanics, which are typically used in soft robot simulation. Then, we

describe more specific energy models which help govern the dynamics of different materials, before briefly discussing discretization.

2.3.1 Kinematics

Soft bodies can be described as spatially varying fields in space. We begin with the simplest quantity: Position. Continuum mechanics defines two notions of position, occurring in two different spaces. The first is the position in *material space*, denoted as $\mathbf{X}$. Material space coordinates are coordinates that refer to the body of the undeformed elastic material. Each point in material space maps to a point in world coordinates, $\mathbf{x}(\mathbf{X})$; this point represents the actual physical coordinates of the point on the material's body when it undergoes deformation. If this is confusing, an analogy often used in geometry can be helpful here. Imagine an ant walking along the surface of a material. Assuming the ant can stretch in size with the elastic material, the coordinates the ant experiences from its point of view are the material coordinates, while the coordinates an outside observer sees of the ant are its world coordinates. We typically drop the parameter $\mathbf{X}$ in the function $\mathbf{x}$, for notational simplicity.

As one moves around in material coordinates, the world coordinates deform smoothly - though they might change differently along different cartesian axes. The deformation gradient, $\mathbf{F} := \frac{\partial \mathbf{x}}{\partial \mathbf{X}}$ describes the rate at which the material is currently deforming as one moves in material space. Note that if $\mathbf{x}, \mathbf{X} \in \mathbb{R}^d$, then per tensor calculus, $\mathbf{F} \in \mathbb{R}^{d \times d}$.

Each point in material space can also have an associated world velocity, $\mathbf{v}(\mathbf{X}) \in \mathbb{R}^d$. The Affine Matrix $\mathbf{C} \in \mathbb{R}^{d \times d}$ computes how the world velocities change as one moves in world coordinates, $\mathbf{C} = \frac{\partial \mathbf{v}}{\partial \mathbf{x}}$. Though this may feel unnatural, it has relevance in a simulation method we will discuss later in this chapter. These position and velocity fields define the kinematics of a soft body.

2.3.2 Dynamics

The story when it comes to soft body dynamics is more complex. First, we make the assumption that soft bodies we are dealing with are hyperelastic (cf elastoplastic), meaning that any soft object which is deformed will return to its undeformed state in the absence of

other forces or boundary constraints. Hyperelastic materials can be thought of as complex, three-dimensional springs. The springy restoring force that returns these hyperelastic soft bodies to their rest configuration is usually what one intuitively thinks about when considering elasticity. In order to describe the dynamics of a soft body, we must first understand the energy functions of hyperelastic bodies.

Strain is defined as a distance measure between $\mathbf{X}$ and $\mathbf{x}$ computing the spatially varying deformation of a soft body. It can be directly computed through $\mathbf{F}$. Many definitions for different strain measures exist and are typically defined for engineering convenience. We will discuss these briefly when we discuss material energy models. For now we define the energy density scalar field as $\Psi(\mathbf{F}(\mathbf{X}), \mathbf{X})$. The energy over a region Ω is then defined as:

$$E = \int_{\Omega} \Psi(\mathbf{F}(\mathbf{X}), \mathbf{X}) d\mathbf{X} \quad (2.3)$$

and, by the definition of force in classical Newtonian mechanics, the elastic force $\mathbf{f}_e$ becomes:

$$\mathbf{f}_e = -\frac{\partial E}{\partial \mathbf{X}} \quad (2.4)$$

Given a (material-dependent) definition of Ψ , this provides a recipe for analytically computing the the elastic forces of a system, which can then be time integrated, along with $\mathbf{v}$, to update soft body system dynamics.

There is one sticking point in the above discussion. Energy is defined as an integral over an energy density function over an infinitesimal region, which is then differentiated in order to compute a force. Except in the most well-behaved of circumstances, this will be impossible to compute analytically. How to accurately and efficiently approximate the spatially varying forces of an elastic body is the topic of discretization, which we discuss briefly.

Before discussing discretization, however, we mention one other property important for soft bodied simulation — the 1st Piola-Kirchhoff stress tensor, $\mathbf{P} \in \mathbb{R}^{d \times d}$. This is a pressure

field that, like strain, provides a useful intermediate quantity when simulating soft bodies (and one which we can directly manipulate in order to inject fluidic actuation into a soft robot model). Its formal definition is $\mathbf{P}(\mathbf{X}) = \frac{\partial \Psi(\partial \mathbf{X})}{\partial \mathbf{F}(\mathbf{X})}$. Since this is similar to the definition of force without the integration step, it can also be thought of as a type of *force density*.

2.3.3 Discretization and Contact

Except in the most idealized of cases, it will be impossible to fully analytically compute the values of the fields described above everywhere in space. Since we also cannot numerically compute the value everywhere in space (doing so would require an infinite amount of computational resources), soft body simulators seek to approximate these values *via* discretization. The idea is that if the discretization is fine enough, the computation converges in accuracy.

There are several ways to go about discretizing a soft body, but typically, representations are always nodal in some fashion. That is, we approximate a large region through a number of sampled points in space. The state of a soft body then is approximated by a number of discrete, finite values — the positions, velocities, and deformation gradients at each sampled point. A soft body is treated as a collection of rigid particles, held together by elastic force.

In order to compute energies, forces, and or stresses throughout the soft body, integrals are approximated *via* quadrature rules. These integration rules are across neighborhoods of nearby points. How these neighborhoods and integration rules are defined depends on the simulation method itself. One popular simulation method is the finite element method, where nodes are meshed, and integration (and force computation) occurs over the mesh elements. Another method, described below, updates system momentum using a background global grid and advecting the nodes (called particles).

Finally, boundary conditions are most easily discussed in the discrete case. For finite elements, similar to the rigid robotics case, resolving contact can be thought of as applying normal and frictional forces on the finite set of rigid nodes. It is equivalent to treating each node as a point on a rigid body. In the case of the material point method, as we will describe, collisions can be handled directly by computing the net momenta on the background grid, and disallowing penetration through the boundary.

2.3.4 Material Models

The final consideration in simulating soft bodies is the elastic energy model used for deriving the elastic force. These different energy models have different tradeoffs. Different energy models can be used to model materials with different chemistries. Some energy models are accurate for large deformations, while others are only accurate for small deformations. Some energy models are still accurate under rotations, while others are not. Finally, some energy models have nice numerical properties, such as differentiability or linearity.

We describe three material models here — the linear material model, the fixed corotated material model, and the NeoHookean material model. In our experiments, we typically rely on the NeoHookean material model, but also sometimes use the fixed corotated material model.

The linear material model is defined as:

$$\psi(\mathbf{F}) = \mu \epsilon : \epsilon + \frac{\lambda}{2} \text{tr}^2(\epsilon) \quad (2.5)$$

where ϵ is the small strain tensor $\epsilon = \frac{1}{2}(\mathbf{F} + \mathbf{F}^T) - \mathbf{I}$. This is referred to as linear elasticity since it yields the stress tensor:

$$\mathbf{P}(\mathbf{F}) = \mu(\mathbf{F} + \mathbf{F}^T - 2\mathbf{I}) + \lambda \text{tr}(\mathbf{F} - \mathbf{I})\mathbf{I} \quad (2.6)$$

which is linear in $\mathbf{F}$. μ and λ are the lamé parameters, which are functions of the system Young's modulus (stiffness) γ and Poisson's Ratio ν (compressibility):

$$\mu = \frac{\gamma}{2(1 + \nu)} \quad (2.7)$$

$$\lambda = \frac{\gamma\nu}{(1 + \nu)(1 - 2\nu)} \quad (2.8)$$

The corotated linear model is defined as having energy model:

$$\psi(\mathbf{F}) = \mu \|\mathbf{S} - \mathbf{I}\|_F^2 + \frac{\lambda}{2} \text{tr}^2(\mathbf{S} - \mathbf{I}) \quad (2.9)$$

which yields:

$$\mathbf{P}(\mathbf{F}) = 2\mu(\mathbf{F} - \mathbf{R}) + \lambda \text{tr}(\mathbf{R}^T \mathbf{F} - \mathbf{I}) \mathbf{R} \quad (2.10)$$

Here, $\mathbf{R}$ is the rotational component of $\mathbf{F}$, computable *via* the (differentiable) polar decomposition and $\mathbf{S}$ is the scaling component.

This is often slightly modified to have increased stability, yielding the “fixed” corotated model:

$$\mathbf{P}(\mathbf{F}) = 2\mu(\mathbf{F} - \mathbf{R}) + \lambda(J - 1)J\mathbf{F}^{-T} \quad (2.11)$$

where $J = \det(\mathbf{F})$. The corotated linear model is named thus because the stress tensor is still linear in $\mathbf{F}$ but is also invariant under rotations, while the linear elastic model is not.

Finally, the NeoHookean model is used for accurate simulation of rubbers and silicones:

$$\psi(\mathbf{F}) = \frac{\mu}{2} (\text{tr}(\mathbf{F}^T \mathbf{F}) - 3) + \frac{\lambda}{2} \log^2(J) \quad (2.12)$$

and

$$\mathbf{P}(\mathbf{F}) = \mu(\mathbf{F} - \mathbf{F}^{-T}) + \lambda \log(J) \mathbf{F}^{-T} \quad (2.13)$$

This nonlinear material model is more accurate for large deformations, but care must be taken as there exists a singularity at $J = 0$

We note that there exists a refinement that is often considered even more accurate than

NeoHookean materials for soft robotics, the Ogden material model [75]. However, its energy density is defined as a function of eigenvalues of the deformation gradient. When these are the same, the gradient is undefined, making the model nondifferentiable. This is problematic for gradient-based optimization, since equal eigenvalues is quite common: see, for example, even the fully undeformed system $\mathbf{F} = \mathbf{I}$!

2.4 Time integration

In rigid body and soft body simulation, the story is the same — from analyzing the state (and thereby energy) of the system at a point in time, one can determine the velocities of that system and the forces acting upon and within it. But, in order to evolve the system, these velocities and forces must be used to determine the new positions and velocities of the system, some amount of time in the future. This is where time integration comes in. It is impractical to perform this update in a continuous matter. Doing so would require an analytical expression of the system’s time evolution, which is typically not possible, or breaking the system down into an infinite number of infinitesimal time steps, which is intractable. Thus, time must be discretized into a finite number of time steps. Between each point in time into which the system is discretized, the evolution of the system between those points is computed. This process is referred to as *time integration*. It is a process that is general to numerically solving differential equations, but holds particular importance in simulation.

Time integration schemes typically come in three flavors. These are referred to as explicit Euler, semi-implicit Euler, and implicit Euler. Consider two points in time, indexed by i and $i + 1$, with the goal of evolving the system from time t_i to time t_{i+1} . In explicit integration, the forces and velocities computed at time t_i are used to update the state positions at t_i to produce system positions and velocities at time t_{i+1} . In implicit integration, the unknown forces and velocities at time t_{i+1} are used to apply the integration. Since they are typically unknown, these values must be solved for in an iterative process, at the same time that the system is time evolved. Finally, semi-implicit integration finds a middle ground between these two approaches, first solving for just the unknown system velocities at time t_{i+1} using

implicit integration, and then using this to directly apply an explicit step for the system positions. While it may seem unnatural to many different types of numerical integration schemes, such diversity is even present in Calculus 101, and can be thought of as computing different types of Riemann sums while simultaneously computing a time-evolving function.

In practice, implicit integration methods have high stability, allowing one to take large time steps. This comes at the cost of accuracy. Meanwhile, explicit and semi-implicit methods are both more accurate, but require smaller time stepping to avoid system explosions or implosions. In the work presented in this thesis, we use implicit integration for rigid-body dynamical systems in which we want to take large time steps (for tractability of the surrounding co-optimization algorithm), and semi-implicit integration for soft-body dynamical systems. Since we GPU accelerate our soft-body simulation algorithm, this offsets the cost of taking more time steps. We note that, despite allowing larger time steps to be taken an implicit integrator does not always have lower computational cost than an explicit integrator, since computing each step of implicit integration is more expensive than an explicit or semi-implicit step.

Although time integration is often performed with a constant time step, there are cases in which a variable time step must be used. The most common of these cases in robot simulation is in the presence of contact with the environment. If a constant time step is used and it is large compared to system velocities, it is possible that a robot will be outside of, say, a wall at time t_i , but inside a wall at time t_{i+1} . Rather than simply choose a static, small time step Δt , which would be inefficient, one may choose to use a large time step and adapt to a smaller time step in the presence of potential collisions.

One simple method for handling such collisions is as follows. Given a state t_i , the simulation is simulated forward with a time step Δt . If, at time $t_i + \Delta t$, the system is in an invalid state (penetrating a surface, for example), the time step is decreased to $\Delta t/2$. The system is then re-simulated forward to $t_i + \Delta t/2$. This process is then repeated, shrinking the time step, until the integrated system is in a valid state; if this process occurs k times before a valid integrated state is found, then we have $t_{i+1} := t_i + \frac{\Delta t}{2^{k-1}}$. More complex (and efficient) methods also exist; these are outside the scope of this thesis, please see [72] for a survey.

Two final notes on time integration in the context of the work presented in this thesis. First, even in the case of variable time steps, it should be clear that system gradients are equally as valid. Second, note that time integration methods that search for penetration always take a step size that is smaller than or equal to Δt , meaning that if a stable Δt is chosen, it will usually still be stable even after applying such methods.

In Chapter 4, we discuss the differentiable material point method, which we use for our soft robotics work. This method features robust, built-in contact handling with the environment and thus we do not use an adaptive time stepping scheme, though it would be interesting to consider in the future.

2.5 Notation

Moving forward, we will begin developing layers of the computational robot design stack for more dynamic, task-driven rigid and soft robots. For this purposes of this thesis, we use the following (consistent) notation, and note the following conventions.

A scalar, a , will be written as lowercase, unbolded text. A vector, $\mathbf{a}$, will be written as lowercase, bolded text. A matrix, $\mathbf{A}$, will be written as capitalized, bolded text. An unbolded symbol with a subscript, a_i , denotes an entry in the symbol's bolded counterpart $\mathbf{a}_i$ (c.f., matrices, A_{ij} vs. $\mathbf{A}_{ij}$). Other uses of subscripting may include time indexing; *e.g.* for a continuous function $f(t)$ parameterized by a continuous time variable t , $f(t) \triangleq f_t$ for notational convenience. Any other use of superscripts (other than exponentiation) or prescripts will be defined in the section in which they are introduced.

This thesis largely covers topics in robotics, control, and computational design, specifically focusing on the interplay between the cyber (control, proprioception, and task planning) and physical (physical shape/topology, sensorization, actuation, and materiality). Throughout this thesis, the following notation will be consistently used. All coordinates are in reduced coordinates unless specified otherwise.

Symbol	Meaning
$\mathbf{x}$	Positional coordinates of robot
$\mathbf{v}$	Velocity coordinates of robot
$\mathbf{q}$	Robot state (position and velocity)
λ	Contact forces
$\dot{\mathbf{a}}$	Time derivative of $\mathbf{a}$
$\mathbf{u}$	Robot actuation vector (for all actuators, maybe multiple time steps)
$\mathbf{f}$	A general force vector
ρ	Robot physical design parameters
θ	Robot control policy parameters for closed-loop control (except when clearly describing geometri
Θ	Proprioceptive model parameters (for soft robots)
γ	Young's Modulus (spatially indexed)
ν	Poisson's Ratio (spatially indexed)
m	Mass (spatially indexed)
Υ	A dataset
$L(\cdot)$	A loss function, for optimization scenarios

Chapter 3

Algorithms for the Co-Optimization of Rigid Robots

The design of robot bodies and brains are innately coupled problems. Changes in physical robot designs often require modifying or completely redesigning their behaviors. Similarly, if behaviors that satisfy user constraints do not exist for a given robot in solving a required task, its mechanical design must be revisited.

Consider, as a motivating example, a Hexapod (Fig. 3-1) that must walk to and press a button 0.15 m off the ground, while remaining planted on the ground itself. With leg lengths of $h = 0.10$ m, this is an impossible task as the Hexapod is not tall enough. However, lengthening the legs increases the torque effected by gravity on the leg joints, making it more difficult for the robot to walk. As the physical design changes, the motion and actuation for walking changes, *e.g.* legs with greater mass require more actuation; longer legs are able to traverse longer distances per gait cycle. Actuator selection needs to be considered as well. If smaller motors are used, link masses may need to be decreased in order to enable the desired robot motion; if larger motors are used, the robot chassis may need to grow to fit them.

Parametric task-driven robot design traditionally involves several steps. Designers must begin by deciding, at a high level, how the robot will look and move, including what its method of movement and actuation will be. Following this, designers typically select actuators, settle on precise robot dimensions, materials, and other physical properties. When the robot has been fabricated, motions must be programmed and tested. This process can

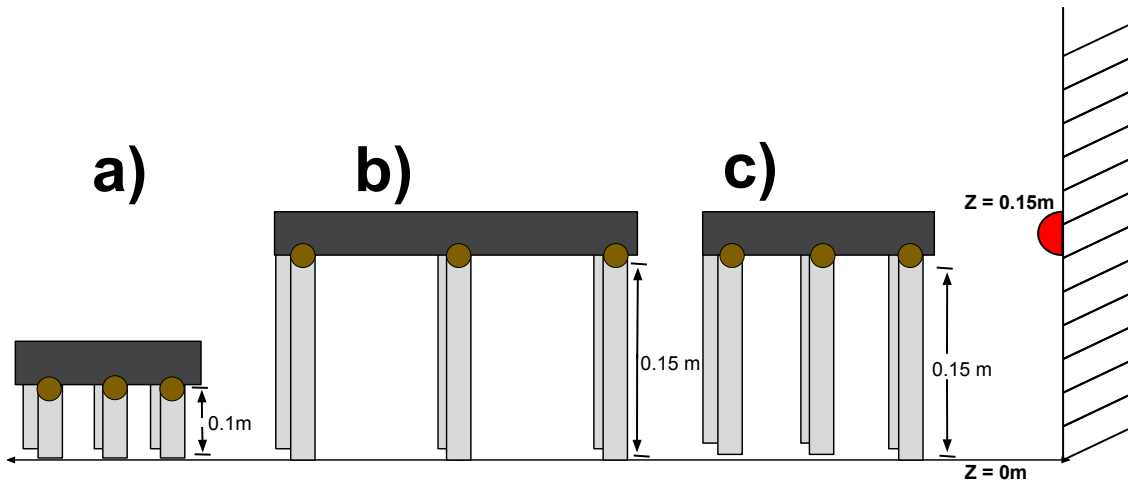


Figure 3-1: A Hexapod robot must walk forward to hit a button on the wall. a) The Hexapod's legs are too short to reach the button. b) Lengthening the legs makes the Hexapod tall enough to reach the button c) Shortening the body allows for motions that are both feasible and require less energy.

require many design iterations before the desired robot is built and able to complete the specified task.

As an alternative approach, consider the following workflow. A computer algorithm could computationally optimize the robot's motion. After that process has converged to a local optimum, it can then pivot to optimize over the physical design domain, optimizing physical body parameters. Then, after *that* process has converged, the algorithm could pivot to optimizing over actuator selection. The algorithm could then pivot back to optimizing over control, and pivot over each domain until all three domains simultaneously converge.

The above workflow appears attractive, injecting computer optimization into the traditional manual workflow. However, this alternating optimization is rife with problems, which can be made especially clear by considering the above Hexapod example. This is a problem with constraints: The robot must be at least a specific height in order to complete the task; elected actuators must fit in the body of the robot, and so on. These constraints affect each of the other domains' decision variables, and potentially restrict them to a very small subspace of solutions. For example, once a body parameterization and a motion have been found that solve the problem, it is impossible to make changes to the body that alter the final height of the chassis while maintaining feasibility of the solution. The robot can then become locked

into a suboptimal design. When both motion and physical body parameters are considered simultaneously, motion parameters may be altered and the body parameters can be altered simultaneously to compensate, maintaining feasibility while also improving the robot’s performance. As an additional concern, such alternating optimization algorithms have no guarantee of converging. We should stress that alternating optimization algorithm schemes *do* have merit, especially in the unconstrained scenario; we will discuss one algorithm which relies on this method in Chapter 5.

Our solution method is rooted in trajectory optimization. For those seeking an introduction to the types of methods used in this chapter, please see [144, 117, 150]. Trajectory optimization is a technique for finding kinodynamically feasible robot motions subject to constraints such as actuation limits, desired position and velocity states of the robot at certain times (*e.g.* starting pose), and environmental constraints, such as non-penetration with terrain or obstacles. Further, trajectory optimization methods optimize these feasible motions for specified objectives, such as minimizing energy expenditure, maximizing robot speed, and so on.

Traditional trajectory optimization methods typically operate on decision variables that include the robot state, actuation, and contact forces. We demonstrate that robot design parameters can be incorporated seamlessly into trajectory optimization problems, enabling the concurrent solution of robot trajectories, actuator selection, and physical designs.

We contribute the following:

- An algorithm for performing trajectory optimization with full dynamics while co-optimizing over physical design parameters and actuation requirements, with no prescribed gaits for walking robots.
- A system in which efficient evaluation of costs and constraints and their gradients and Jacobians respectively is achieved through symbolic representation of the robot’s state, actuation, contacts, and parameters.
- Demonstrations of motions on a physical prototype created using our approach.

We introduce parametric methods for co-optimizing a robot’s physical design and its motion. In this chapter, we focus on rigid robots; in the next chapter, we turn our attention

to soft robots. (These provide a basis for layer 4 of the robot design stack.) Specifically, here we focus on geometric/inertial parameters, discrete actuator selection, and open-loop control.

In this chapter, we describe an algorithm (dubbed parametric trajectory optimization) for co-optimizing rigid robots over kinematic and inertial physical design parameters, actuation requirements, and robot motions. As input, the method requires only a coarse guess of the target robot motion and parameterized robot topology. We demonstrate that by optimizing robot body parameters alongside robot trajectories, motion planning problems which would otherwise be infeasible can be made feasible, and actuation requirements can be significantly reduced. We demonstrate our results on four simulated robots, one of which we physically fabricated in order to demonstrate physical consistency.

3.1 Parametric Trajectory Optimization

We frame co-design of robot structure, actuation, and motion as a parametric trajectory optimization problem, naturally folding in design parameters as additional variables atop the traditional trajectory optimization framework. By incorporating parameters directly in the optimization step, our method is able to alter the robot’s physical structure to make otherwise infeasible tasks feasible, as well as optimize robots for tasks such as path following or forward walking. Our method exploits symbolic differentiation in order to compute gradients (Jacobians) of constraints with respect to physical parameters, enabling efficient optimization of these structural parameters. While we pay special interest to ground locomotion due to the complexity of the task, our method is general to a gamut of robot models and tasks.

Our algorithm solves for robot designs and fully dynamic motions. It ensures dynamical consistency by employing kinodynamic direct transcription trajectory optimization, solving for contact forces *via* implicit contact constraints as in [117]. Our algorithm operates in the continuous domain by solving for the robot state at discretized knot points and integrating the robot dynamics between them.

Together, these objectives and constraints combine to form a complex nonlinear program

of the form:

$$\begin{aligned} & \min_{\mathbf{h}, \mathbf{q}_1 \dots \mathbf{q}_k, \dot{\mathbf{q}}_1 \dots \dot{\mathbf{q}}_k, \lambda_1 \dots \lambda_k, \mathbf{u}_1 \dots \mathbf{u}_k, \rho} \mathcal{G} \\ & \text{subject to dynamics, contact constraints,} \\ & \text{actuation constraints, parametric constraints,} \\ & \text{design constraints, and task constraints} \end{aligned} \tag{3.1}$$

where $\mathcal{G}$ is an objective function operating on the variables, and $\mathbf{q}$, λ and $\mathbf{u}$ are robot state, contact forces, and actuation, respectively. We describe this model in more detail throughout the remainder of this section.

3.1.1 Robot Specification and Dynamics

Formally, let R be a user-specified robot model comprised of links and joints with design parameters $\rho \in \mathbb{R}^p$. These design parameters may include, for each link, a parametric representation of its geometry and mass, as well as functions which define the link's inertial tensor, center of mass, and so on. Each joint may be actuated or non-actuated, and each actuator may possess torque limits.

Consider, for example, our Hexapod model. Here, ρ includes as geometric variables the body length, width, and height and the length of each leg link, with the other dimensions fixed. ρ also includes mass variables, namely the body mass and the leg mass (the same for each leg). The geometry of each link is a rectangular prism. We model each link as uniform density. We set the inertial tensor of each leg L specifically to be:

$$\begin{pmatrix} m(w_L^2 + d_L^2)/12 & 0 & 0 \\ 0 & m(\mathcal{L}_L^2 + d_L^2)/12 & 0 \\ 0 & 0 & m(\mathcal{L}_L^2 + w_L^2)/12 \end{pmatrix}$$

where $\mathcal{L}_L$ and m_L are parameters length and mass parameters of link L respectively. The width w_L and depth d_L of link L is constant in this case.

Key to our algorithm is paying special care to the effect of such parameters on robots'

motion constraints. As dynamic constraints, we require:

$$\forall t, [\dot{\mathbf{q}}(t), \ddot{\mathbf{q}}(t)]^T = f(\mathbf{q}(t), \dot{\mathbf{q}}(t), \mathbf{u}(t), \lambda(t); \rho) \quad (3.2)$$

Here f is a function governing the dynamics of a robot, and $\mathbf{q}(t) \in \mathbb{R}^n$, $\dot{\mathbf{q}}(t) \in \mathbb{R}^n$, $\mathbf{u}(t) \in \mathbb{R}^m$, $\lambda(t) \in \mathbb{R}^{6l}$, and $\rho \in \mathbb{R}^p$ are the robot state, time derivative of the robot state, actuation inputs (torques or forces), and contact forces at time t , and the design parameters, respectively. Parameters are a static attribute of the robot and are thus *not* a function of time.

By discretizing t into a vector of K (ordered) knot points $\mathbf{t} \in \mathbb{R}^K$, we can write, employing the backward Euler integration method:

$$\begin{aligned} \forall k \in 1 \dots K, [\mathbf{q}_{k+1}, \dot{\mathbf{q}}_{k+1}]^T &= [\mathbf{q}_k, \dot{\mathbf{q}}_k]^T + \\ &\quad f(\mathbf{q}_k, \dot{\mathbf{q}}_k; \mathbf{u}_k; \lambda_k; \rho) dt_k \\ \text{where } dt_k &= \mathbf{t}_{k+1} - \mathbf{t}_k \end{aligned}$$

Again, note that ρ is constant over *all* knot points. Specifically, for f , we compute the forward dynamics of the robot in order to compute the robot's manipulator equations and invert the equations in order to solve for $[\dot{\mathbf{q}}_k, \ddot{\mathbf{q}}_k]^T$. In other words, $f(\mathbf{q}_k, \mathbf{u}_k, \rho)$ is computed by solving the parameterized manipulator equations $\mathbf{H}(\mathbf{q}_k; \rho)\ddot{\mathbf{q}}_k + \mathbf{C}(\mathbf{q}_k, \dot{\mathbf{q}}_k; \rho)\dot{\mathbf{q}}_k + \mathbf{g}(\mathbf{q}_k; \rho) = \mathbf{B}(\mathbf{q}_k)\mathbf{u} + \mathbf{J}\lambda_k$ for $[\dot{\mathbf{q}}_k, \ddot{\mathbf{q}}_k]^T$. Here, $\mathbf{B}$ is the robot's control matrix, $\mathbf{J}$ is its contact point Jacobian matrix, $\mathbf{H}$ is its inertial matrix, $\mathbf{C}$ is its Coriolis matrix, and $\mathbf{g}$ are forces due to gravity. $\mathbf{H}$ is computed using the Composite-Rigid Body algorithm, and $\mathbf{C}$ is computed using the recursive Newton-Euler method.

For the contact forces, we add $\lambda = [\lambda_z, \lambda_x^-, \lambda_x^+, \lambda_y^-, \lambda_y^+, \gamma]$. Here, λ_z is the normal for each of l potential contact forces, λ_x^+ , λ_x^- , λ_y^+ , and λ_y^- are the bilateral sliding contact forces along the friction cone, and γ is a slack variable in order to allow for sliding of the contacts. In practice, similar to [117], we add additional slack variables when solving the nonlinear complementarity constraints to make the problem better conditioned numerically.

For our Hexapod example, we model the center of each foot as being a potential contact

point, giving $l = 6$. We then add constraints at each knot point k as:

$$\phi(\mathbf{q}_k, \rho) \geq 0 \quad (3.3)$$

$$\lambda_{k,z}, \lambda_{k,x}^+, \lambda_{k,x}^-, \lambda_{k,y}^+, \lambda_{k,y}^-, \gamma_k \geq 0 \quad (3.4)$$

$$\mu\lambda_{k,z} - \lambda_{k,x}^+ - \lambda_{k,x}^- - \lambda_{k,y}^+ - \lambda_{k,y}^- \geq 0 \quad (3.5)$$

$$\phi(\mathbf{q}_k, \rho)^T \lambda_{k,z} = 0 \quad (3.6)$$

$$\gamma_k + \psi(\mathbf{q}_k, \dot{\mathbf{q}}_k, \rho)^T \hat{\mathbf{e}}_x \geq 0 \quad (3.7)$$

$$\gamma_k - \psi(\mathbf{q}_k, \dot{\mathbf{q}}_k, \rho)^T \hat{\mathbf{e}}_x \geq 0 \quad (3.8)$$

$$\gamma_k + \psi(\mathbf{q}_k, \dot{\mathbf{q}}_k, \rho)^T \hat{\mathbf{e}}_y \geq 0 \quad (3.9)$$

$$\gamma_k - \psi(\mathbf{q}_k, \dot{\mathbf{q}}_k, \rho)^T \hat{\mathbf{e}}_y \geq 0 \quad (3.10)$$

$$(\mu\lambda_{k,z} - \lambda_{k,x}^+ - \lambda_{k,x}^- - \lambda_{k,y}^+ - \lambda_{k,y}^-) \gamma_k \geq 0 \quad (3.11)$$

$$(\gamma_k + \psi(\mathbf{q}_k, \dot{\mathbf{q}}_k, \rho)^T \hat{\mathbf{e}}_x) \lambda_{k,x}^+ = 0 \quad (3.12)$$

$$(\gamma_k - \psi(\mathbf{q}_k, \dot{\mathbf{q}}_k, \rho)^T \hat{\mathbf{e}}_x) \lambda_{k,x}^- = 0 \quad (3.13)$$

$$(\gamma_k + \psi(\mathbf{q}_k, \dot{\mathbf{q}}_k, \rho)^T \hat{\mathbf{e}}_y) \lambda_{k,y}^+ = 0 \quad (3.14)$$

$$(\gamma_k - \psi(\mathbf{q}_k, \dot{\mathbf{q}}_k, \rho)^T \hat{\mathbf{e}}_y) \lambda_{k,y}^- = 0 \quad (3.15)$$

Here, $\phi_i(\cdot)$ is the signed distance between contact i and the terrain, and $\psi_i(\cdot)$ provides the relative tangential velocity of each contact. (Note we have dropped the contact indexing i in the constraints above for readability.) The first constraint encodes non-penetration; the second constraint is necessary for breaking the contact forces into slack and bilateral components. The third constraint enforces friction cone constraints, and the fourth constraint encodes the strict complementarity of contacts. The remaining constraints introduce the tangential velocity term to the previous constraints in order to allow for sliding in the contacts.

The decision to use an implicit contact constraint formulation is important for two reasons. First, it lessens the specification requirements for the user. As we are concerned with allowing users to design robots from high-level design goals, we consider such details to be a burden. Second, constraining foothold patterns can shrink the design space. This may make it impossible for the optimizer to find feasible designs and motions for a given

task, even if feasible designs would otherwise exist; or, those solutions the optimizer finds may be highly suboptimal. By solving for contacts as we search for motions, we allow the optimizer to search over the space of all robot designs and motions at once.

3.1.2 Constraints

In addition to the robot, users may specify three types of problem constraints - parametric constraints, task constraints, and design constraints.

As a bare minimum for parametric constraints, we require an upper and lower bound on each design parameter in order to prevent degenerate solutions. If a user desires, auxiliary constraints may be added which establish a necessary relationship between parameters; these are useful for describing fabrication constraints. For example, the minimum possible mass of a robot may be a function of its size. For our Hexapod example, we require the (obvious) constraint that masses and dimensions must be positive, and specify an upper and a lower bound on each.

Task constraints, which define the robot's function, are kinematic in nature. These constraints allow users to specify the configuration (including world pose) of the robot, the position of any point *on* the robot in the world frame, and prescribe velocities of any of the robot's degrees of freedom. For walking robots, we also automatically add the constraint that links without contact points must not collide the ground (unless such a collision is explicitly requested by a user).

In our Hexapod task example, we add three task constraints: that our robot is upright on the ground with $\dot{\mathbf{q}}_1 = \mathbf{0}$ at the first knot point, that our robot is upright on the ground with $\dot{\mathbf{q}}_K = \mathbf{0}$ at the final knot point, and that the front and center of the robot's body is at a height of 0.15 m.

Design constraints define a mapping between actuation and design parameters. Higher power-output motors tend to be larger in both size and mass. We assume designers have access to a discrete collection of motors with which to build their robots. Each motor is rated with a maximum power output, and comes with its own design restrictions; specifically, they must fit on the robot in the specified locations, and the robot must be able to manage their

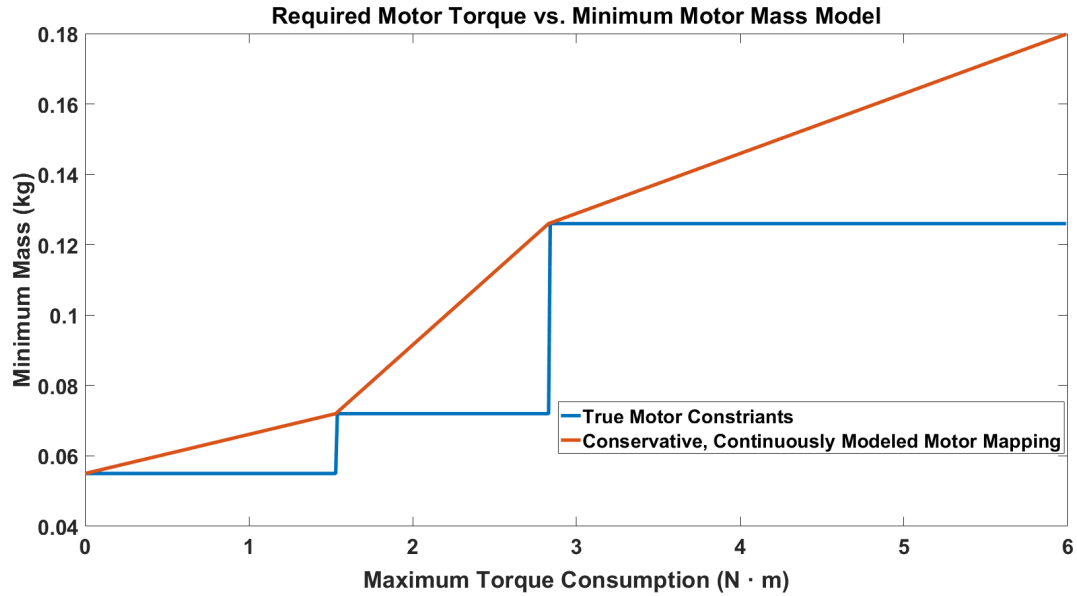


Figure 3-2: A conservative mapping of the necessary output torque to individual minimum motor mass used to preserve continuity. We avoid flat regions in our model to ensure that our optimizer has meaningful gradients to work with.

load.

To ground this discussion, consider our Hexapod example. Our Hexapod topology calls for three actuators on each side. Any solution motion imposes constraints on the minimum motor size. The height of the Hexapod's body link must be at least the height of the motor; its width must be at least twice the width of the motor, and its length must be at least three times the length of the motor. Further, the mass of the body must be at least six times the mass of the chosen motor, plus the mass of the building materials.

Such a set of constraints is discontinuous and can cause numerical challenges for our optimizer. Therefore, we model the necessary lower bounds on mass and dimensions conservatively, linearly interpolating the motor masses and link dimensions between the jumps in power output. Although this constraint is nonsmooth at a few points, we have not found this to be an issue for our optimizer. Fig. 3-2 shows our model for conservatively estimating the added motor mass; similar models are used for the other variables as well.

As will be described in Section 3.1.3, we seek to minimize the maximum power output throughout a motion, thus allowing our optimizer to choose motors with the smallest design restrictions possible. We currently require all motors in a robot to be the same for ease of

programming the electronics.

3.1.3 Objectives and Actuator Selection

Our algorithm can optimize over any smooth function of the robot state, actuation, contact, and design parameters. However, we mostly focus on the objective function

$$\mathcal{G} = \alpha \mathcal{G}_{act} + \beta \mathcal{G}_{reg} \quad (3.16)$$

where $\mathcal{G}_{act}$ is an actuation minimizing term, $\mathcal{G}_{reg}$ is a parameter regularizer, and α and β are user-specified weights.

By choosing $\mathcal{G}_{act}$ as part of our objective function, our method automatically chooses the actuators necessary for the designed robots by attempting to *minimize* the necessary power consumption. Since motors with higher power requirements are often larger, the decision to use higher power output motors constrains designs. We seek to minimize the motor size by minimizing the maximal power output.

We write this objective $\mathcal{G}_{act}$ as:

$$\mathcal{G}_{act} = \max_j \max_i \mathbf{u}_{ij} \equiv \|u\|_\infty \quad (3.17)$$

where $\mathbf{u}_{ij}$ denotes the necessary actuation (torque) output of actuator i and knot point j . The electrical power consumption of the motors we consider are linearly proportional to their torque applied, so we use torque as a proxy.

Through use of an auxiliary variable ξ , this objective can be written as:

$$\begin{aligned} & \xi \\ \text{subject to } & \forall i, j \quad \xi - \mathbf{u}_{ij} \geq 0 \\ & \xi + \mathbf{u}_{ij} \geq 0 \end{aligned}$$

The total actuation of each motor at each knot point must also be limited by the specifications of the largest motor available in a designer's library.

For aesthetics, functional purposes, and perhaps fabrication concerns (*e.g.* reducing

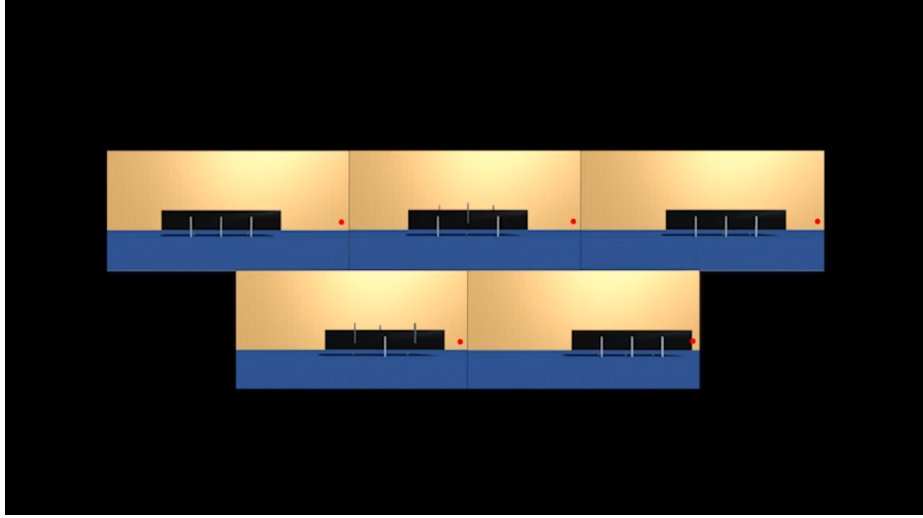


Figure 3-3: Example input keyframes, used for seeding the Hexapod co-optimization.

material consumption), the designer may prefer that the output designs respect her vision as closely as possible. Therefore, we add the regularization term $\mathcal{G}_{reg}$ to the objective cost. We use L_2 regularization as its smoothness makes it easier to optimize over, though other regularizers may be employed as well. In particular,

$$\mathcal{G}_{reg} = \frac{1}{2} \|\rho - \rho_0\|_2^2 \quad (3.18)$$

The choice in α and β represents a user-specified trade-off between the desire to minimize motor size and the desire to keep the design close a designer's original vision.

3.1.4 Initialization

Finally, users must specify guesses for the robot's configuration q at each time step and a maximum time for task completion. We stress here that the initial guess need not be physically feasible. Our algorithm generates initial poses for each knot point by linearly interpolating over a few sparse user-specified keyframes. An example five keyframe sequence for the Hexapod, inspired by the tripod gait, is shown in 3-3.

Velocities $\dot{q}$ are automatically initialized to 0. Knot points are linearly spaced between $[0, T]$, where T is a user-specified maximum duration of the motion. The contact force vector for contact point i at knot point k is set to be normal to the terrain with magnitude

equal to the robot's weight if its keyframe interpolation is in contact with the terrain, and 0 otherwise. For actuation initialization, $\mathbf{u}_{ij} \sim \mathcal{U}(-u'_i, u'_i)$, where u'_i is maximum torque-limit for actuator i .

3.1.5 Optimization

A complete description of the algorithm can be found in Alg. 1. For the minimization step, we use SNOPT [53], a sparse sequential quadratic programming (SQP) solver which has been shown to be effective in solving direct transcription optimization problems due to their sparse nature. Our parameterized problem is in fact still sparse, since the number of design parameters in each constraint is constant and does not grow with the size of other problem parameters such as knot points or robot state dimensionality. SNOPT internally approximates the necessary Hessian information in the optimization step in a numerically robust manner. We break the optimization into two successive calls to SNOPT; once to find a feasible solution (with no objective function), and once using the feasible point as a seed to the objective optimization. Since SNOPT converts constraints to costs to find feasible points, minimizing over costs and constraints simultaneously can sometimes lead SNOPT to deem locally feasible problems as infeasible. By splitting the optimization into two steps, we avoid this scenario.

Algorithm 1 Co-optimization Of Trajectories and Parameters

- 1: **procedure** CO-OPTIMIZE($R, U, \mathcal{G}$) $\triangleright R$ is a parameterized robot design, $\mathcal{G}$ is a user-specified objective function, U is the set of user constraints, $\mathcal{G}$ is a user-specified objective function.
 - 2: Constraints $\leftarrow \{U\}$
 - 3: Costs $\leftarrow \{\}$.
 - 4: **for** $k = 0 \dots K - 1$ **do**
 - 5: Constraints $\leftarrow$ Constraints $\cup \{\Lambda_k, M_k, \mathbf{q}_{k+1} = \mathbf{q}_k + f(\mathbf{q}_k, \mathbf{u}_k, \rho)dt_k, \rho_{lower} \leq \rho \leq \rho_{upper}\}$
 - 6: $\triangleright \Lambda_k$ are the contact constraints, M_k are the motor constraints.
 - 7: $(q, \dot{q}, u, dt, \lambda, \rho) \leftarrow$ SNOPT($q_{init}, \dot{q}_{init}, u_{init}, dt_{init}, \lambda_{init}, \rho_{init}$, Constraints, Costs)
 - 6: $\triangleright$ Solve for feasibility.
 - 8: Costs $\leftarrow \mathcal{G}$
 - 9: **return** (Success, $q, \dot{q}, u, dt, \lambda, \rho$) $\leftarrow$ SNOPT($q, \dot{q}, u, dt, \lambda, \rho$, Constraints, Costs) $\triangleright$ Solve with costs.
-

3.2 Implementation

Our system was implemented atop Drake [145], a MATLAB toolbox for robot simulation and trajectory optimization.

Trajectory optimization approaches often rely on gradient-based methods (such as SQP) to solve the formulated nonlinear programs. The form of these Jacobians is well understood for robot dynamics when taken with respect to actuation, state, and time. It is less obvious how to calculate these Jacobians with respect to arbitrary physical parameters. One approach is to calculate these Jacobians numerically. However, computing finite differences can be expensive and it can be numerically inaccurate in regions where function curvature is large.

Our solution is to represent costs and constraints symbolically in all decision variables. Since typical constraints vary smoothly with respect to physical parameters, we can use the symbolic representation to efficiently calculate these Jacobians. This has benefits for designer usability as well. A symbolic representation enables complex user costs and constraints with no user overhead in specifying Jacobians. This representation also makes it natural for users to define metaparameters of other physical parameters (*e.g.* moments of inertia that are functions of masses and link dimensions).

Generating symbolic objectives and constraints is a pre-processing procedure that only has to be performed once per robot or task. In order to speed up the evaluation of the constraints and their Jacobians, we compile them to C.

3.3 Analysis

Complexity The algorithm has two phases of computation - pre-processing, in which dynamic, contact, and user-specified constraints and objectives are composed, and the optimization step itself. We examine each in turn.

Constraints And Objectives The dynamics of the robot need only be computed once in order to formulate the dynamics constraints. This routine relies on two steps: Composite Rigid-Body and Recursive Newton-Euler algorithms, which run in $O(n)$ time, and solving the manipulator equations. Solving the manipulator equations is the most time-consuming

operation, requiring $O(n^3)$ time. Computing the Jacobian of the dynamics requires differentiating each of the $O(n)$ dynamics expression coordinates with respect to $(n + m + p + l)$ state, actuation, contact force, and parameter variables, taking $O(n^3(n(n + m + p + l)))$ time.

Computation of the contact constraint relies on computing the distance from each contact point on the robot to each potential terrain contact point. Computation of each of the contact points in the world frame can be computed in $O(n)$ time using forward kinematics. Thus, the distance from each contact point in the world to each terrain point may be computed in $O(n)$ time. For piecewise defined terrain with τ grid cells, and l contact points, $O(l\tau)$ constraints must be composed, this computation takes $O(l\tau n)$ time. For Jacobians, each of $O(l\tau)$ constraints must be differentiated with respect to $O(n + l + p)$ state, contact force, and parameter variables, taking $O(nl\tau(n + l + p))$ time.

Beyond the constraints that we add automatically, users are free to add their own constraints as described above.

Optimization Each of our two SNOPT calls requires, at worst, $O(\|z_0 - z^*\|_2)$ iterations to converge, where z_0 is the initial decision vector of each SNOPT call and z^* is the local optimal feasible solution. For each iteration, every constraint and its Jacobian must be evaluated. The time needed to evaluate these constraints and Jacobians is equal to the time needed to formulate them, as described above. $O(K)$ dynamics, non-terrain penetration constraints, and contact constraint sets must be evaluated at each iteration.

3.3.1 Convergence Guarantees

Our optimization problem is highly nonlinear over a very high dimensional space; state, design, and slack variables sum to $(2n + m + 6l)K + p + 1 + (K - 1)(5l + 1)$ variables per problem. For nonlinear problems, SNOPT is only guaranteed to converge in the neighborhood of the solution. If the problem is initialized within a small enough neighborhood of a feasible solution, then SNOPT (and indeed our algorithm) is guaranteed to find a feasible design. Otherwise, no guarantee can be made. Theoretically, this neighborhood is very hard to define explicitly. In practice, we find that local feasibility of our problems is most

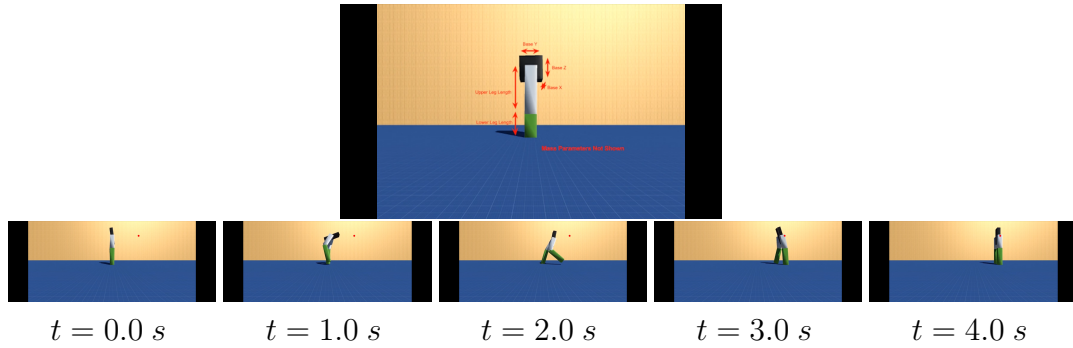


Figure 3-4: A montage of our simulated Biped in motion for the flat walking task, along with the original parameterized version. The Biped is too short and heavy in its original configuration to reach the goal circle.

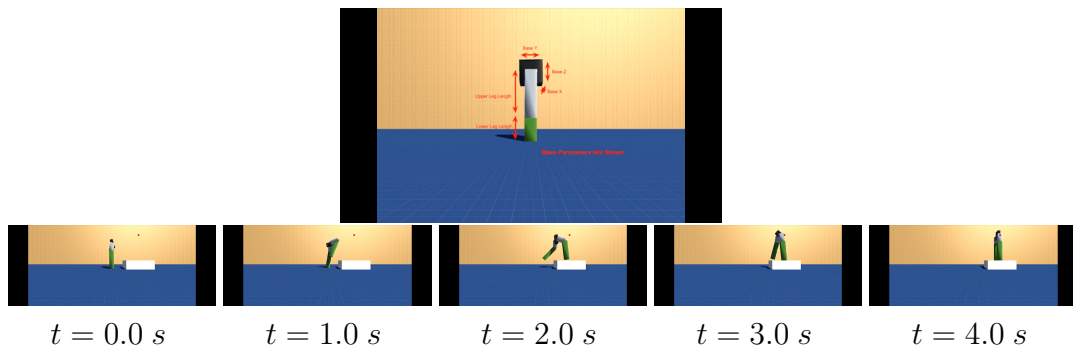


Figure 3-5: A montage of our simulated Biped in motion for the step walking task, along with the original parameterized version. The Biped is too short and heavy in its original configuration to reach the goal circle, and its links cannot clear the terrain without collision.

sensitive to the initial guesses for the actuation. In the event of failures, random restart can eventually find a solution with most well-behaved problems.

3.4 Experiments

We present virtual experiments on four model robots to demonstrate the flexibility and power of our algorithm. Our model robots include a Hexapod, Biped, Quadruped, and Quadcopter, all with fully dynamic motion. Further results can be found in Figs. 3-4, 3-5, 3-6, 3-7, 3-8.

We model all of our virtual and physical experiments (with the exception of the Quadcopter) using the Dynamixel family of servo motors. A chart mapping the necessary power output to motors we have selected from the Dynamixel family, along with their masses and dimensions can be seen in Table 3.1.

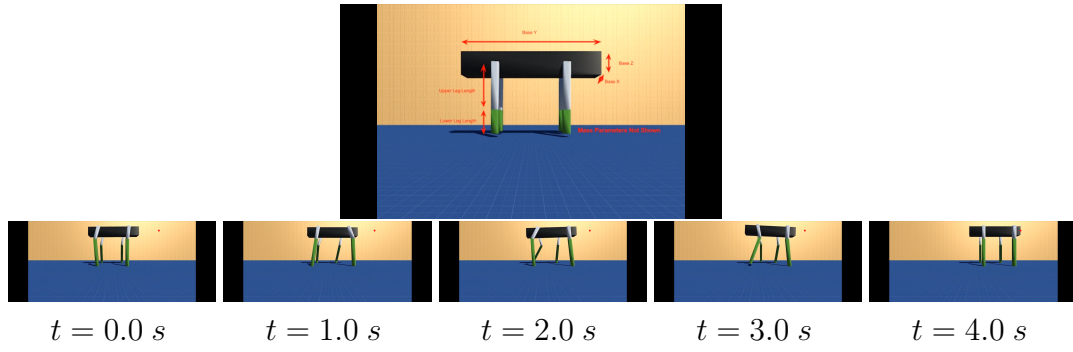


Figure 3-6: A montage of our simulated Quadruped in motion for the energy efficient task, along with the original parameterized version. The Quadruped is too short and heavy in its original configuration to reach the goal circle.

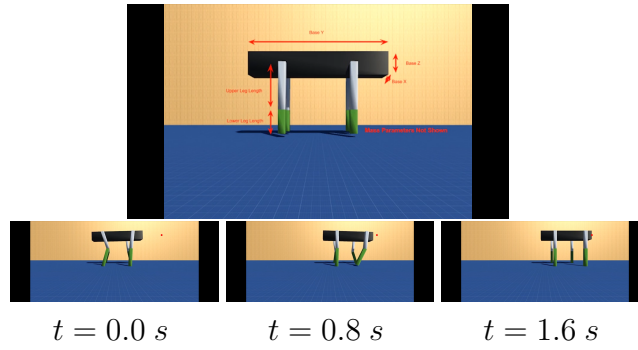


Figure 3-7: A montage of our simulated Quadruped in motion for the fast motion, along with the original parameterized version. The Quadruped is too short and heavy in its original configuration to reach the goal circle.

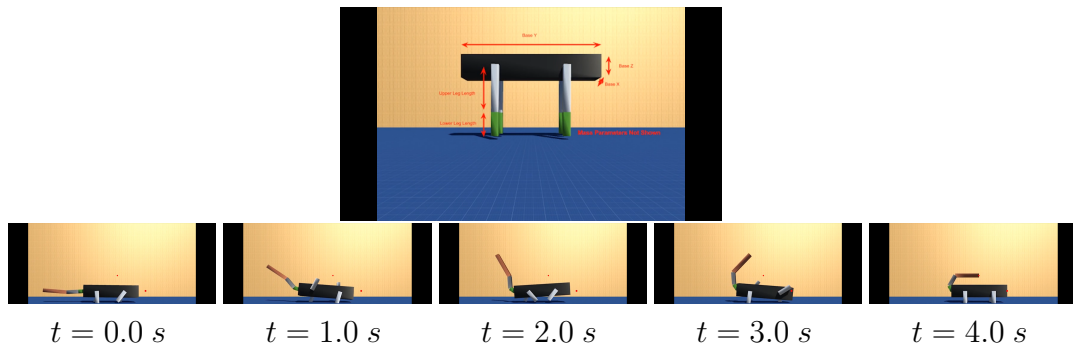


Figure 3-8: A montage of our simulated Squirrel in motion for the energy efficient task, along with the original parameterized version. Because of the 60 degree limit on the tail links, only one set of geometric configurations for legs and tail extents will result in a squirrel that can reach both red circles.

Motor Name	X Extents (m)	Y Extents (m)	Z Extents (m)	Mass (kg)	Torque (N · m)
AX-12a	0.032	0.04	0.05	0.055	1.53
RX-28	0.0355	0.0356*	0.41	0.072	2.83
MX-64T8	0.0402	0.041	0.0611	0.126	6.0

Table 3.1: The motors from our selected Dynamixel motor library, ordered by increasing torque and size. All motors consume 12 V and have a top speed of approximately 2π rad / s. *Though this motor’s Y extents are 0.0356, we model it as 0.0405 in order to keep the Y extents monotonically increasing.

All walking robots were given 4 seconds to complete their tasks; the Quadcopter was given 6 seconds. $K = 16$ knot points were used for all experiments. For legged robots, we model the legs’ inertial tensors as rectangular prisms as described in Section 3.1.1; we choose constant values for the body links, assuming we can realize these moments by judiciously distributing the mass in our fabrication process. All joints on our robots are actuated, and there are no joint limits. Our robot models are shown in Figs. 3-9 and 3-10.

3.4.1 Hexapod

Our Hexapod demonstrates our running example being optimized using our algorithm. We require the Hexapod to walk forward 0.45 m and touch a height of 0.15 m, while keeping all legs planted on the ground. This task is infeasible without adjusting its parameters, since the center of its body initially is only 0.1 m tall. The Hexapod has 6 parameters in total: the X (length), Y (width), and Z (height) extents of the base, the (equal) height of the legs, the mass of the body, and the (equal) mass of the legs. The optimization problem has 1528 decision variables.

We use the Hexapod as an opportunity to demonstrate the results of our regularization objective. In particular, we run our Hexapod under two conditions: $\beta = 0$, and $\beta = 1.0$. We set $\alpha = 1.0$ for all of our experiments. For each condition, we ran 10 optimizations. Initial parameters and the results under each condition can be seen in Table 3.2.

As expected, the legs of the Hexapod grow to satisfy the height constraints. Further, the leg masses decrease, reducing the power output needed to propel the robot forward.

As β increases, the Hexapod approaches its initial parameter values where possible. Larger β results in larger link masses; larger motors may be necessary depending on the

β	Actuation Objective (N · m)	Regularization Objective	Body X (m) 0.9 / 1.2 / 1.05	Body Y (m) 0.05 / 0.10 / 0.10	Body Z (m) 0.1 / 0.15 / 0.15	Leg Mass (kg) 0.08 / 0.12 / 0.10	Body Mass (kg) 0.3 / 0.5 / 0.4	Opt. Time (s)
0	1.49 (0.572)	0.394 (0.019)	1.01 (0.154)	0.092 (0.008)	0.138 (0.023)	0.08 (0.0)	0.027 (0.46)	2,246 (2,230)
1.0	1.28 (0.189)	0.447 (0.026)	1.50 (0.014)	0.1 (0.0)	0.15 (0.0)	0.08 (0.0)	0.433 (0.019)	2,049 (2,007)

Table 3.2: Results of the simulated Hexapod optimizations for two β values. Parentheses denote standard deviations. Headers of parameter columns denote lower bound / upper bound / initial value. The task was feasible for 7 of the 10 initializations.

trajectories selected. For the $\beta = 0$ case, the motors had to be increased from the AX-12a model to the RX-28 in one trial, and in the $\beta = 1.0$ case, the motors had to be increased in four of the trials. As expected, larger relative β values decrease the emphasis on reducing motor size.

We found in our $\beta = 0$ experiments that some of the optimized parameter values were spread fairly uniformly throughout their range; for larger β values these parameters snap to their initial parameter values (*e.g.* the body Z extents). The regularizer can automatically detect parameters which have little effect on the actuation objective and keep them in their preferred configuration with little to no cost.

We fabricated a real Hexapod in order to demonstrate that our simulations are physically realizable. We selected one optimized Hexapod design from our $\beta = 0$ experiments and fabricated it with a light laser cut acrylic frame, 3-D printed legs, and 6 Dynamixel AX-12a servos with PID position and velocity control. An Arduino MEGA 2560 was used to control the electronics and record data. Our fabricated Hexapod body is 0.9 m long, 0.1 m wide, and 0.15 m tall. Its legs are 0.15 m tall legs.

A montage of one fabricated Hexapod alongside its simulated counterpart can be found in Fig. 3-9. Compared with its planned motion, the center of mass of the fabricated Hexapod had an average integrated squared error of 1.93 cm^2 in x (the direction of motion), 0.078 cm^2 in the $y - z$ plane, and 1.97° in its angular orientation. Although geometrically the motions look quite similar, there is a large discrepancy in tracking in the x direction since the actual friction coefficients were higher than those used for planning the virtual model. Throughout its trajectory, the center of mass of the physical model gradually fell behind that of the virtual model due to slip. Improving the accuracy of the frictional coefficients or tightly constraining the allowed slip in motion planning can reduce these types of errors.

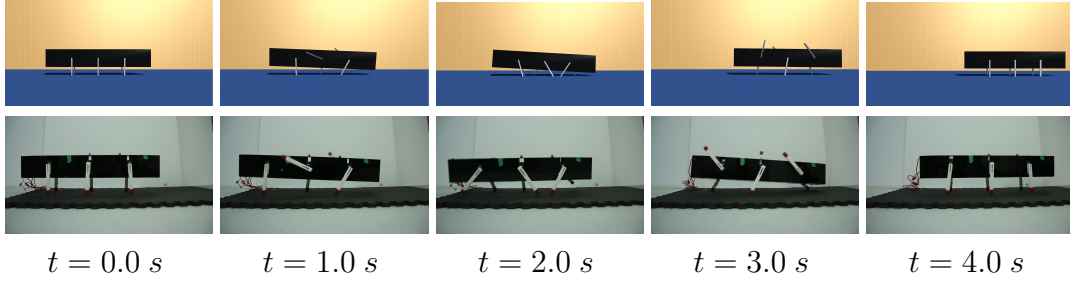


Figure 3-9: A montage of our simulated and fabricated Hexapod in motion.

3.4.2 Biped

The Biped example demonstrates that our walking robots can locomote on more than just flat terrain. In this example, we present a Biped, consisting of a base, hip joints, and knee joints. It consists of the following eight parameters: the length of the lower leg links (equal), the length of the upper leg links (equal), the x , y , and z dimensions of the base, the mass of the lower leg links (equal), the mass of the upper leg links (equal), and the mass of the body.

We present two tasks for the Biped. In the first task, we require the Biped to walk forward 0.3 m. In the second task, we require the Biped to walk forward 0.3 m and up a 0.1 m step. We set $\alpha = 1.0$ and $\beta = 0$ for both tasks. Further, we restrict the final height of the Biped to lie between 0.25 m and 0.35 m tall. This problem has 750 decision variables.

We ran 30 optimization runs for each task. Table 3.3 presents statistics about the optimized design and motion, along with valid parameter ranges. Two items are of particular note. First, resulting masses are near their lower limits. Lowering the masses of the links decreases the power needed to actuate them, and allows the optimizer to select the smallest motors available (which it does for all tasks). Notably, the masses are rarely all at their absolute minimum - some excess mass may be added by the optimizer to increase the controllability of the design. Second, leg links tend to get longer, meaning they have to actuate over a smaller angular range. For the step task in particular, the algorithm discovers that the upper leg link should be shorter than in the flat walking task in order to clear the step *via* a forward motion.

Task	Actuation Objective (N · m)	Lower Leg Length (m) 0.5 / 4.0 / 1.0	Upper Leg Length (m) 0.5 / 4.0 / 2.0
Flat	0.506 (0.117)	0.216 (0.044)	0.100 (0.032)
Step	0.651 (0.267)	0.2328 (0.509)	0.085 (0.044)

Task	Body X (m) 0.3 / 2.0 / 1.0	Body Y (m) 0.3 / 2.0 / 1.0	Body Z (m) 0.3 / 2.0 / 1.0	Body Mass (kg) 0.5 / 10.0 / 4.0
Flat	0.080 (0.00)	0.085 (0.079)	0.133 (0.044)	0.14 (0.058)
Step	0.088 (0.024)	0.097 (0.076)	0.130 (0.043)	0.126 (0.026)

Task	Upper Leg Mass (kg) 0.5 / 10.0 / 4.0	Lower Leg Mass (kg) 0.5 / 10.0 / 4.0	Opt. Time (s)
Flat	0.064 (0.037)	0.068 (0.030)	685 (1,830)
Step	0.066 (0.034)	0.059 (0.019)	460 (500.2)

Table 3.3: Results of the simulated Biped optimizations for the flat and step terrain. Parentheticals denote standard deviations. Headers of parameter columns denote lower bound / upper bound / initial value.

3.4.3 Quadruped

The Quadruped presents the most articulated robot of our examples. Our Quadruped model consists of four legs attached to a base. Similar to our Biped, each leg has an upper and lower limb; the parameterization used for the Quadruped, in fact, is the same as that of the Biped but with two extra legs (and some different parameter ranges). We require the robot to walk forward 0.25 m, with a final height between 0.35 m and 0.4 m. Note the Quadruped only starts at 3 m tall; the algorithm must discover that it must grow. The Quadruped problem has 1284 decision variables.

Although we have put a focus on actuation minimization and regularization objectives, our method can extend to other objectives which may be of interest to other roboticists. To demonstrate this, we optimize our Quadruped over two sets of experiments. First, we optimize our Quadruped over our standard objectives with $\alpha = 1.0$ and $\beta = 0.0$. Second, we optimize our Quadruped in a different set of experiments where we minimize time to task completion.

Each experiment was run 10 times. This is a difficult and time-consuming optimization problem, sensitive to initial actuation guesses, and so for each experiment we only received a handful of successes (three each). Thus, we note mostly qualitative results here.

In the actuation minimization experiments, the heights of the legs increased to their maximum combined lengths of 0.4 m. This allows the robot to locomote while expending as little torque possible, since its joints have to cover less of an angular range to travel the required distance. Meanwhile, in the velocity tasks, the legs decreased to their minimum combined lengths of 0.35 m. Shorter legs give the robot greater acceleration and deceleration, allowing the robot to achieve the zero velocity start and end constraints.

In both sets of experiments, the base and upper leg link masses decreased to nearly their minimum bounds; the total robot masses decreased overall as well. However, the bottom link mass decreased for the actuation minimization task, while increasing slightly for the time minimization. Increasing mass in the lower leg links increases their inertial moments, making overall velocity of the robot easier to control. This is again important for allowing time-objective designs to easily decelerate. Controllability is not such a concern for the lower-actuated design, and so the optimizer is free to decrease masses to decrease power output.

Finally, we notice a large contrast in the objective values between the two tasks. For the actuation minimization task, the full 4 seconds was required, while the maximum torque required ranged between 0.294 and 0.494 N · m, resulting in the smallest motors. Meanwhile, for all time minimization trials, the largest motors were selected and the maximum torque was used (6 N · m), with most of these high torques occurring during the push-off and landing of contacts. Times required for task completion, however, ranged from 1.252 s to 1.268 s, marking a significant improvement.

3.4.4 Quadcopter

As a final example, we present a Quadcopter, which we use to validate our assertion that our algorithm is able to find known optimal parameterizations, with better objectives than any other (valid) fixed parameterization. The Quadcopter also presents an opportunity to demonstrate our algorithm on a non-walking example, demonstrating its generalizability.

Our Quadcopter consists of a circular base with motors spaced evenly around the circle half the overall size of the Quadcopter. We model the base as being uniform density with

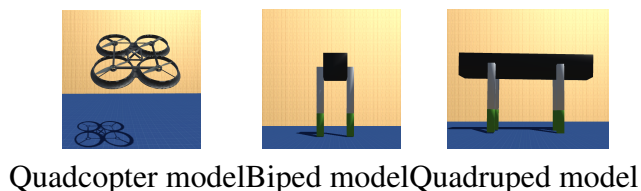


Figure 3-10: The non-Hexapod models.

fixed inertia. Each motor acts as a force element which can provide an impulsive linear force, which provides linear thrusts and torques to the Quadcopter’s base. Our Quadcopter has two parameters - the radius of the base (which in turn positions the motors), and an adjustable mass which can be increased at each motor’s base. We consider the motors chosen *a priori* and massless for this single example.

For our task, we require the Quadcopter to fly in a circle of radius 1 m by hitting 16 waypoints. This simple model and task (274 decision variables) admits an obvious optimal solution: in order to maneuver with as little power output as possible, torquing the robot and maintaining altitude must be as cheap as possible. This is achieved when the motors are extended as far out radially as possible and when the mass is reduced to its minimum.

In our experiments, we allow the Quadcopter radius to vary between 0.1 m and 0.5 m (initialized at 0.3 m,) and allow the mass to vary between 0.3 kg and 0.7 kg (initialized at 0.5 kg). Over 20 trials, our optimizer successfully found the optimal configuration every time in less than 5 seconds. Beyond this, we ran 20 trials fixed at 400 different parameter configurations. A plot showing the resulting average costs of these trials can be seen in Fig. 3-11. The minimum can be found at 0.5 m and 0.3 kg as claimed; our algorithm recovered the optimal parameterization and corresponding motion.

The methods and results presented here build upon a rich body of work on computational design of rigid robots, soft robots, and robot actuators applicable to either domain.

3.5 Related Work

Optimization techniques have shown great promise over recent years as a means of generating complex behaviors from higher level input. Work in [97] demonstrated how gaits could be designed interactively for walking robots. Work in [143] incorporated dynamics into the

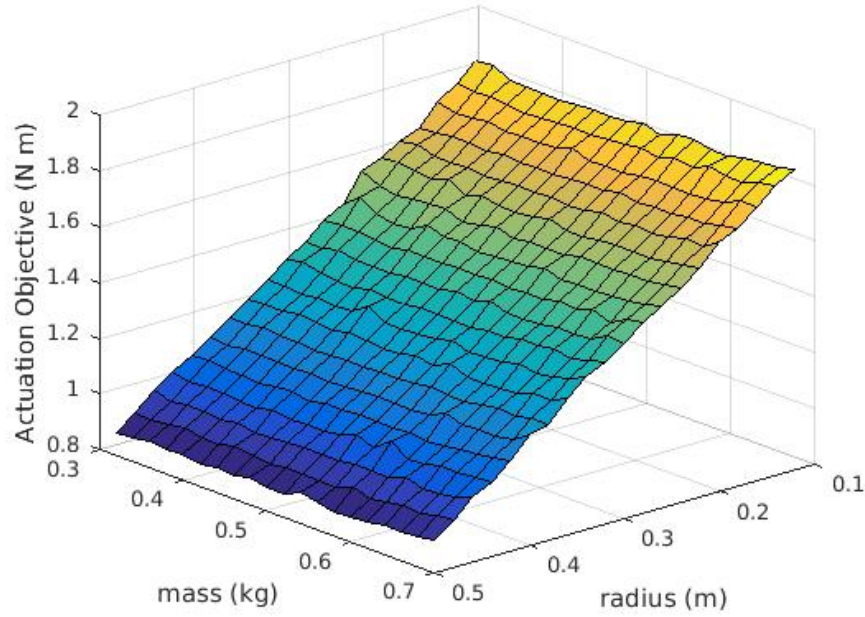


Figure 3-11: A surface plot of the actuation costs versus various parameterizations for the Quadcopter. Our parameterized trajectory optimization is successfully able to find the optimal parameterization at the bottom-left corner.

optimization and demonstrated that an online formulation of trajectory optimization could be used to create complex motions for humanoids by varying the constraints and dynamics. Similarly, [104] and [103] demonstrated that emergent behaviors can be synthesized for arbitrary morphologies from high level specification, and [156] demonstrated that trajectory optimization can be used to synthesize controllers for arbitrary morphologies and high-level gait specification. However, none of these works have incorporated physical parameter changes.

On the other hand, recent results related to design optimization include [27], where a kinematic approach to optimal linkage synthesis is proposed, [99], where robot function is specified in natural language to automate component selection, and [98], which demonstrates data-driven co-design for mechanical and electronic subsystems of functional robots. [18] demonstrated the ability to select discrete robot components, including batteries and actuators, based on constraints operating on mixed discrete and continuous variables, but does not consider geometry or motions.

The work most similar to ours is [38]. In this work, the authors demonstrate the potential

for simultaneously optimizing robot gaits and design parameters. Their method does not factor in contact dynamics and allows only biologically-inspired subsets of potential quadrupedal foothold patterns. By contrast, our method for finding walking motions, which builds upon [117], does not rely on expert knowledge for gait patterns and computes contact forces implicitly as part of the optimization. Further, compared with their evolutionary algorithm-based approach, which provide few theoretical guarantees, we employ a gradient-based optimization approach that is locally optimal in costs and constraints.

Although the co-design of robot structure and motions has been gaining attention recently, it is worth noting that a related problem, the co-design of robot structure and robot *controllers* has been studied for far longer. For instance, a method for simultaneously designing a high-speed, dynamic arm with an optimal PD controller was explored in [111], and the simultaneous design of electric DC motors and their optimal PID controllers was explored in [121].

Concurrent work has been presented on computational co-optimization of kinodynamic system parameters with controllers using reinforcement learning. This approach is attractive because it provides a means for creating closed-loop controllers. Methods of note using this approach are [124] and [58].

3.6 Outlook: Rigid Robot Co-Optimization

We have introduced an algorithm for the co-optimization of robot design and function, and we demonstrate that a natural parametric representation and complete symbolic formulation of constraints, costs, and robot dynamics allows for direct and efficient of physically realizable designs. Our method is able to simultaneously design the robot motion, robot body parameters, and select from available actuators with very little user overhead. Our work suggests two natural follow-up problems whose solutions would aid the effectiveness and usefulness of our method.

First, our method would benefit from a better initialization procedure. Our method assumes that the robot designer can identify initial guesses for the robot motions and can specify them as waypoints. In our examples these initial guesses were fairly obvious, typi-

cally amounting to linear interpolation between starting and ending poses and considering natural biological motion of legs. Further, our method chooses the initial actuation vector at random, which can sometimes lead to local infeasibilities, requiring a random restart. In order to automate the design process further, a similar coarse planning method such as a parametric variant of RRT [82] is needed.

Second, topology is fixed in our method. A method for searching over morphology while searching over design parameters is necessary to explore larger design spaces. Such an optimization would likely be a mixture of discrete and continuous variables, and thus much harder to solve efficiently, though we have already started in this direction using conservative continuous models for motor selection. However, for those interested in methods that optimize robot topology while restricting themselves to a discrete design domain, please see the recent methods of [168, 163].

The introduction of this thesis set forth an ambitious goal: to co-design all of the “organ systems” of a robot such that they perform in optimal harmony. Tying back to that goal, this chapter demonstrated that we can algorithmically co-optimize over a robot’s *skeletal* shape parameters, open-loop control, and actuators. But rigid robots do not encompass the dynamical richness that allows for organisms in nature to be so dexterous and efficient, namely due to a lack of compliance and elasticity. In the next chapter, we will extend the co-optimization ideas established so far to the domain of *soft robots*, which we will focus on for most of the remainder of this thesis. In particular, we will discuss how to co-optimize over such robots’ materials and (the more sophisticated class of) *closed-loop* controllers – and undertaking that will also necessitate the development of a new soft-bodied simulator.

Chapter 4

The (Differentiable) Material Point Method and Soft Robot Co-Optimization

In the previous chapter, we presented an algorithm for co-optimizing dynamic, parametrically specified rigid robots over form and control. Unfortunately, the Parametric Trajectory Optimization algorithm does not directly extend to the soft and compliant robotics regime, as elastic energy potentials make searching over states with direct transcription methods difficult. This chapter introduces and demonstrates a new co-optimization algorithm that scales beyond the soft robot scenario, to the higher dimensional and more kinematically and dynamically complex soft robotics co-design problem. While simpler in form, it relies on a highly optimized, differentiable, soft-bodied simulator as an inner loop of its optimization procedure. For that reason, we begin this chapter by describing our fast, accurate, differentiable soft robot simulator, and then we examine algorithms for the co-design of spatially varying materials and open-/closed-loop controllers for soft robots.

The previous paragraph mentioned something called *differentiable* simulation (layer 3 in the robot design stack). A differentiable simulator is one where any variable of a system (state or design) can be differentiated with respect to any other variable. Differentiable simulation is a bit more powerful than a simulator in which symbolic gradients are computed (as in Chapter 3), since differentiable simulation can have complex program structure such as branch statements or loops. Yet, the simulator demonstrated here will also scale to very high parameter dimensions, since it relies on efficient backpropagation schemes for gradient

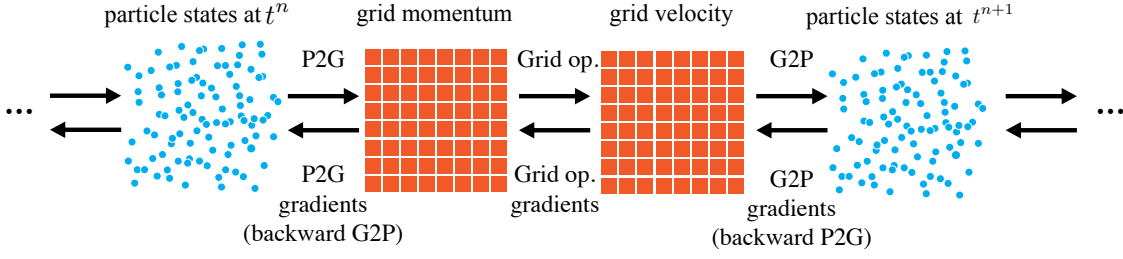


Figure 4-1: One time step of MLS-MPM. Top arrows are for forward simulation and bottom ones are for back propagation. A controller is embedded in the P2G process to generate actuation given particle configurations.

computation. In fact, the idea of a differentiable system is far from new, and readers familiar with deep learning may be quick to note that the differentiability of high-dimensional deep learning systems is what (like our soft robot co-optimizer) allows them to be trained so efficiently *via* gradient-based methods. In fact, for the remainder of this discussion, if it helps, one can think of a single time-step of a simulation as a single physically-based layer of a (very deep and nonlinear) neural network.

We now proceed by describing the implementation of a differentiable material point method (dubbed ChainQueen), as in [66]. We describe the simulator in-depth here, as this simulation method is used for all of the remaining algorithms in this thesis involving soft robot co-design. Further, we describe extensions to the core simulator in order to adapt it to the soft robot design setting.

4.1 Preliminaries

ChainQueen has two features — forward simulation, in which a soft-body simulation is timestepped forward in time, and backward propagation, in which gradients of some *loss function* are computed with respect to physical and controller parameters of the robot. Unlike passive soft body simulators, ChainQueen is a soft robot simulator, meaning that control signals and actuation are applied at appropriate times during the simulation. A simulation occurs over a user-specified T timesteps. The backward gradient computation procedure involves the same forward simulation operations carried out in MPM, but in a reverse order; a visual depiction can be seen in Fig. 4-1.

The loss function is a function which takes in a measure of the robot state (positions, velocities, deformation gradients, *etc.*) and actuation signal at each time step i and a measurement of the robot’s performance. For convenience, we also define an explicit final loss for the design and the final state T . ChainQueen thus is well-suited for computing gradients with respect to loss functions of the form:

$$f(\mathcal{M}) = g(\mathbf{s}_T, \boldsymbol{\phi}) + \sum_{i=0}^{T-1} h(\mathbf{s}_i, \mathbf{u}_i) \quad (4.1)$$

Here, $\mathcal{M}$ is referred to as the “Memo”, a descriptor of the robot state evolution throughout simulation, $\mathbf{s}_i$ is the state of the robot at timestep i , $\mathbf{u}_i$ is the received actuation signal at timestep i , and $\boldsymbol{\phi}$ is the static design of the robot. While $\mathbf{u}$ is typically a compact vector, as one can imagine, $\mathbf{s}$ and $\mathbf{u}$ can be particularly complex, having to describe a very high-dimensional system. We describe each in turn later in this section.

4.2 Forward Simulation

ChainQueen uses the moving least squares material point method (MLS-MPM) [64] to discretize continuum mechanics, which is governed by the following two equations:

$$\rho \frac{D\mathbf{v}}{Dt} = \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{g} \quad (\text{momentum conservation}), \quad (4.2)$$

$$\frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{v} = 0 \quad (\text{mass conservation}). \quad (4.3)$$

Here we only briefly cover the basics of MLS-MPM and readers are referred to Jiang et al[71] and Hu et al[64] for a comprehensive introduction of MPM and MLS-MPM, respectively. The material point method is a hybrid Eulerian-Lagrangian method, where both particles and grid nodes are used. Simulation state information is transferred back-and-forth between these two representations. We summarize the notations we use in this paper in the Table in Appendix ???. Subscripts are used to denote particle (p) and grid nodes (i), while superscripts such as n and $n + 1$ are used to distinguish quantities in different time

steps. The MLS-MPM simulation cycle has three steps:

1. **Particle-to-grid transfer (P2G).** Particles transfer mass m_p , momentum $(m\mathbf{v})_p^n$, and stress-contributed impulse to their neighbouring grid nodes, using the Affine Particle-in-Cell method (APIC) [70] and moving least squares force discretization [64], weighted by a compact B-spline kernel N : ($\mathbf{G}_p^n$ below is a temporary tensor)

$$m_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) m_p, \quad (4.4)$$

$$\mathbf{G}_p^n = -\frac{4}{\Delta x^2} \Delta t V_p^0 \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n, \quad (4.5)$$

$$\mathbf{p}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) [m_p \mathbf{v}_p^n + \mathbf{G}_p^n (\mathbf{x}_i - \mathbf{x}_p^n)]. \quad (4.6)$$

2. **Grid operations.** Grid momentum is normalized into grid velocity after division by grid mass:

$$\mathbf{v}_i^n = \frac{1}{m_i^n} \mathbf{p}_i^n. \quad (4.7)$$

Note that neighbouring particles interact with each other through their shared grid nodes, and collisions are handled automatically. Here we omit boundary conditions and gravity for simplicity.

3. **Grid-to-particle transfer (G2P).** Particles gather updated velocity $\mathbf{v}_p^{n+1}$, local velocity field gradients $\mathbf{C}_p^{n+1}$ and position $\mathbf{x}_p^{n+1}$. The constitutive model properties (e.g. deformation gradients $\mathbf{F}_p^{n+1}$) are updated.

$$\mathbf{v}_p^{n+1} = \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) \mathbf{v}_i^n, \quad (4.8)$$

$$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2} \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) \mathbf{v}_i^n (\mathbf{x}_i - \mathbf{x}_p^n)^T, \quad (4.9)$$

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \mathbf{C}_p^{n+1}) \mathbf{F}_p^n, \quad (4.10)$$

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1}. \quad (4.11)$$

4.2.1 Boundary conditions (Contact models)

ChainQueen adopts four commonly used MPM boundary conditions from computer graphics [138]. These boundary conditions happen on the grid, after Eqn. 4.7. Denoting the grid node velocity as $\mathbf{v}$ and local boundary normal as $\mathbf{n}$, the four boundary conditions are

Sticky Directly set the grid node velocity to 0. *i.e.*, $\mathbf{v} \leftarrow \mathbf{0}$.

Slip Set the normal component of the grid node velocity to 0. *i.e.*, $\mathbf{v} \leftarrow \mathbf{v} - (\mathbf{v} \cdot \mathbf{n})\mathbf{n}$.

Separate If the velocity is moving away from the boundary ($\mathbf{v} \cdot \mathbf{n} > 0$) then do nothing. Otherwise use **Slip**. This can be considered as a special case (coefficient of friction = 0) of **Friction**.

Friction If the velocity is moving away from the boundary ($\mathbf{v} \cdot \mathbf{n} > 0$) then do nothing. Otherwise apply Coulomb’s law of friction to compute the new tangential and normal components of velocity after collision and friction.

4.3 Material Models

We extend ChainQueen’s original material model, which was simply fixed corotated material, in order to further support NeoHookean materials as well. We emphasize that other material models can be simply, modularly added as an option in ChainQueen, so long as they are differentiable.

The difference between NeoHookean, corotated, and linear materials has been well studied and documented [15, 91, 131], especially in the static load case. A linear versus a nonlinear, NeoHookean materials provides a modest but not insignificant effect on a soft robot’s dynamical response, making more accurate models preferable. The (wall-time) difference in the computational cost of the NeoHookean and corotated models is negligible.

4.3.1 Fixed Corotated

The fixed corotated material model [137] is defined as having first Piola-Kirchoff Stress function $\mathbf{P}$:

$$\mathbf{P}(\mathbf{F}) = 2\mu(\mathbf{F} - \mathbf{R}) + \lambda(J - 1)J\mathbf{F}^{-T} \quad (4.12)$$

where $J = \det(\mathbf{F})$, and λ and μ are the first and second Lamé parameters, determined by the material's Young's modulus and Poisson's ratio. Here, $\mathbf{R}$ is the rotational component of $\mathbf{F}$, computable *via* the (differentiable) polar decomposition.

4.3.2 Neohookean

The Neohookean material model is especially popular in modeling nonlinear rubbers and silicones. The Neohookean elastic stress tensor is defined as:

$$\mathbf{P}(\mathbf{F}) = \mu(\mathbf{F} - \mathbf{F}^{-T}) + \lambda \log(J)\mathbf{F}^{-T} \quad (4.13)$$

Although this energy model possesses a pole at $J = 0$, this corresponds to the situation where the material is compressed to a singularity. While instability caused by this singularity can happen in practice in general, it rarely occurs in the types of systems explored here, and never happened in any of our experiments.¹

4.3.3 Incompressible materials

Materials may be made approximately incompressible by adding a stress, as described by Bonet and Wood [15]. In this scenario, an additional incompressibility stress tensor is added to the original stress tensor: $\mathbf{P} = \mathbf{P}_{elastic} + \mathbf{P}_{inc}$:

$$\mathbf{P}_{inc}(\mathbf{F}) = \frac{E}{3(2 - 2\nu)} p(J - 1)^{p-1} J\mathbf{F}^{-T} \quad (4.14)$$

¹Note that in MPM the update of J (i.e. $\det(F)$) is multiplicative instead of additive, therefore getting $J \leq 0$ is rare.

where p is a parameter ≥ 2 . The larger p is chosen, the more strongly incompressibility will be enforced. The ability to simulate incompressible materials is included in our ChainQueen extensions.

4.4 Actuation Models

Designed to model real-world soft robots, our extended ChainQueen implementation supports two common actuation models found in soft robots. The first, aimed at modeling fluidic actuators, is a stress-based actuation model. The second, which we present here and aimed at modeling cable-based actuation, is a force-based actuation model. These models are physically based and do not inject any net fictitious forces or pressures to the system (thus respecting Newton’s third law). We modify the classical MLS-MPM formulation with two additional steps in order to account for application of the actuation.

In order to model stress-based (including fluidic) actuators, we have designed an actuation model that stretches a given particle p with an additional cauchy stress:

$$\mathbf{A}_p = \mathbf{F}_p \boldsymbol{\sigma}_{pa} \mathbf{F}_p^T \quad (4.15)$$

Here, $\boldsymbol{\sigma}_{pa} = \text{diag}(a_x, a_y, a_z)$. This equation corresponds to the stress in material space. Particles corresponding to a given actuator are assigned at robot design time; each actuator thus affects many corresponding particles applying the same stress. Note that this formulation naturally allows for both isotropic and anisotropic pressure actuators, useful for modeling directionally constrained actuators (*e.g.* Sun et al [140]). Already, this model is well-suited for modeling stress-based actuators (*e.g.* Hara et al [60]) and pneumatic actuators (*e.g.* Marchese et al[94]). This model is similarly well-suited to simulating hydraulic actuators, modeling the actuator’s particles as incompressible, as previously described in Section 4.3.3.

In order to model force-based actuators, such as cable-driven actuators, we simply integrate these forces when updating our particle velocity, creating an additional summable velocity term, $\mathbf{v}_{act}$:

$$\mathbf{v}_{act} = \mathbf{f}_{act}dt/m \quad (4.16)$$

Velocities, forces, and masses here refer to particle states. Here, $\mathbf{f}_{act}$ is computed as some user-defined force model. We detail one such model here, which also provides a concrete example of how actuation signals can be applied.

Let $\{\mathbf{p}_i\}$ be an ordered sequence of r particles through which a cable is routed. Let the actuation signal associated with this actuator be the scalar u , and let R be the rest length of the cable. We then model a cable energy, based off Hooke’s law:

$$E_{\text{cable}} = \frac{k}{2} (\max(0, (\sum_i^{r-1} d(\mathbf{p}_i, \mathbf{p}_{i+1})) - \max(0, Ru)))^2 \quad (4.17)$$

Here, and throughout our simulator, we assume actuation signals to be normalized between -1 and 1 . d is the L_2 distance function and k is a user-defined cable stiffness. Here, the summation computes the distance the cable must span, and Ru is the true length of the cable given the current actuation parameters (L can be thought of as a fraction of unspooled cable since we clamp it to be positive). If the true length of the cable is larger than the summed particle distance, then there is slack in the cable, and thus no stored energy. Otherwise, the cable stores an energy related to the difference in these quantities, much like a linear spring. Forces are computed by differentiating the energy with respect to positions (which can be computed easily in TensorFlow *via* `tf.gradients` with respect to the positions at time step i) and integrated as described above.

4.5 Back-Propagation

In order to compute gradients of losses of the form of equation 4.1, we apply the following backward propagation scheme. First, we assume a memo $\mathcal{M}$ computed *via* a forward simulation is available. We begin by computing gradients with respect to the final loss g , as $\frac{\partial l_T}{\partial \mathcal{M}} = \frac{\partial g}{\partial s_T} + \frac{\partial g}{\partial \phi}$ the partial derivative with respect to the final state, we then recursively

compute: $\frac{\partial f(\mathcal{M}_T)}{\partial \mathcal{M}_{i-1}} = \frac{\partial f(\mathcal{M}_T)}{\partial \mathcal{M}_i} \frac{\partial \mathcal{M}_i}{\partial \mathcal{M}_{i-1}} + \frac{\partial h_{i-1}}{\partial \mathcal{M}_{i-1}}.$

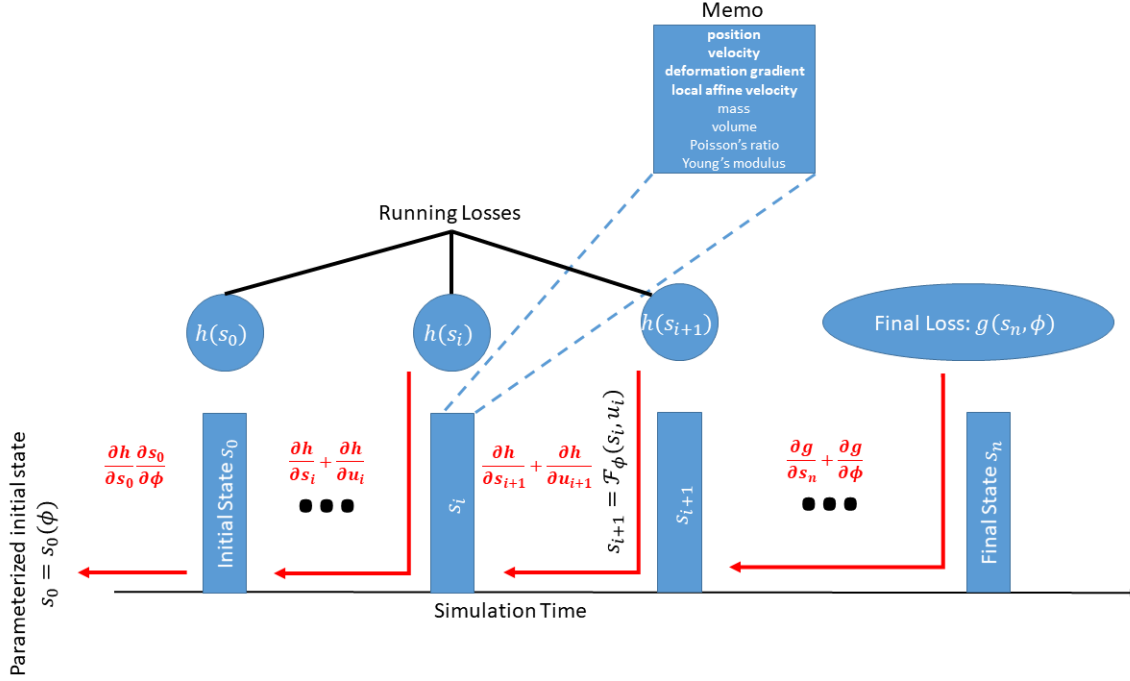


Figure 4-2: The backpropagation graph for a time step (which may include a final state and/or a final loss). The forward simulated system is reverse-mode autodifferentiated in order to compute gradients with respect to state, actuation, and/or controller and design parameters. A memo containing all of the relevant information for analysis is computed and maintained according to the checkpointing strategy discussed in 4.5.1.

Computing $\frac{\partial \mathcal{M}_i}{\partial \mathcal{M}_{i-1}}$ is highly nontrivial, and, in the absence of an autodifferentiation framework, requires extensive manual derivation. While one can implement an autodifferentiated version (in TensorFlow), implementation's enormous graph is far too slow (both in compilation and runtime) to be practical for complex tasks. To remedy this, we manually derived the simulation gradients $\frac{\partial \mathcal{M}_i}{\partial \mathcal{M}_{i-1}}$ and packaged them along with forward simulation in highly optimized GPU kernels. For the full details of the backward propagation derivation, please see Appendix A.3.

4.5.1 Implementation

Although we have provided all the ingredients for our fully differentiable MPM simulation, we describe two optimizations which make practical implementation fast and efficient.

First, it may be tempting to create a TensorFlow graph which adds each instance of

our simulation kernel (along with any control code) to one large graph, and backpropagate through the entire graph with a single call to `tf.gradients`. Unfortunately, we have found the TensorFlow’s compiler is unable to efficiently optimize such a graph (despite its simplicity). Therefore, we build a graph that simply includes a single simulation step (with control), and manually perform back-propagation by iteratively calling `tf.gradients`. That graph, whose construction we describe in more detail in Section 4.7, can be seen in Fig. 4-3.

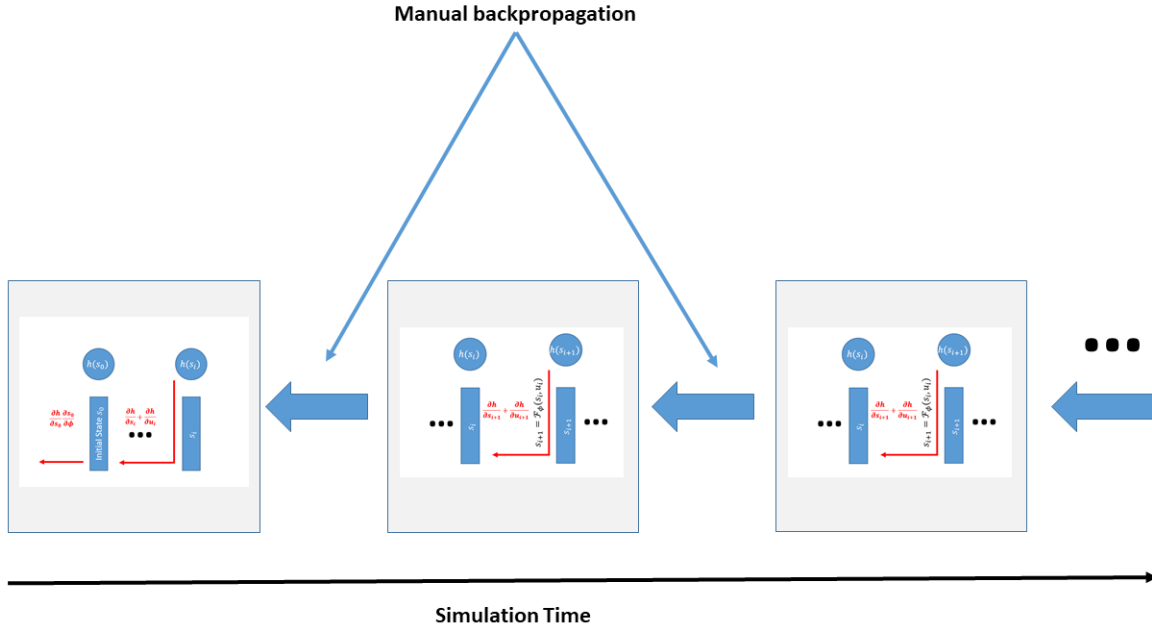


Figure 4-3: Each step is, in itself, a large TensorFlow graph (from Fig. 4-2), which the TensorFlow compiler is ill-suited to optimize. For practical efficiency, we compute only the gradients with respect to individual time steps, and manually backpropagate them using `tf.gradients`.

Second, although a scene may need to be simulated at a small dt for physical stability, real-world controllers typically run at a much lower frequency. In order to account for this, we allow for substepping, in which the simulation kernel is executed many times at a smaller time step between simulation steps for stability. The number of substeps, b , to use is a user parameter that can be chosen, and the effective time step used in that kernel is set to $\frac{dt}{b}$. Since TensorFlow-native controllers are typically orders of magnitude slower to execute than our MPM kernel, substepping has the added bonus of providing faster simulation and

back-propagation. This is a new optimization included in our ChainQueen extension which has two benefits. First, it greatly accelerates robot optimizations, since everything aside from the (highly optimized) timestepping kernel is the bottleneck. Second, it allows us to more faithfully model real-world robots, which might have limited controller frequencies.

The memory consumption to store all simulation steps is proportional to the number of time steps. In practice when the number of time steps N is high, the memory consumption may grow as $O(N)$. In order to reduce memory consumption, we apply a checkpointing trick here to trade time for space. We partition the whole simulation into $O(\sqrt{N})$ segments, each with $O(\sqrt{N})$ time steps. During forward simulation, we memorize not only the latest simulation state, but also the initial simulation state of each segment. This means we only have to memorize $O(\sqrt{N} + 1) = O(\sqrt{N})$ states. During backpropagation, we compute the gradients in a segment-wise manner, from the final segment to the initial segment. For each segment, we first recompute all the time steps and store the results. This would result in $O(\sqrt{N})$ temporary simulation states. Then we can back propagate within this segment. The total memory consumption of this checkpointing scheme is $O(\sqrt{N} + \sqrt{N}) = O(\sqrt{N})$, which is a significant improvement over $O(N)$. At the same time, the number of forward time steps is $O(2N) = O(N)$, and the number of backward time steps stays $O(N)$.

4.6 Pitfalls and Resilience

Despite our simulator’s strengths, there are two common failure modes which can cause soft simulation to fail. We describe each in turn, below.

Large time steps Our simulator uses explicit forward euler time-stepping, meaning that the time steps taken for each substep must be small in order to ensure stable simulation. In particular, as shown in Fang et al [43], the Δt limit for explicit time integration is $C\Delta x\sqrt{\frac{\rho}{E}}$, where C is a constant close to one, ρ is the density, and E is the Young’s modulus. This is the speed of sound, or the speed of vibrations as it travels through an elastic medium. In our workflow, users specify the time step size, but must take care; if the time step is chosen to be significantly larger than this limit, numerical explosions can occur.

4.6.1 Numerical Fracture

A more subtle failure mode, but one that is present as a drawback of MPM is numerical fracture, or “tearing” of the elastic medium. However, even just a few particles per cell can show resilience to numerical fracture.

4.7 Soft Robotics API

In order to streamline the design and optimization of soft robots, we have developed a simple-to-use, high-level API for interfacing with the ChainQueen physics engine. Our API abstracts the robot simulation process into four parts: observation, control, actuation, and physical timestepping. To optimize a robot for a task, users must specify an initial robot morphology (shape and materials), place actuators on the robot, choose an “observer” model, choose a controller, and specify an objective of the form in equation 4.1. Users may also select from an array of constrained and unconstrained optimizers with which to optimize the robot. Using this API, a robot co-design problem may be specified in fewer than 50 lines of code. (Here, we define a line of code as 80 characters in length, with statements longer than 80 characters extended to multiple lines. We do not include comments or import statements.) Further and perhaps more importantly, users only need to specify the above seven items (topology, materials, actuators, observer, controller, objective, optimizer), making the conceptual workflow simple and amenable to rapid prototyping and experimentation.

In the remainder of this section, we describe in more detail how these components combine to form the TensorFlow computational graph, describe the components currently available in the API, and finally present a simple, example program to demonstrate the simplicity of our API.

4.7.1 Graph Construction — Preliminaries

Previously, we have referred to two parts of our simulation - physical simulation, and control. In truth, what we describe as “control” actually is composed of multiple subcomponents.

We describe these here.

Assume a robot is at a state s_i . First, that state s_i is fed into what is referred to as an *observer*, which transforms the (very) large state description and summarizes it in what is typically a more compact form, the observation o_i . This observation o_i is then fed into a controller in order to produce an actuation signal, u_i . The actuation signal is fed into the actuators in order to produce appropriate stress or forces to apply. These stresses and forces are then fed into the physical simulator, along with the state s_i , and substepped b times, producing state s_{i+1} . At this point, we can apply the final loss g or the running loss h , depending on if $i = T$ or not, and accumulate it into our total loss.

We make one final note about this accumulation process. Although as described above, the backpropagation scheme accumulates running losses through a summation, our system is easily modifiable such that other differentiable accumulation functions may be used, including max, min, product, and so on. Derivation of the backpropagation scheme is mechanically similar for each of these.

The entire process is demonstrated in Fig. 4-4.

For the remainder of this section, we describe the available options for the components described above, before providing a complete example of specifying a soft robot control optimization example using our API.

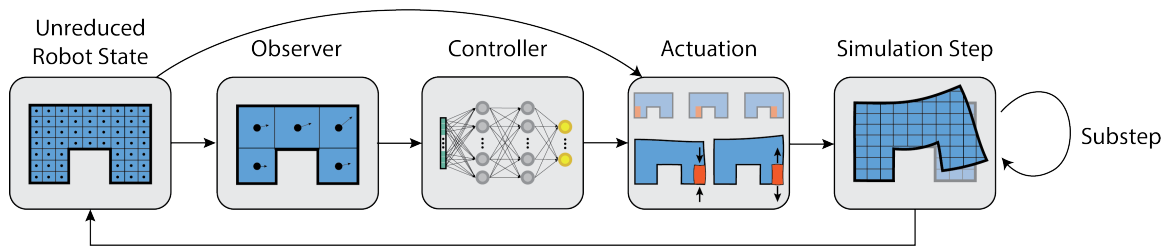


Figure 4-4: One full step of simulation. The full state of the robot (masses, positions, and velocities of particles and grids, particle material properties, and particle volumes deformation gradients) is fed into an observer. The observer transmits a more compact summary of the full state to the controller function, which produces an output actuation. The system is then substepped a number of user-specified times with the computed actuation, resulting in the robot state for the next time step.

4.7.2 Choosing the Objective Function

Choosing the objective function amounts to choosing a final cost g , a running cost h , and an accumulator function (which we have chosen to be summation). Examples of final costs may be (the negation of) how far forward a robot has traveled, measured by the average position of its particles, or for design problems, a regularizer to keep design parameters close to their initial values. Examples of running costs include an actuation penalty (say, $\frac{1}{2} \mathbf{u}_i^T \mathbf{u}_i$), which is popular for keeping robots energy efficient. Objectives may be weighted in order to trade off their relative importance.

4.7.3 Observers

As described, observers translate potentially high dimensional states into typically low dimensional observations. Our API currently supports two observation models. In the first observation model, the centroid observer computes the center of position, velocity, and/or acceleration of manually, or automatically segmented particle clusters, and concatenates these vectors into a long observation vector. The second observation model (also known as the convolutional variational autoencoder, CVAE) observer, applies a convolutional encoder to the grid cells and returns the latent space as the observer. The autoencoder may be trained offline or in-the-loop (as in Spielberg et al[136]). We expose the API as described in Appendix Listing 1 in order to allow users to easily define their own custom observation functions.

4.7.4 Controllers

Controllers translate (typically compact) observations into actuation signal. Our API currently supports two controller models. The first is an open-loop controller, which simply maintains an actuation vector for each time step and returns it (notice this controller does *not* consume the observation in any way). The second, a closed-loop neural network controller, applies a multi-layer perceptron (MLP) and returns an actuation output. The API for users to extend is described in Appendix Listing 1.

4.7.5 Actuators

Actuators translate actuation signals into stresses or forces to be applied in simulation. An example of each was described in Section 4.4. The API for users to extend is presented in Appendix Listing 3.

4.7.6 Optimizers

We have created wrappers for PyGMO[37] and TensorFlow’s internal first-order optimizers in order to allow users to easily experiment with both first-order and second-order optimizers. Our first-order optimizer wrapper also allows for variable bounds (maintained *via* a projection operator, in order to keep physical design parameters within reasonable bounds. Our second-order optimizer allows users to solve problems with complex equality and inequality constraints. Further optimizers could include, for example, the Learning-In-The-Loop optimizer from Spielberg et al[136]. Users are free to define their own optimizers; the API is described in Appendix Listing 4 A.2.

4.7.7 Other Utilities

In addition to the aforementioned components, a library of useful functions is also provided to the user. These include the ability to load robot geometries directly from image files and meshes, in order to simplify the topological design of the robot.

4.7.8 An Example

Appendix A.2 presents an example with fewer than 50 lines of code in which a 2D soft walker is optimized in control and material, while regularizing on actuation. The code is heavily commented in order to obviate each line’s purpose. Results for this problem are presented in section 4.9.

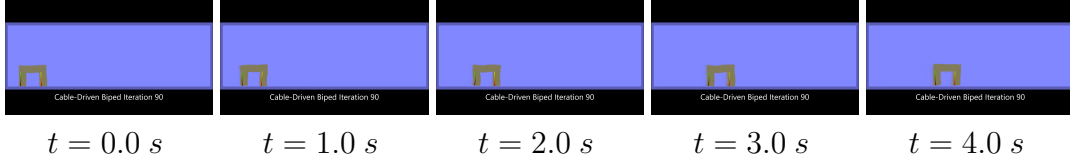


Figure 4-5: A montage of our simulated Cable-Driven Biped in motion. Colored lines demonstrate cable placement.

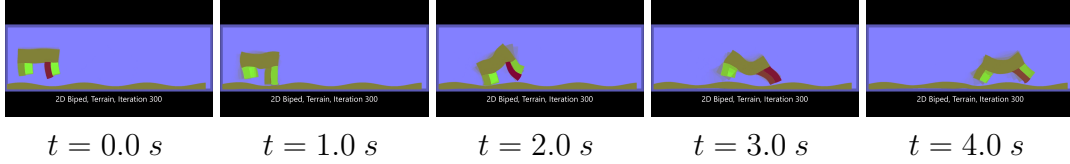


Figure 4-6: A montage of our simulated terrain-traversing Biped in motion. Green colored actuators are expanding; red, contracting.

4.8 Applications

We demonstrate our enhancements to ChainQueen on two classes of applications: inference tasks, in which we algorithmically reason about the physical properties of a simulated system and pure control tasks, in which we optimize control parameters for some target robot performance. All units used in the following experiments are in a dimensionless system; for scale, robot models are on the order of around 0.3 – 0.5 in max length. Please see the Figs. 4-13, 4-6, 4-7, 4-5 for simulations and optimized motions. For even more experiments, including experiments which demonstrate dominance over model-free approaches such as reinforcement learning, please see the original ChainQueen manuscript[66].

For all experiments, the more physically realistic NeoHookean materials were chosen, as fixed co-rotated materials were showcased in the original ChainQueen paper.

All experiments were performed on a laptop with a 2.9GHz i7 Intel processor, NVIDIA GeForce GTX 1080 graphics card, and 32GB of RAM.

In all drawings of robots at rest in the following sections, visible pneumatic actuators have been segmented with black lines and labeled with red circles.

4.8.1 Inference

We demonstrate the ability of ChainQueen to determine the physical parameters of simulated objects. Given a sequence of keyframes to match or a final pose to match, we define l , which

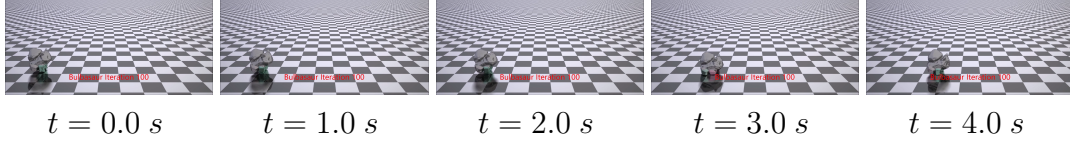


Figure 4-7: A montage of our simulated 3D Bulbasaur in motion. Green colored actuators are expanding; red, contracting.

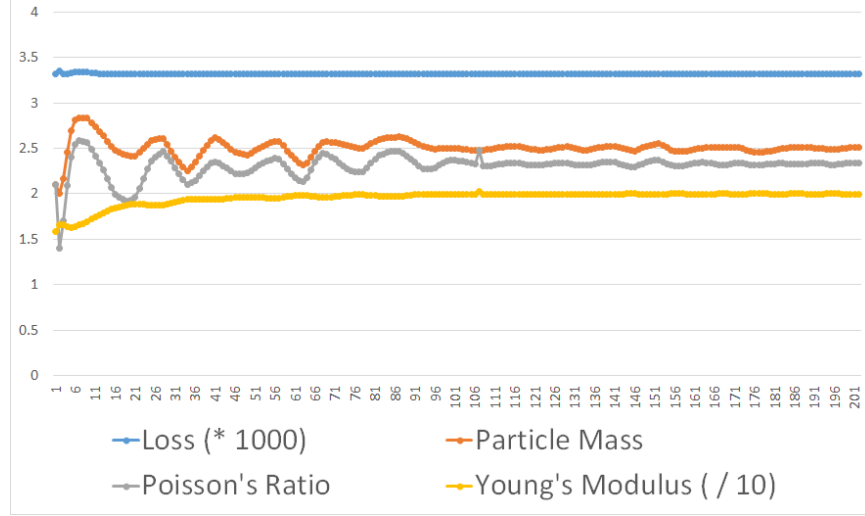


Figure 4-8: Convergence of system identification for the soft finger. All material parameters converge to their initial values within around 100 epochs of SGD. The x -axis presents epochs of SGD, while the y -axis meaning depends on the trend-line observed. While the loss does not appear to change much in the absolute scale, this is a local (and ground truth) minimum. It cannot converge completely to 0 due to the noise added to the dataset.

corresponds to the running loss h or the final loss g as:

$$l(s_i) = \sum_{\tau \in \mathcal{T}} \|s_i^\tau - r_i^\tau\|_2^2, \quad (4.18)$$

where r is some target reference pose at that point in time. The summation τ is over different trajectories in a set $\mathcal{T}$, generated by different fixed, specified actuation sequences, allowing users to fit robustly to multiple data sequences.

We minimize this loss using first-order optimization, using the Adam[76] optimizer. In the case where one optimizes over multiple trajectories, stochastic gradient descent is performed over the set of given trajectories.

In our demonstration, we attempt to fit three homogeneous material parameters —

Young’s modulus, Poisson’s ratio, and density (as specified by particle mass) — to reconstruct the trajectory of a dynamically bending, 2D, cable-driven soft finger. The finger has two cables, one on the left side and one on the right side, and is pulled in 10 different leftward-bending trajectories (generated from ten different actuation sequences. Keyframes (200) are recorded from each trajectory. Ten copies of each trajectory are created, with noise added normally to each of 100 tracked particles (used for measuring trajectory fitting accuracy). This is used in lieu of the ground-truth trajectory, in order to emulate motion capture error. Ground truth for trajectory generation was set to 20 for Young’s Modulus, 2.33 for particle mass, and 0.25 for the Poisson’s ratio. Before optimizing, initial guesses were set to 15 for Young’s Modulus, 3.0 for particle mass, and 0.3 for the Poisson’s ratio. The loss was chosen to be the average squared particle distance of 100 chosen particles on the soft arm over all trajectories; this was optimized using stochastic gradient descent over the dataset of trajectories. Results can be seen in Fig. 5, featuring convergence of loss and the three constitutive material parameters. (The loss and Young’s Modulus were rescaled in this plot in order to put all plots on the same plot.)

4.8.2 Control

We begin our control experiments by demonstrating a few pure control tasks. These include a 2D cable-driven Biped, a 2D pneumatically powered biped on graded terrain, and a pneumatically powered “Bulbasaur” (modeled after the Pokémon) loaded directly from a .stl file. We describe each briefly here; specifics of the parameters of each experiment can be found in Appendix A.4. The cable-driven 2D walker is an open-loop control example, the others are all closed-loop control using a MLP (two hidden layers of 64 neurons) controller. All examples in this section were optimized *via* the Adam optimizer. For all tasks demonstrated here, the objective is for the robot to walk forward as far as possible in the allotted time, thus $g(\mathbf{s}_T) = -\mathbf{x}_T$, where $\mathbf{x}_T$ is the mean of the robot’s particles. A small actuation regularizer $h(\mathbf{u}_i) = \frac{10^{-6}}{2} \mathbf{u}_i^T \mathbf{u}_i$ was added as a running cost to each experiment to regularize the motion. Each experiment in this section was repeated 10 times; 90% confidence intervals are presented for each experiment.

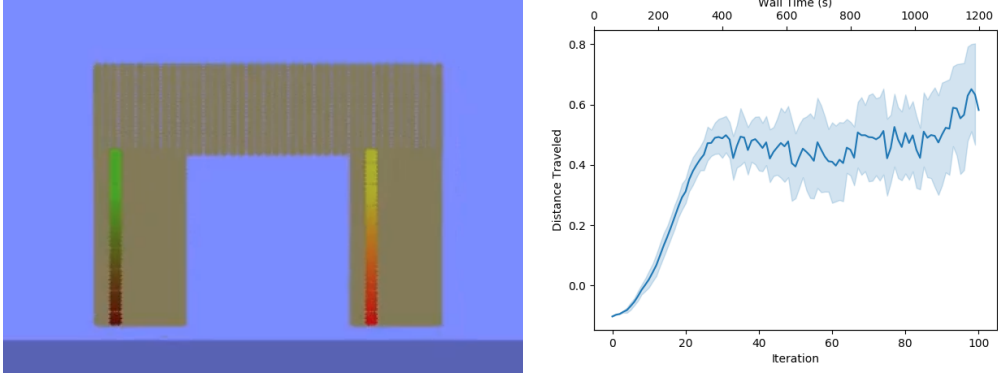


Figure 4-9: Progress of the 2D Cable-Driven Biped vs. iteration and wall-time. By pulling in an alternating motion on each of the two cables, the biped can ambulate forward.

Please see Figs. 4-13, 4-6, 4-7 for demonstrations of the optimized controllers.

2D Cable-Driven Walker

In this example, we demonstrate a first, simple 2D locomotion example. A 2D biped with a cable running down each of its legs is tasked with walking forward. These cables are able to contract from their rest pose (but not expand). They are offset from the center; contracting a cable causes the leg to bend toward that cable, allowing the robot to walk. This problem has 5024 particles, for 10048 degrees of freedom. In this example, the goal is for the robot to walk as far forward (to the right) as possible in the allotted time. Fig. 4-9 presents a rendering of the Biped and the progress of the Biped with optimization iteration and wall time, demonstrating that this problem can make significant progress (walking two body lengths) within just 5 minutes.

2D Biped With Terrain

In this example, we demonstrate the ability of ChainQueen to simulate and optimize over varying terrain. In this demonstration, a 2D Biped must walk as far to the right as possible in the allotted time. This task is tricky for a few reasons. First, gradients must accurately capture the effects inter-particle collisions. Second, the optimizer must learn a control scheme that is dependent on the robot's location in space. Third, the task itself is dynamically challenging, at times requiring the robot to be able to run up slopes of nontrivial steepness. ChainQueen

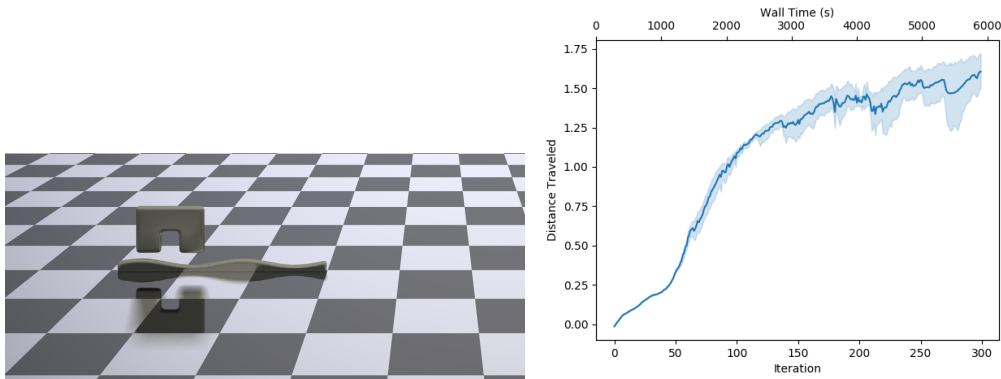


Figure 4-10: Progress of the 2D Biped along terrain vs. iteration and wall-time.

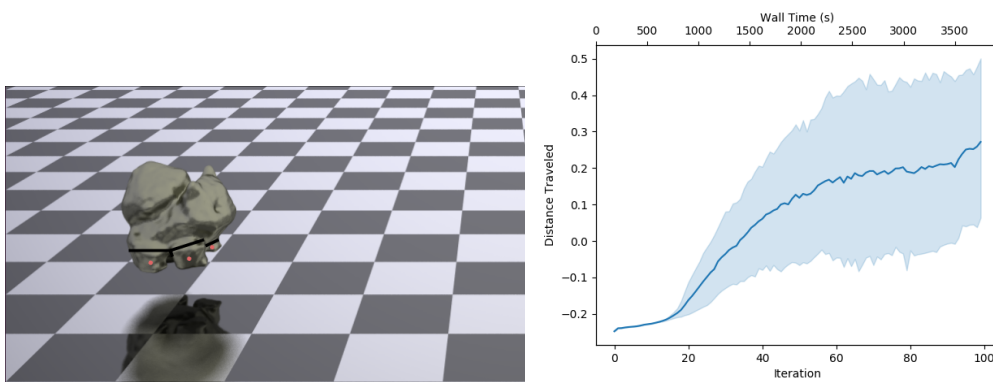


Figure 4-11: Progress of the 3D Bulbasaur vs. iteration and wall-time.

is capable of all of this, and efficiently optimizes the controller for this soft robot; see Fig. 4-10 for details. This problem has 36200 particles (including the terrain), for 72400 degrees of freedom.

3D Bulbasaur

Here, we present our first 3D control optimization example, in which the robot must run as far to the right as possible in the allotted time. The Bulbasaur is instantiated directly from a .stl mesh file based off the popular video game character (with actuators then specified manually). The actuators are placed in the four quadrants of the bottom quarter of the design, which roughly places one actuator in each leg. Because of the curvature of the legs, a single actuator in each is enough to enable a lopping motion forward. Progress of the Bulbasaur optimization can be seen in Fig. 4-11. This problem has 22084 particles, for 66252 degrees of freedom.

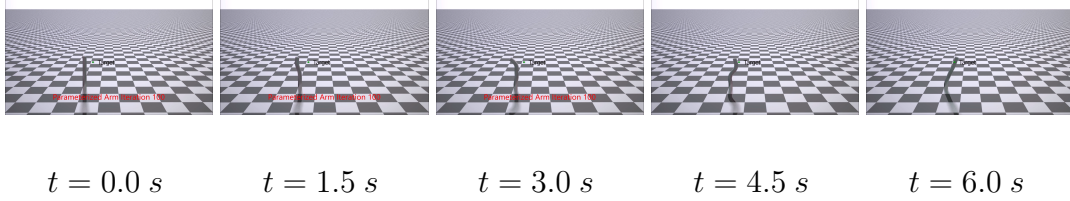


Figure 4-12: A montage of our simulated co-optimized Arm in motion, along with the original parameterization. Green colored actuators are expanding; red, contracting. This problem is impossible to solve without changing the design. The initial arm parameterization is less than half the final optimized height.

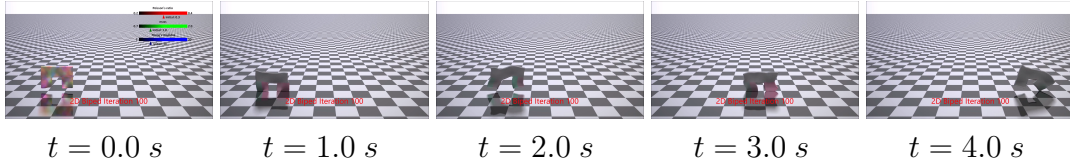


Figure 4-13: A montage of our simulated co-optimized Biped in motion, with the first frame showing the optimized material distribution. Green colored actuators are expanding; red, contracting.

4.9 Soft-Body Co-Optimization

We demonstrate how our differentiable soft body simulation can be used to co-optimize over both closed- and open-loop controllers as well as high-dimensional parametric soft robot designs. We give particular attention to optimizing over neural network controllers and spatially varying materials of soft robots. Our method is a direct application of gradient-based optimizers (Adam or a second-order optimizer, depending on the problem), through a differentiable MPM simulator. We describe the specifics on an example-by-example basis, and demonstrate how co-optimization can lead to marked improvements in robot performance over single domain optimization.

In this section, we present a suite of open-loop and closed-loop tasks, in which we simultaneously optimize over spatially varying material parameters of the robots. Each of these examples uses pneumatic actuators.

Please see Figs. 4-13, 4-14, 4-15, 4-16, 4-12 for demonstrations of the optimized controllers and robot designs.

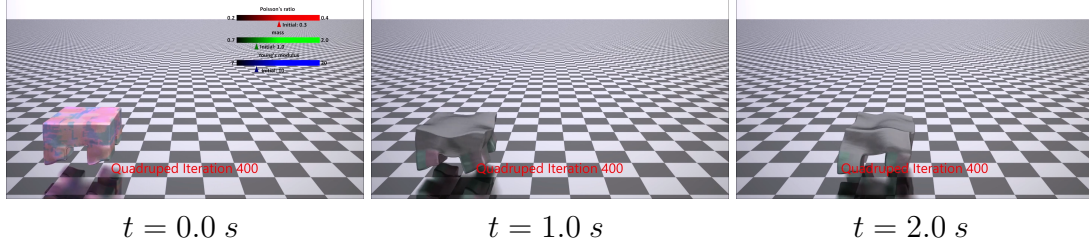


Figure 4-14: A montage of our simulated co-optimized Quadraped in motion, with the first frame showing the optimized material distribution. Green colored actuators are expanding; red, contracting.

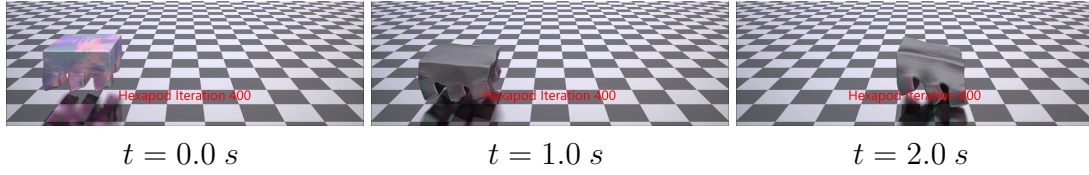


Figure 4-15: A montage of our simulated co-optimized Hexapod in motion, with the first frame showing the optimized material distribution. Green colored actuators are expanding; red, contracting.

Open-Loop Control

In these demonstrations, we show the ability of ChainQueen to efficiently optimize over arm reaching tasks. In these tasks, the arms must reach a target goal ball region.

3D Arm The 3D arm based on the 2D arm from the original ChainQueen manuscript is tasked with reaching a goal region with the end of its arm, shown in Fig. 4-17. It must further stop at this goal with 0 velocity. The arm hangs upside down from the ceiling; in this task, gravity is enabled, requiring the arm to have to “swing up” against gravity. This is a challenging task, as the stiffness of the arm and gravity requires the arm to swing back and forth in order to build up enough momentum to reach its target, like a soft analogue to a pendulum. For this experiment we used the constrained sequential quadratic programming solver, WORHP [16]. The experiment is performed in three configurations - with a fixed Young’s modulus = 300, at fixed Young’s modulus = 400, and with Young’s modulus initialized at 300 but co-optimized with control in a range between 150 and 400. Co-optimization dominates the fixed soft arm designs in optimization steps needed to converge to the goal. This problem has 17280 particles, for 51840 degrees of freedom.

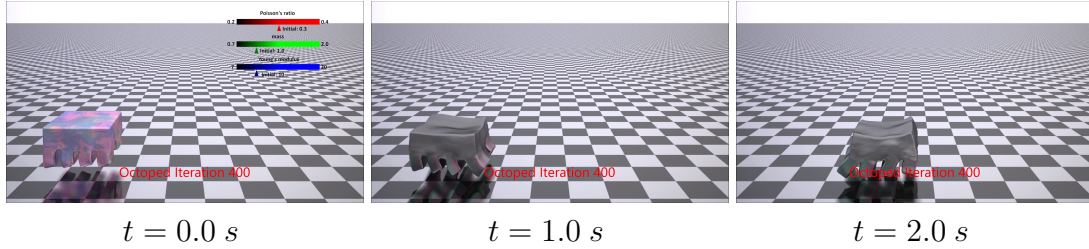


Figure 4-16: A montage of our simulated co-optimized Octoped in motion, with the first frame showing the optimized material distribution. Green colored actuators are expanding; red, contracting.

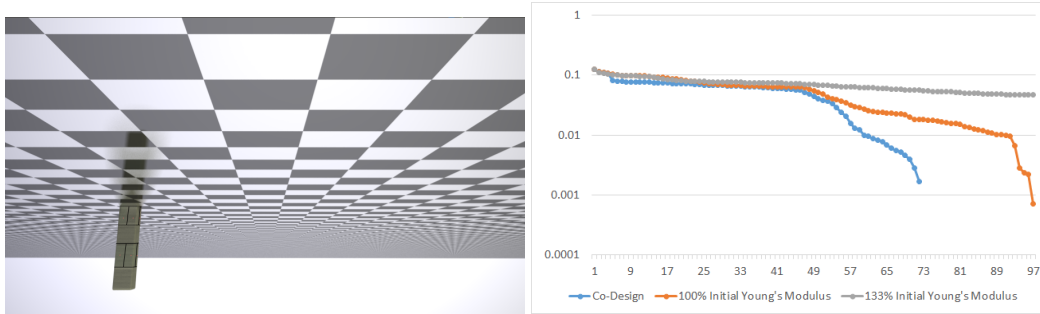


Figure 4-17: Convergence of the 3D arm reaching task for co-design vs. fixed arm designs. The fixed designs take significantly longer to optimize than with co-design; it's worth noting that the final total sum-squared actuation cost of the co-optimized 3D arm is less than 75% that of the full 3D arm. Constraint violation is the norm of two constraints: distance of end-effector to goal and mean squared velocity of the particles.

An Example Impossible Without Co-Design In a further open-loop control example, we present a geometrically parameterized arm, optimized *via* Adam. This soft arm has only a single parameter, its length, with an objective of minimizing distance of the end-effector to the goal. It must reach a target point that is too far away for it to reach in its default configuration, even with elastic stretching. In order to solve this task, the optimizer must automatically discover that the arm must lengthen to reach the goal; naturally, gradients can provide this information. This problem is successfully optimized with lower than 0.001 distance error within 100 steps; please see the Fig. 4-12 for a demonstration.

This example demonstrates our system's support for geometric parameters, and shows the power of ChainQueen in enabling co-optimization tasks which transform otherwise infeasible tasks into feasible ones.

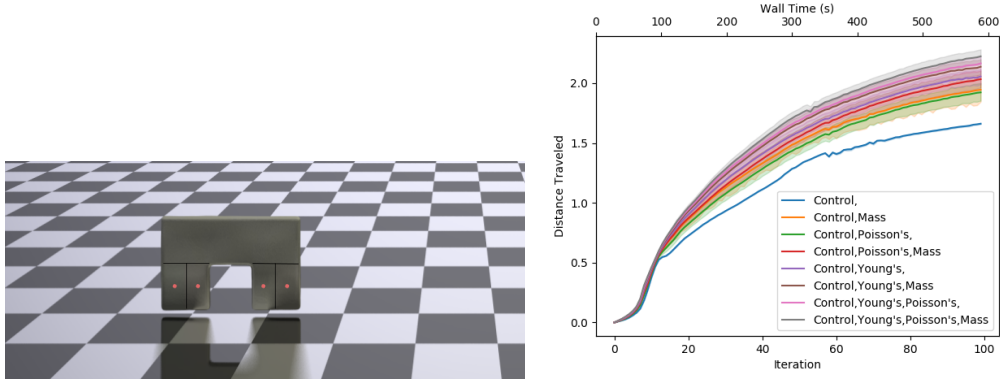


Figure 4-18: Progress of the 2D Biped vs. iteration and wall-time.

Closed-Loop Control

Here, we present four additional closed-loop control co-optimization tasks using an MLP (again, with two hidden layers of 64 neurons). These demonstrations — a 2D biped, a 3D quadruped, a 3D Hexapod, and a 3D Octoped — show ChainQueen’s ability to co-optimize over neural network controller parameters and spatially varying Young’s modulus, Poisson’s ratio, and density. In these examples, each particle has its own parameter for these three properties, meaning these problems have tens to hundreds of thousands of decision variables). Still, each of these robots can be optimized to (nearly) optimal gaits within 1 hour, and often much faster. The 2D Biped was run 10 times; the 3D experiments were run 6 times each.

Please see Figs. 4-13, 4-14, 4-15, 4-16 for full renderings of the material optimized robots. For each problem, the Young’s modulus was constrained between 7 and 20 (initial: 10); the Poisson’s ratio between 0.2 and 0.4 (initial 0.3); and the mass between 0.7 and 2.0 (initial 1.0). The maximum pressure per actuation chamber was set to 2.

2D Biped This example demonstrates a simple co-optimization example over the same biped demonstrated in the terrain example, with a goal of running as far to the right as possible in the allotted time. Every combination of the three material parameters (eight in total) were turned on and off, and run 10 times each. An interesting result, which will also follow in all 3D examples, is that each additional material parameter adds additional benefit to the co-optimization, allowing the robot to walk even farther. Furthermore, there is a clear ordering to the effect each of the material parameters has on the robot’s performance, with

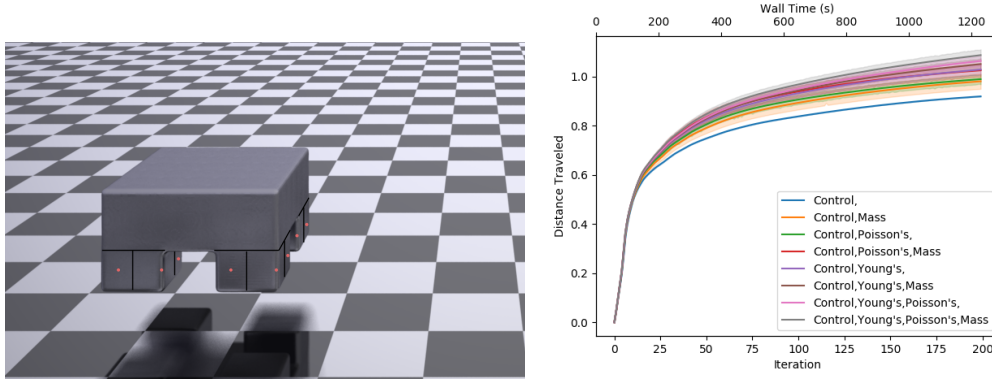


Figure 4-19: Progress of the 3D Quadruped vs. iteration and wall-time.

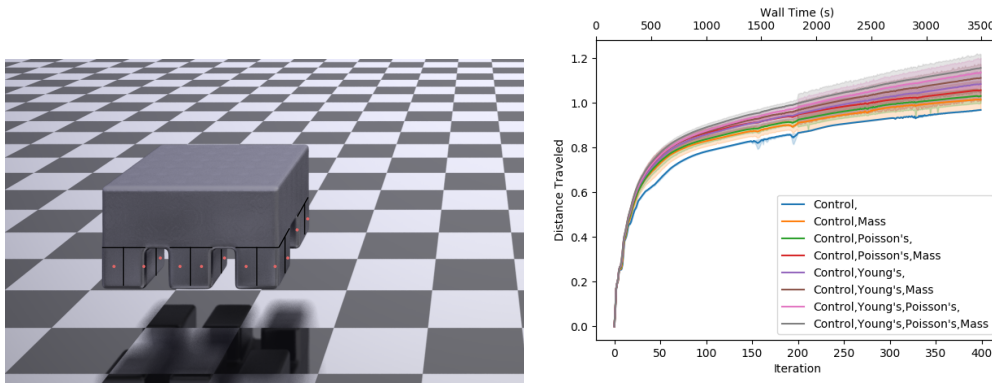


Figure 4-20: Progress of the 3D Hexapod vs. iteration and wall-time.

the Young's modulus having the most significant impact, followed by material density, and finally Poisson's ratio. This makes intuitive sense; the flexibility of the material will have the greatest impact in its ability to achieve optimal deformation; the density of the material then affects how easily the robots can overcome inertial forces and the effects of gravity; finally, Poisson's ratio, while not negligible in impact, have far less of an obvious intuitive effect on soft robot locomotion. Full results can be found in Fig. 4-18. We highlight that optimization is extremely fast, with each Biped walking several body lengths within 5 minutes. This problem has 12800 particles, for 25600 degrees of freedom.

Perhaps particularly stunning is the the large impact co-optimization can have on robot performance, providing a nearly 50% improvement in gait over non-co-optimized counterparts.

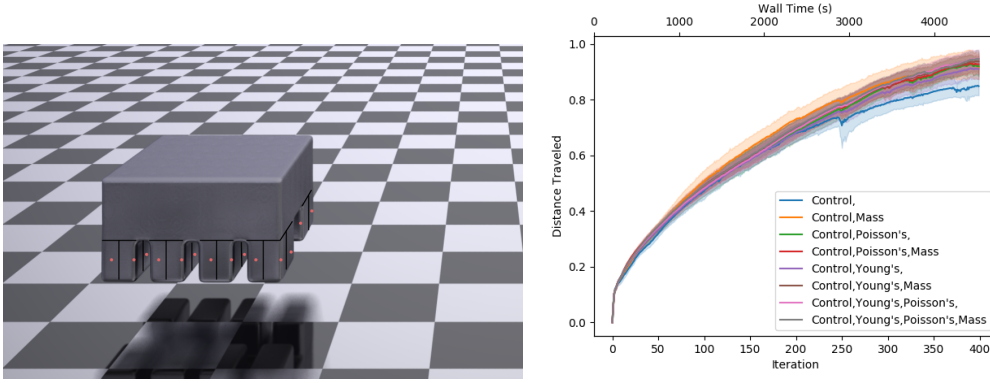


Figure 4-21: Progress of the 3D Octoped vs. iteration and wall-time.

3D Quadruped, Hexapod, and Octoped We further perform co-optimization on three complex 3D walkers, a Quadruped, a Hexapod, and an Octoped. Each of these robots has four pressure chambers in each of its legs arranged in a square pattern. As we increase the number of legs, the robots become more dynamically complex (having more actuators) and heavier, making these tasks more challenging. Full results can be found in Figs. 4-19, 4-20, 4-21 ; these material optimization results mirror those of the 2D Biped. Again, the impact of co-optimization is underscored by these graphs, as co-optimization provides a roughly 33% gain in performance for these tasks. The 3D Quadruped, Hexapod, and Octopeds have 84375, 111375, and 138375 particles respectively, for 253125, 334125, and 415125 degrees of freedom respectively.

4.10 Related Work

4.10.1 Differentiable Simulation For Soft Robots

The interplay between machine learning techniques - both classical and “deep” - and computational dynamics for computer graphics is long and storied. In general, this interplay has traditionally fallen into three different categories. The first category is machine learning techniques to aid dynamics. These techniques use machine learning in order to build more accurate, faster, or more expressive models of virtual characters and environments. The second category is machine learning that replaces dynamics. Learned models have the capability of being more compact in representation and less computationally expensive than

physics models derived from first principles. The techniques in this category aim to learn as-accurate-as-possible representations of physical phenomena from recorded real-world or virtual data, aiming to provide fast physical evaluation while sacrificing little in the way of accuracy: the best of both worlds. Finally, the third category is comprised of fast physics engines which are differentiable, allowing them to be plugged directly into a machine learning pipeline as part of the learnable model itself. From a deep learning perspective, a physics simulation (or part of a simulation) can be thought of as yet another layer type in a model.

Differentiable simulators are not new, but the advent of deep learning has put them in the spotlight. Recent focus has been on making them faster and extending them to new domains. Differentiable simulation has long been studied for classical Newtonian mechanics, specifically in the case of rigid robot control. Being able to differentiate a system with respect to system positions, velocities, accelerations, control inputs, and external forces such as contact is key to the problem of controllability [144] which is core to so much in robotics. Techniques such as LQR, trajectory optimization, and certain model-predictive control algorithms (*e.g.* [84, 117, 162]) directly rely on this information for motion planning and control. Thus, it’s unsurprising that various simulation engines have been established for differentiable robot simulation. [145, 42, 52, 34] provide complete, fully differentiable toolboxes for computing robot (kino)dynamics, and inverse (kino)dynamics, based off methods such as Newton-Euler iteration [73] and Featherstone’s algorithm [45]. These approaches rely on techniques such as symbolic or autotomatic differentiation and efficient factorizations for performance. Many approaches have also been taken to improve simulation speed, often focusing on efficient collision detection and fast, differentiable, physically-based contact [32, 161].

Outside of Newtonian mechanics, differentiable simulators have recently been proposed for other domains. One relevant domain is soft bodied systems, where differentiable models are useful for animation and soft robotics. Differentiable soft bodied methods such as material point method [66, 135] and finite element methods [40] are popular in computer animation, and both can handle contact and some multiphysics such as solid-fluid coupling. A second interesting domain is that of cloth physics [119], including realistic handling

of complex contact. Recently emerging are APIs and languages for quickly prototyping efficient, GPU-accelerated physical simulation, such as DiffTaichi [63], whose compiler is optimized for sparse spatial computing. Languages like this have an opportunity to play an analogous role to that of tools such as TensorFlow or PyTorch, but for physical simulation and sparse visual computing instead of traditional machine learning building blocks.

Continuum mechanics have been less studied in differentiable frameworks, not least of all because of the high computational complexity they incur. Schenck and Fox[125] presented differentiable simulation of position-based fluids, using neural networks to approximate the dynamics in a differentiable way. A learned, differentiable, hierarchical particle-based representation of continuum systems was presented in Mrowca et al[106]. Rather than learn smooth, neural-network-based models of physics from other simulators, ChainQueen directly differentiates MLS-MPM for much more accurate simulation results and gradients. More recently, Liang et al [86] presented a differentiable simulation framework for spring-based cloth simulation without the need for neural-network approximations.

4.10.2 Co-Optimization of Control and Materials

There exist two main threads of work in which robots are co-designed over morphology and control — gradient-free and gradient-based. Most of the work in model-free co-optimization of soft-robots is based on evolutionary algorithms. [24], [28], and [22] have demonstrated task-specific co-optimization of soft robots over materials, actuators, and topology. These approaches are less susceptible to local minima than gradient-based approaches but are vastly more sample inefficient. For instance, a single evolved robot in [24] requires 30000 forward simulations; by comparison, optimized robots in our work are optimized in the equivalent of 400 simulations (treating gradient calculation as equal to 3 forward simulations). Further, their approach was limited to simple open-loop controllers tied to form, while ours solves for more robust, closed-loop control.

While some algorithms exist for gradient-based co-optimization of rigid robots ([157, 134, 59, 159]), results in model-based co-optimization of soft robots have been sparse. Closest to our work, [65] presented a method for gradient-based co-optimization of soft

robotic arms using a fully-differentiable simulator based on the material point method (MPM). As a limitation, their method relied on ad-hoc features that needed to be labeled at the time the robot topology was specified and could not be easily measured in the physical world, making them ill-suited for physical control tasks. Work in this thesis addresses this shortcoming.

4.11 Outlook: Differentiable Simulation

In any field, many competing approaches means many open questions, and differentiable physics is no exception. Here, we outline a few key open questions in the intersection of physics and machine learning. First, for most (non-continuum) physical systems, contact must be modeled as a discrete event in order to model it accurately. However, such discrete contact models also have non-smooth, noisy derivatives, making it difficult to use the resulting gradients for downstream applications. By contrast, in the application of control, model-free approaches such as reinforcement learning have been very successful in systems with dense contact, and do not require gradients for control optimization; these approaches lack the type of efficiency that model-based approaches can have [120]. Naturally, one wonders if these two approaches can be combined in order to achieve the best of both worlds — efficient optimization when gradients are smooth, and robust optimization when they are not. The work in [115] demonstrates methods on learning more accurate contact models, for combination with model-based optimization methods. The work in [80] similarly demonstrates the interplay of learned models and models derived from first principles, for applications in manipulation. Finally, the analysis in [69] provides a warning: though learned models may be accurate in the forward simulation, developing learned models that are accurate in gradient computation remains an open problem.

4.12 Outlook: Soft Robot Co-Design

The impact of efficient differentiable simulation on soft robot co-design should be abundantly clear: fast simulation allows for *fast* gradient computation *via* backpropagation schemes; fast

gradient computation allows for *efficient* optimization, and efficient optimization directly enables gradient-based co-optimization methods. We are hopeful that such algorithms can be made to work on more physically realistic materials, made robust to perturbations in system dynamics, and extended to more expressive shape parameterizations – although solving this last aspect of the soft robot co-design problem involves dealing with problems such as reparticlization in MPM (and remeshing in FEM), which could affect the problem dimension.

At this point, we summarize the story of this thesis thus far. First, we have demonstrated that co-optimization can help to design new robots that solve problems that are infeasible with the default input designs. Second, we have demonstrated that c-optimization can lead to designs that perform vastly better than default or naïve solutions. And, finally, co-optimizing over more design domains can lead to better results and even scale to very large problem dimensions. In the next chapter (and indeed for much of the remainder of this thesis), we observe what happens when we introduce complex or combinatorial design domains which cannot be solved *via* direct optimization methods (layer 4 in the robot design stack), and examine learning-based methods as a solution (graduating to layer 5).

Chapter 5

Co-Learning Algorithms For Proprioceptive Modeling Of Soft Robots

In the previous two chapters we discussed co-optimization algorithms for co-designing robots over multiple domains, including geometry, control, actuation, and materiality. While demonstrably effective in allowing robots to solve tasks unsolvable in their default morphology, and while able to vastly improve the performance of such robots, these techniques are limited. Each design iteration exists as an island, relatively uninformed by other designs tested in the iterative optimization. In other words, while plenty of simulations are being performed, none of this information is being leveraged in order to *learn* anything about the structure of the problem. This data is being effectively wasted.

In this chapter, we demonstrate how intelligently using this data can extend the co-design pipeline to more complex problems, both in cognitive robotics and hardware design. In particular, we introduce what we dub *co-learning* algorithms which employ learning as part of the co-design problem. By learning the structure of problems, we are able to learn complex proprioceptive models of soft robots, build generative models of designs with *discrete* variables, and escape local minima to which direct optimization algorithms are so prone. Further, (as will be examined in Chapter 5), learning algorithms allow us to learn models that are robust against an entire target dataset, implicitly opening up opportunities in modeling and multitask design. Tying back to the robot design stack, we begin an exploration of layer 5.

5.1 Co-Learning Control, Materials, and Proprioceptive Models

Soft robots have continuum solid bodies that can deform in an infinite number of ways. Controlling soft robots with closed-loop control is very challenging as there are no compact, closed form solution for robot representation. In Chapter 4’s control examples, we manually chose clusters of to track in center of position and velocity; however, such features are difficult to track in the physical world and must be specified manually in an *ad hoc* manner.

In this section, we describe how we can leverage learning techniques to co-design soft robots over not only control and materials (as was discussed in Chapter 4), but also proprioceptive models. Our solution marries hybrid particle-grid-based simulation with deep, variational convolutional autoencoder architectures that can capture salient features of robot dynamics with high efficacy. We demonstrate our dynamics-aware feature learning algorithm on both 2D and 3D soft robots, and show that it is more robust and faster converging than the dynamics-oblivious baseline. We validate the behavior of our algorithm with visualizations of the learned representation.

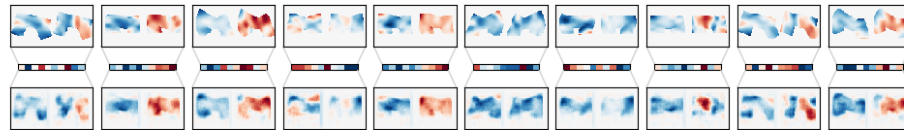


Figure 5-1: Our algorithm learns a latent representation of robot state which it uses as input for control. Above are velocity field snapshots of a soft 2D biped walker moving to the right (top), the corresponding latent representations (middle), and their reconstructions (bottom) from our algorithm. In each box, the x (left) and y (right) components of the velocity fields are shown; red indicates negative values, blue positive.

Recent breakthroughs have demonstrated capable computational methods for both controlling ([61], [126], [88]) and designing ([59], [134], [157]) rigid robots. However, control and design of *soft* robots have been explored comparatively little due to the incredible computational complexity they present. Due to their continuum solid bodies, soft robots’ state dimensionality is inherently infinite. High, but finite dimensional approximations such as finite elements can provide robust and accurate forward simulations; however, such representations have thousands or millions of degrees of freedom, making them ill-suited

for most control tasks. To date, few compact, closed-form models exist for describing soft robot state, and none apply to the general case. In this paper, we address the problem of learning low-dimensional robot state while simultaneously optimizing robot control and/or material parameters. In particular, we require a representation applicable to physical control of real-world robots.

We propose a computer-vision-inspired approach which makes use of the robot’s observed dynamics in learning a compact observation model for soft robots. Our task-centric method interleaves controller (and material) optimization with learning low-dimensional state representations. Our “learning-in-the-loop optimization” method is inspired by recent advances in hybrid particle-grid-based differentiable simulation techniques and deep, unsupervised learning techniques. In the *learning phase*, simulation grid data is fed into a deep, variational convolutional autoencoder to learn a compact latent state representation of the soft robot’s motion. In the *optimization phase*, the learned encoder function creates a compact state description to feed into a parametric controller; the resulting, fully differentiable representation allows for backpropagating through an entire simulation and directly optimizing a simulation loss with respect to controller and material parameters.

Because learning is interleaved with optimization, learned representations are catered to the task, robot design (including, *e.g.*, discrete actuator placement), and environment at hand, and not just the static geometry of the soft structure. Because of our judicious choice of a physics engine which operates (in part) on a grid, we are able to easily employ modern, deep learning architectures (convolutional neural networks) to extract robust low-dimensional state representations while providing a representation amenable to real-world control through optical flow. Because of our fully-differentiable representation of the controller, observations, and physics, we can directly co-design robot performance.

To our knowledge, our pipeline is the first end-to-end method for optimizing soft robots without the use of a pre-chosen, fixed representation, minimizing human overhead. In this section, we contribute: 1) An algorithm for control and co-design of soft robots without the need for manual feature engineering; 2) experiments on five model robots evaluating our system’s performance compared to baseline methods; 3) visualizations of the learned representations, validating the efficacy of our learning procedure.

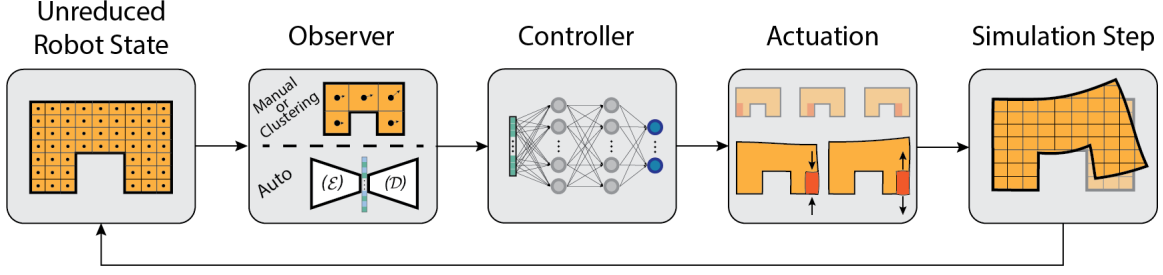


Figure 5-2: At each step of our simulation, the following procedure is performed. First, the unreduced state is fed into an observer function — centroids of a segmentation, as in [65], or, as we demonstrate in this section, an automatically learned latent space. Regardless, the observer outputs features to be processed by a parametric controller, which converts these features to actuation signals. Finally, the actuation is fed into our MPM simulator, which performs a simulation step. The entire pipeline is differentiable and therefore we can compute derivatives with respect to design variables even when executing the workflow for many steps.

5.2 Overview and Preliminaries

We seek an algorithm for co-optimizing soft robots over control and design parameters without manually prescribing a state description for the controller to observe. Our solution will be to periodically learn an updated observation model from the simulation data generated during optimization. For the remainder of this chapter, we refer to the dimensionally reduced representation of the soft robot as the *latent* representation and the unreduced representation as the *full* representation. To avoid confusion, we use the term *learning* to refer to the procedure of learning a latent representation and the term *optimization* to exclusively refer to the procedure of improving a robot’s controller or design.

A full overview of our system is shown in Fig. 5-2. At each time step, the full representation is fed into a (learned) *observer* function, which reduces it down to a latent representation. The latent representation is fed into an (optimized) controller function, which produces control signals for the robot’s actuators. Those control signals are applied to the full robot state to simulate the robot forward one time step, producing the next full state. At the end of the simulation, a specified *final loss* function L is computed. Direct optimization of this loss function over controller and physical design parameters is possible since each component of our system, including our simulator, is differentiable.

Formally, let $\mathbf{v}^t \in \mathbb{R}^u$ denote a robot’s full state at time t , let $\mathbf{q}^t \in \mathbb{R}^r$ denote the corresponding latent space, and let $\mathbf{u}^t \in \mathbb{R}^m$ denote the actuation control signal at time t . The observer function, $\mathcal{O} : \mathbb{R}^u \rightarrow \mathbb{R}^r$ maps a full state to a latent state and is governed by

observer parameters Θ . The controller function, $\mathcal{C} : \mathbb{R}^r \rightarrow \mathbb{R}^m$ maps a reduced state to deterministic actuation output and is governed by control parameters θ . The simulation step function, $f : \mathbb{R}^u \times \mathbb{R}^m \rightarrow \mathbb{R}^u$, time steps the system for some specified Δt given the full state and the actuation, and is governed by physical design parameters ϕ . In other words, $\mathbf{v}^{t+\Delta t} = f(\mathbf{v}^t, \mathcal{C}(\mathcal{O}(\mathbf{v}^t; \Theta); \theta); \phi)$. For brevity, we will omit writing the parameterizations explicitly except when necessary. The final loss $L : \mathbb{R}^u \rightarrow \mathbb{R}$ at final time T operates on some final full state $\mathbf{v}^T$ to produce a scalar loss; this could be the distance the robot has traveled, its final velocity, *etc.* - anything that's dependent on the robot's final state. Computing L amounts to iteratively applying f to generate states $\mathbf{v}^0, \mathbf{v}^{\Delta t}, \mathbf{v}^{2\Delta t} \dots \mathbf{v}^T$ and applying L to the final one. We use $\mathcal{S}$ to denote the full process of simulating a robot and then computing the loss value l ; in other words, $l = L(\mathbf{v}^T) = \mathcal{S}(L, \mathbf{v}^0)$ for some simulation time length T . The chain rule can then be used to backward-propagate through this chain of functions to compute gradients for optimization. In our algorithm, the learning phase learns Θ while the optimization step optimizes ϕ , and θ .

$$\begin{aligned}
& \underset{\theta, \phi}{\text{minimize}} && L(\mathbf{v}^T) \\
& \text{where} && \forall t, \mathbf{v}^{t+\Delta t} = f(\mathbf{v}^t, \mathcal{C}(\mathcal{O}(\mathbf{v}^t; \Theta); \theta); \phi) \\
& \text{subject to} && \phi_{\min} \leq \phi \leq \phi_{\max}
\end{aligned}$$

Though Θ is not part of the optimization, it is an auxiliary variable that must be learned in tandem.

5.3 Method

Simulation and Data Generation We use a simulator based on ChainQueen [65], the differentiable Moving Least Squares Material Point Method (MLS-MPM) [64] simulator, for the underlying physics of robots. In ChainQueen, robots are represented as collections of particles, and a background velocity grid is used for particle interaction and is exposed to users. ChainQueen also provides *analytical gradients* of functions of simulation trajectories with respect to controller parameters and robot design. Our algorithm is not simulator

specific, and can operate on any fully differentiable simulator where differentiable grid velocity data can be extracted.

In the remaining of this section we describe our LITL optimization algorithm. First, we assume we have a large dataset of robot motion data from the simulator, representative of the way the robot will move when completing the prescribed task, and describe the learning phase of the algorithm. Next, we describe how we use the simulation data to optimize the controller and design. Finally, we describe how to combine these two phases into a cohesive, complete algorithm.

Learning During the learning phase, we seek to learn a compact, expressive representation of robot state to feed to the controller. As input, learning takes in snapshots of robot simulation; namely, the robot’s velocity field on a background grid. Note that this field implicitly also provides robot positional information. As output, weights for an observer function with a descriptive latent space are learned.

In particular, we learn a variational autoencoder [77, 122] that takes, as input, a state description of a robot and minimizes the reconstruction cost of said state. Our assumption is that features which allow reconstruction are highly expressive. Fig. 5-3 presents the architecture we used for all experiments. We experimented with various network depths; ours was chosen for stability and generality across experiments. We use a convolutional architecture, which operates naturally on input velocity grid data and generalizes to robot translation due to equivariance.

Formally, for an unreduced u -dimensional object, let $\mathcal{E} : \mathbb{R}^u \rightarrow \mathbb{R}^r$ be an encoder function with parameter weights $\Theta_{\mathcal{E}}$, and $\mathcal{D} : \mathbb{R}^r \rightarrow \mathbb{R}^u$ be a decoder function with parameter weights $\Theta_{\mathcal{D}}$. For an input training dataset grid velocity data Υ , the reconstruction loss is defined as:

$$\mathcal{L}_R(\Upsilon) = \frac{1}{|\Upsilon|} \sum_{\mathbf{v} \in \Upsilon} \|\mathcal{D}_{\Theta_{\mathcal{D}}}(\mathcal{E}_{\Theta_{\mathcal{E}}}(\mathbf{v})) - \mathbf{v}\|_2^2.$$

We omit the details of the VAE formulation, which adds a variational representation and regularization term. For a more extensive treatment, please refer to [39]. We minimize this

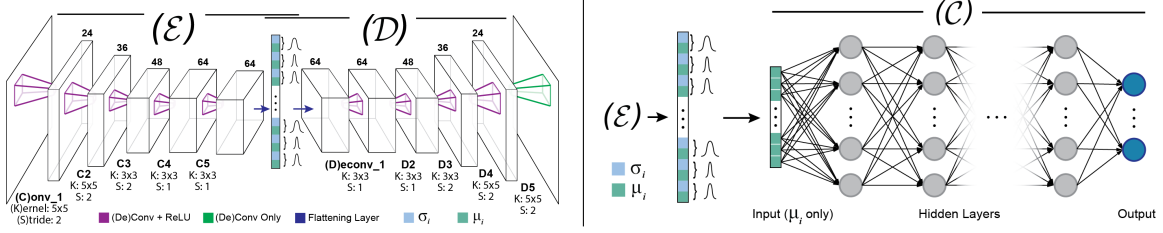


Figure 5-3: **Left:** The architecture of our convolutional variational autoencoder. The autoencoder takes in 2-channel pixel grid data as input, with each channel representing the x or y velocity field at that pixel. We use five layers of strided convolutions followed by ReLU operations. This is followed by a final flattening operation which coalesces the weights into latent variables. The latent variables parameterize Gaussians, used in our variational loss ($\mathcal{L}_v$). The architecture is mirrored on the opposite side (without a final ReLU, to allow for negative outputs). The 3D version is completely analogous, but takes in 3-channel voxel grid velocity field data and applies 3D convolutions. Above, the filter sizes are specified with K , and the strides are specified with S . **Right:** At inference time, simulation data is fed into the encoder, $\mathcal{E}$, which produces a latent $[\mu, \sigma]^T$ vector. The mean μ variables are then fed as inputs to the controller, $\mathcal{C}$.

loss by performing mini-batch stochastic gradient descent on our input dataset. For updates, we employ the Adam [76] first-order optimizer. We also experimented with a non-variational autoencoder. However, in the majority of experiments, that network overfit to a $1D$ manifold. This caused control optimization to fail, quickly resulting in unpredictable, erratic behaviors. The regularization from the variational formulation is necessary for avoiding collapse in the latent space.

Optimization Our optimization procedure is similar to that of [65]. At each optimization iteration, we compute $\nabla_{\theta, \phi} L$, providing a gradient of our loss with respect to all of our decision variables. We then use this gradient to apply a gradient descent update step to our parameters. Finally, we account for potential bounds in our design variables (e.g., maximum and minimum Young’s Modulus) by projecting the variable bounds back to the feasible region; i.e. $\phi_i \leftarrow \max(\min(\phi_i, \phi_{\max}), \phi_{\min})$. In practice, we use the Adam optimizer. At each iteration of the optimization, during forward simulation, we record snapshots of the grid data to be used in the learning phase.

Algorithm Our algorithm is an *alternating minimization*. First, we optimize the robot controller and design parameters for a fixed number of iterations, during which we record

snapshots of grid velocities. Then, we use these grid velocities to learn an observer for a fixed number of iterations. With the observer and latent representation improved, we return to our optimization procedure, and keep alternating until convergence. The initial grid velocity dataset is generated from simulating just once with the initial, untrained controller. This is enough to bootstrap our learning. Two key design decisions are discussed below:

Alternative vs. Simultaneous Minimization Learning a descriptive latent encoding is harder than optimizing the controller, and therefore requires more minimization iterations. When trained simultaneously, the controller tends to get trapped into a local minimum under a non-descriptive latent space. Performance-wise, evaluating $\nabla_{\Theta} L$ is orders of magnitude more expensive than evaluating $\nabla_{\Theta} \mathcal{L}_R$ since it requires backpropagating through an entire simulation. The alternating scheme allows us to economically draw a large amount of historical snapshots, and to evaluate $\nabla_{\Theta} \mathcal{L}_R$ only, for the autoencoder training.

Continuous vs. One-Shot Autoencoder Training Since robot dynamics change with changing control and design, continual retraining is critical. See, for example, Fig. 5-7 a. (robot arm control). With one-shot autoencoder training using only initial motion, the autoencoder only disambiguates motions of a mostly static arm, and optimization fails.

Our algorithm has no obvious guarantee of convergence; here, we describe three specific ways this algorithm can theoretically fail, and the steps we take to make our algorithm work reliably in practice.

Overfitting to Historical Snapshots It is important to fit to an entire trajectory, and not just the trajectory’s individually captured historical snapshots. Overfitting the autoencoder to history will degrade feature quality on future scenarios. Therefore, we employ *early stopping* to be conservative with autoencoder training. Before training the autoencoder, we evenly split the snapshots into a training and validation set. We early stop the training when the validation loss has remained worse than the best seen loss value, for a certain number of consecutive iterations.

Algorithm 2 Learning-In-The-Loop Co-Optimization

Hyperparameters: Max episode K , Max optimization iterations M , max learning iterations N , minibatch size b , max replay buffer size B , target update step size α , latent space dimensionality r .

Given: user-specified robot morphology $\mathcal{R}$, loss function L , design parameter bounds $\phi_{\min}, \phi_{\max}$, initial design parameters ϕ_0 , and initial full state $\mathbf{v}^0$.

Randomly initialize network weights θ , and Θ (with latent space of dimension r), and initialize autoencoder copy $\Theta' \leftarrow \Theta$.

Initialize empty replay buffer $\mathcal{I} \leftarrow []$ with maximum size B .

for episode $i = 1 \dots K$ **do**

for optimization iteration $j = 1 \dots M$ **do**

 Compute loss l_j and simulation snapshots $\Upsilon_j: l_j, \Upsilon_j = \mathcal{S}(L, \mathbf{v}^0)$.

 Store snapshots Υ_j in $\mathcal{I}$.

 Update θ, ϕ using the analytical simulation loss gradients $\nabla_{\theta, \phi} L$.

 Clamp physical design variables $\phi_i \leftarrow \max(\min(\phi_i, \phi_{\max}), \phi_{\min})$.

 Split $\mathcal{I}$ randomly and evenly into training set $\mathcal{I}_\tau$ and validation set $\mathcal{I}_v$.

for learning iteration $j = 1 \dots N$ **do**

for minibatch $1 \dots \text{len}(\mathcal{I}_\tau)/b$ **do**

 Sample minibatch I_τ from $\mathcal{I}_\tau$ (without replacement)

 Update Θ' using analytical autoencoder loss gradients $\nabla_{\Theta} \mathcal{L}(\Upsilon)|_{\Upsilon=I_\tau}$

 Compute validation loss $\ell_j = \mathcal{L}(\mathcal{I}_v)$

if ℓ_j has not decreased for q iterations **then**

 Break (Early Stopping).

 Update Autoencoder weights using target: $\Theta \leftarrow \Theta' \alpha + (1 - \alpha) \Theta$

Return: $\mathcal{R}$ with optimized design ϕ and controller θ .

Overfitting to Recent Trajectories The autoencoder tends to prioritize learning the most recent trajectories, harming generalization to future snapshots. To remedy this problem, we maintain an *experience replay buffer* [102] of snapshots from multiple simulations. We use all snapshots currently in the replay buffer to train the autoencoder. This increases the diversity of autoencoder training inputs, and stabilizes our algorithm against a changing controller.

Feature Oscillation Despite our precautions thus far, the autoencoder can still change rapidly, which injects instability to the controller optimization. Inspired by the smoothed target network update scheme in [88], we perform learning on a *copy* of the autoencoder network weights, and use the original autoencoder network throughout an episode. After each episode, we step the original autoencoder toward the updated copy Θ' ; *i.e.*, $\Theta \leftarrow \Theta' \alpha + (1 - \alpha) \Theta$.

We combine these refinements with our learning and optimization phases in our final algorithm (See Alg. 2 for details).

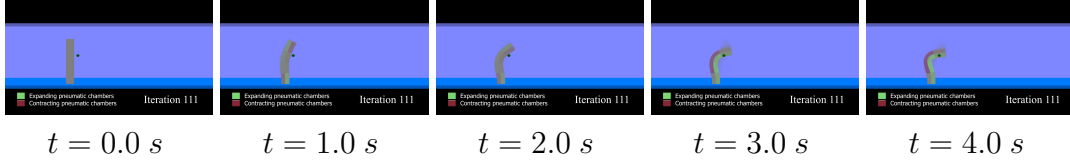


Figure 5-4: A montage of our simulated optimized arm in motion. Green colored actuators are expanding; red, contracting.

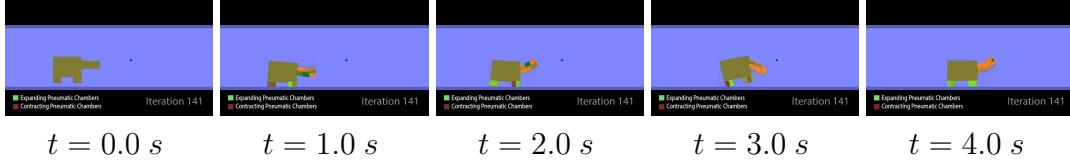


Figure 5-5: A montage of our simulated optimized Elephant in motion. Green colored actuators are expanding; red, contracting.

5.4 Results and Discussion

In this section, we summarize the results of our experiments on four of our model robots: 2D Biped, 2D Arm, 2D “Elephant,” 2D “Bunny,” and briefly describe demonstrations on four further robots: 2D Rhino, 3D Quadruped, 3D Curved Quadruped, and 3D Hexapod. Robot morphologies can be seen in Figs. 5-7 and 5-12. The Biped and its variants are co-design experiments, while the others are pure control, as we found co-design matters much more for locomotion tasks. Montages of optimized model robot tasks that were compared to baselines can be found in Figs 5-4, 5-9, 5-6, 5-5. For each 2D example, we compare to another automated procedure, a k -means clustering baseline, inspired as an automated version of the manual labels from [65]. In this baseline, the particles were clustered prior to optimization based on their Euclidean distance in the robot’s rest configuration; the average position and velocity of each cluster in each Cartesian coordinate is fed as input to the controller network. Each experiment was run ten times. All results are presented with a 90% confidence interval. We provide some ablation tests in each experiment to justify the necessity of certain aspects of our algorithm. For all 2D experiments, iteration vs. loss is presented; autoencoder training time is trivial compared to simulation, and both the VAE and k -means simulations/backpropagation times are similar. Each 2D simulation and corresponding backpropagation is computed in less than 20 seconds. All experiments were performed on a computer with an Intel i7 2.91-GHz processor, NVIDIA GeForce GTX 1080

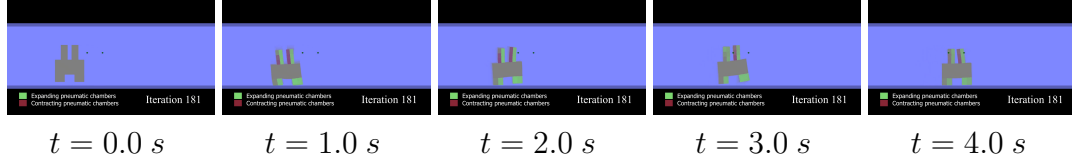


Figure 5-6: A montage of our simulated optimized Bunny in motion. Green colored actuators are expanding; red, contracting. This is a particularly challenging problem that cannot be solved with zero error.

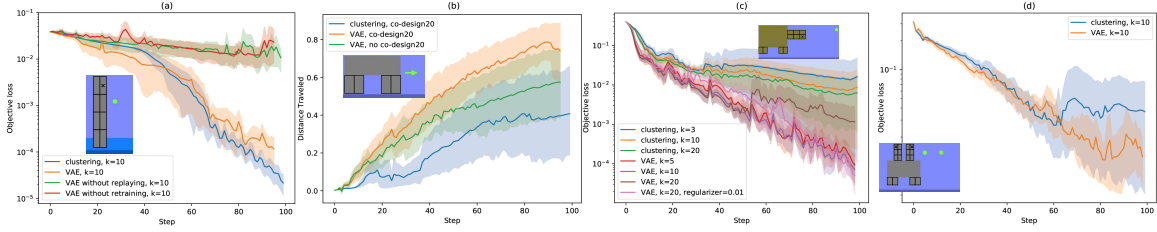


Figure 5-7: Progress of robot performance vs. optimization iteration, along with drawings of our 2D demos. Each black rectangle denotes an actuated region; in precision tasks, regions denoted with black X's are those which must reach target locations, denoted with green circles.

GPU, and 16GB of RAM.

2D Arm The 2D arm presents the simplest of all of our tasks in which the centroid of a region of a fixed-base soft robot arm must reach a prescribed point in space with no gravity. While geometrically simple, the problem is not dynamically trivial. The actuators are too weak to allow the robot to directly bend to the goal; it must swing back and forth to build up momentum to reach its target. This is the easiest problem we present; in this one example, k -means clustering is competitive, and at finer resolutions, faster converging than our VAE (though slower at coarser resolutions). See Fig. 5-7 a.

We use the 2D arm as an opportunity to present ablation tests for what happens if the replay buffer is eliminated. In the former case, the representation oscillates wildly, making control optimization impossible. In the latter case, since earlier iterations have less dynamic motion, they provide less dynamically descriptive, insufficient representations of the robot's full motion.

2D Biped The 2D Biped presents a locomotion task in which the robot must run to the right as far as possible in the allotted time. The biped's progress can be seen in Fig. 5-7 b. In

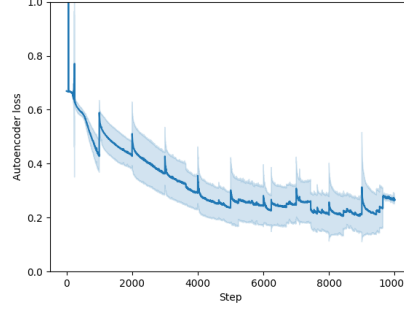


Figure 5-8: The average reconstruction loss $\mathcal{L}_R$ per pixel for the 2D Biped scaled as measured in average pixel distance vs. stochastic gradient descent step, demonstrating that not only does our algorithm converge in objective value, but also in model learning.

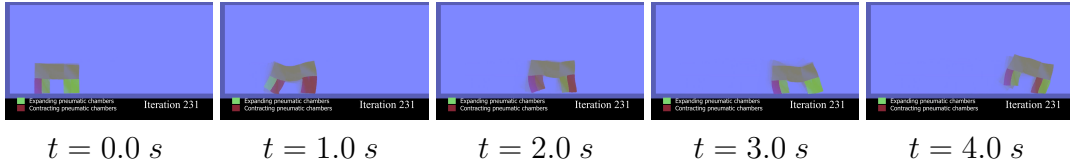


Figure 5-9: A montage of our simulated Co-Optimized Biped in motion, with bluer regions corresponding to stiffer materials. Green colored actuators are expanding; red, contracting.

the Figs. 5-10 and 5-11, we present two design variations in the robot’s shape. We also show the results of a typical VAE training procedure in Fig. 5-8, showing typical convergence. In four out of the ten trials, k –means clustering completely failed to converge, landing in poor local minima near the robot’s starting configuration.

We further use the Biped as an opportunity to show the minimal adverse effects of re-training. Table 1 presents the change in the distance traveled after retraining the autoencoder and then performing a single optimization step. The single optimization step cancels out virtually all backward progress caused by retraining.

2D Elephant The 2D Elephant presents a task which is a mixture of locomotion and manipulation. The elephant must walk to the right while a part of the trunk must reach a prescribed location. A subset of the results are seen in Fig. 5-7 c for readability, further results can be found in the Appendix. We use the Elephant to perform experiments over a wide range of latent variable and cluster sizes. While we try to compare the same number of clusters and latent variables in experiments (since inputs from a cluster give highly dependent data) we acknowledge that each cluster (in 2D) provides six inputs to the controller. Thus,

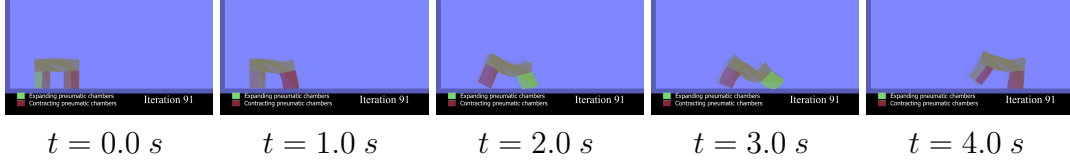


Figure 5-10: A montage of our simulated Co-Optimized Lanky Biped in motion, with bluer regions corresponding to stiffer materials. Green colored actuators are expanding; red, contracting.

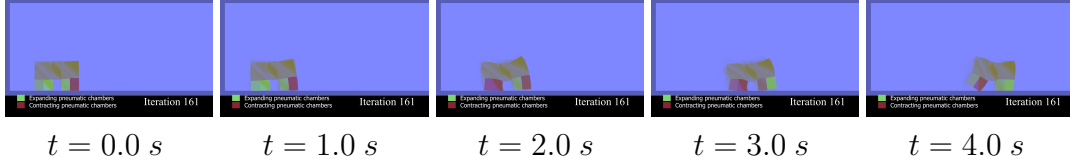


Figure 5-11: A montage of our simulated Co-Optimized Stubby Biped in motion, with bluer regions corresponding to stiffer materials. Green colored actuators are expanding; red, contracting.

this experiment also allows comparisons over controllers of similar size.

The VAE dominates k -means over all cluster/latent variable counts. As can be seen, the latent variable procedure has a weakness that the autoencoder can suffer the well-known “collapse” phenomenon as the latent variable size grows; increasing the VAE’s regularizer weight combats this phenomenon.

2D Bunny The 2D Bunny provides a task in which two arms must reach two target locations in space. The robot must walk forward and bend the arms to reach the target points as closely as possible. This is our most dynamically challenging task and cannot be solved perfectly. Results are in Fig. 5-7 d.

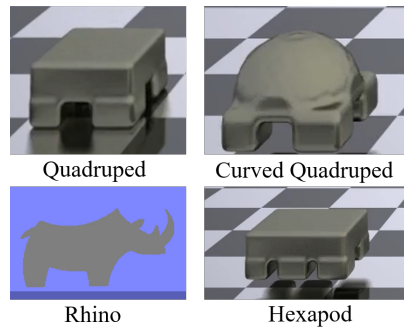


Figure 5-12: Four further robot demonstrations of we present montages.

Retrain #	Mean	Standard Dev.	Retrain #	Mean	Standard Dev.
1	-1.08×10^{-2}	2.64×10^{-2}	9	-8.53×10^{-3}	6.07×10^{-2}
2	-1.68×10^{-2}	1.53×10^{-2}	10	-1.26×10^{-2}	1.80×10^{-2}
3	-1.25×10^{-2}	3.06×10^{-2}	11	-2.70×10^{-5}	2.61×10^{-2}
4	3.56×10^{-2}	8.36×10^{-2}	12	-1.00×10^{-2}	1.36×10^{-2}
5	-1.86×10^{-2}	2.88×10^{-2}	13	-6.54×10^{-3}	4.29×10^{-3}
6	-1.17×10^{-2}	1.28×10^{-2}	14	5.93×10^{-3}	1.16×10^{-2}
7	-4.13×10^{-3}	1.55×10^{-2}	15	-3.27×10^{-4}	8.39×10^{-3}
8	-9.69×10^{-3}	1.44×10^{-2}			

Table 5.1: The mean backward progress remaining from retraining after a single optimization iteration on the 2D Biped locomotion task, with corresponding standard deviations. A negative value indicates a decrease in the distance traversed. As can be seen, the backward progress is a very small negative number, or positive in all cases, indicating that a single optimization almost completely reverses the adverse effects of retraining.

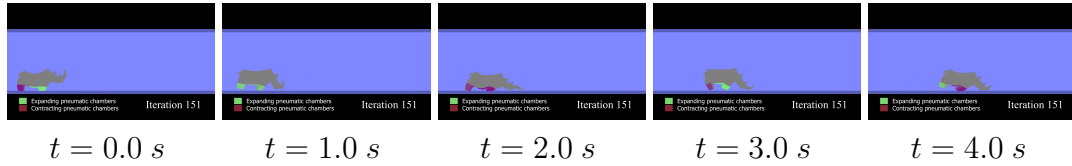


Figure 5-13: A montage of our control optimized Rhino in motion. Green colored actuators are expanding; red, contracting.

Further Demonstrations Finally, we present further control tasks, including extensions to 3D and curvier designs (Fig. 5-12). Like the 2D Biped, these robots must run as far to the right as possible in the allotted time. The 2D rhino is instantiated directly from a .png file. 3D optimizations take much longer, since the 3D autoencoder, and the corresponding simulation data, are much larger. They take more time to process, and the VAE capacity is larger, meaning it requires larger minibatches, a larger replay buffer, and more iterations during learning. Each 3D walker takes over a day to optimize, and thus was only performed once; please see Figs. 5-14, 5-13, 5-15, 5-16, for demonstrations of the four additional walkers.

Discussion As can be seen, our VAE observer tends to converge faster and get stuck in poor local minima much more rarely than k -means clustering. A natural question is to ask why k -means clustering performs worse. There are several reasons why clustering-based observers can lead to worse outcomes. First, k -means clustering can lead to poorly selected regions to track on the robot. For illustrative purposes, Fig. 5-17 shows two clusterings of the

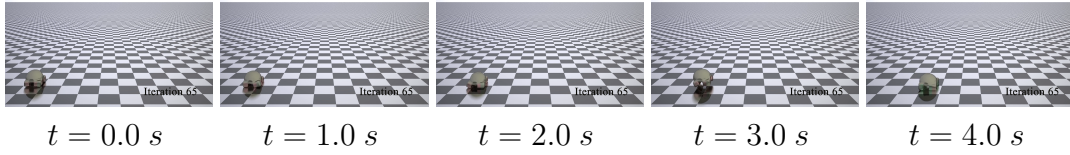


Figure 5-14: A montage of our control optimized Hemisphère Quadruped in motion. Green colored actuators are expanding; red, contracting.

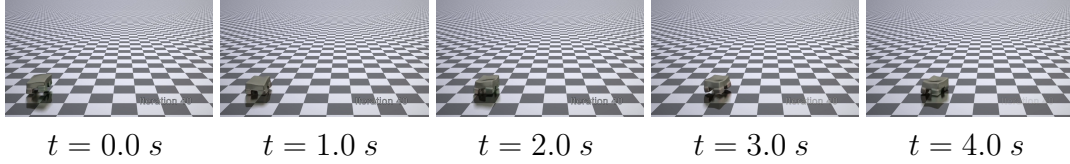


Figure 5-15: A montage of our control optimized Boxy Quadruped in motion. Green colored actuators are expanding; red, contracting.

bunny. In the first, because the Euclidean distance is used, clustering leads to two segments that are geometrically close but *dynamically* dissimilar to be clustered together; especially a problem when the task demands that they ideally should be tracked separately (we note that a geodesic distance might not suffer as seriously from such a behavior). Second, clustering does not gracefully handle changes in robot feature size. Even though the top arms are more dynamically interesting than the body of the robot, the body is allocated the majority of the clusters. This can be compensated for in a brute-force manner by adding more clusters, but experiments showed that as the number of clusters grows large, simulation time slows tremendously (if, say, $k = 1000$ is used, simulation on the same problems can take minutes). Finally, clustering is dynamics-oblivious; it cannot adapt as different motions are explored, nor does it consider other design or task specifics like actuator placement.

Fig. 5-20 provides a visualization of the extracted latent features for the 2D Biped and describes their computation. The emergence of natural “physical modes” arises as the procedure continues, with some more significant latent feature representing more rigid motion modes (such as velocity to the right), and less significant latent features representing higher-frequency, dynamic deformations. Such representations are not only valuable for robust control, but can make it easier to understand learned observers and controllers in relation to the underlying physical processes.

Finally, we note that while our VAE observer dominates on more challenging problems, k -means is still sufficient for simpler problems, as can be seen with the arm. One other

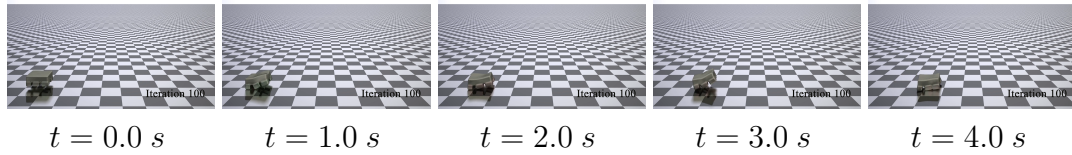


Figure 5-16: A montage of our control optimized Hexapod in motion. Green colored actuators are expanding; red, contracting.

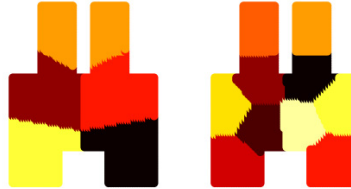


Figure 5-17: Two suboptimal clusterings for the bunny with different k values. For $k = 5$ (left) the upper arms are clustered together; for $k = 10$ (right) clusters overemphasize the importance of the body compared with the feet or arms.

example where the k -means observer performs better is in the case of the 2D biped when the problem is made sufficiently easier, by giving it much stronger actuators and dropping it from a high initial height to give it initial momentum. In this case, the walker can quickly learn to “bounce” forward; however, in the much more difficult case shown here, the k -means observer often completely fails.

5.4.1 Problem Hyperparameters

The following hyperparameters are the same for all problems. Note that we work in a fictitious unit system for convenience.

Hyperparameter	Value
Autoencoder learning rate	1×10^{-4}
Adam Parameter β_1	0.9
Adam Parameter β_2	0.999
Early Stopping Iterations q	10
Minibatch Size	$\frac{1}{40}$ dataset size
Simulation Δt	5×10^{-2} s (with 100 substeps)
Simulation Time T	4 s
Target Weight α	0.75
Poisson ratio	0.3
Density (kg / m ³)	1.0
Allowed Young’s Modulus variability	$\pm 10\%$

Table 1: Fixed Hyperparameters

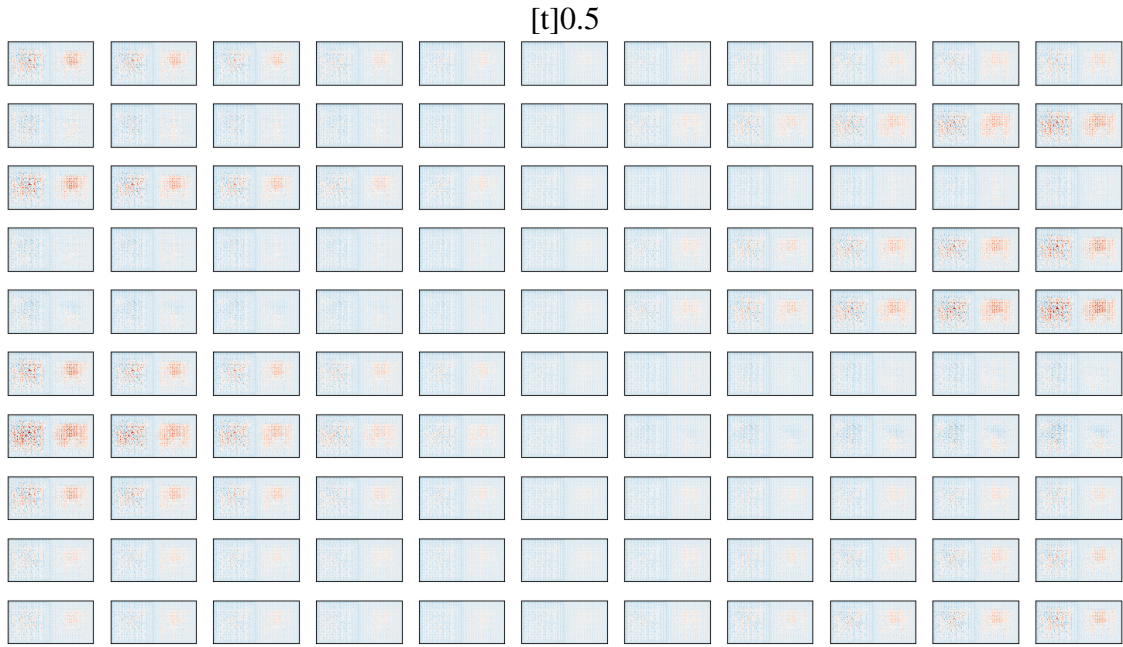


Figure 5-18: After Episode 1

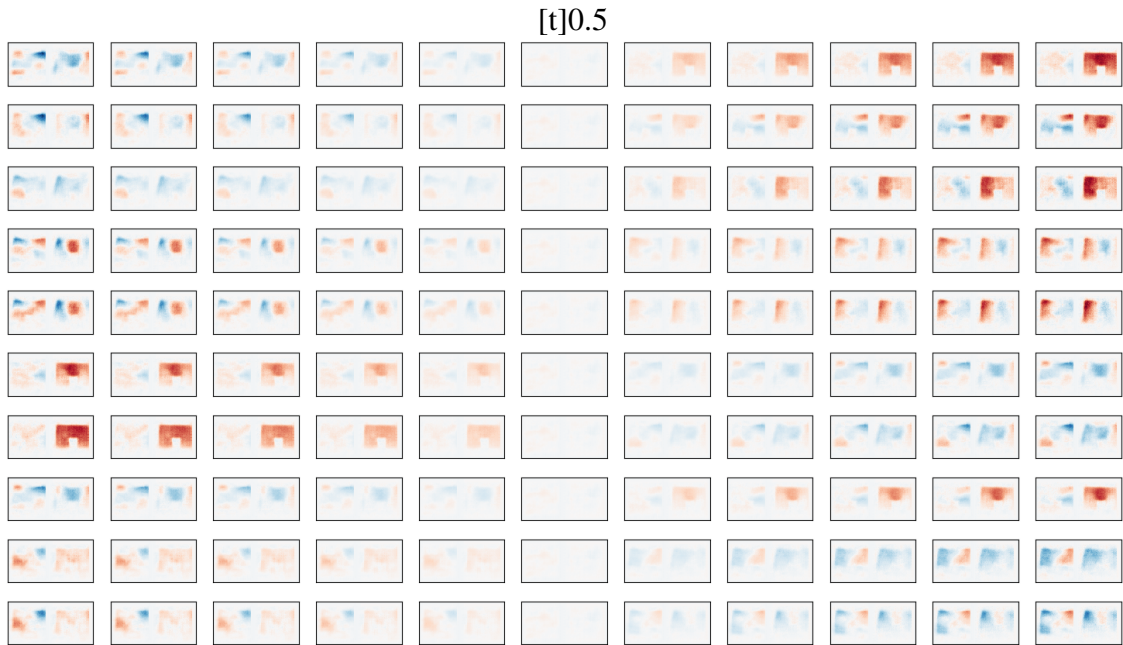


Figure 5-19: After Episode 10

Figure 5-20: Visualization of the latent space of the autoencoder (2D Biped) with 10 latent variables. Each row represents a (normalized) decoded output for a one-hot latent feature, varied from -1 to 1 . As the algorithm proceeds, the latent features become more descriptive. Formatting per box is the same as in Fig. 5-1.

The following hyperparameters were used for each problem. The following rules were used to choose these hyperparameters. 1) Controller learning rate is most dependent on the

dynamics of the system, and can be the same for any representation, but the more sensitive the system is to actuation the lower the learning rate should be. 2) The slower the objective changes, the larger M should be. 3) The larger the physical footprint of the robot and larger the dynamic range of its velocity profile, the more training iterations are necessary for convergence. 4) The faster the trajectory of the system changes between episodes, the larger the replay buffer should be.

Young's Modulus is purely a parameter of the problem and was chosen to make the problem most challenging and interesting.

Hyperparameter	2D Arm	2D Biped	2D Elephant	2D Bunny
Controller learning rate	5×10^{-4}	5×10^{-3}	5×10^{-3}	5×10^{-3}
Max Optimization Iter. M	6	6	10	10
Max Learning Iter.	10	15	20	30
Replay Buffer Size	4800	4800	2400	2400
Young's Modulus (kg / (m s ²))	30	10	30	50

Table 2: Variable Hyperparameters

5.4.2 Complete 2D Elephant Experimental Results

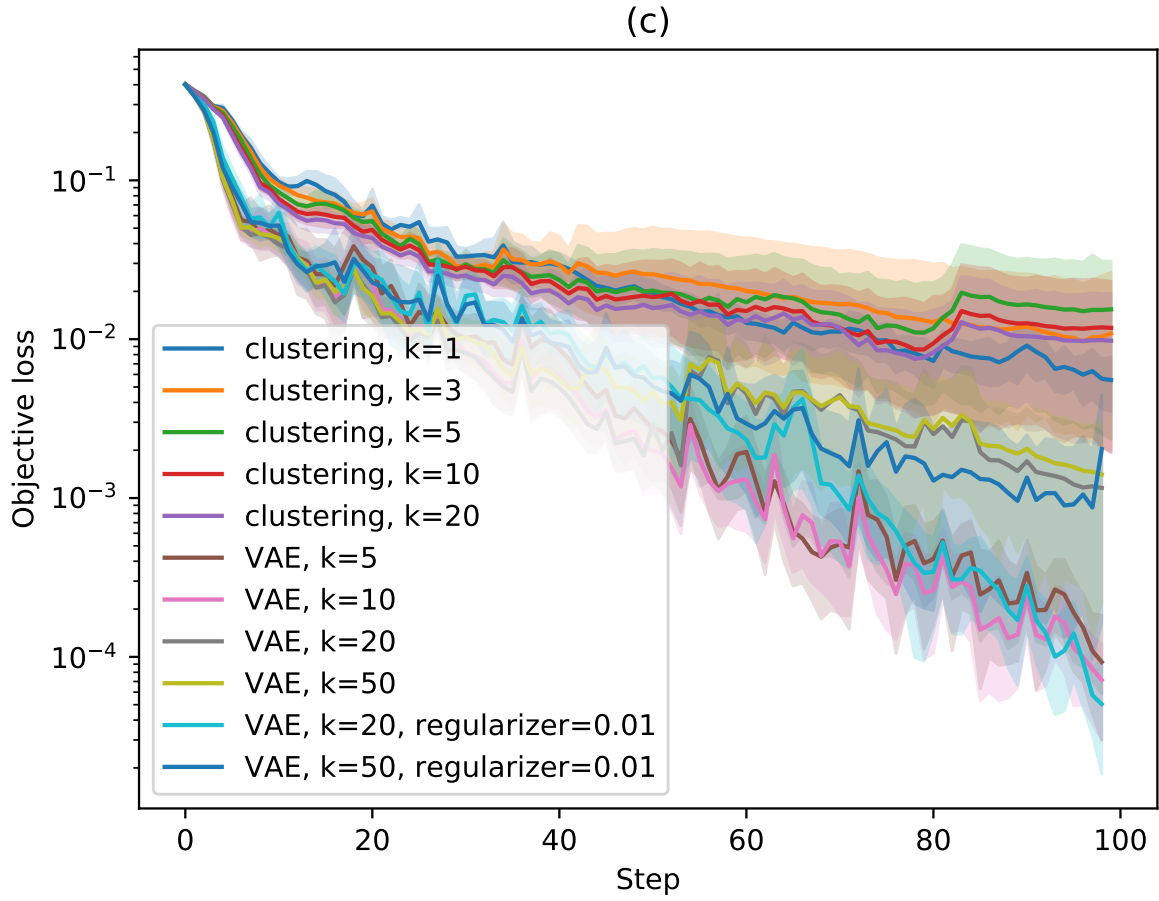


Figure 5-21: Progress of robot performance vs. optimization iteration for the 2D Elephant example.

5.5 Related Work

A compact, descriptive latent space is crucial for tractably modeling and controlling soft robots. Methods for extracting and employing such spaces for control typically fall into two categories: *a)* analytical methods, and *b)* learning-based methods.

Analytical methods examine the underlying physics and geometry of soft structures in order to extract an optimal subspace for capturing low-energy (likely) deformations. Most popular among these methods are modal bases [128], formed by solving a generalized eigenvalue problem based on the harmonic dynamics of a system. These methods suffer from inadequately modeling actuation, contact, and tasks and only represent a linear approximation of system’s dynamics. Still, such representations have been successfully applied to real-time linear control (LQR) in [13] and [146], and (with some human labeling) animation [12], but lack the physical accuracy needed for physical fabrication. In another line of work, [20] presented a method for using analytical modal bases in order to reduce the degrees of freedom of a finite element system for faster simulation while maintaining physical accuracy. However, the resulting number of degrees of freedom are still impractical for most modern control algorithms. For the specific case of soft robot arms, geometrically-inspired reduced coordinates may be employed. [35] developed a model for accurately and compactly describing the state of soft robot arms by exploiting segment-wise constant curvature of arms.

5.6 Outlook: Co-Learning of Proprioceptive Models

In this chapter, we demonstrated a method for end-to-end co-optimization of soft robots that requires minimal human intervention. Our method interleaves optimization with learning a deep latent space representation, allowing improved state estimates to improve the control and design, and vice versa. We have demonstrated our algorithm’s superior reliability to naïve dynamics-oblivious methods.

Our method has two notable drawbacks. First, although the 2D version of our algorithm can be applied on a visual cross-section in 3D, the fully 3D version can be hard to realize

in the physical world. Further, autoencoder training times can be quite large in the 3D convolutional architecture. Second, retraining of the autoencoder, while necessary, can sometimes undo some forward progress and interfere with momentum in optimization, both slowing the tail-end of convergence in optimization.

Finally, we envision a few future extensions to our work. First, since we learn a low-dimensional latent space, it would be interesting to use the learned latent-space outside of the context of its counterpart controller - namely for, *e.g.* optimal control such as LQR control. Second, we would like to apply our control algorithm to other soft robot simulation methods. For example, the nodes of an FEM simulation could be similarly rasterized to a background grid (though with additional overhead; MPM generates this grid “for free”) to which our algorithm could be directly applied. Third, we hope to demonstrate our optimized controllers on real, physical soft robots using vision-based sensors and optical flow.

Finally, although the Learning-In-The-Loop algorithm presented here shows promise for extrinsic observer models (in other words, observers built from observation sources external to the robot plant, such as cameras or motion capture systems) and may be well-suited for fixed-base robot plants such as robot arms, such systems are most likely impractical for floating-base robot plants intended to roam untethered around a large environment. Use of an extrinsic observer in such a scenario could require instrumenting intractably large areas. In the next chapter, we will discuss learning algorithms which focus on building observer models from *intrinsic* (cf *extrinsic*) data, such as strain sensors, while co-designing the hardware for these sensorizations.

Chapter 6

Design and Fabrication For Learning-Based Soft Sensing

A limitation of the method presented in the learning-in-the-loop demonstration is the choice of measuring the latent space from *extrinsic* coordinates, that is derived from an external sensor. Having an external camera model as input to the control algorithm limits the deployability of the optimized robots when locomotion is at play, since it would (impractically) require a camera to follow the soft robot. Preferable to extrinsic coordinates is the use of *intrinsic* coordinates. Such coordinates are derived from intrinsic sensing, that is, sensing that occurs within the robot's (possibly moving) frame of reference, or material space. In particular, we care about strain (specifically stretch) sensing, since such sensors are easy to fabricate and common in soft robotics. The choice of measuring strain is also more natural from a biomimetic perspective, as strain is one of many measurements organisms sense and use from neurons embedded in their bodies.

The choice to work with intrinsic sensing emits a related problem — where do we put sensors on the soft robot? We obviously cannot sensorize the entire robot, as that is physically impossible. To address this need for intrinsic soft robot sensing, in this chapter we focus on the problem of co-learning optimal sensor placements and models for general supervised tasks. Different locations on a soft robot experience different dynamic responses during motion; thus, sensor distribution can have a profound impact on a robot's ability. Better understanding the impacts of sensor placement can create the rich representation of

soft bodies needed for complex modeling and control tasks. Further, in this chapter, we demonstrate a novel soft sensing method computationally powered by deep learning, as a promising future deployment method for our co-learning algorithm.

6.1 Co-Learning Of Task And Sensor Placement For Soft Robotics

We propose a neural architecture for simultaneously learning soft robotic tasks and the optimal sensor placement for that task. Our model relies purely on intrinsic measurements — specifically, strains and strain rates — and is amenable to physical realization through off-the-shelf sensors. Since many soft robot representations are nodal in nature, we propose a novel architecture which adopts existing work in point-cloud-based learning and probabilistic sparsification. Our method treats sensor design as the dual of learning, combining physical and digital design in a single end-to-end training process.

Although computational sensing is traditionally thought of as an algorithmic problem, the choice of sensor placements has vital ramifications on hardware as well. For many sensor fabrication methods, such as directly embedding strain sensors into a solid elastic substrate, physical sensor placement is immutable and often laborious. Even in cases where sensor design allows for reconfiguration, sensorizations robust to a wide variety of environments or state distributions are still needed in order to maximize deployment time without expensive human-in-the loop hardware modifications.

To our knowledge, ours is the first method for *general* task learning for *dynamic* soft robots while optimizing sensor placement, focusing on simulated soft robots. We contribute the following: 1) A neural architecture for reasoning about soft robot state through measured strains and strain rates. 2) A probabilistic sensor representation suited to sparsification to a minimal set for downstream tasks, and a co-design algorithm for achieving this which outclasses both automated and human baselines. 3) Demonstrations of task learning and sensor placement co-design in two applications — tactile sensing and proprioception — across seven soft robot morphologies. The representations learned from these tasks can be

applied to unlock a novel task — closed-loop computational control of soft walking robots from solely intrinsic sensor measurements, and no external sensors.

Although our architecture is general to supervised learning tasks with nodal inputs, we demonstrate its utility to important soft robotic problems, such as in-hand sensing and full state reconstruction. Despite the fact that these problems have not been successfully handled with manually specified sensor locations, we are able to demonstrate success on a wide range of topologies of fully dynamic soft robots operating on intrinsic sensor data.

6.1.1 Preliminaries

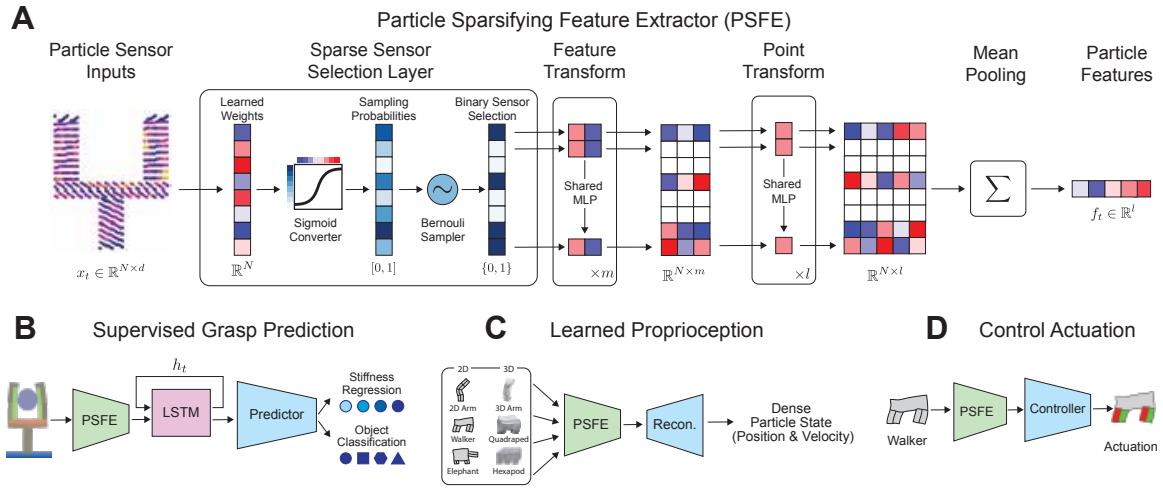


Figure 6-1: **A)** The foundation of our models is a Particle Sparsifying Feature Extractor (PSFE) which takes as input the full, dense sensory information (left) and extracts a global feature representation (right) from a sparse subset of the inputs. Our model simultaneously learns this representation and sparsification of the input. Since the input is an unordered point cloud, the PSFE also maintains order invariance through shared feature and point transformations as well as global pooling operations. We employ the PSFE on various complex tasks: **B)** Supervised regression and classification of object characteristics from grasp data. **C)** Learned proprioception by combining PSFE with a variational decoder network. **D)** Learned control policies for a soft robot.

Given a budget for the maximum number of sensors we wish to place on a robot, τ , we seek to learn both the optimal locations on a soft robot body to place those sensors, along with a model that uses those sensor inputs to solve complex tasks. Such optimal sensor locations depend both on the type of motion the robot will experience as well as which regions of the robot have the highest task relevancy. The combination of complex geometries

and dynamics of soft robots make this a highly difficult task, not easily solved by naïve strategies such as simply sensorizing the most dynamic portions of the robot (which may be densely clustered and are possibly irrelevant to the target task). Furthermore, while this chapter deals with soft robots in simulation, we wish for our sensor designs to be physically deployable, meaning only the intrinsic strain and strain rate information operating in a relative reference frame (*i.e.*, material space) can be used.

In this section, we consider dynamic soft robots simulated using the material point method (MPM). Recall that MPM approximates continuum robots as collections of particles. For a d -dimensional robot ($d = 2$ or 3), each particle i contributes a small bit of volume to the overall soft robot, and stores relevant information about its dynamic state; these include its position $\mathbf{p}_i \in \mathbb{R}^d$, its velocity $\mathbf{v}_i \in \mathbb{R}^d$, its deformation gradient $\mathbf{F}_i \in \mathbb{R}^{d \times d}$ (a measure of kinematic stretch and shear), and its affine velocity $\mathbf{C}_i \in \mathbb{R}^{d \times d}$ (a measure of kinematic stretch and shear rate). We employ the open source, physically-based ChainQueen simulator [66], whose differentiability can be used to solve motion-planning tasks, useful for generating task-driven trajectory datasets. We emphasize that although we choose MPM as our soft robot simulation method and its particle representation as input to our models, our learning method is not specific to MPM. Our method is amenable to any simulation method which has a node-based representation, including finite element methods. For the curious reader, however, [71] and [67] present in-depth tutorials on MPM.

We make the following assumptions about our strain sensor model: 1) Strain (rate) measurements have perfect accuracy and no latency. 2) Strain sensors can be placed on each particle and measure the strain (rate) over the volume of that particle. 3) Each sensor provides both strain and strain rate information along each (material space) axis. We relax the first assumption in experiments in Sec. 6.1.3. The Green Strain $\mathbf{S}_i$ of a particle i is defined as: $\mathbf{S}_i = \frac{1}{2}(\mathbf{F}_i^T \mathbf{F}_i - \mathbf{I})$. The Green strain rate $\frac{d\mathbf{S}_i}{dt}$ of a particle i can be efficiently computed analytically as: $\frac{d\mathbf{S}_i}{dt} = \frac{1}{2}\mathbf{F}_i^T(\mathbf{C}_i^T + \mathbf{C}_i)\mathbf{F}_i$. We provide a derivation later in this section. As most real-world strain sensors can only measure stretch, but not shear, we work only with the diagonal entries of $\mathbf{S}$ and $\frac{d\mathbf{S}}{dt}$ in this section. Most real-world soft robot sensing systems rely purely on stretch sensors (*e.g.* [148]), which are cheaper and easier to fabricate than shear sensors. Despite this limitation, efficient task learning is still possible,

emphasizing that there is a *likely* subspace of robot motions that can be learned from sparse sensory input, a subspace which can be leveraged through data-driven methods like ours.

The remainder of this section is organized as follows. First, we describe our point-based neural network architecture, which we use for all tasks considered. Next, we describe our sparse neuron model, which populates the input layer of our network and adaptively down-selects salient sensor locations. We then explain how we combine sensor and task learning in a simple learning algorithm. Finally, we demonstrate our co-learning approach on two novel soft robotics problems — dynamic in-hand sensing and dynamic state reconstruction — and show applications to further tasks such as control.

6.1.2 Method

In this section, we formulate a neural architecture suited to sparse sensorization for soft robotics tasks. Our network is modular and takes intrinsic sensor readings of the body as input. A point sparsification and feature extraction (PSFE) network simultaneously learns a representation of these sensor readings along with a set of sparse sensor locations. Our PSFE (Fig. 6-1A) is used as the core building block for all demonstrations and applications presented in this section, including object grasping prediction (B), learned proprioception (C), and control (D).

Point Sparsification and Feature Extraction Core to our algorithm is a neural network (NN) architecture for learning representations of soft robots from their raw, dense, and intrinsic sensor readings. By using a NN architecture, our method can employ drop-out techniques targeted to NN *input* layers as a means of choosing sparse sensor location.

To achieve accurate state representation during simulation, soft robots must be discretized into many individual particles. Candidate sensor locations are co-located with our MPM particles, creating a particle set which can be viewed as a point cloud. Thus, our architecture builds upon prior models for point cloud scene understanding, but makes important adaptations for the physical soft robotics setting.

Our representation builds upon PointNet [118], treating our robot particle set as a set of N order-invariant points in space. This invariance is created using a series of shared

feature transformations followed by point transformations into point features. The final representation is computed by pooling the features, again using order-invariant (mean) pooling. PSFE contributes two modifications in order to make the PointNet architecture amenable to soft robotics and sensor placement.

First, the original PointNet architecture reasoned purely on measured (x, y, z) Cartesian coordinates in space as input. Our sensors, however, are only able to measure axial strain and strain rates. While there is nothing preventing using these strain “coordinates” in lieu of Cartesian coordinates, such a representation would fail, due to PointNet’s inherent symmetric interpretation of point distributions. Each input to PointNet is processed as an unordered set, not an ordered list. Thus, a distribution of strain and strain rates would be interpreted the same way regardless of where they occurred on a soft robot; wholly different system dynamics could be interpreted as equivalent states. To avoid this problem, we carry along the local coordinates of the *undeformed* soft robot. Since the undeformed coordinates are the same regardless of robot state, this provides no feature information, but rather disambiguates particles in the point set while also providing spatial coherence which the architecture exploits. The result is that each point is represented by a $3 \times d$ vector: strains, strain rates, and static, undeformed position (material space) coordinates.

Second, the input layer to the PointNet architecture is deterministic. We modify our architecture to allow for stochastic weights in the input layer during training. During each training step, neurons corresponding to input i are sampled on or off according to Bernoulli random variable parameterized by some θ_i . Our training procedure will push θ_i to 0 or 1, thus learning a deterministic model of important inputs, and equivalently, optimal sensor placements.

Sparse Sensor Selection Layer Our representation uses stochastic neurons to sample different sensor candidates for extracting features as part of our Sparse Sensor Selection Layer (SSSL). By combining this representation with a sparsifying loss, we devise a training objective to learn an optimal sensor placement. Unlike classical NN weights, which are typically unbounded real values, our SSSL learns a set of binary $\{0,1\}$ weights as a sparse mask over our dense input. This is done by first learning a continuous weight vector,

transforming to probabilities $\in [0, 1]$, and using these probabilities to Bernoulli sample the binary mask. More concretely, given a set of continuous weights, W_i , we compute the probabilities, θ_b , and sample binary weights, b_i , as follows:

$$\theta_{b_i} = \sigma(W_i) \quad (6.1)$$

$$b_i = \text{Bernoulli}(\theta_i) = \text{ceil}(\theta_i - U) \quad (6.2)$$

where $\sigma(\cdot)$ is the sigmoid function, and $U \sim \text{Unif}(0, 1)$ is a uniform random number between 0 and 1. While the forward pass through these functions is well-defined, the backward pass is not, since the gradient of $\text{ceil}(\cdot)$ is zero for almost all inputs. To enable backpropagation we override the gradient:

$$\frac{\partial b}{\partial \theta_i} = \theta_i - U. \quad (6.3)$$

This modification, similar to BinaryConnect [29], allows the SSSL to forward propagate continuous, learned weights through a binary mask, transforming dense sensor candidates into sparse sensor locations. Fig. 6-2 illustrates the transformation of these random variables into discrete sensor locations.

Co-Learning Algorithm While a stochastic representation of neurons is useful for testing their saliency (evidenced by gradients with respect to θ), this sampling does not sparsify the input layer in its own right. In order to sparsify over the input layer and sensor placement, we introduce a regularizer that can be added to the task loss $L_t(\mathbf{w})$ of any learning problem, where $\mathbf{w}$ are the non-sparsifying weights of the learning model. Given a target sensor budget τ and a max sensor count/input layer size N , we have a sparsifying regularizer $L_s(\theta)$ and a total loss $L(\mathbf{w}, \theta)$ to be minimized:

$$L_s(\theta) = \left| \frac{\sum_i \theta_i - \tau}{N} \right|, \quad L(\mathbf{w}, \theta) = L_t(\mathbf{w}) + w_s L_s(\theta)$$

where w_s is a user-defined weight that trades off sparsification and the target learning task. The absolute value efficiently pushes most of the θ_i values to 0. Despite the sparsifying

Algorithm 3 Training algorithm

Given: Dataset $\mathcal{I}$, task loss function L_t , learning architecture with N inputs.

Randomly initialize network weights.

while $\sum_i \theta_i \geq \tau$ or $\sum_i \text{round}(\theta_i) > \tau$ **do**

for Minibatch $I \in \mathcal{I}$ **do**

$$L = L_t + w_s L_s$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L \quad (\text{Sec. 6.1.2})$$

$\theta \leftarrow \text{constant}(\text{round}(\theta))$

while Not converged **do**

for Minibatch $I \in \mathcal{I}$ **do**

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L \quad (\text{Sec. 6.1.2})$$

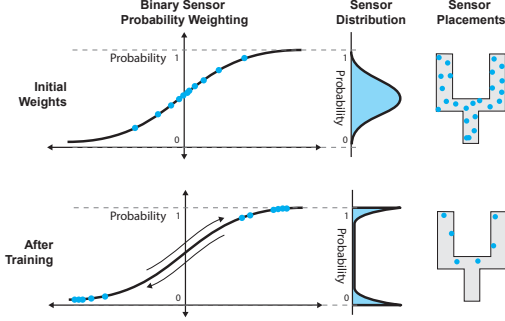


Figure 6-2: Learning sparsification. A 1D sigmoid function independently parameterizes the probability of each sensor location with weight W_i . Sampled gradients evolve W_i toward $\pm\infty$, pushing the corresponding θ_i probabilities toward 0 or 1, sparsifying the candidate sensor set.

nature of L_s , it is possible for a few weights to have partial probability at convergence, achieving a “mixed strategy Nash” against a target dataset. To alleviate this concern, once $\sum_i \theta_i < \tau$ and $\sum_i \text{round}(\theta_i) \leq \tau$, we appropriately round all θ_i to 1 or 0, fix them, and continue training (Alg. 1).

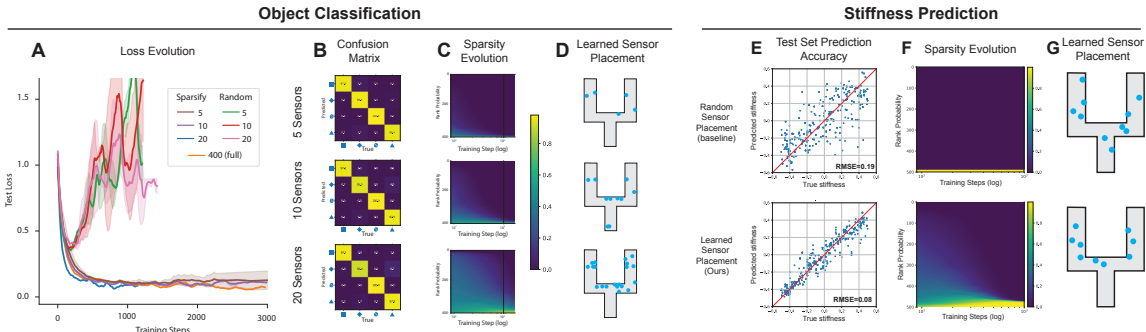


Figure 6-3: Results for object classification (A-D) and stiffness regression (E-G) tasks. (A) Convergence of the loss for $\tau = 5, 10, 20$ on the sparsifying algorithm, with comparison to baselines and full sensorization. (B) Confusion matrices for our co-learning algorithm across different sensor budgets. (C) Evolution of sensor probabilities over training time, validating that each θ_i converges toward 0 or 1, with final sensor placements shown in (D). (E) Prediction accuracy for the stiffness regression task, comparing the baseline (top) to our algorithm (bottom). (F) Evolution of sensor probabilities. (G) Optimized sensors have more task-relevant locations.

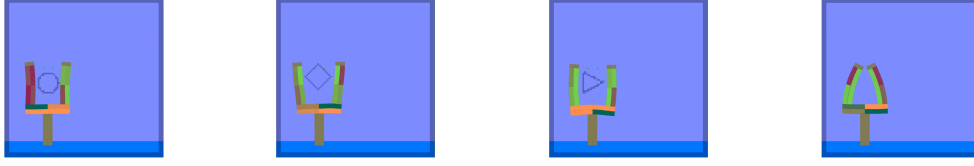


Figure 6-4: Exemplar frames of the soft gripper manipulating the four shape classes.

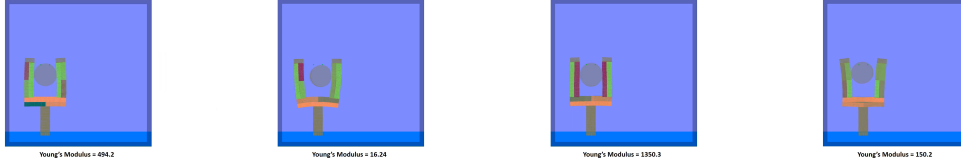


Figure 6-5: Exemplar frames of the soft gripper manipulating four disks of different stiffnesses.

6.1.3 Results

We evaluate our method on two supervised learning problems — grasped object classification and proprioceptive state reconstruction. Since previous work [108, 79] is inapplicable to a general task learning regime, we focus on two baselines: human-specified sensor locations and random placement. These demonstrations show the full power of a sensor placement model in our co-learning pipeline as opposed to the fixed-sensor, static representation learning problem.

Tactile Sensing Given a dynamic, soft robot gripper with a fixed, open loop trajectory, we seek an optimal sensor placement and classification/regression network that can best reason about a grasped object. In the classification task, the gripper must disambiguate between four shapes (square, diamond, triangle and *no shape*) through manipulation. 120 simulations are performed for each class in total. In the regression task, a ball with unknown Young’s modulus is manipulated and its stiffness must be inferred. 1000 simulations are captured in total for the stiffness regression problem with Young’s modulus varying from 10^0 to 10^4 . Example stills from simulations across shapes, materials, and actuation inputs can be seen in Figs. 6-5, 6-4.

In all tasks, the location and size of the objects is varied between runs, with noise added to actuators. This makes the learning tasks challenging as the robot must learn to reason

about the qualities of the different tactile responses – subtle differences that can be difficult to detect. The network cannot simply memorize motion trajectories, especially given the added noise. The classification task amplifies this difficulty, since contact with the object only happens at a few rather discrete moments, making supervision sparse.

Each sensor on the gripper provides $2 \times d$ inputs at each time step, over a time series of $T = 50$ simulation steps. $\tau \times 4 \times T$ time series inputs are captured from the gripper and provided to the network.

To solve these tactile sensing tasks, we augment our PSFE with a recurrent LSTM module to process the entire trajectory of data. At every timestep, features are extracted using our PSFE, fed into our LSTM with 50 recurrent units, and finally passed through a single fully connected layer which predicts the output. For the shape classification task, the network was to predict a categorical distribution, $f(\mathbf{x}; \mathbf{w})$ over K possible objects with softmax activation, and was trained using a cross entropy loss function. For the stiffness regression task, the final layer was to output a real scalar, and was trained using a mean squared error loss from the ground truth stiffness. These losses can be expressed as,

$$\underbrace{L_t = - \sum_{k=1}^K \mathbf{y}_k \log(f(\mathbf{x}; \mathbf{w})_k)}_{\text{(shape classification)}} \quad \underbrace{L_t = ||\mathbf{y} - f(\mathbf{x}; \mathbf{w})||_2}_{\text{(stiffness regression)}}$$

Results for both tasks can be found in Fig. 6-3. We used Adam as our optimizer, set the sparsity regularizer to 0.01, the learning rate to 5×10^{-4} , and the minibatch size to 8. We ran our pipeline for $\tau = 5, 10, 20$. We compare to a baseline where τ sensors are assigned randomly *a priori* and fixed. Our co-learning method vastly outperforms the baseline in generalization on test sets. While the baseline overfits quickly due to poor sensor placement, our algorithm places sensors in regions most relevant to the tactile sensing task. Specifically, it places sensors at common gripper-object interaction points, such as the interior gripper base and fingertips.

Proprioception Given strain and strain rate inputs from a collection of sensors (a $2 \times d \times \tau$ input space), the goal is to reconstruct the entire position map (deformation) and velocity map (deformation rate) of the robot (a $2 \times d \times N$ prediction space). Here, n is a set

number of particles whose pose we wish to predict. This is difficult for two reasons. First, $\tau \mathbf{I}N$; a low input dimension must predict a much larger output dimension. Second, four of our six robots are floating base robots; neither rigid degrees of freedom nor ground contact information are provided, making state estimation challenging.

We employ a similar architecture as used in Sec. 6.1.3. The PSFE outputs an 8 dimensional latent space. Then, a fully connected MLP is appended of hidden layers (64-128) with ReLU activations everywhere but the final layer translates the latent space to pose predictions. While the bottlenecking latent space may seem a handicap, we still achieve excellent pose prediction, since soft robots have natural subspaces that are captured by neural architectures [136]. Further, a compact latent space unlocks applications such as motion subspace visualization and control, which we explore later.

Our loss function was chosen to be the L_2 difference of the reconstruction of the normalized positions and velocities of each particle compared to ground truth. We also add a small variational autoencoder loss [77, 122] to ensure a descriptive latent space. Formally, our task loss $L_t(\mathbf{w})$ is:

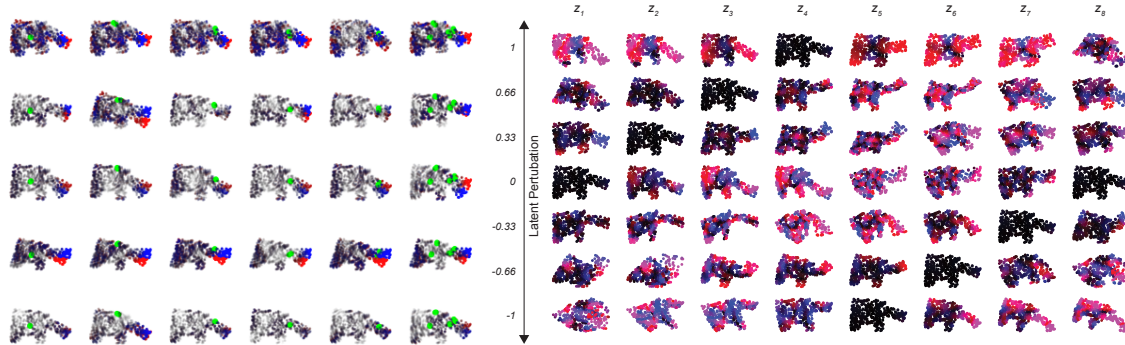


Figure 6-6: **Left:** Visualization of the effects of sensor locations (large green circles) on the 2D elephant with different sensor placements (columns) over five exemplar poses (rows). Frames show the reconstruction error with all but the shown sensor turned off. Brighter colors indicate larger deviation of the reconstructed robot (blue) from ground-truth (red). Sensors lower reconstruction errors in their immediate neighborhoods the most. The rightmost column shows all five sensors turned on, yielding the best reconstructions. **Right:** Latent space of the 2D elephant. Columns represent different latent dimensions of the elephant, generated by one-hot latent vector activations. Each column ranges latent activations from -1.0 to 1.0 . Redder particles indicate higher speeds in x ; bluer particles indicate higher speeds in y .

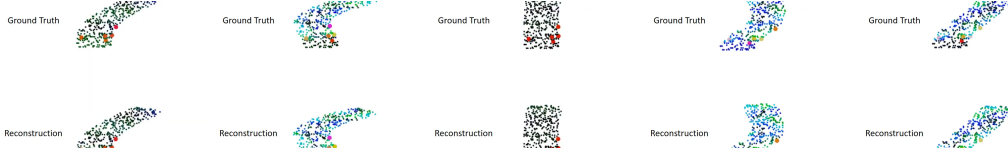


Figure 6-7: Example reconstructions from a 2D Arm proprioception co-learning example. Sensors are shown as large, reddened circles. Points near the base are most important to track as they kinematically affect everything above them. All coordinates are normalized. Green channels represent the velocity in x (higher velocity is greener); blue channels represent the velocity in y (higher velocity is bluer).

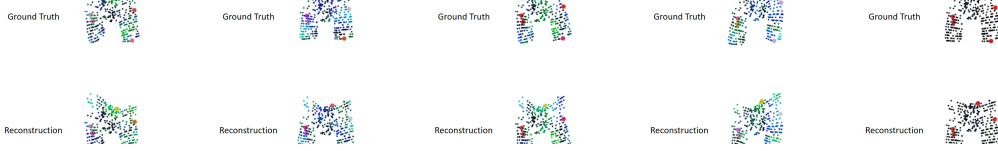


Figure 6-8: Example reconstructions from a 2D Biped proprioception co-learning example. Sensors are shown as large, reddened circles. Points on the legs are the most important to track as they have the most dynamic motion. All coordinates are normalized. Green channels represent the velocity in x (higher velocity is greener); blue channels represent the velocity in y (higher velocity is bluer).

$$L_t = \sum_{(\mathbf{x}, \mathbf{y}) \in \Upsilon} \|f(\mathbf{x}; \mathbf{w}) - \mathbf{y}\|_2 + w_r L_r(\mathbf{x}, \mathbf{y}; \mathbf{w}) \quad (6.4)$$

where Υ is the dataset of strain (rate) ($\mathbf{x}$) and position/velocity ($\mathbf{y}$) pairs, $f(\cdot; \mathbf{w})$ is our network which predicts robot pose for a given input, and L_r is the VAE regularizer loss (see, *e.g.*, [39]). $w_r = 10^{-4}$ is a regularization weight. Adam was used as the optimizer with a learning rate 10^{-3} . For each problem, 400 randomly downsampled particles were simultaneously used as reconstruction targets and candidate sensor locations.

Our dataset was generated by capturing frames from trajectories generated during robot motion optimization. We used the control optimization from [66], which performs gradient descent on model-based simulation gradients. By capturing a variety of motions from different parts of the optimization, a wide variety of motion frame data was captured. Six pneumatically-powered robots were explored based on models in prior related work: a 2D Biped, a (fixed-base) 2D Arm, a 2D Elephant, a 3D Arm, 3D Quadruped, and a 3D Hexapod. Examples of trajectories used for data collection, along with sample reconstructions, can

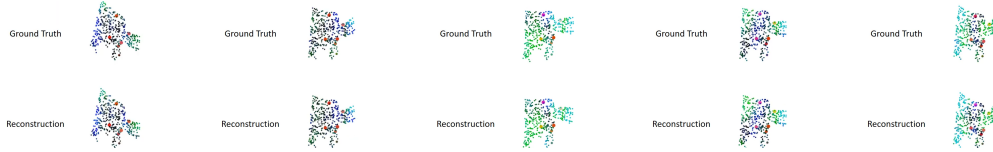


Figure 6-9: Example reconstructions from a 2D Elephant proprioception co-learning example. Sensors are shown as large, reddened circles. Points on the legs are the most important to track as they have the most dynamic motion. Points near the base of the trunk or on the trunk are most important to track as they comprise the most dynamic portion of the robot. Only two sensors are reserved for the more static base. All coordinates are normalized. Green channels represent the velocity in x (higher velocity is greener); blue channels represent the velocity in y (higher velocity is bluer).

be found in Figs. 6-7, 6-8, 6-9, 6-10, 6-11, 6-12. Each of the learning experiments was repeated five times for $\tau = 4, 5, 6$, and compared to a baseline of random sensor locations. Results can be seen in Table 6.1; convergence plots can be found in Fig. 6.1.6.

Our adaptive co-learning of sensors and task vastly outperforms baselines for all tasks, often with strong statistical significance. We performed a paired t-test for the aggregate results of all robot morphologies; strong statistical significance was found for all morphologies (Table 6.3) except for the 2D Biped, which is the easiest example. For the Arm examples, our algorithm achieves a 30% reduction in error compared to the baseline; the 2D Elephant and 3D Hexapod generally perform 10% better in the adaptive case. Reasonably, our algorithm delivers the largest improvements on robots with highly dynamic deformations, as these are more difficult problems where sensor placement matters most. Our algorithm outperforms baselines on dynamically consistent robots such as the Biped and Quadruped as well, but with smaller gains.

Fig. 6-6 presents sample reconstructions of the elephant, along with the contributions of individual sensors. Here, three of the five sensors are placed on or near the trunk, which experiences the most extreme deformation (rates) and thus requires the most information to reconstruct accurately.

Comparison To Human Intuition The main goal in this section is to develop an *automated* means of co-learning soft robotic tasks and optimizing sensor placement. In order to further emphasize the value of such an automated sensor selection, we compare it to manual human selection. Similar to the random baselines, for these experiments, sensor locations

were chosen *a priori* to training.

In order to gather data for these experiments, ten participants (all engineers with knowledge and experience in continuum mechanics) were provided renderings of all of the 2D robot morphologies, and provided videos from the captured motion dataset as well as a description of the target task. The participants were then asked to select particle locations, choosing where they thought would be the best sensor locations to learn the task. 2D morphologies were used in these experiments in order to facilitate the user data collection process. On the manipulation task of both object classification and material regression, humans were evaluated on their ability to select sensor placements for 5, and 10 sensors, aligning with Fig. 6-3.

These experiments provide striking results, which can be found in Table 6.1 (reconstructions) and Table 6.2 (manipulation). Notably, it appears that humans are actually antagonistically bad at choosing sensor locations for some tasks. Not only does our method dominate human baselines, but even the random baseline is competitive with the human baseline, outperforming it in the case of reconstruction and within statistical bounds in the case of manipulation. For example, in the case of the 2D Elephant, many labelers chose to put four sensors on the legs, and only one on the trunk — even though the legs experience relatively minor deformations while the trunk experiences rapid, large deformations. In other cases, the sensors were more equitably spaced in regions of high deformation, but human biases for geometrically salient locations, rather than locations relevant to the experienced strains and task, were noticeable. This insight emphasizes the importance of computational aid to compensate for flawed human intuition for sensor coverage.

Resilience To Sensor Noise To further demonstrate our method’s applicability to real-world systems, we compared our adaptive method to the random baseline for both the 2D Arm and the 3D Hexapod, in the presence of sensor noise. These morphologies were chosen because they experience highly dynamic motion where noise would have the most impact on results. For five runs, random Gaussian noise was added to the strain and strain rates, centered at the ground truth, with 1% standard deviation (roughly matching the accuracy of physical strain gauges). While reconstruction was predictably worse than in the no-noise

case, our adaptive method still vastly outperformed the random baseline (Table 6.4), with lower variance.

6.1.4 Applications

We have judiciously trained our translational proprioception network through a small latent space. This affords two advantages: interpretable latent motion subspaces, and coordinates for robot control.

Motion Subspace Visualization Our latent space provides natural, low-dimensional coordinates through which to represent the soft robot. Figure 6-6 visualizes our eight latent variables, demonstrating interpretable coordinates representing different aspects of the robot’s motion.

Control We demonstrate how the latent feature space learned during proprioception training can be used as an observer model for control of the 2D Bipod. To our knowledge, our control demonstration is the first example of closed-loop computational control of terrestrial soft robots from intrinsic sensors. The Bipod was chosen, since its cyclic motion means a stable gait can be learned from limited training data. We save the learned encoder network during Bipod proprioception training and use it as the observation model during robot control tasks. We optimize a neural network MLP controller with the fixed PSFE observation model. *Only* intrinsic strain and strain information is provided as input to the observer network, meaning no translation, rotation, or contact information is provided in any way to the controller, making this task extra difficult. 20 sensors were required for a robust observation model. As the robot optimizes, it finds a cyclic gait. As one would expect, the latent space observed during this motion is cyclic as well. Please see Fig. 6-13 for the optimized robot and a visual analysis of the latent space during simulation.

6.1.5 Derivation of the Strain Rate Tensor

The Green strain rate is defined as:

$$\mathbf{S}_i = \frac{1}{2}(\mathbf{F}_i^T \mathbf{F}_i - \mathbf{I}) \quad (6.5)$$

differentiating with respect to t , we get:

$$\frac{d\mathbf{S}_i}{dt} = \frac{1}{2} \left(\left(\frac{d}{dt} \mathbf{F}_i^T \right) \mathbf{F}_i + \mathbf{F}_i^T \frac{d}{dt} \mathbf{F}_i + \frac{d}{dt} \mathbf{I} \right) \quad (6.6)$$

$$= \frac{1}{2} \left(\left(\frac{d}{dt} \mathbf{F}_i^T \right) \mathbf{F}_i + \mathbf{F}_i^T \frac{d}{dt} \mathbf{F}_i \right) \quad (6.7)$$

The deformation gradient $\mathbf{F}_i$ is the derivative of a point i in the world $\mathbf{x}_i$ with respect to its location in the material coordinate space $\mathcal{X}_i$; in essence this tell us how all vectors along the undeformed material have changed after deformation.

$$\frac{d}{dt} \mathbf{F}_i = \frac{d}{dt} \nabla_{\mathcal{X}_i} \mathbf{x}_i \quad (6.8)$$

$$= \nabla_{\mathcal{X}_i} \frac{d}{dt} \mathbf{x}_i \quad (6.9)$$

$$= \nabla_{\mathcal{X}_i} \mathbf{v}_i = \nabla_{\mathbf{x}_i} \mathbf{v}_i \nabla_{\mathcal{X}_i} \mathbf{x}_i \quad (6.10)$$

$$= (\nabla_{\mathbf{x}_i} \mathbf{v}_i) \mathbf{F}_i \quad (6.11)$$

where $\mathbf{v}_i$ is the velocity of a point i in world coordinates. The last two identities come from application of the chain rule and definition of the deformation gradient, respectively. $\nabla_{\mathbf{x}_i} \mathbf{v}_i$ is a quantity computed in MPM simulation known as the *affine matrix*, which we denote as $\mathbf{C}_i$. Thus we have the result that $\frac{d}{dt} \mathbf{F}_i = \mathbf{C}_i \mathbf{F}_i$. Substituting this into Eqn. 6.7, we are left with:

$$\frac{d\mathbf{S}_i}{dt} = \frac{1}{2} (\mathbf{F}_i^T \mathbf{C}_i^T \mathbf{F}_i + \mathbf{F}_i^T \mathbf{C}_i \mathbf{F}_i) = \frac{1}{2} \mathbf{F}_i^T (\mathbf{C}_i^T + \mathbf{C}_i) \mathbf{F}_i \quad (6.12)$$

6.1.6 Proprioception Problem Hyperparameters

Hyperparameters For Reconstruction Task						
Robot	# Frames of Data	# Particles to Reconstruct/Sensor Locations Considered	Sparsity Regularizer	VAE Regularizer	Minibatch Size	
2D Arm	32040	400	0.001	10.0	20	
2D Biped	48060	400	0.001	10.0	20	
2D Elephant	48060	400	0.001	10.0	20	
3D Arm	48060	400	0.0075	10.0	20	
3D Quadraped	48060	400	0.0075	10.0	20	
3D Hexapod	20100	400	0.0075	10.0	20	

Robot + Sensor Count	Minimum Reconstruction Error By Task (Mean/Std.)							
	Test Adapt.	Test Rand.	Train Adapt.	Train Rand.	Test Human	Train Human	Full Sensor Test	Full Sensor Train
2D Arm 4 Sensors	9.29/0.403	12.0/2.16	4.09/0.114	5.44/0.510	-	-	5.27/0.058	3.25/0.027
2D Arm 5 Sensors	8.93/0.821	12.2/2.22	3.93/0.228	5.38/0.569	14.24/5.538/1.91	6.88/0.71	"	"
2D Arm 6 Sensors	7.89/0.43	10.1/1.34	3.73/0.092	4.66/0.307	-	-	"	"
2D Elephant 4 Sensors	9.36/0.333	10.0/0.687	6.95/0.330	8.03/0.524	-	-	6.29/1.69	5.60/0.14
2D Elephant 5 Sensors	8.56/0.516	9.89/0.460	6.58/0.252	7.83/0.353	10.03/1.00	7.69/0.48	"	"
2D Elephant 6 Sensors	8.21/0.280	8.86/0.420	6.32/0.150	7.10/0.247	-	-	"	"
2D Biped 4 Sensors	11.97/0.7106	12.16/0.5069	8.82/0.1919	9.319/0.2472	-	-	8.81/0.162	7.57/0.173
2D Biped 5 Sensors	11.90/0.5839	12.02/0.4421	8.724/0.2418	9.264/0.2446	12.60/0.86	9.46/0.44	"	"
2D Biped 6 Sensors	11.84/0.5308	11.95/0.5392	8.692/0.2264	9.222/0.2792	-	-	"	"
3D Hexapod 4 Sensors	6.04/0.146	6.68/0.263	4.80/0.153	5.03/0.045	-	-	5.53/0.110	4.71/0.113
3D Hexapod 5 Sensors	6.01/0.142	6.67/0.187	4.81/0.131	5.06/0.063	-	-	"	"
3D Hexapod 6 Sensors	6.01/0.134	6.58/0.229	4.843/0.132	5.019/0.085	-	-	"	"
3D Quadruped 4 Sensors	6.60/0.260	7.08/0.321	4.38/0.070	4.56/0.125	-	-	5.63/0.097	4.36/0.096
3D Quadruped 5 Sensors	6.45/0.294	6.85/0.345	4.34/0.087	4.51/0.113	-	-	"	"
3D Quadruped 6 Sensors	6.36/0.322	6.70/0.373	4.28/0.123	4.48/0.115	-	-	"	"
3D Arm 4 Sensors	3.19/0.118	3.95/0.433	3.14/0.114	3.86/0.390	-	-	2.65/0.078	2.60/0.076
3D Arm 5 Sensors	3.17/0.103	3.88/0.437	3.11/0.101	3.78/0.400	-	-	"	"
3D Arm 6 Sensors	3.14/0.099	3.79/0.400	3.08/0.096	3.70/0.367	-	-	"	"

Table 6.1: Average minimum reconstruction errors (lower is better) of our six robots for three sensor combinations, for our adaptive method, for the random sensor placement baseline, and where applicable, for the human baseline. For our adaptive method, minima were taken only once the training reached the clamped, fine-tuning phase. Our method always outperforms the random and human baselines and typically beats them by a significant margin; bolded results indicate where our algorithm outperforms the random baseline by at least a standard deviation on either side. An estimated “best” reconstruction error for our architecture is presented in the far right columns, computed by training with a sensor on each target particle.

	Classification			Regression		
	5	10	Full	5	10	Full
Random	0.19/0.03	0.18/0.03	0.003/0.005	0.18/0.07	0.18/0.04	0.01/0.0
Human	0.21/0.10	0.06/0.06	-	0.13/0.01	0.12/0.02	-
Sparse	0.009/0.01	0.01/0.005	-	0.06/0.02	0.08/0.03	-

Table 6.2: Performance scores (lower is better) for manipulation tasks on both classification (object detection) and regression (material stiffness) with varying number of sensors. Our method significantly outperforms both random as well as human labelling baselines.

P-Values For Proprioceptive Tasks					
2D Arm	2D Elephant	2D Biped	3D Hexapod	3D Quadruped	3D Arm
5.4×10^{-4}	2.6×10^{-3}	0.58	2.5×10^{-3}	0.011	1.64×10^{-5}

Table 6.3: P-values from a paired t-test comparing our adaptive algorithm to the random baseline approach.

Minimum Reconstruction Error By Task With 1% Noise (Mean/Std.)				
Robot (Sensor #)	Test Adapt.	Test Rand.	Train Adapt.	Train Rand.
2D Arm (4)	9.42/0.638	11.24/1.17	4.24/0.071	5.04/0.381
2D Arm (5)	9.07/0.980	11.36/2.05	4.16/0.222	5.03/0.572
2D Arm (6)	8.70/0.969	11.09/1.74	4.06/0.245	4.91/0.502
3D Hexapod (4)	6.09/0.03	6.82/0.325	4.85/0.091	5.06/0.100
3D Hexapod (5)	6.13/0.104	6.56/0.365	4.89/0.109	5.01/0.096
3D Hexapod (6)	6.08/0.142	6.53/0.348	4.87/0.125	4.98/0.094

Table 6.4: Average minimum reconstruction errors (lower is better) of our two most dynamic robots for three sensor combinations with 1% Gaussian noise added, for our adaptive method, for the random sensor placement baseline. Analysis method is the same as in the no-noise case. Our method always outperforms the random and human baselines; bolded results indicate where our algorithm outperforms the random baseline by at least a standard deviation on either side.

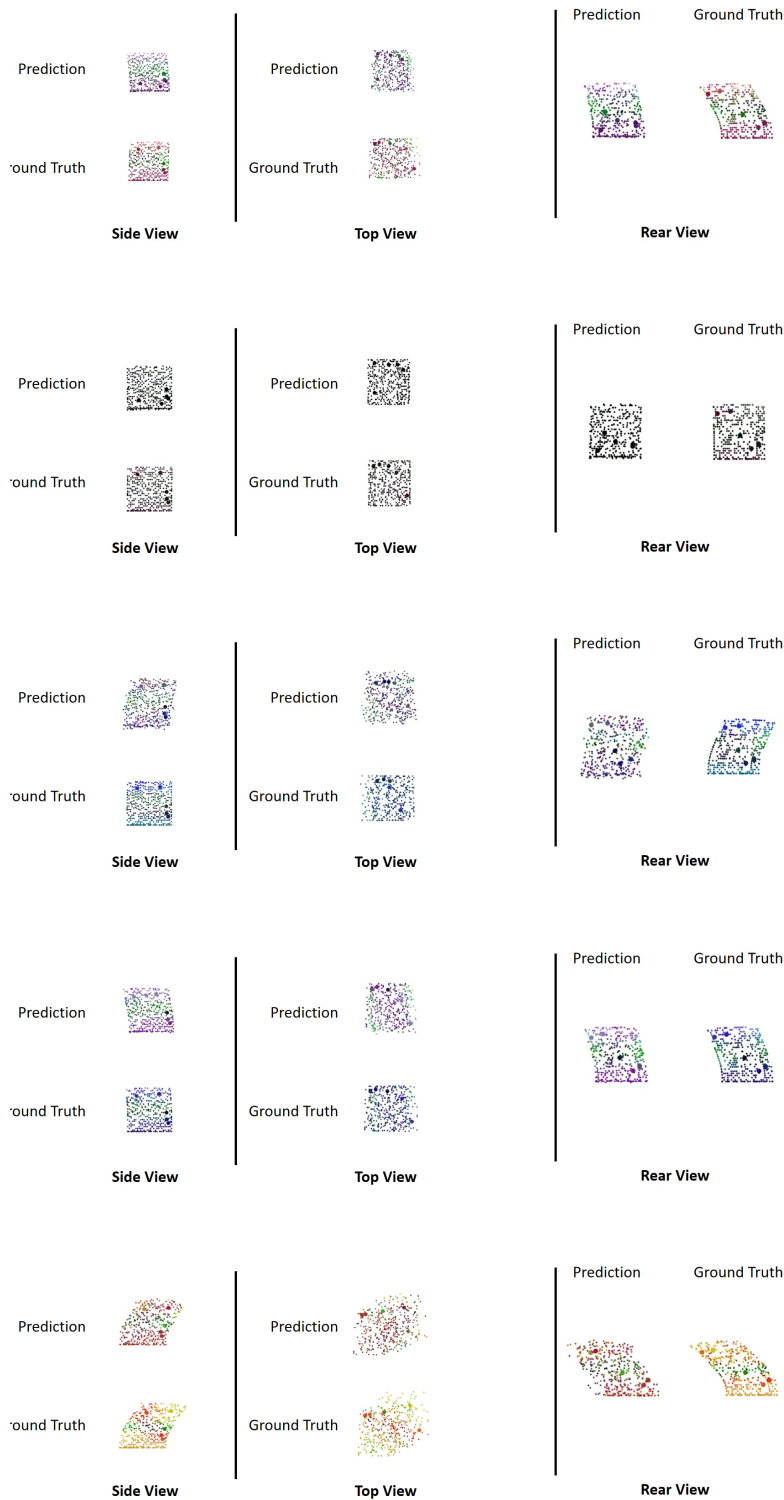


Figure 6-10: Example reconstructions from a 3D Arm co-learning example. Sensors are shown as large circles. All coordinates are normalized. Red channels represent the velocity in x (higher velocity is redder); green channels represent the velocity in y (higher velocity is greener); blue channels represent the velocity in z (higher velocity is bluer).



Figure 6-11: Example reconstructions from a 3D Quadruped co-learning example. Sensors are shown as large circles. All coordinates are normalized. Red channels represent the velocity in x (higher velocity is redder); green channels represent the velocity in y (higher velocity is greener); blue channels represent the velocity in z (higher velocity is bluer).

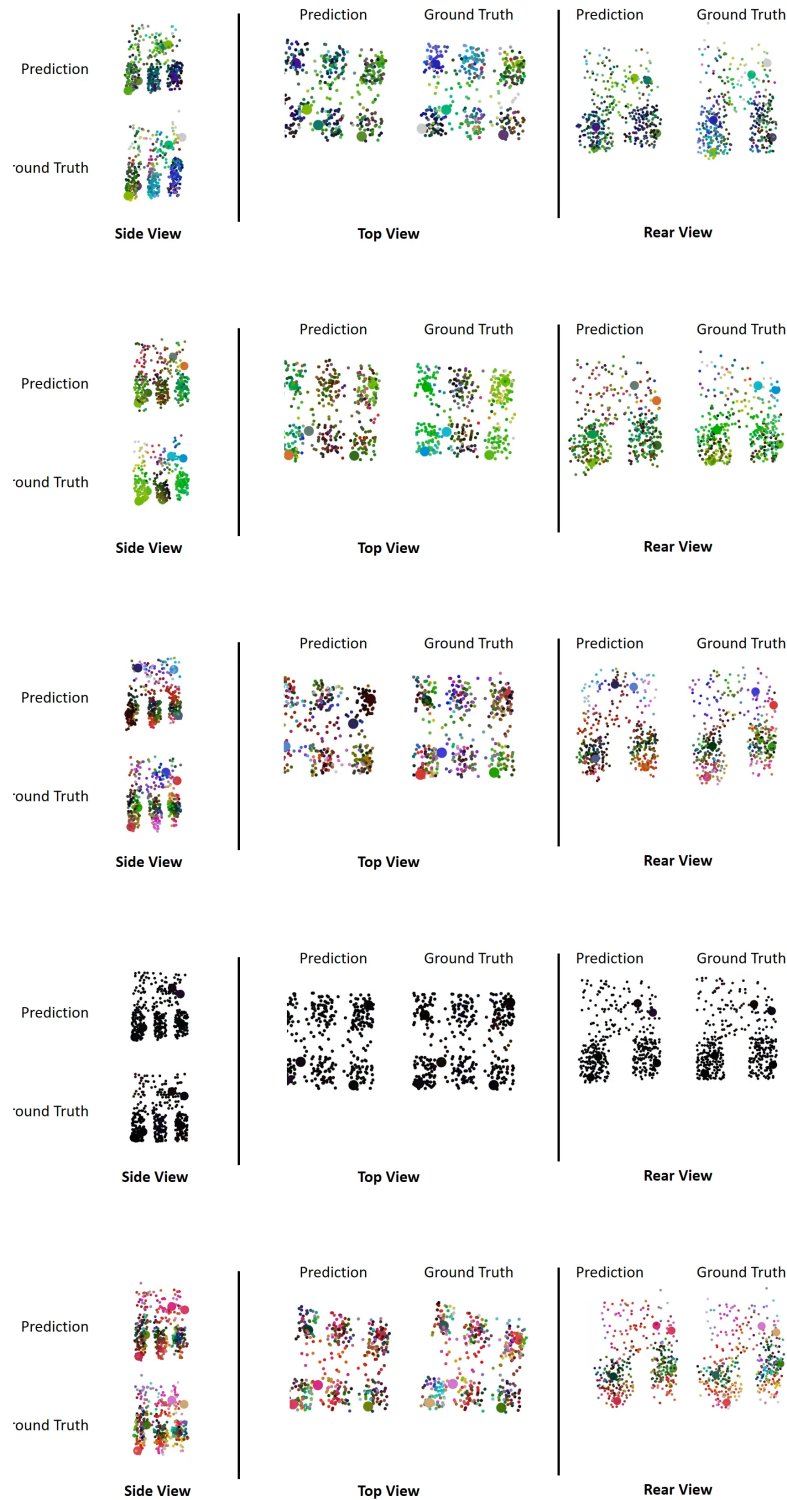


Figure 6-12: Example reconstructions from a 3D Hexapod co-learning example. Sensors are shown as large circles. All coordinates are normalized. Red channels represent the velocity in x (higher velocity is redder); green channels represent the velocity in y (higher velocity is greener); blue channels represent the velocity in z (higher velocity is bluer).

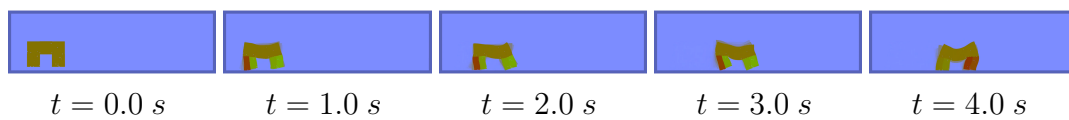
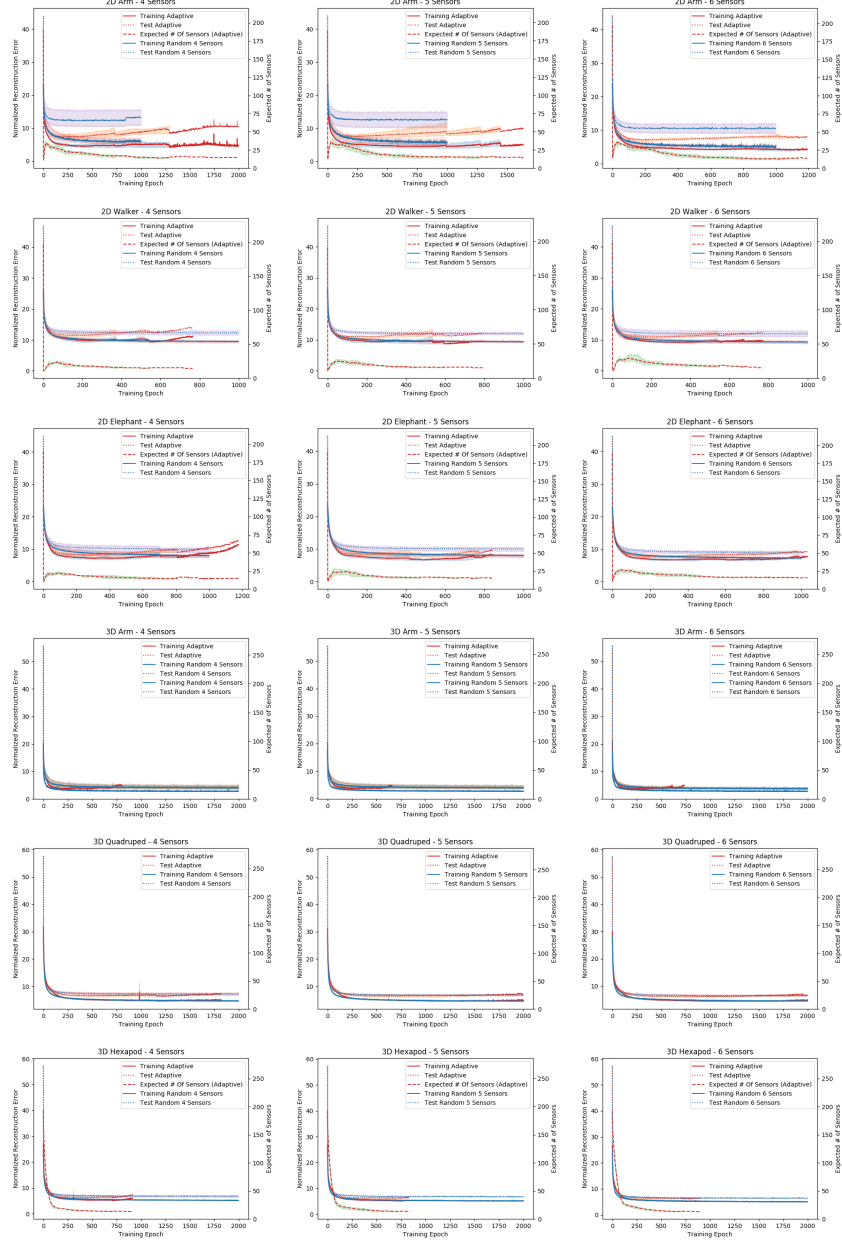


Figure 6-13: A montage of the 2D Biped, controlled purely from strain sensors, colored in blue.

Additional Convergence Plots for the Proprioception Task



Convergence of the reconstruction loss for our adaptive sensor placement algorithm and the random sensor baseline for the state reconstruction task with $\tau = 4, 5, 6$, along with a 100% (full range) confidence interval. Each experiment was run at least 5 times. The convergence of the expected number of sensors over epochs is also plotted. Our algorithm outperforms the random baseline in all cases.

6.1.7 Outlook

We have demonstrated our task and sensor co-learning algorithm on a wide array of learning tasks, including tactile sensing and dynamic state reconstruction. Our adaptive algorithm vastly outperforms baselines and our representation has useful applications to state visualization and control.

Our method reveals several directions for future exploration. First, many strain sensors only measure single-directional strain; thus, it would be useful to have a method for choosing sensor orientation during learning. Second, while we focused on soft robots in this manuscript, we believe our method is general enough to be applied to high-dimensional sensor co-design problems in other fields, and hope it inspires such research.

Finally, it would be exciting to fabricate our simulated robots and demonstrate trained sensing capabilities in the wild. Developing a general pipeline for translating learned models to physical hardware would prove physical practicality of our method and address the difficult "sim-to-real" research question. In the next section, we present a novel, rich, learning-based strategy for sensing in soft systems, whose design we hope could be optimized *via* the methods described here and deployed directly on soft robot hardware. (Such a fabrication method, combined with this sim-to-real pipeline, would constitute layer 7 of the robot design stack.)

6.2 Sensor Fabrication - A Simple Sensing Architecture For Soft Robotics

In this section, we describe our work toward fabricating sensing architectures that can be useful for soft deformable systems, including wearables and soft robotic systems. Our textile-based system is demonstrated in a human-wearable glove, called MemGlove.

6.2.1 System Overview

MemGlove is based on the idea that resistive sensing can be used in order to measure in-hand (pose) deformation, and fluidic pressure sensing can be used in order to measure

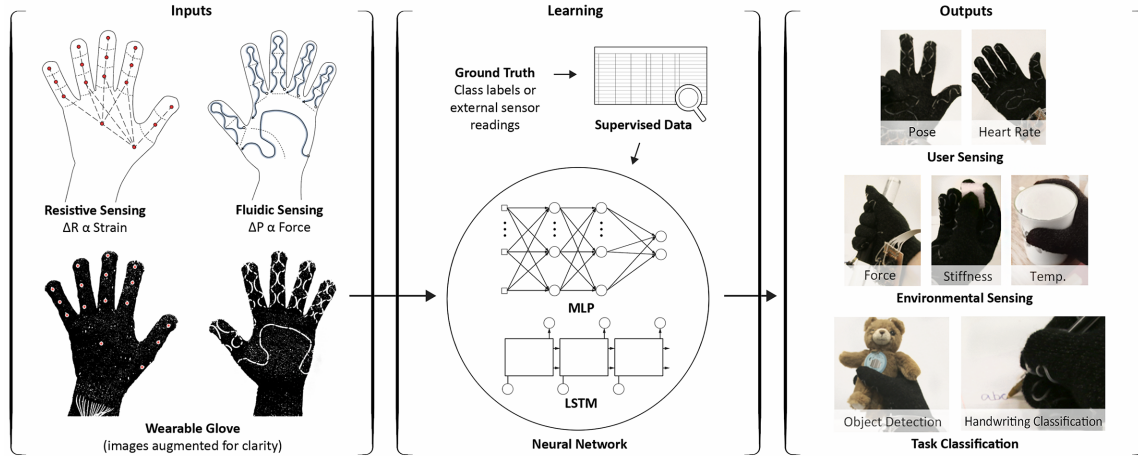


Figure 6-14: Our wearable glove system. Resistive sensing and fluidic sensing combine to provide signals for strain and contact force information, respectively. These signals are fed into a pre-trained neural network architecture for inference, which is trained from captured data labeled with ground-truth knowledge. Depending on the neural network used, the output of the network provides inference for a wide variety of downstream tasks, which we demonstrate. Electrode points which are usually not visible have been colored and enlarged in the left-hand images for clarity.

out-of-hand (tactile) deformation, *i.e.* contact pressures. Strategic design choices ensure that these readings are mostly disjoint in the aspects of the interaction they capture, and they do not interfere with each other. Thus, these readings provide maximum information to then be (jointly) reasoned about by a downstream neural network model. This neural network model is responsible for translating raw signals to task-specific inference, such as hand pose reconstruction, grasped object classification, and so on, and is trained offline in a supervised manner from labeled ground-truth data.

One of our major contributions is the coupling of two complementary sensors into the glove, unlocking powerful downstream inference applications. We use a fully conductive thread glove as a substrate for the resistive sensing and sew in thin, tube-based sensors for fluidic pressure sensing. Electrodes are placed in strategic locations so as to maximize deformation information provided by resistance measurements. The resistance between all unique pairs of electrodes is measured to allow large area measurements of strain information. The tube-based fluidic sensors provide pressure signals induced by compression, as per the ideal gas law. The fluidic tube sensor's fibrous nature allows it to be strategically sewn into the glove in a manner to optimize the response to normal force. The signals from

the resistive and fluidic pressure sensing are read synchronously, and the data is fed into a machine learning model for task-based inference.

For comfort, electronics are packaged into a small form-factor and mounted on the user's wrist. This is another advantage of our fiber-based system — both the conductive and fluidic sensors require only compact transducers and minimal signal condition electronics which can be unloaded from the hand. The electronics are connected to a laptop that reads raw input signals (resistance measurements and pressure signals) and feeds them forward through a (context-dependent) neural network. The choice of network is modular and depends on the user's task, be it hand pose reconstruction, grasped object stiffness estimation, *etc.* We train these neural networks with recorded sensor data and ground-truth labels captured for the task at hand. The system is summarized in Fig. 6-14.

In the remainder of this section, we describe and characterize the resistive and pressure sensor design, detailing their decoupled nature, before describing our machine learning pipeline that synthesizes sensor data into higher-level sensing.

Resistive Knit Strain Sensing In this work, we explore a commercial off the shelf knitted glove (Aglove) which has been developed to allow gloved use of capacitive touch screens. Due to the inclusion of the high conductance silver threads within the knit, the glove has a resistance of approximately $5\Omega cm^{-1}$ which varies when a resistance is applied. Due to the knit pattern, the material can undergo approximately 70% strain and when relaxed, the knit returns to a similar state. The knit pattern is directional, and thus the sensitivity and sensing capabilities of the knit are highly directional. Previous work has demonstrated the capabilities of custom knit-based strain sensors [5], identifying how fabric and knit parameters can change the properties of the sensor [4, 3]. Our decision to add a finite number of electrode connections to the 'off-the-shelf' glove enables rapid creation.

To characterize the sensing capabilities of this knit, a section of the material was placed in an Instron Machine and a strain of 0-50% applied for 80 cycles. This was repeated for directions along and across the knit to characterize the directionality of the sensing capabilities. Fig. 6-15 shows that the knit shows high sensitivity to strain *along* and low sensitivity to strain *across* the knit. This is adequate for our purposes, as the majority of

deformations of the hand (particularly in the DIP and PIP joints of the fingers) bend in the direction along the knit. The response along the knit shows high repeatability but also a very high hysteresis. This is most likely due to the relaxation and nonlinear behaviors of the knitted fibers. The average response time was found to be 0.35 seconds, whereas the recovery time was longer, and for the 80 cycles performed was found to be approximately 0.8 seconds. This does potentially limit the use of the knitted glove as a sensor for some application which require a fast response, such as slip detection, but was not a major inhibiting concern for any of the applications presented in this paper.

The choice of a fully conductive knitted glove as a substrate for our sensors admits three further benefits. First, it is fiber-based, making it easy to attach additional sewable components, such as our fluidic tubes or conductive fibers for connecting wrist-mounted electronics. This latter connection helps keep electronics off the hand itself, avoiding the need for flexible electronics and promoting ergonomic use. Second, the knit structure of the glove has minimal strain-based response to normal force/pressure. This ensures that the (pressure) responses of our fluidic tubes are maximally decoupled from the response of the knit. Finally, because the glove is fully conductive, hand self-collision (including fingertip-fingertip or fingertip-palm contact, as seen in an “OK” or “thumbs-up” gesture respectively) emits a noticeable signal, making it easy to computationally detect these types of gestures.

Fluidic Tube Sensing The principle behind our fluidic tube sensing is based off the ideal gas law: $PV = nRT$. Compression of the fluidic tube caused by contact results in an inversely correlated response in internal pressure. As we will later show, the choice of fluidic sensors has additional benefits — its correlation to temperature means we can also detect changes in held objects’ temperatures (given constant holding force). The fluidic sensors complement the sensing of the knit matrix, providing out-of-plane deformation information typically caused by environmental interaction. Our fluidic sensors are constructed from soft flexible tubes, sealed at one end and connected to a pressure transducer with an airtight seal at the other end. Although any airtight tubes or bladders could be used, a thin soft tube allows for easy integration into the glove.

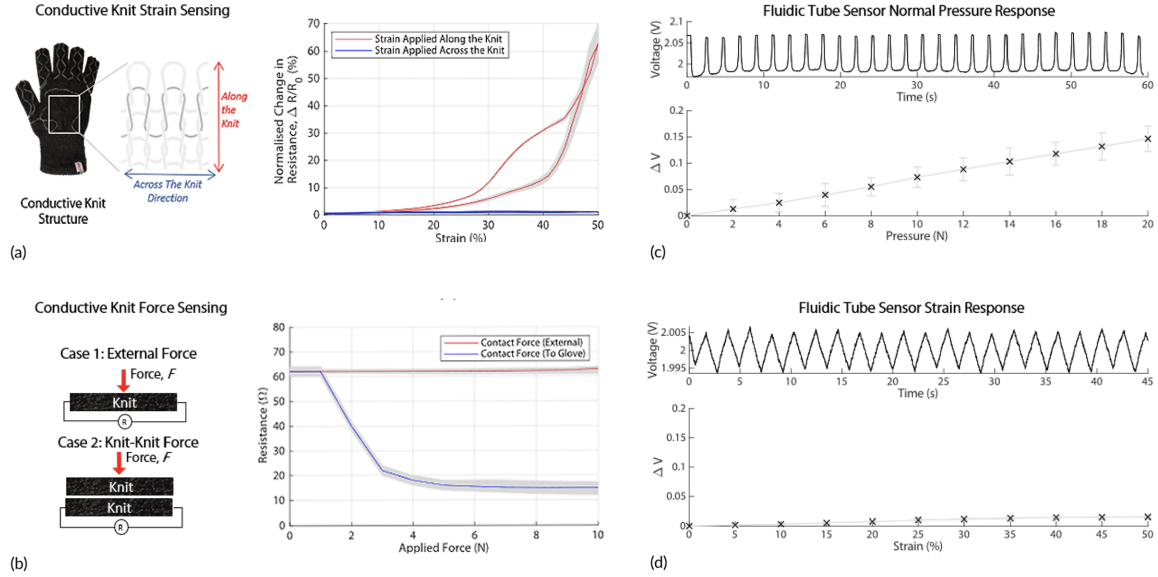


Figure 6-15: a) Characterization of the conductive knit to applied strain in both directions; b) Characterization of the conductive knit to external and inter-glove contact forces; c) Exemplar time series of the response of the fluidic tube sensor to a fixed force and normal pressure; d) Exemplar time series of the response of the fluidic tube sensor to a fixed strain.

To characterize the fluidic tube sensors, a cyclic normal force profile from 0-50 N was applied using an Instron Machine, repeated 80 times. The results from this characterization of normal force are shown in Fig. 6-15. The sensor is sensitive to normal force and shows a linear and repeatable response with minimal hysteresis.

The strain response of the sensor is also investigated; ideally the sensor should show a low sensitivity to strain so that normal force is isolated. Strain was applied to the sensor incorporated in the knit and cycled between 0-50% strain 60 times. This response is also shown in Fig. 6-15. Although there is some response, it is far lower than that arising from normal force.

Learning Architecture Our goal is to learn pose reconstruction and task classification in a supervised manner. In order to do this, we implement neural network architectures that combine our 6 tube pressure readings and our 120 multiplexed resistance readings (from 16 electrodes) into single or multidimensional (in the case of hand-pose reconstruction) predictions. Our goal is to choose a simple neural-network-based architecture that can be re-used for all tasks. Since MemGlove is carefully designed to provide robust signals

regarding wearer interactions, we find that even simple learning architectures are capable of providing highly accurate task-based inference. Since our data is not organized in a particular grid-based or sequential fashion, we employ a fully connected multilayer perceptron (MLP), which is well-suited for general unstructured data. We further train a long short-term memory (LSTM) model, using an MLP for each cell, in order to improve inference on temporally correlated tasks (such as hand motion). For each task, we compare the MLP and LSTM architecture; typically, the LSTM outperforms the MLP when samples are highly temporally correlated and constantly changing, and vice versa.

For each task, we record combined pressure and resistance readings, along with the ground-truth target for prediction — this can either be labels (for classification tasks, such as object identification), or real numbers (for regression tasks, such as temperature prediction), and can be one or multiple output labels. While we learn different neural network weights for each task, we re-use the same network architectures for nearly every task, which we have found work with high efficacy. The fact that the same architectures can be re-used for each task makes learning new tasks simple and straightforward, and requires little hyperparameter search.

Our network architectures employ standard, state-of-the-art techniques to improve inference performance. We apply layer normalization at each layer of the neural network [7]. The activation at each layer of the neural network is a rectified linear unit (ReLU), except for the final layer, which uses the tanh or softmax function, for regression and classification respectively. Each network used two layers of 64 hidden units, except for the more complex multidimensional prediction of hand pose reconstruction. Inspired by the work of [55], hand pose reconstruction uses a 2048-2048-2048-2048-1024 architecture for the MLP and a 512-512 architecture for each cell of the LSTM. We use standard loss functions for training; for regression tasks, such as temperature classification or hand-pose reconstruction, we minimize the mean-squared error of the target output prediction; for classification tasks such as object classification we minimize the cross-entropy measure.

As described, our architectures are based on established, vetted components. This makes implementing highly optimized versions of our models easy and compact. In our case, all models were implemented in TensorFlow [1].

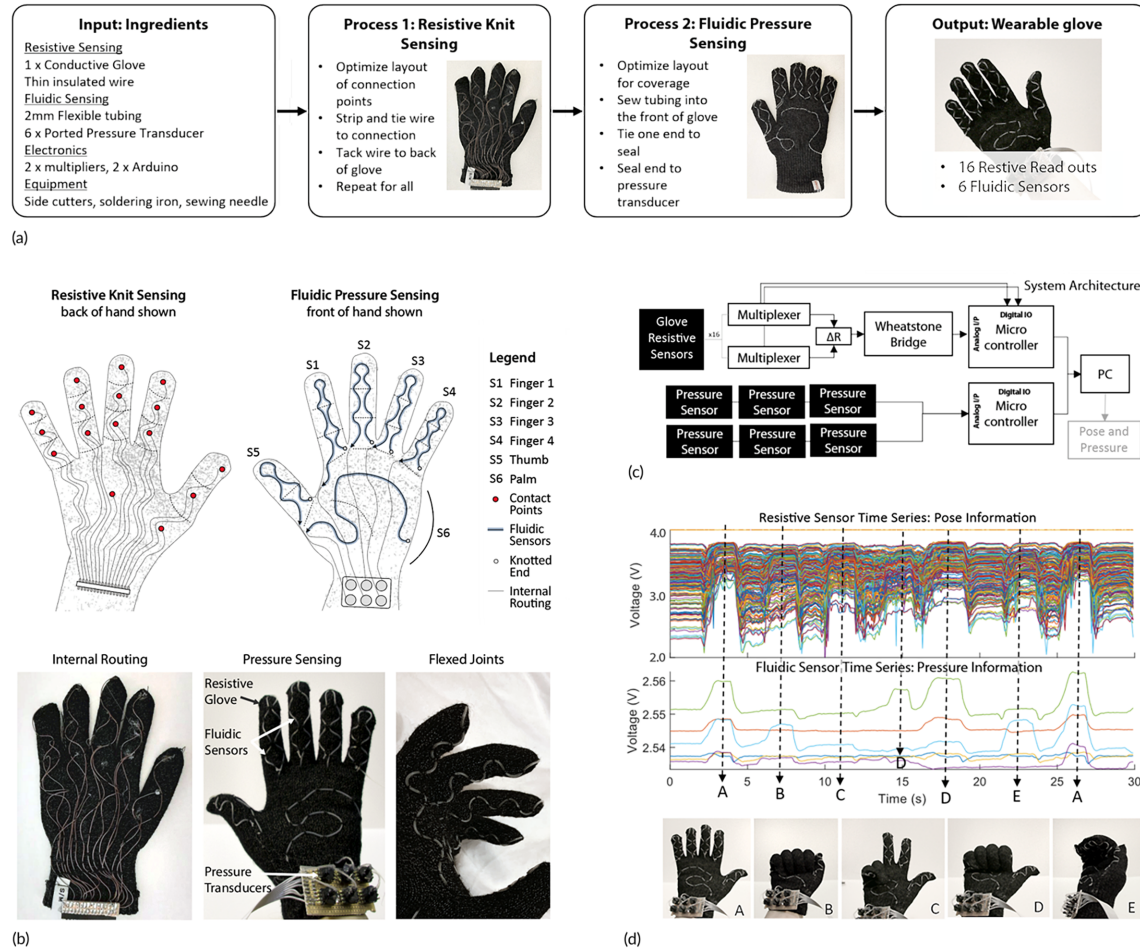


Figure 6-16: a) Recipe for the fabrication of our wearable glove; b) The layout of the sensors on the glove (top) and the fabricated glove including sensor integration; c) System architecture diagram showing the interactions between key components; d) Raw exemplar time series data from both the resistive and fluidic sensors for a variety of different hand poses.

6.2.2 Implementation

Fabrication The wearable glove is fabricated in two processes, as summarized in Fig. 6-16. First, the electrode connection points are added to the knitted glove “substrate,” after which the fluidic pressure sensors are integrated.

Process 1: Resistive Knit Sensor Design & Fabrication Using a conductive knitted glove as a base structure for the wearable device, a number of fixed connection points are made. By sequentially measuring the resistance between all pairs of n connection points,

there are $\frac{n(n-1)}{2}$ resistance readouts, providing quadratic scalability in sensing. This is a similar approach to impedance spectroscopy. In this work we consider 16 readout points which provides 120 unique sensor readouts.

A key design consideration is the placement of the electrode connections to the glove. This is a finite number, limited by electronic and fabrication constraints. The locations chosen should maximize the information that can be gained. Electrode connections have been made at points between joints to optimize the capture of strain resulting from joint bending. The 16 connection points chosen provide one connection point per major facet (*i.e.* bone between joints) of the hand, consisting of one connection per finger joint and two on the palm, as shown in Fig. 6-16b). While the knit sensing only provides sensing along the fingers, the change in bulk material or contact between fingers allows sensing of stretch and movement in the direction across the hand. Although more readout points would provide more information, this design provides a good trade off in the complexity of hardware and the information gained.

The electrode connections to the glove have been created using a thin, soft, flexible, insulated multi-core wire. The ends have been stripped and tied to the glove in the appropriate location on the inner side of the glove. The insulated wire is then tacked to the back of the glove, using a zigzag pattern to provide slack such that the wire does not prevent or limit movement while being comfortable for the wearer. To achieve the sequential resistance reads from the glove, each of the wired connections to the glove connection points is connected to two 16 channel multiplexers. The analog outputs from the two multiplexers are connected to a balanced Wheatstone bridge. By cycling through all unique combinations of resistor pairs, this builds up a map of resistance network of the glove. An Arduino Due microcontroller is used to control the multiplexers, with the output from these connected to the 12-bit Arduino Analog to Digital Converter (ADC). The balancing resistors have been chosen to maximize the output to the ADC. The microcontroller reads from each of the unique combinations of the glove and then transmits these over serial to the control PC.

Process 2: Integration of Fluidic Sensors The pressure sensors use a 2mm outer diameter soft tube, which is highly flexible and can be sewn or woven into the glove with few

limitations in form. One end is sealed by knotting and the other end is given an airtight connection to the pressure transducer using an epoxy resin. Our design uses six pressure sensors — one on each of the five fingers, and one on the palm. The tubes have been integrated such that they run along centrally along the flats of the fingers. They are then routed in a gentle curvature along the fingers to flex outward at the joints when bent. This is to maximize the response to normal force and minimize false-positive reads induced by finger bending, while also ensuring they do not limit or obstruct finger movement. This layout can be seen in Fig. 6-16b.

The pressure is measured using a Freescale MPXV6115V ported pressure transducer which provides an analog response. Each sensor is connected to an analog input on an Arduino Due which provides 12-bit resolution. The changes in pressure that are measured are very small (ranging from around 3-100 Pa), and the transducer has a sensitivity of 38.26 mV/kPa; thus we perform oversampling to increase the resolution. By capturing 256 samples and averaging we increase the resolution to 16-bit corresponding to detection of 0.05mV changes; this allows us to capture the small changes in pressure which occur.

Electronics & Data Capture The data from both microcontrollers is communicated via serial to a PC where it can be read live or stored locally. Data is read simultaneously from the two microcontrollers, with data captured at approximately 16Hz, which is sufficient for all the applications considered in this work. If needed, the hardware could be later optimized to increase the read-out frequency.

In many of the experiments, an additional sensing system was used to record ground-truth labels for learning. Of particular note, in order to obtain ground-truth hand-posture for hand pose reconstruction, a Leap Motion hand tracking system was used. This is a system that has been successfully used for other motion tracking research [107], and has been demonstrated to have a positional accuracy of up to 200 μm [160]. In order to obtain synchronized motion tracking data, the raw data from the Leap Motion was also read over serial alongside the data from the glove. The data was converted to the hand model from [149], ignoring the (irrelevant) global and wrist degrees of freedom. This model is visualized in 6-17b. In other experiments, a temperature sensor or a force sensor was incorporated to

provide a ground truth. Similarly, these were connected to a microcontroller, with the data simultaneously read over a serial connection.

In order to ensure classification and regression tasks transfer between different glove-wearing sessions, all datasets were captured in multiple sessions. Between each session, the glove was fully removed and put back on again after a time period of at least 30 minutes. This rules out wear on the glove as a source of potential overfitting. Data was recorded in at least five sessions for classification tasks, and at least three sessions overall for each regression task.

Training Procedure Training data was randomly divided into three sets: 45% of the collected data was used as a training set, 45% of the data was used as a validation set, and 10% of the data was used as a test set. Data was normalized in the training sets; those normalization constants were then reused for normalizing validation and test sets. As much as possible, data from the same recording session was kept together in the same set in order to ensure generalization of learned models on one session’s captured data to another’s. For all tasks, training was stopped when the validation loss ceased to decrease any further.

For all tasks we report the loss, as defined earlier, as well as a more human interpretable measure of real-world performance. For classification tasks, we report the probability of correct label classification. For regression tasks, we compute the average absolute error of the measurement. In the case of pose estimation, which has multidimensional prediction, this is the average error over all output angles.

6.2.3 Results

In this section we demonstrate the sensing and inference capabilities of the glove system on a wide array of tasks. First, we demonstrate the ability of MemGlove to measure the intrinsic state of the user, including state-of-the-art accurate proprioception (hand pose reconstruction), as well as heart-rate monitoring. Second, we demonstrate how MemGlove can be used to infer wearer interactions with the physical environment. Finally, we demonstrate how our sensing suite combines for complex task classification, specifically classification of grasped objects and written letters. All training experiments were performed on a laptop

with a 2.9GHz Intel i7 processor and a NVIDIA GeForce GTX 1080 GPU. In the following section we present smoothed training, test and validation data which was calculated with a smoothing window of 200 such that the general trends can be better observed.

Wearer-Intrinsic Sensing We begin by demonstrating MemGlove’s capability to reconstruct wearer hand pose, traditionally a very difficult problem with major implications for applications in virtual and mixed reality. Second, we demonstrate a simple application in health monitoring, specifically the capability of MemGlove to accurately predict the user’s heartbeat. For both of these applications we use only the resistive information of the glove, as we are not interested in interactions with the environment.

Hand Pose Reconstruction Our first application is hand-pose reconstruction, in which our system must infer the pose of the wearer’s hand. We aim to faithfully reconstruct all joint angles of the hand, as per the kinematic hand model of [149]. In this model, each finger (including the thumb) is modeled as having a ball joint at the base with three degrees of freedom, and two revolute joints further along the finger. See figure 6-17b for a reference of the hand model. Pose reconstruction is notable as this example has a multidimensional output (25 joints) to predict.

The resistance data from the glove and the ground truth of the joint configuration were captured, with the (approximate) ground truth data being captured from a Leap Motion device. Approximately 30 minutes of training data was captured, where the user aimed to move the hand in varying poses. Our trained MLP and LSTM were able to achieve an average per-joint accuracy of 6.40 and 4.78 degrees respectively. Training curves are presented in Fig. 6-17; as this task involves a lot of motion, the LSTM is able to leverage this dynamic information for higher reconstruction rates. We note that our LSTM-based reconstruction dominates even the best results of the [55], the previous state-of-the-art which boasted on average 5.3 degree per joint error for the most fine-tuned model. Training is also realistically fast: for even this, our largest and most computationally expensive task and model, wall-time for training is 4.2s per epoch for the MLP, and 19s for the LSTM (we note all other experiments ran faster). Table 6.5 presents a breakdown of how that error

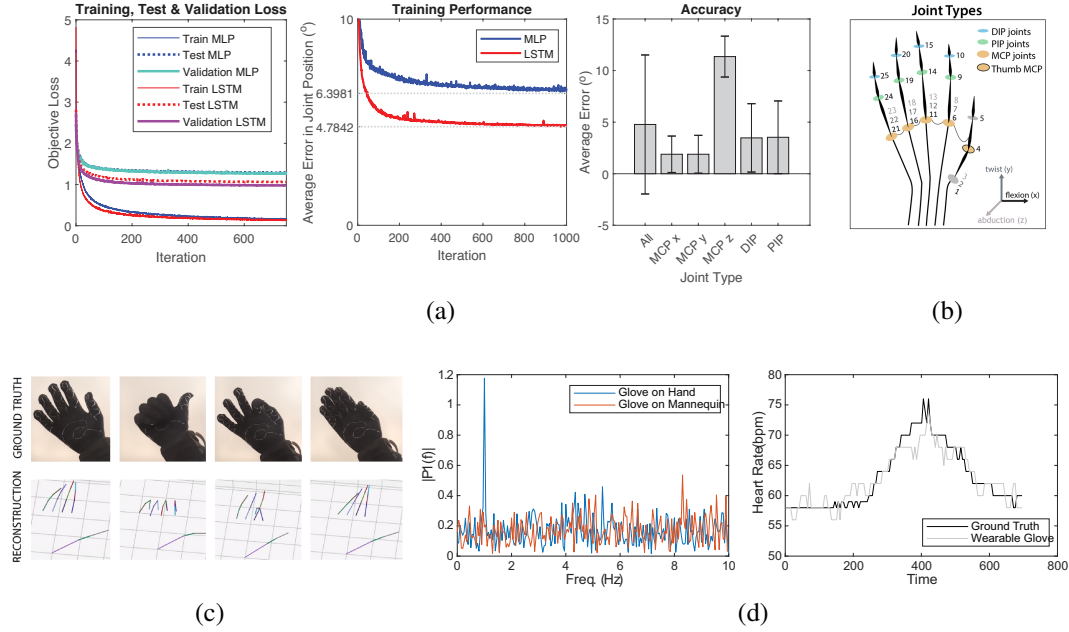


Figure 6-17: a) Training, test and validation data for pose estimation and accuracy of the reconstruction across the different joint types; b) The different joints and their types; c) Different hand poses when wearing the glove and corresponding visualizations of the the reconstructed positions, along with our kinematic and biological hand model; d) Measured heart rate spectrum analysis, and comparison of our measured heart rates to those of a Fitbit.

is distributed across each particular joint; we further include a summary of how that error is distributed among the different biological joint classes. As can be seen, the majority of the joint errors are exceptionally low, even much lower than 5 degree on average. The primary exception is the abduction of the fingers, which has 10 – 15 degree average error. This is likely primarily because MemGlove has poor sensitivity to strain response caused by lateral positional motions, due to the structure of the knit. The large standard deviations (and a nontrivial contribution of the error) are caused from spurious frame captures from the leap motion which occasionally invert the entire hand — we obviously cannot model this, leading to a heavy tail in our reconstruction distribution. Still, even with these combined errors, MemGlove demonstrates superior overall average reconstruction.

Heart Rate In this demonstration, we show MemGlove’s ability to infer the wearer’s heartbeat. For this one application, we did not use a neural network estimator, but rather used standard signal processing tools and techniques in order to capture the periodic response of

Per Joint Reconstruction Errors					
Joint #	Mean Error	Std. Deviation of Error	Joint #	Mean Error	Std. Deviation of Error
1	1.33	2.14	14	3.13	6.60
2	6.50	13.11	15	4.98	9.88
3	17.28	27.54	16	2.07	3.86
4	3.62	6.05	17	0.36	0.73
5	4.57	7.95	18	5.12	47.63
6	2.28	3.99	19	3.29	6.54
7	1.05	1.59	20	5.35	9.24
8	18.00	82.56	21	1.90	3.50
9	3.85	7.32	22	1.27	2.22
10	6.09	11.23	23	12.92	24.64
11	1.92	4.16	24	3.85	7.73
12	0.33	0.71	25	5.08	8.54
13	3.45	47.46			

Table 6.5: Mean and standard deviations of joint error for LSTM-based reconstruction, in degrees.

the heartbeat.

Although the most significant changes to the measured resistance of the glove correspond to postural changes, there are small changes that correspond to slight expansion/contractions resulting from the heart rate. Due to the many data channels in the glove, it is possible to average the frequency component over all sensors, filter, and extract the heart rate, even though the effective induced strains are small.

To obtain the heart rate, the raw data from the glove held in a static pose is windowed over a 5 second period. For each of the resistive channels on the glove, the signal is converted into the frequency domain using a Fast Fourier Transform (FFT) and applying a low-pass filter, over which the responses from all sensors combined. By computing the power density of FFT and performing peak detection, the heart rate can be estimated. Due to the 16 Hz sampling rate of our current glove, the precision of the heart rate is only to the nearest 1 or 2 beats per minute (bpm); still, this is sufficient for many medical and fitness heart rate monitoring applications and is competitive with consumer-grade monitors, such as Fitbit.

Data was collected with the user holding their hand in a neutral, approximately still pose. Approximately 10 minutes of data was captured during which the user performed light exercise (walking in place) to vary their heart rate. A wearable Fitbit heart rate monitor

which has a precision of approximately 2 bpm was also worn to provide ground truth. To ensure that the glove is responding to only the heart rate measurements and not to, say, small changes in hand kinematics, the glove was placed on a mannequin hand (which has no pulse) as a control, and the experiment was repeated, ruling out this possibility.

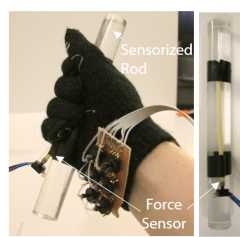
Fig. 6-17c shows an example of the power spectrum obtained from the FFT. There is a clear peak at around 1Hz, corresponding to a heart rate of approximately 60bpm. A comparison between the heart rate determined by the glove, and that measured by the Fitbit for the 10 minute period is also shown. Overall there is a strong agreement between the two devices' estimates.

Exteroceptive Sensing In the following section we demonstrate the ability of MemGlove to sense external stimuli as a wearer interacts with the environment.

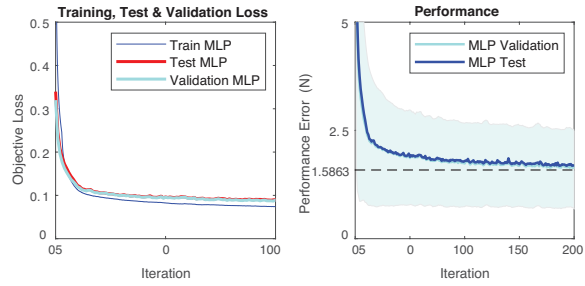
Holding Force As a first, straightforward demonstration of exteroceptive sensing, we show how MemGlove can be used to estimate the force with which a wearer is grasping a target object. This is a straightforward task, as the data captured from the fluidic pressure sensors naturally correlates with force. Therefore, for this task, we only use the fluidic pressure data. Holding force is a useful statistic, as it can be an indicator of health and recovery from several medical conditions, including muscular atrophy. To capture ground-truth data, a calibrated force sensing resistor (FSR) was attached to a rigid plastic bar. The bar was held and repeatedly grasped with different forces with data recorded from the fluidic sensors. Approximately 2 minutes of test data was obtained.

Fig. 6-18b shows the experimental setup, the test and training loss, and the performance using the trained networks. Because the grasps were changed rapidly throughout data collection, training data varied rapidly, and so only an MLP was considered here. The trained network shows an average error of 1.74 N, which is low in comparison to grasping forces which can range from 0-80 N.

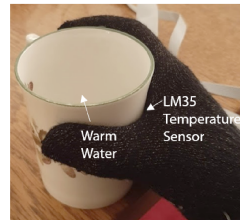
Object Stiffness In a slightly more complex but similar task, we seek to measure the stiffness of grasped elastic objects. As the pressure tubes are also compliant, different object stiffnesses induce different deformations on the tubes at similar grasp strength, which



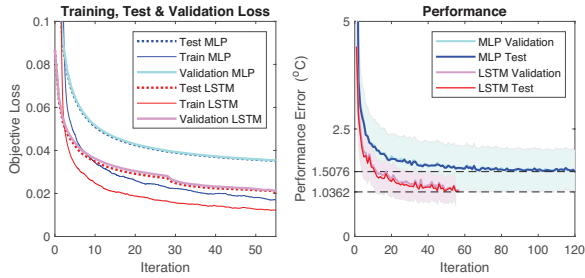
(a)



(b)



(c)



(d)

Figure 6-18: a) Holding Force Experiments; b) Evolution of the loss function for training, test, and validation sets, as well as real-world performance (in terms of average error) for Holding force estimation; c) Temperature experiments; d) Evolution of the loss function for training, test, and validation sets, as well as real-world performance (in terms of average error) for Holding force estimation.

leads to different pressure readings. Similar to the holding force experiments, this task also only relies on the pressure data readings. To investigate the ability of MemGlove to infer grasped material stiffness, 6 geometrically identical cubes of side 25mm were fabricated from materials of varying stiffness including 3D printed materials, EcoFlex silicones, and foams. Each object was then grasped with the same hand pose and similar grasp strength. Time series data from the two sensors was obtained for 6 objects, all with 10 repetitions.

Fig. 6-19a shows the training, validation, and test losses as well as classification probabilities, alongside a visual demonstrate of the experiment. As expected, since this is mostly static task, the MLP outperformed the LSTM, but both networks approached 100% classification rates.

Object Temperature Sensing Although not a mechanical task, both our pressure sensors and our resistive sensors have the ability to inform estimates of grasped object temperature. Changes in temperature have a direct correlation with the resistivity of the knit glove; further, the fluidic pressure sensor readings are directly correlated temperature through the ideal gas law. Both measurements are used to provide an estimate of the temperature of grasped objects. Although both sensor modalities show some hysteresis in raw signal, the combination provides increasingly unique signatures for temperature change offering the potential of improved temperature detection. Further, the LSTM model provides smoothing and increased accuracy by considering signal history, resulting in no noticeable hysteresis or lag in prediction.

In order to obtain temperature training data, a thin coffee cup was equipped with an LM35 temperature sensor and hot water added to the cup. As the water cooled, the cup was held with a constant grasp. Data was recorded from the glove and ground truth obtained from the temperature sensor. This process was repeated 5 times with the cooling process accelerated *via* the addition of small amounts of cool water. Fig. 6-18b shows the experimental setup, the training and test loss of the trained networks, and the performance of the trained network in terms of estimated absolute temperature difference. Both the MLP and LSTM network perform well, but since temperature readings vary smoothly over time, the LSTM network outperforms MLP, achieving an average temperature estimation error of just over $1.0^{\circ}C$.

We further use this experiment to examine the benefit of using two sensing modalities instead of one. Here, we examine the mean error and standard deviation of the minimum mean test error recorded during training, averaged over three training runs. Our experiments demonstrate that pure resistive knit sensing provides an error of $0.96^{\circ}C \pm 0.02^{\circ}C$, fluidic pressure sensing provides an error of $1.2^{\circ}C \pm 0.03^{\circ}C$, while both modalities combined provides an error of only $0.92^{\circ}C \pm 0.01^{\circ}C$.

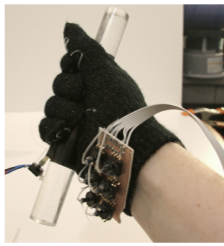
Material Conductibility The resistive nature of the glove allows for identification of the conductibility of grasped materials. When the glove comes in contact with a conductive material, the material will act as a conductive path, changing the resistance profile of the glove. Such capability could have applications, for example, in sorting in recycling units, for separating metallic and non-metallic recyclables. We treat this as a binary detection (conductive or non-conductive) task and do not attempt to estimate the conductability of the grasped object. As only the resistive sensing is sensitive to conductibility, it was the only sensing modality used for this task.

Data was collected for five different objects (box, pen, small ball, tube and soft toy). In one training data set they were covered in aluminium foil to make them conductive, and in the other, they were not, remaining non-conductive. Each object was grasped 5 times in both configurations.

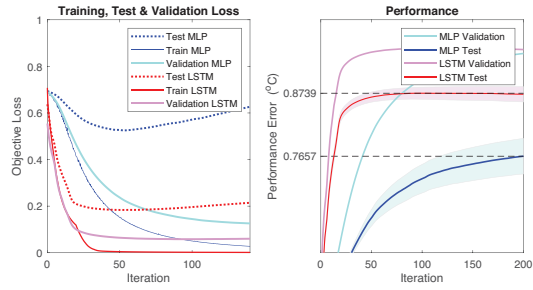
This is a particularly challenging task, as the changing grasp interferes with the readings caused by the conductance changes. Ultimately, our networks are able to achieve 84.7% accuracy in classification between non-conductive and conductive objects — significantly better than random (Figure 6-19). This demonstrates that conductive pathways can be detected by the glove despite changing postures.

Complex Task Classification Tasks

In this section, we demonstrate MemGlove on two more complex tasks, one static, and one dynamic, in the form of grasped object classification and handwriting classification. For both of these complex tasks, both resistive and pressure sensors were needed and used in order to achieve the accuracies reported.



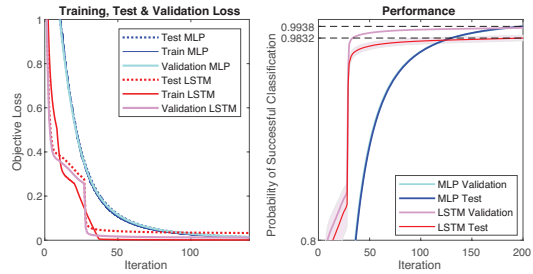
(a)



(b)



(c)



(d)

Figure 6-19: a) Conductability Experiments; b) Evolution of the loss function for training, test, and validation sets, as well as real-world performance (in terms of average error) for conductability estimation; c) Stiffness experiments; d) Evolution of the loss function for training, test, and validation sets, as well as real-world performance (in terms of average error) for stiffness estimation.

Object Identification: Static Task To demonstrate the combined capabilities of MemGlove’s sensors on a complex classification task, we seek to identify classification of the natural grasps of 30 objects. This reflects a scenario in which the glove could be used to assist with task identification through identification of a task-relevant, held item. The 30 household items (Fig. 6-20a) selected were chosen to test the capabilities of the glove. These objects are of variable stiffness, shape, size, density, and material. To challenge the ability of the glove to differentiate similar objects, we included a number of objects with the same shape, but with different material properties, *e.g.* metallic versus plastic, or soft versus hard.

The training data set was generated by grasping each object 10 times and recording time series data from both sensors after the object was grasped. Fig. 6-20b shows the training and test loss and the performance of the classifier. As expected in this static task, the MLP slightly outperforms the LSTM network and achieves close to 100% classification success.

Handwriting Identification: Dynamic Task As a final, challenging demonstration of MemGlove’s capabilities, we investigate its ability to disambiguate handwritten letters. This task not only highlights the glove’s precision (as handwriting leads to only small changes in posture and pressure distribution within the hand), but also presents a practical use case, providing another mode as an input device (*e.g.* for giving regular pens the capabilities of smart pens, such as the LiveScribe). It is also a task in which there is large noise in the dataset; no handwritten letter is ever the same twice.

For each of the first 10 letters in the English language alphabet (*a* through *j*), data from the sensors when the user wrote each letter was recorded, with 15 samples collected for each letter.

Fig. 6-20 shows the experimental results, showing that each letter has a distinct enough signature to allow for accurate classification. The MLP and LSTM both providing classification rates in the mid 80% range, compared to the 10% accuracy of true random.

6.2.4 Related Work

Our work builds on previous results in computational modeling and learning-based sensing hardware.

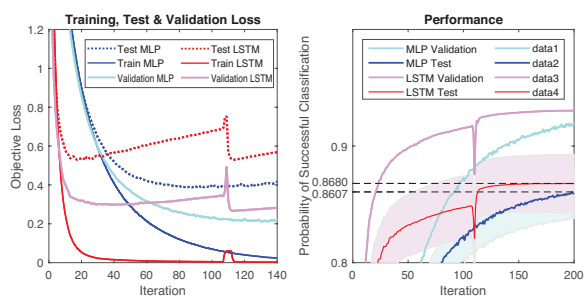
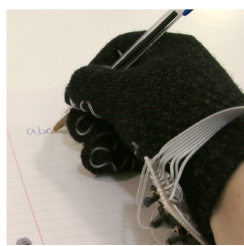
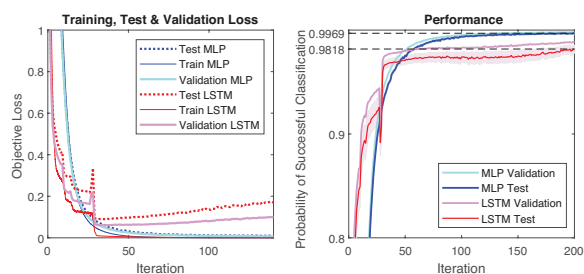


Figure 6-20: a) Our object classification dataset, with associated grasps. Evolution of the loss function for training, test, and validation sets, as well as real-world performance (in terms of correct classification probability) for a) Grasped object classification; b) Handwriting recognition classification.

Soft skins and soft electronics provide a promising means for achieving sensors for soft robots. Realization of such technologies has recently become more feasible, given low cost electronics and wireless communication devices [26, 164]. These leverage a number of different sensing modalities including capacitive [25, 54], resistive [68, 87, 30, 10], knitted resistive sensors [154, 109] and stretchable electronics [92, 155]. Many wearable devices focus on providing tactile information or hand-pose information, advancing from previous work where the main focus was on the detection of inertial forces [50] or acceleration [114]. A key limitation of many of these sensors is the inability to decouple strain and normal force responses. In addition, the sensors are often challenging to fabricate, making it difficult to integrate multiple modalities of sensing.

6.2.5 Outlook: Sensor Fabrication

We presented a novel wearable glove design that combines knitted resistive sensing and fluidic pressure sensing with machine learning-powered reasoning for advanced real-world, real-time inference applications. We have demonstrated that MemGlove not only provides state-of-the-art hand pose reconstruction, but also unlocks applications in heart rate, temperature, force, stiffness, and conductivity sensing, as well as static and dynamic task classification. The range of different stimuli which the glove can reason about is significant, and outperforms previous work. Further, our fiber-based glove has low material cost and is exceptionally simple to fabricate using COTS components and is amenable to semi-automated or fully automated machine textile manufacturing methods.

MemGlove currently possesses a few limitations that would be worth investigating in future work. First, MemGlove has currently only been tested in single domain inference tasks. This means we can infer, *e.g.* hand pose, temperature, or holding force independently, but not jointly. Since similar changes in signal are used to learn each output label, we would likely need to improve the design or augment the glove with sensors and a more sophisticated reasoning pipeline that can better disambiguate the cause of the signal. Second, the temporal and signal resolution of MemGlove is limited by our choice of electronic hardware. Better electronics with higher resolution could be incorporated at additional monetary cost, bulk

(affecting comfort), and complexity. It would be interesting to investigate these trade offs. Third, although our sensor architecture is effective and was chosen strategically, there is no evidence that our sensor placement is optimal. Computational or data-driven methods for optimizing sensor placement, as described earlier in this chapter, could further improve performance, although that method would have to be adapted to work with the sensor models presented here. Fourth, it would be helpful to add additional sensors to improve accuracy and precision of the abduction estimation. Finally, as this glove is a fiber-based wearable, it would be useful if its connections could be modified to be resilient to machine washing.

Given its accuracy, robustness, and versatility, we believe MemGlove can unlock exciting applications in many fields. Soft robotic hands can use a modified version of our sensor architecture and machine learning pipeline for better proprioception and tactile manipulation. Virtual and mixed reality games and other applications could benefit from more immersive gestural input. Further, from heart rate sensing to hand force and motion monitoring, we believe MemGlove has many exciting opportunities in application to the medical field. We look forward to further investigation and application of our work in these and other fields.

Co-learning algorithms like the one presented in this chapter as well as Chapter 5, when combined with fabrication methods for building soft robot sensor networks, provide yet another robotic organ system to be co-designed — a nervous system, realized through a densely sensorized robot skin. We are hopeful that physical considerations will dovetail with algorithmic capabilities in the future to produce a future in which cyber and physical will be codeveloped in one unified cyberphysical framework.

Chapter 7

Conclusion

Computational robot co-design algorithms and systems have the potential to be a force multiplier in the field of robotics at large, creating tools that make it easier for researchers, hobbyists, and industry professionals to ideate, specify, optimize, and fabricate customized, adept, task-driven robots. Computational tools have ability to shorten design cycles and thereby revolutionize the way we think about robots in our everyday lives: their ubiquity and their utility, their diversity and their dexterity.

The ultimate vision of a world in which computational co-design takes a central world in robotics, however, is not one in which computers singularly rule the day. We have, in this thesis, posited workflows by which a computer completely takes over a design process and co-designs all relevant aspects of a robot simultaneously. But the ways in which robots are designed, even when using computation, will be far more nuanced than that. A human designer will (likely gesturally, rather than mathematically) specify tasks, constraints, and preferences, defining a design space and a high-level goal for their robot; a computer will take over, exploring designs, playing with possibilities, generating a large collection of possibilities to be fed back to the user. As the user sees these results, they refine their design requirements, goals, and preferences, and feed this back to the computer, and the dance continues until convergence. This is the true alternating optimization algorithm that will lie in the core of computational robotics. As such, we see the methods presented in this paper as tools to be integrated into a system like Interactive Robogami, rather than ones to replace it. These capabilities will be modularly swapped as users tackle robot designs across

different domains, but ultimately interacting gracefully through a unified stack-based API, and, ideally, an integrated CAD suite.

The impact of computational robot design promises to be twofold. First, by accelerating design workflows, computational robot design promises to hasten design iteration, allowing robot developers to go deeper during each phase of the design process. Robots will be designed and built more quickly and specified to more detail and with more guarantees. This will make it easier and faster to build robots and boost their performance, hopefully one day rivalling or surpassing the performance of organisms in nature. Second, the ability to explore the space of robot designs algorithmically will make it easier to discover new robot morphologies and behaviors that are more esoteric, riskier, and novel, boosting the creative output of robot engineers. We emphasize that the combination of form and function is a Cartesian product, and we will see more surprising output both in design and performance space. Through this boost in design exploration will we achieve the type of robotic diversity that can one day start to approach the organismic diversity of nature.

7.1 The Future Of The Future of Robot Design

This thesis set out to introduce methods that will provide a first step in revolutionizing the way we design robots; in this section, we turn our attention to what the future of this field will hold, what are the important, big, open questions, and what we as a community need to focus on.

7.1.1 Full Co-Design Algorithms

This thesis presented algorithms which co-designed robots over bodies and brains — shape, materials, and sensors on the physical side, and controllers, proprioceptive models, and more general task classifiers and regressors. Going back to our organ system analogy, this allowed us to co-optimize robot brains with skeletal, integumentary, and nervous systems — but only one or two of those physical systems at a time. This presents two challenges.

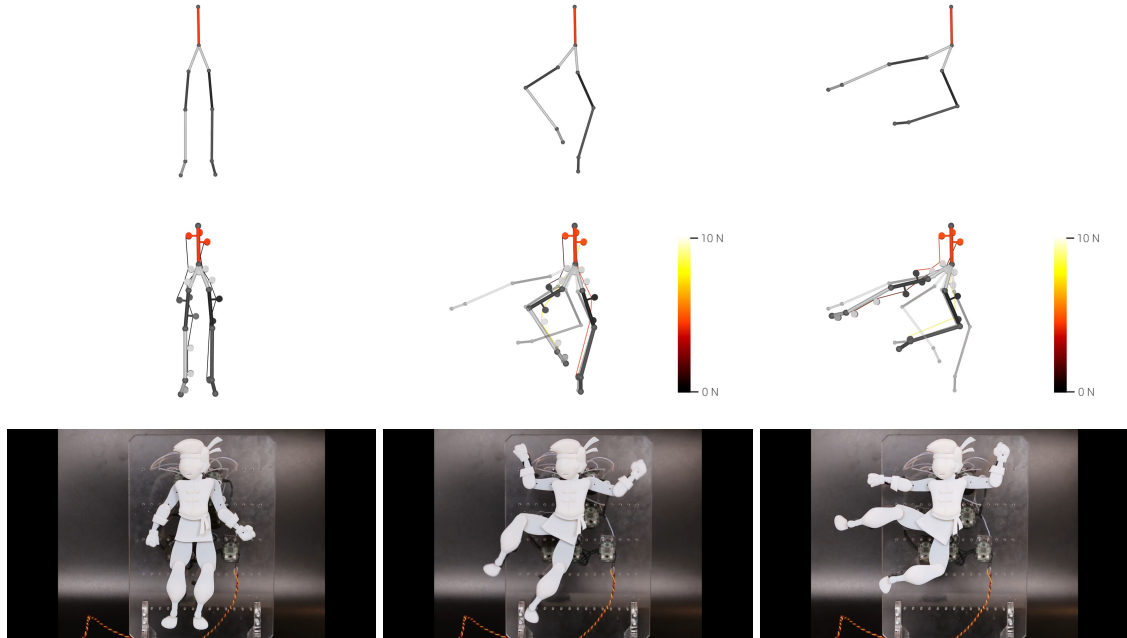


Figure 7-1: Row 1: The rest and target poses for the Fighter character; Row 2: The cable network along with realized simulated poses; Row 3: The fabricated character reaching the desired poses.

Actuator Co-Design The first challenge, which is quite concrete, is to fill in one obvious gap — the co-design of robot actuators and dynamical behavior, thus co-optimizing robots’ brains and their *de facto* muscular systems. Although we touched upon actuator selection in Chapter 3, we did not deal with placement or soft actuator structure, and this remains an open challenge.

In work related to this thesis [96], we co-optimized characters’ cable-driven actuation networks and kinematic controllers. In that work, we presented a method that designed a cable network, aiding the artist and hobbyist with the design of cable-driven kinematic chains and tress that closely match a set of specified poses or keyframes when actuated. While two cables per rotational degree of freedom is sufficient for this task, it is more than necessary for many desired motion subspaces. We placed torsional springs at the joints to allow for restoring torques, transforming unidirectional cable-pulling into a reversible action, and we co-optimized the number and placement of cables together with control forces needed to drive the mechanical hierarchies.

Our solution to this problem was to take a two-step approach, where we first identified the

topology of a network by solving a complex statics problem in a scenario where hundreds or thousands of cables are provided as potential input. Because solving for the forces necessary to hit target keyposes then reduces to solving a linear system, we can solve for sparse solutions over cable input forces in a manner similar to the LASSO problem. Similar to chapter 6, where solving for important input strains implicitly selected hardware for sensing, this process of solving for important input forces implicitly selects important hardware for actuation. This first step reduces the number of cables down from hundreds or thousands to a handful. In the second part of the optimization algorithm, the network is refined by parameterizing routing points, and reducing the network and control forces further if possible in a continuous co-optimization process. Resulting designs were fabricated. Figure 7-1 shows the input target poses, optimized cables and actuation patterns, and fabricated realized version for the example “Fighter” character, while Figs. 7-2 and 7-3 demonstrate a robot gripper and a robot hand, optimized for multiple target poses.

Although this work omitted dynamics since it only deals with the kinematic, quasistatic case, this provides one promising avenue of optimizing robot actuator structure. Other methods have looked at similar ideas for fully soft characters [14, 93], but all thus far have avoided the challenge of contact. One possible promising direction moving forward is to apply a neuron structure similar to the sensor learning method presented in Chapter 6 to the output of a control network, and wrap its optimization in a reinforcement learning framework.

A Unifying Theory The second challenge toward full co-design is more general. If we ever wish to realize the dream of full organism co-optimization, we need ways of reasoning about *all* physical aspects of a robot simultaneously. And as, robots become more sophisticated and become hybrid entities which include both rigid and soft elements, this may even mean reasoning about aspects of rigid and soft robots simultaneously. But, for this to be possible, we will need computational tools which enable such robots to be modeled.

7.1.2 More On The Horizon

We now more briefly summarize other important topics that will shape the future of this field.

Differentiable Multiphysics In order to co-design robots with varying types of physical components and more exotic phenomena, fast, accurate, and differentiable multiphysics simulation engines will be necessary. This is true for the same reason that such requirements were necessary for soft robot co-optimization. While these first two requirements (to varying degrees) are satisfied by certain commercial simulation packages, there exists little research on multiphysics simulators in which gradient computation is available. This problem is complicated by interrelated two computational hurdles — the fact that different phenomena operate on different *time* scales; and the fact that different phenomena operate on different *energy* scales (thus affecting gradient scales).

The last several years have yielded an amazing proliferation of differentiable simulators, bringing us differentiability in the domains of soft body simulation [66], rigid body simulation [32], fluid simulation [41], and cloth simulation [86], to name a few. Frameworks by which to unify these discrete domains will have a profound impact in promoting more diverse robot typologies and solving challenging problems in their control, design, and system identification.

Multi-task Design Most physical tasks performed by humans or other animals require being adept at multiple skills. For example, an octopus hunting prey must be proficient at swimming and tactile manipulation; a duck in migration must be capable of locomotion through land, air, and sea; a human hurdler must be fast at running along bends and straightaways as well as jumping. If such animals were only capable of single behaviors, they would not be very successful. But our philosophy toward computational robot design has been focused toward forms that are capable of only a single task. Contrast simple pick-and-place manipulators with the rich dexterity of motion found in the tentacles of an octopus, which are capable of aiding in swimming, grasping, tool manipulation, and crawling through tight spaces. Or, compare the legs of simple walking robots with the claws

of a chameleon which enable running, climbing, and balancing on narrow tree branches.

An approach of directly optimizing the Pareto front of designs would greatly ease human workflows for multi-task design, and move us toward robots which are capable in more complex, unstructured environments, enabling robots which can seamlessly transition between subtasks in larger, higher-level tasks such as cooking or manufacturing. Further, such a capability could lead to robots which could organically adapt to novel situations.

Sim-To-Real For *Any* Robot The analysis of sim-to-real transfer is a burgeoning field, founded on the radical notion that the robots that exist in simulation should actually predict the behavior of their fabricated counterparts. Despite being an obvious desired outcome of robot simulators and control algorithms, it is only until recently that researchers are starting to develop robust algorithms to guarantee this transfer [142, 167, 2]. Still, these algorithms are currently only deployable on a single robot morphology. If researchers change the robot’s physical design, there is no guarantee that designs will transfer.

We wish to realize a dream of robot design where completely novel, never-before-seen robots can efficiently be designed and fabricated, in which they have instant functionality as soon as they are removed from their digital machining device. But such capability will require more robust sim-to-real transfer methods which leverage data from a collection of well-understood physical robots to bootstrap or ideally eliminate the computational transfer process for a novel robot. Sim-to-real transfer for a single robot is difficult and fabrication is expensive; while it may be a while before this challenge can be efficiently solved, its existence only underscore while it is so important that useful computational tools be developed sooner.

Tools For Users, Not Just For Researchers The field of robot co-design has exploded in research prevalence over the last decade. But you wouldn’t know it if you weren’t immersed in this subfield. In fact, very few computational co-design tools have made their way into any easy-to-use available software, open-source or commercial. In short, the field has become an ivory tower.

If we want robot co-design tools to part of real-world robot design cycles, we need to

get these tools in front of real practitioners. This is important for several reasons. First, it will allow us to truly assess, ethnographically and statistically, if these tools do indeed live up to their promise of accelerating and facilitating real-world robot design cycles. Secondly, if these algorithms deliver on their promise, their successes will translate into increased focus (and scrutiny) of the field, both of which will bolster co-design as a field. Third, putting these algorithms in front of end-users will give incentive to pay attention to the human-computer interaction aspect of robot co-design, something which is oft ignored in the field, but important for real-world adoption. Finally, it will give researchers a better connection with the real-world needs of robot builders, and allow us to focus and scope out the most important future research problems that will have maximal impact.

7.2 Lessons Learned

The successes in this thesis, as in any research endeavor, were only made possible by many more failures. In order to steer readers of this thesis away from our mistakes and toward what works, we now briefly summarize some of the biggest lessons learned during the development of this thesis.

Avoid Constrained Optimization For Soft Robot Pose It is extremely tempting to try to directly apply the methods from Chapter 3 in the soft robotics domain. One might imagine specifying target poses that the soft robot should reach, and solving a trajectory optimization problem in order to plan a soft robot’s motion. However, this is a poor strategy for (at least) two reasons. First, allowing each MPM particle’s or FEM node’s position to be a variable in an optimization problem can lead to elements shifting past one another, which can lead to element inversions or tearing. Second, it is hard to even specify rest or target goal poses. Since soft robots deform under gravity or contact, static simulations are needed to determine what are even reasonable, feasible rest poses. While it is possible these problems could be alleviated by more carefully specifying constraints and objectives, that remains an open research question, and readers should take note of these pitfalls. We would advise sticking to unconstrained optimization methods in this regime.

Contact Modeling Sim-To-Real Transfer For Soft Robotics Is Difficult Readers may notice that physical instantiations of co-optimized soft robots are missing from this thesis. While we did attempt to fabricate these robots in various media, we had difficulty getting the physical robots to match the simulated robots in the interesting cases of problems with contact. Real-world friction and contact damping is still very difficult to match in simulation, especially because of the sharp motion bifurcations that can be caused by phenomena such as slipping. While some successes at alleviating these problems have been demonstrated in the rigid robotics arena through domain randomization techniques, these methods are data hungry and require even faster simulation than we demonstrated in this thesis.

Fast Simulation Is Important Optimization and learning algorithms can both be iteration-hungry processes. Some of the results demonstrated in this thesis took over a day to generate. It is important that simulations, which can be significant bottlenecks of these processes, be as fast as possible. This is true for generating results, but especially true during the research experimentation phase of a project. As a researcher, you want to be able to see if an algorithm is going to work quickly, and if not, rapidly make changes to the method. If algorithms which rely on simulation are too slow, research progress also slows considerably. Even if approximations must be made in order to accelerate the simulation (such as larger time steps, less realistic material models, *etc.*), this is a worthwhile sacrifice to make during the algorithmic development phase of a research project. It is also worth taking the time (even at the expense of research hours) to optimize the simulation’s performance. Make your simulation fast and get the algorithm to work on anything; then refine it.

Differentiability Saves The Day Many of the algorithms in this paper are a direct consequence of differentiability. We cannot understate how useful and powerful differentiability is when it comes to design optimization. Gradient-based methods converge much faster than gradient-free methods such as evolutionary algorithms (at least when dealing with continuous variables) and are much, much less hardware intensive. Co-optimization algorithms in some cases become an obvious consequence of this differentiability. We do note, however, that readers, in their own experiments, should be cautious about gradient-scaling

when dealing with first order optimization methods.

Don't Throw Away Your Data Traditional optimization algorithms throw away all the data generated by a simulation, retaining only gradient information in order to improve the design. But, data generated as a side-effect of this gradient computation can be important in its own right, as we demonstrated in Chapter 5. Such learning-in-the-loop algorithms that leverage trajectory data in concurrence with model gradients will hopefully play an important role in the future of AI, providing means of leveraging learning-based methods in model-based workflows, and vice versa.

Dual Formulations Can Vastly (And Elegantly) Simplify Design Problems In Chapter 3, we saw how minimizing over actuation can implicitly select for part of a design, namely, the actuator. In Chapter 6, we saw how dropping out unimportant inputs can implicitly determine where sensors should be placed. In the conclusion, we saw a brief example transforming a pose reconstruction problem to a sparse selection problem. In all of these cases, a dual formulation was used — one where morphological designs can be solved for *implicitly* through optimizing cyberphysical performance. It is vital to always think of these aspects a robot as coupled, as solving a robot's cyber design can implicitly inform the physical design, and vice versa, thus vastly simplifying search problems.

Learning Is The Future Over the course of the development of this thesis, deep learning has taken off in a huge way; to both overstate and understate the truth, machine learning finally works. This has been the result of many breakthroughs in the field over several decades, both in deep and classical statistical methods. With this enhanced computational power comes great opportunity, and we have seen many examples in which learning-based approaches have led researchers to solve problems previously out of reach [136, 133, 168, 163]. Still, the use of machine learning as part of the robot design pipeline is relatively new — meaning it still houses so much opportunity to be mined in terms of exciting, efficient co-design algorithms. Learning-based methods will form a new, exciting frontier in robot co-design, one which should be aggressively pursued.

7.3 Closing Remarks

In this thesis, we have presented algorithms for co-optimizing and co-learning optimal robot designs, and provided a path toward fabricating these robots. We have demonstrated the co-design of robots over geometry, inertial parameters, materiality, open- and closed-loop control, proprioceptive models, sensor design, and higher-level task planning. Further, we have provided a roadmap toward the fabrication of some of these classes of devices.

Looking forward, we are hopeful that this combination of stack-like thinking for robot design and computational algorithms for realizing designs will provide a framework for the field of computational robot design. Computational design algorithms, and the system that surrounds them, have the potential to revolutionize the way we design robots, and unlock a future of robots that are more diverse, faster to design, more complex, and, most importantly, more capable and useful in our daily lives.

The future of the future of robot design is ripe with areas for innovation. We hope that the design of new processes which automate the virtual design and digital fabrication of custom robots will unlock the proliferation of custom robots and usher in a new age of robot design and experimentation.

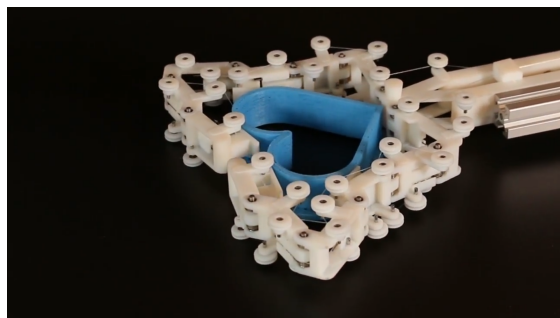
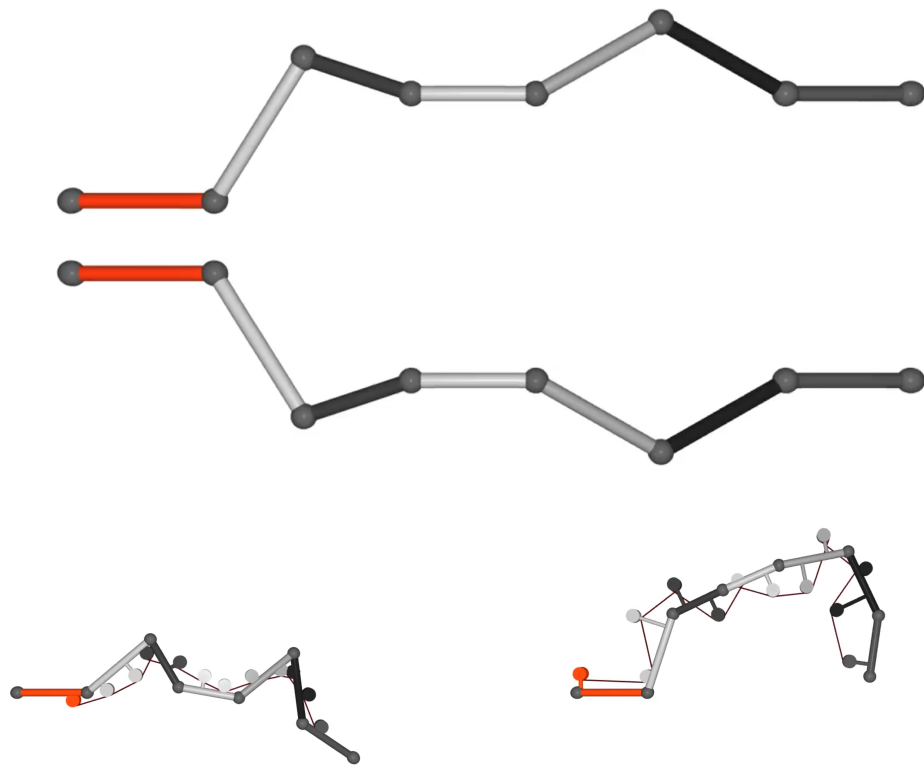


Figure 7-2: Row 1: The rest pose of the gripper; Row 2: The cable network along with realized simulated poses; Row 3: The fabricated gripper reaching the desired poses in order to pick up two differently shaped objects.

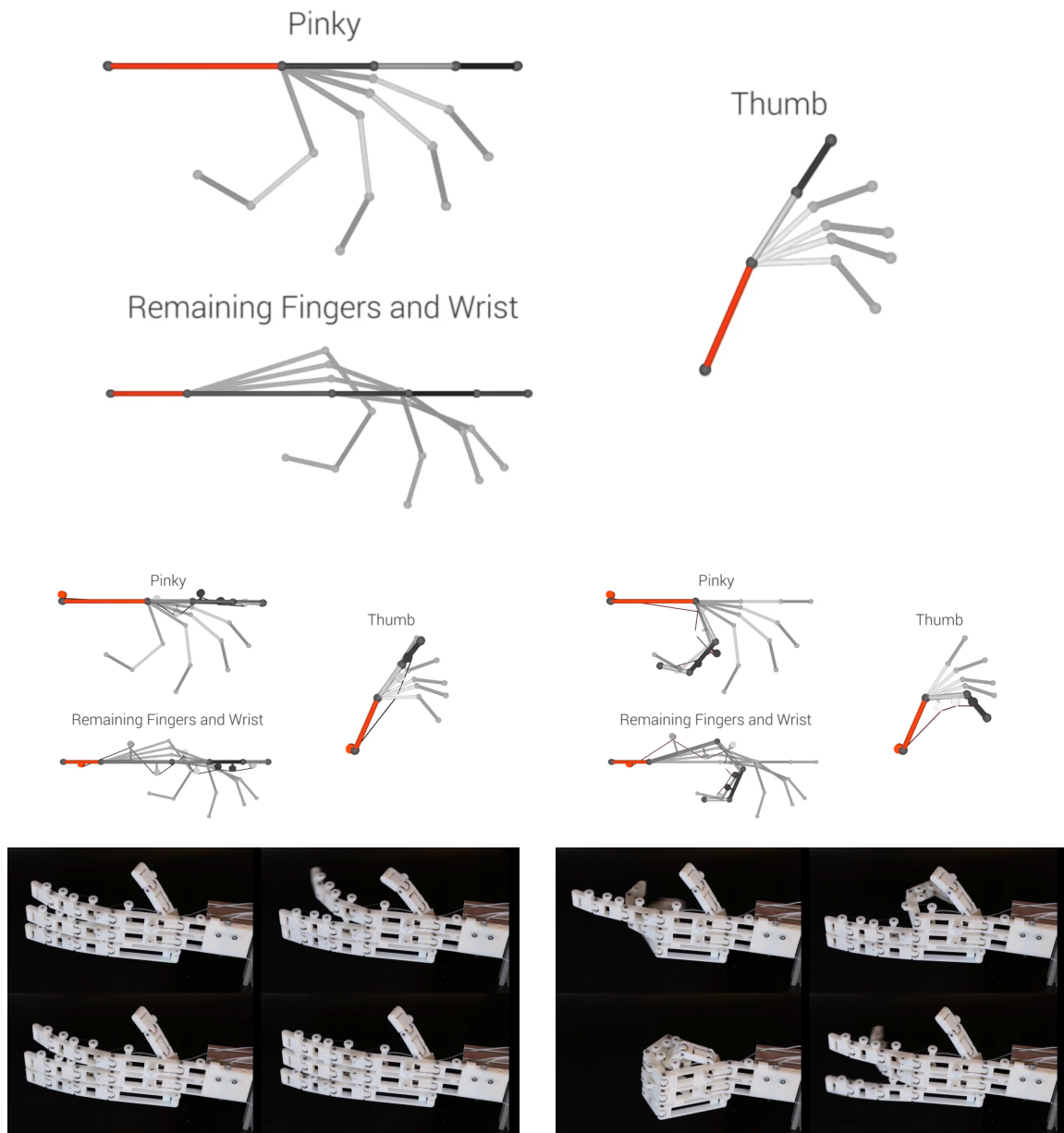


Figure 7-3: Row 1: The rest pose of the hand, along with several desired target poses; Row 2: The cable network along with a subset of the realized simulated poses; Row 3: The fabricated hand reaching various combinations of desired poses in order to create a variety of finger poses.

Appendix A

A.1 Soft Robotics API Listings

Listing 1 The user-defined Observer class, which converts high-dimensional robot state to a compact representation.

```
class Observer:

    def get_dim(self):
        """
        Return the total dimension
        of the observation space
        as an int.
        """
        raise NotImplementedError

    def observation(self, state):
        """
        Return an observation
        vector of size self.get_dim(),
        computed from the state.
        """
        raise NotImplementedError

    def get_observation_range(self):
        """
        Return the element-wise
        maximum and minimum value of the
        observation space
        (as a pair of vectors).
        """
        raise NotImplementedError
```

Listing 2 The user-defined Controller class, which computes an actuation signal from an observation.

```
class Controller:

    def get_controller_function(self):
        """
        Return a function that maps an
        input observation (vector)
        to an output actuation signal (vector).
        """
        raise NotImplementedError
```

Listing 3 The user-defined Actuator class, which converts actuation signal to output force or pressure.

```
class Actuator:

    def __init__(self, particles, is_force_actuator):
        """The user must pass in an iterable
        of ints corresponding to the sequence
        of particles to be actuated.
        In some cases, order may matter (cables),
        in others order may not matter (fluidic actuators).
        """
        self._particles = particles

    def get_actuation(self, control, state):
        """
        Given a control input signal normalized between -1 and 1
        and the state of the simulation, return a
        force vector (n * dim) or a pressure tensor (n * dim * dim).
        (NOTE: Not all actuators need be state-dependent.)
        Whether this is a force or a stress-based actuator
        is inferred from the dimensionality of the output.
        """
        raise NotImplementedError
```

Listing 4 The user-defined Optimizer class, which optimizes a robot given user-defined objectives and constraints.

```
class Optimizer:

    """
    assume that objectives and constraints return a pair of a value
    and its corresponding gradient.
    Our API provides helper functions to compute these gradients.
    """

    def __init__(self, trainables, visualizer=None, **kwargs):
        self._visualizer = visualize
        self._trainables = trainables
        self._equality_constraints = []
        self._inequality_constraints = []

    def setObjective(self, objective):
        """
        objective is the function to be minimized.
        """
        self._objective = objective

    def addInequalityConstraint(self, constraint):
        """
        constraint is a function  $f$  interpreted in the form
         $f(\text{memo}) \leq 0$ , intended to be satisfied
        during optimization.
        """
        self._inequality_constraints.append(constraint)

    def addEqualityConstraint(self, constraint):
        """
        constraint is a function  $f$  interpreted in the form
         $f(\text{memo}) = 0$ , intended to be satisfied
        during optimization.
        """
        self._equality_constraints.append(constraint)

    def optimize(self):
        """
        Optimize the objective function, subject to the constraints.
        Entirely up to the user to define. Algorithms may be unconstrained
        and ignore constraints at the implementer's discretion.
        """
        raise NotImplementedError
```

A.2 A robot optimization in 50 lines of Python code

```
#Some imports we'll want, including our building blocks and numpy and tensorflow
import tensorflow as tf
import numpy as np
from optimizers.gradient_descent_optimizer import *
from actuators.axis_aligned_pressure_actuator import *
from observers.kmeans_observer import *
from controllers.tanh_nn_controller import *
from controllable_mechanism import ControllableMechanism
from group import *
from math_tools import *
from simulation import get_bounding_box_bc

#Set up our TF environment
sess_config = tf.ConfigProto(allow_soft_placement=True)
sess_config.gpu_options.allow_growth = True
sess_config.gpu_options.per_process_gpu_memory_fraction = 0.4

with tf.Session(config=sess_config) as sess:

    #Some constants we'll use.
    sample_density = 60 #Each group will be 60 by 60 particles.

    time, dt = 4.0, 0.005
    num_steps = int(time / dt)

    #The geometric parameters of our robot
    leg_width, leg_height, body_height, body_width = 0.1, 0.1, 0.1, 0.3
    dim = 2 #the dimension of the problem

    #Compose this robot from block groups of particles - what are their locs and sizes?
    group_offsets = [(0, 0), (leg_width / 2.0, 0), (0, leg_height),
                     (leg_width, leg_height),
                     (leg_width, leg_height + body_height / 2.0),
                     (body_width - leg_width, leg_height),
                     (body_width - leg_width, 0),
                     (body_width - leg_width / 2.0, 0)]

    group_sizes = [(leg_width / 2.0, leg_height), (leg_width / 2.0, leg_height),
                   (leg_width, body_height),
                   (body_width - 2.0 * leg_width, body_height / 2.0),
                   (body_width - 2.0 * leg_width, body_height / 2.0),
                   (leg_width, body_height), (leg_width / 2.0, leg_height),
                   (leg_width / 2.0, leg_height)]
```

```

groups = []
#Build the groups from the specifications
for i in range(len(group_offsets)):
    groups.append(
        Group(np.array(offset=group_offsets[i]), size=np.array(group_sizes[i]),
            particle_density=np.array([sample_density, sample_density])))

#How many particles did we allocate in total?
num_particles = sum(map(Group.get_num_particles, groups))

#Our robot will have one a list of one observer -
#based on the mean positions and velocities of clustered portions.
observers = [KMeansObserver(particles_to_track=list(range(num_particles)),
    n_clusters=7)]

#Allocate pressure actuators to certain groups.
#Set the affected particles, direction, and maximum actuation
actuators = [AxisAlignedPressureActuator(get_group_particles(groups, a),
    1, max_act=2.0) for a in [0, 1, 6, 7]]

#Create MLP with ReLUs in the hidden layers and tanh as the final activation
controller = TanhNNController(use_relu=True, hidden_layers = (64, 64))

#What are our objective functions?

#Maximize average forward progress in the x direction
def objective(final_state):
    return -tf.reduce_mean(final_state.position[0, :])

#Minimize running squared actuation
def running_objective(state):
    return tf.linalg.norm(state.controller(state))**2

optimizer = GradientDescentOptimizer #default Adam

#Feed our specifications into the robot. Defaults to non-trainable design:
#Default Young's = 10.0, Poissons = 0.3, particle mass and volume = 1.0
cm = ControllableMechanism(sess, dim=dim, groups=groups, observers=observers,
    controller = controller, actuators = actuators, optimizer = optimizer,
    objectives = [(objective, 1.0)],
    running_objectives = [(running_objective, 1e-8)],
    use_neohookean=True)

```

```

gravity = (0, gravity_strength)
bc = get_bounding_box_bc((400, 64)) #grid size
cm.set_world(res, bc, gravity)

#How long will we simulate for? #How many steps will we optimize for?
cm.initialize(dt = dt, num_steps = num_steps, num_opt_steps=100)
cm.optimize() #Optimize over controller

```

A.3 Forward Simulation and Differentiation

In this section, we discuss the detailed steps for backward gradient computation in , *i.e.* the differentiable Moving Least Squares Material Point Method (MLS-MPM) [64]. We assume fixed particle volume V_p^0 , hyperelastic constitutive model (with potential energy ψ_p or Young's modulus E_p and Poisson's ratio ν_p) for simplicity.

The list of notation for MLS-MPM is below:

Symbol	Type	Affiliation	Meaning
Δt	scalar		time step size
Δx	scalar		grid cell size
$\mathbf{x}_p$	vector	particle	position
V_p^0	scalar	particle	initial volume
m_p	scalar	particle	particle mass
E_p	scalar	particle	particle Young's Modulus
ν_p	scalar	particle	particle Poisson's ratio
$\mathbf{v}_p$	vector	particle	velocity
$\mathbf{C}_p$	matrix	particle	affine velocity field [70]
$\mathbf{P}_p$	matrix	particle	PK1 stress ($\partial\psi_p/\partial\mathbf{F}_p$)
$\boldsymbol{\sigma}_{pa}$	matrix	particle	actuation Cauchy stress
$\mathbf{A}_p$	matrix	particle	actuation stress (material space)
$\mathbf{F}_p$	matrix	particle	deformation gradient
$\mathbf{x}_i$	vector	node	position
m_i	scalar	node	mass
$\mathbf{v}_i$	vector	node	velocity
$\mathbf{p}_i$	vector	node	momentum, i.e. $m_i\mathbf{v}_i$
N	scalar		quadratic B-spline function

Variable dependencies

The MLS-MPM time stepping is defined as follows:

$$\mathbf{P}_p^n = \mathbf{P}_p^n(\mathbf{F}_p^n, E_p, \nu_p) + \mathbf{F}_p \boldsymbol{\sigma}_{pa}^n \quad (\text{A.1})$$

$$m_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) m_p \quad (\text{A.2})$$

$$\mathbf{p}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) \left[m_p \mathbf{v}_p^n + \left(-\frac{4}{\Delta x^2} \Delta t V_p^0 \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n \right) (\mathbf{x}_i - \mathbf{x}_p^n) \right] \quad (\text{A.3})$$

$$\mathbf{v}_i^n = \frac{1}{m_i^n} \mathbf{p}_i^n \quad (\text{A.4})$$

$$\mathbf{v}_p^{n+1} = \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) \mathbf{v}_i^n \quad (\text{A.5})$$

$$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2} \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) \mathbf{v}_i^n (\mathbf{x}_i - \mathbf{x}_p^n)^T \quad (\text{A.6})$$

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \mathbf{C}_p^{n+1}) \mathbf{F}_p^n, \quad (\text{A.7})$$

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1} \quad (\text{A.8})$$

The forward variable dependency is as follows:

$$\mathbf{x}_p^{n+1} \leftarrow \mathbf{x}_p^n, \mathbf{v}_p^{n+1} \quad (\text{A.9})$$

$$\mathbf{v}_p^{n+1} \leftarrow \mathbf{x}_p^n, \mathbf{v}_i^n \quad (\text{A.10})$$

$$\mathbf{C}_p^{n+1} \leftarrow \mathbf{x}_p^n, \mathbf{v}_i^n \quad (\text{A.11})$$

$$\mathbf{F}_p^{n+1} \leftarrow \mathbf{F}_p^n, \mathbf{C}_p^{n+1} \quad (\text{A.12})$$

$$\mathbf{p}_i^n \leftarrow \mathbf{x}_p^n, \mathbf{C}_p^n, \mathbf{v}_p^n, \mathbf{P}_p^n, \mathbf{F}_p^n, m_p^n \quad (\text{A.13})$$

$$\mathbf{v}_i^n \leftarrow \mathbf{p}_i^n, m_i^n \quad (\text{A.14})$$

$$\mathbf{P}_p^n \leftarrow \mathbf{F}_p^n, \boldsymbol{\sigma}_{pa}^n, E_p, \nu_p \quad (\text{A.15})$$

$$m_i^n \leftarrow \mathbf{x}_p^n, m_p \quad (\text{A.16})$$

During back-propagation, we have the following reversed variable dependency:

$$\mathbf{x}_p^{n+1}, \mathbf{v}_p^{n+1}, \mathbf{C}_p^{n+1}, \mathbf{p}_i^{n+1}, m_i^{n+1} \leftarrow \mathbf{x}_p^n \quad (\text{A.17})$$

$$\mathbf{p}_i^n \leftarrow \mathbf{v}_p^n \quad (\text{A.18})$$

$$\mathbf{x}_p^{n+1} \leftarrow \mathbf{v}_p^{n+1} \quad (\text{A.19})$$

$$\mathbf{v}_p^{n+1}, \mathbf{C}_p^{n+1} \leftarrow \mathbf{v}_i^n \quad (\text{A.20})$$

$$\mathbf{F}_p^{n+1}, \mathbf{P}_p^n, \mathbf{p}_i^n \leftarrow \mathbf{F}_p^n \quad (\text{A.21})$$

$$\mathbf{F}_p^{n+1} \leftarrow \mathbf{C}_p^{n+1} \quad (\text{A.22})$$

$$\mathbf{p}_i^n \leftarrow \mathbf{C}_p^n \quad (\text{A.23})$$

$$\mathbf{v}_i^n \leftarrow \mathbf{p}_i^n \quad (\text{A.24})$$

$$\mathbf{v}_i^n \leftarrow m_i^n \quad (\text{A.25})$$

$$\mathbf{p}_i^n \leftarrow \mathbf{P}_p^n \quad (\text{A.26})$$

$$E_p \leftarrow \mathbf{P}_p^n \quad (\text{A.27})$$

$$\nu_p \leftarrow \mathbf{P}_p^n \quad (\text{A.28})$$

$$\mathbf{P}_p^n \leftarrow \boldsymbol{\sigma}_{pa}^n \quad (\text{A.29})$$

$$m_p \leftarrow m_i^n, \mathbf{p}_i^n \quad (\text{A.30})$$

We reverse swap two sides of the equations for easier differentiation derivation:

$$\mathbf{x}_p^n \rightarrow \mathbf{x}_p^{n+1}, \mathbf{v}_p^{n+1}, \mathbf{C}_p^{n+1}, \mathbf{p}_i^{n+1}, m_i^{n+1} \quad (\text{A.31})$$

$$\mathbf{v}_p^n \rightarrow \mathbf{p}_p^n \quad (\text{A.32})$$

$$\mathbf{v}_p^{n+1} \rightarrow \mathbf{x}_p^{n+1} \quad (\text{A.33})$$

$$\mathbf{v}_i^n \rightarrow \mathbf{v}_p^{n+1}, \mathbf{C}_p^{n+1} \quad (\text{A.34})$$

$$\mathbf{F}_p^n \rightarrow \mathbf{F}_p^{n+1}, \mathbf{P}_p^n, \mathbf{p}_i^n \quad (\text{A.35})$$

$$\mathbf{C}_p^{n+1} \rightarrow \mathbf{F}_p^{n+1} \quad (\text{A.36})$$

$$\mathbf{C}_p^n \rightarrow \mathbf{p}_i^n \quad (\text{A.37})$$

$$\mathbf{p}_i^n \rightarrow \mathbf{v}_i^n \quad (\text{A.38})$$

$$m_i^n \rightarrow \mathbf{v}_i^n \quad (\text{A.39})$$

$$\mathbf{P}_p^n \rightarrow \mathbf{p}_i^n \quad (\text{A.40})$$

$$E_p \rightarrow \mathbf{P}_p^n \quad (\text{A.41})$$

$$\nu_p \rightarrow \mathbf{P}_p^n \quad (\text{A.42})$$

$$\sigma_{pa}^n \rightarrow \mathbf{P}_p^n \quad (\text{A.43})$$

$$m_p \rightarrow m_i^n, \mathbf{p}_i^n \quad (\text{A.44})$$

In the following sections, we derive detailed gradient relationships, in the order of actual gradient computation. The frictional boundary condition gradients are postponed to the end since it is less central, though during computation it belongs to grid operations. Back-propagation in is essentially a reversed process of forward simulation. The computation has three steps, backward particle to grid (P2G), backward grid operations, and backward grid to particle (G2P).

Backward Particle to Grid (P2G)

(A, P2G) For $\mathbf{v}_p^{n+1}$, we have

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1} \quad (\text{A.45})$$

$$\Rightarrow \frac{\partial L}{\partial \mathbf{v}_{p\alpha}^{n+1}} = \left[\frac{\partial L}{\partial \mathbf{x}_p^{n+1}} \frac{\partial \mathbf{x}_p^{n+1}}{\partial \mathbf{v}_p^{n+1}} \right]_{\alpha} \quad (\text{A.46})$$

$$= \Delta t \frac{\partial L}{\partial \mathbf{x}_{p\alpha}^{n+1}}. \quad (\text{A.47})$$

(B, P2G) For $\mathbf{C}_p^{n+1}$, we have

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \mathbf{C}_p^{n+1}) \mathbf{F}_p^n \quad (\text{A.48})$$

$$\Rightarrow \frac{\partial L}{\partial \mathbf{C}_{p\alpha\beta}^{n+1}} = \left[\frac{\partial L}{\partial \mathbf{F}_p^{n+1}} \frac{\partial \mathbf{F}_p^{n+1}}{\partial \mathbf{C}_p^{n+1}} \right]_{\alpha\beta} \quad (\text{A.49})$$

$$= \Delta t \sum_{\gamma} \frac{\partial L}{\partial \mathbf{F}_{p\alpha\gamma}^{n+1}} \mathbf{F}_{p\beta\gamma}^n. \quad (\text{A.50})$$

Note, the above two gradients should also include the contributions of $\frac{\partial L}{\partial \mathbf{v}_p^n}$ and $\frac{\partial L}{\partial \mathbf{C}_p^n}$ respectively, with n being the next time step.

(C, P2G) For $\mathbf{v}_i^n$, we have

$$\mathbf{v}_p^{n+1} = \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) \mathbf{v}_i^n \quad (\text{A.51})$$

$$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2} \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) \mathbf{v}_i^n (\mathbf{x}_i - \mathbf{x}_p^n)^T \quad (\text{A.52})$$

$$\Rightarrow \frac{\partial L}{\partial \mathbf{v}_{i\alpha}^n} = \left[\sum_p \frac{\partial L}{\partial \mathbf{v}_p^{n+1}} \frac{\partial \mathbf{v}_p^{n+1}}{\partial \mathbf{v}_i^n} + \sum_p \frac{\partial L}{\partial \mathbf{C}_p^{n+1}} \frac{\partial \mathbf{C}_p^{n+1}}{\partial \mathbf{v}_i^n} \right]_{\alpha} \quad (\text{A.53})$$

$$= \sum_p \left[\frac{\partial L}{\partial \mathbf{v}_{p\alpha}^{n+1}} + \frac{4}{\Delta x^2} \sum_{\beta} \frac{\partial L}{\partial \mathbf{C}_{p\alpha\beta}^{n+1}} (\mathbf{x}_{i\beta} - \mathbf{x}_{p\beta}) \right]. \quad (\text{A.54})$$

Backward Grid Operations

(D, grid) For $\mathbf{p}_i^n$, we have

$$\mathbf{v}_i^n = \frac{1}{m_i^n} \mathbf{p}_i^n \quad (\text{A.55})$$

$$\Rightarrow \frac{\partial L}{\partial \mathbf{p}_{i\alpha}^n} = \left[\frac{\partial L}{\partial \mathbf{v}_i^n} \frac{\partial \mathbf{v}_i^n}{\partial \mathbf{p}_i^n} \right]_{\alpha} \quad (\text{A.56})$$

$$= \frac{\partial L}{\partial \mathbf{v}_{i\alpha}^n} \frac{1}{m_i^n}. \quad (\text{A.57})$$

(E, grid) For m_i^n , we have

$$\mathbf{v}_i^n = \frac{1}{m_i^n} \mathbf{p}_i^n \quad (\text{A.58})$$

$$\Rightarrow \frac{\partial L}{\partial m_i^n} = \frac{\partial L}{\partial \mathbf{v}_i^n} \frac{\partial \mathbf{v}_i^n}{\partial m_i^n} \quad (\text{A.59})$$

$$= -\frac{1}{(m_i^n)^2} \sum_{\alpha} \mathbf{p}_{i\alpha}^n \frac{\partial L}{\partial \mathbf{v}_{i\alpha}^n} \quad (\text{A.60})$$

$$= -\frac{1}{m_i^n} \sum_{\alpha} \mathbf{v}_{i\alpha}^n \frac{\partial L}{\partial \mathbf{v}_{i\alpha}^n}. \quad (\text{A.61})$$

Backward Grid to Particle (G2P)

(F, G2P) For $\mathbf{v}_p^n$, we have

$$\mathbf{p}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) \left[m_p \mathbf{v}_p^n + \left(-\frac{4}{\Delta x^2} \Delta t V_p^0 \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n \right) (\mathbf{x}_i - \mathbf{x}_p^n) \right] \quad (\text{A.62})$$

$$\Rightarrow \frac{\partial L}{\partial \mathbf{v}_{p\alpha}^n} = \left[\sum_i \frac{\partial L}{\partial \mathbf{p}_p^n} \frac{\partial \mathbf{p}_p^n}{\partial \mathbf{v}_p^n} \right]_{\alpha} \quad (\text{A.63})$$

$$= \sum_i m_p \frac{\partial L}{\partial \mathbf{p}_{i\alpha}^n}. \quad (\text{A.64})$$

(G, G2P) For $\mathbf{P}_p^n$, we have

$$\begin{aligned} \mathbf{p}_i^n &= \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) \left[m_p \mathbf{v}_p^n + \left(-\frac{4}{\Delta x^2} \Delta t V_p^0 \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n \right) (\mathbf{x}_i - \mathbf{x}_p^n) \right] \\ \Rightarrow \frac{\partial L}{\partial \mathbf{P}_{p\alpha\beta}^n} &= \left[\frac{\partial L}{\partial \mathbf{p}_i^n} \frac{\partial \mathbf{p}_i^n}{\partial \mathbf{P}_p^n} \right]_{\alpha\beta} \end{aligned} \quad (\text{A.66})$$

$$= - \sum_i \frac{4}{\Delta x^2} \Delta t V_p^0 \sum_{\gamma} \frac{\partial L}{\partial \mathbf{p}_{i\alpha}^n} \mathbf{F}_{p\gamma\beta}^n (\mathbf{x}_{i\gamma} - \mathbf{x}_{p\gamma}^n). \quad (\text{A.67})$$

(H, G2P) For $\mathbf{F}_p^n$, we have

$$\mathbf{F}_p^{n+1} = (\mathbf{I} + \Delta t \mathbf{C}_p^{n+1}) \mathbf{F}_p^n \quad (\text{A.68})$$

$$\mathbf{P}_p^n = \mathbf{P}_p^n(\mathbf{F}_p^n, E_p, \nu_p) + \mathbf{F}_p \boldsymbol{\sigma}_{pa}^n \quad (\text{A.69})$$

$$\mathbf{p}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) \left[m_p \mathbf{v}_p^n + \left(-\frac{4}{\Delta x^2} \Delta t V_p^0 \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n \right) (\mathbf{x}_i - \mathbf{x}_p^n) \right] \quad (\text{A.70})$$

$$\Rightarrow \frac{\partial L}{\partial \mathbf{F}_{p\alpha\beta}^n} = \left[\frac{\partial L}{\partial \mathbf{F}_p^{n+1}} \frac{\partial \mathbf{F}_p^{n+1}}{\partial \mathbf{F}_p^n} + \frac{\partial L}{\partial \mathbf{P}_p^n} \frac{\partial \mathbf{P}_p^n}{\partial \mathbf{F}_p^n} + \frac{\partial L}{\partial \mathbf{p}_i^n} \frac{\partial \mathbf{p}_i^n}{\partial \mathbf{F}_p^n} \right]_{\alpha\beta} \quad (\text{A.71})$$

$$\begin{aligned} &= \sum_{\gamma} \frac{\partial L}{\partial \mathbf{F}_{p\gamma\beta}^{n+1}} (\mathbf{I}_{\gamma\alpha} + \Delta t \mathbf{C}_{p\gamma\alpha}^{n+1}) + \sum_{\gamma} \sum_{\eta} \frac{\partial L}{\partial \mathbf{P}_{p\gamma\eta}^n} \frac{\partial^2 \Psi_p}{\partial \mathbf{F}_{p\gamma\eta}^n \partial \mathbf{F}_{p\alpha\beta}^n} + \sum_{\gamma} \frac{\partial L}{\partial \mathbf{P}_{p\alpha\gamma}^n} \\ &\quad + \sum_i - \sum_{\gamma} \frac{\partial L}{\partial \mathbf{p}_{i\gamma}^n} \frac{4}{\Delta x^2} \Delta t V_p^0 \mathbf{P}_{p\gamma\beta}^n (\mathbf{x}_{i\alpha} - \mathbf{x}_{p\alpha}^n). \end{aligned} \quad (\text{A.73})$$

(I, G2P) For $\mathbf{C}_p^n$, we have

$$\begin{aligned} \mathbf{p}_i^n &= \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) \left[m_p \mathbf{v}_p^n + \left(-\frac{4}{\Delta x^2} \Delta t V_p^0 \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n \right) (\mathbf{x}_i - \mathbf{x}_p^n) \right] \\ \Rightarrow \frac{\partial L}{\partial \mathbf{C}_{p\alpha\beta}^n} &= \left[\sum_i \frac{\partial L}{\partial \mathbf{p}_i^n} \frac{\partial \mathbf{p}_i^n}{\partial \mathbf{C}_p^n} \right]_{\alpha\beta} \end{aligned} \quad (\text{A.75})$$

$$= \sum_i \frac{\partial L}{\partial \mathbf{p}_{i\alpha}^n} m_p (\mathbf{x}_{i\beta} - \mathbf{x}_{p\beta}^n). \quad (\text{A.76})$$

(J, G2P) For $\mathbf{x}_p^n$, we have

$$\mathbf{x}_p^{n+1} = \mathbf{x}_p^n + \Delta t \mathbf{v}_p^{n+1} \quad (\text{A.77})$$

$$\mathbf{v}_p^{n+1} = \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) \mathbf{v}_i^n \quad (\text{A.78})$$

$$\mathbf{C}_p^{n+1} = \frac{4}{\Delta x^2} \sum_i N(\mathbf{x}_i - \mathbf{x}_p^n) \mathbf{v}_i^n (\mathbf{x}_i - \mathbf{x}_p^n)^T \quad (\text{A.79})$$

$$\mathbf{p}_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) \left[m_p \mathbf{v}_p^n + \left(-\frac{4}{\Delta x^2} \Delta t V_p^0 \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n \right) (\mathbf{x}_i - \mathbf{x}_p^n) \right] \quad (\text{A.80})$$

$$m_i^n = \sum_p N(\mathbf{x}_i - \mathbf{x}_p^n) m_p \quad (\text{A.81})$$

$$\mathbf{G}_p := \left(-\frac{4}{\Delta x^2} V_p^0 \Delta t \mathbf{P}_p^n \mathbf{F}_p^{nT} + m_p \mathbf{C}_p^n \right) \quad (\text{A.82})$$

$$\implies \quad (\text{A.83})$$

$$\frac{\partial L}{\partial \mathbf{x}_{p\alpha}^n} = \left[\frac{\partial L}{\partial \mathbf{x}_p^{n+1}} \frac{\partial \mathbf{x}_p^{n+1}}{\partial \mathbf{x}_p^n} + \frac{\partial L}{\partial \mathbf{v}_p^{n+1}} \frac{\partial \mathbf{v}_p^{n+1}}{\partial \mathbf{x}_p^n} + \frac{\partial L}{\partial \mathbf{C}_p^{n+1}} \frac{\partial \mathbf{C}_p^{n+1}}{\partial \mathbf{x}_p^n} + \frac{\partial L}{\partial \mathbf{p}_i^n} \frac{\partial \mathbf{p}_i^n}{\partial \mathbf{x}_p^n} + \frac{\partial L}{\partial m_i^n} \frac{\partial m_i^n}{\partial \mathbf{x}_p^n} \right]_\alpha \quad (\text{A.84})$$

$$= \frac{\partial L}{\partial \mathbf{x}_{p\alpha}^{n+1}} \quad (\text{A.85})$$

$$+ \sum_i \sum_\beta \frac{\partial L}{\partial \mathbf{v}_{p\beta}^{n+1}} \frac{\partial N(\mathbf{x}_i - \mathbf{x}_p^n)}{\partial \mathbf{x}_{i\alpha}} \mathbf{v}_{i\beta}^n \quad (\text{A.86})$$

$$+ \sum_i \sum_\beta \frac{4}{\Delta x^2} \left\{ -\frac{\partial L}{\partial \mathbf{C}_{p\beta\alpha}^{n+1}} \mathbf{v}_{i\beta}^n + \sum_\gamma \frac{\partial L}{\partial \mathbf{C}_{p\beta\gamma}^{n+1}} \frac{\partial N(\mathbf{x}_i - \mathbf{x}_p^n)}{\partial \mathbf{x}_{i\alpha}} \mathbf{v}_{i\beta}^n (\mathbf{x}_{i\gamma} - \mathbf{x}_{p\gamma}^n) \right\} \quad (\text{A.87})$$

$$+ \sum_i \sum_\beta \frac{\partial L}{\partial \mathbf{p}_{i\beta}^n} \left[\frac{\partial N(\mathbf{x}_i - \mathbf{x}_p^n)}{\partial \mathbf{x}_{i\alpha}} (m_p \mathbf{v}_{p\beta}^n + [\mathbf{G}_p(\mathbf{x}_i - \mathbf{x}_p^n)]_\beta) - \mathbf{G}_{p\beta\alpha} \right] \quad (\text{A.88})$$

$$+ m_p \sum_i \frac{\partial L}{\partial m_i^n} \frac{\partial N(\mathbf{x}_i - \mathbf{x}_p^n)}{\partial \mathbf{x}_{i\alpha}} \quad (\text{A.89})$$

Further, we have:

$$\implies \quad \frac{\partial L}{m_p} = \sum_i \frac{\partial L}{m_i^n} \frac{\partial m_i^n}{\partial m_p} = \sum_i \frac{\partial L}{m_i^n} N(\mathbf{x}_i - \mathbf{x}_p^n) \quad (\text{A.90})$$

(K, G2P) For σ_{pa}^n , we have

$$\mathbf{P}_p^n = \mathbf{P}_p^n(\mathbf{F}_p^n, E_p, \nu_p) + \mathbf{F}_p \boldsymbol{\sigma}_{pa}^n \quad (\text{A.91})$$

$$\Rightarrow \frac{\partial L}{\partial \boldsymbol{\sigma}_{pa\alpha\beta}^n} = \left[\frac{\partial L}{\partial \mathbf{P}_{elast,p}^n} \frac{\partial \mathbf{P}_{elast,p}^n}{\partial \boldsymbol{\sigma}_{p\alpha}^n} \right]_{\alpha\beta} \quad (\text{A.92})$$

$$= \sum_{\gamma} \frac{\partial L}{\partial \mathbf{P}_{p\gamma\beta}^{n+1}} \mathbf{F}_{p\gamma\alpha}^n. \quad (\text{A.93})$$

Here, we also note the gradients for the constitutive material constants, E and ν . The precise formulation of the gradients depends on the constitutive model. For the equations below, we use the Neo-Hookean model; other gradients may be derived similarly.

$$\Rightarrow \frac{\partial L}{\partial E_p^n} = \frac{\partial L}{\partial \mathbf{P}_p^n} \frac{\partial \mathbf{P}_p^n}{\partial E_p^n} \quad (\text{A.94})$$

$$= \sum_{\beta, \gamma} \frac{\partial L}{\partial \mathbf{P}_{p\gamma\beta}^{n+1}} \frac{1}{E_p}. \quad (\text{A.95})$$

$$\lambda_p^n = \frac{E_p \nu_p}{(1 + \nu_p)(1 - 2\nu_p)} \quad (\text{A.96})$$

$$J_p^n = \det(\mathbf{F}_p^n) \quad (\text{A.97})$$

$$\phi_p^n = \quad (\text{A.98})$$

$$\mathbf{P}_p^n - \lambda(J_p^n - 1)J_p^n \mathbf{F}^{-T} \quad (\text{A.99})$$

$$\Rightarrow \frac{\partial L}{\partial \nu_p^n} = \frac{\partial L}{\partial \mathbf{P}_p^n} \frac{\partial \mathbf{P}_p^n}{\partial \nu_p^n} \quad (\text{A.100})$$

$$= \sum_{\beta, \gamma} \frac{\partial L}{\partial \mathbf{P}_{p\gamma\beta}^{n+1}} \left(\frac{\phi_{p\gamma\beta}^n}{1 + \nu_p} + \frac{E J_p^n (J_p^n - 1) \mathbf{F}_{p\beta\gamma}^{-T} (1 + 2\nu_p)^2}{(1 - 2\nu_p)^2 (1 + \nu_p)^2} \right) \quad (\text{A.101})$$

A.4 Soft Robot Problem Parameters

Problem	# Particles	Actuators	Observer	Controller
2D Cable Biped	5024	4 Cables	None	Open Loop
3D Bulbasaur	22084	4 Pressure	K-Means	MLP
2D Biped w/ Terrain	36200	4 Pressure	Centroid	MLP
3D Arm	17280	8 Pressure	None	Open Loop
2D Biped	12800	4 Pressure	Centroid	MLP
3D Quadruped	84375	16 Pressure	Centroid	MLP
3D Hexapod	111375	24 Pressure	Centroid	MLP
3D Octoped	138375	32 Pressure	Centroid	MLP

Problem	Time (s)	dt (s)	substeps	Learning Rate
2D Cable Biped	4	10		2×10^{-2}
3D Bulbasaur	4	0.005	10	2×10^{-3}
2D Biped w/ Terrain	4	0.02	100	1×10^{-3}
3D Arm	2	0.0	100	2×10^{-2}
2D Biped	4	0.02	10	1×10^{-3}
3D Quadruped	3	0.05	10	5×10^{-3}
3D Hexapod	3	0.05	10	5×10^{-3}
3D Octoped	3	0.05	10	5×10^{-3}

Bibliography

- [1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.
- [2] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.
- [3] Ozgur Atalay and William Kennon. Knitted strain sensors: Impact of design parameters on sensing properties. *Sensors*, 14(3):4712–4730, 2014.
- [4] Ozgur Atalay, William Kennon, and Muhammad Husain. Textile-based weft knitted strain sensors: Effect of fabric parameters on sensor properties. *Sensors*, 13(8):11114–11127, 2013.
- [5] Ozgur Atalay, Asli Tuncay, Muhammad D Husain, and William R Kennon. Comparative study of the weft-knitted strain sensors. *Journal of Industrial Textiles*, 46(5):1212–1240, 2017.
- [6] Ed Ayyappa. Normal human locomotion, part 1: Basic concepts and terminology. *JPO: Journal of Prosthetics and Orthotics*, 9(1):10–17, 1997.
- [7] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *stat*, 1050:21, 2016.
- [8] Moritz Bächer, Bernd Bickel, Doug L James, and Hanspeter Pfister. Fabricating articulated characters from skinned meshes. *ACM Transactions on Graphics (TOG)*, 31(4):1–9, 2012.
- [9] Moritz Bächer, Stelian Coros, and Bernhard Thomaszewski. Linkedit: Interactive linkage editing using symbolic kinematics. *ACM Transactions on Graphics (TOG)*, 34(4):1–8, 2015.
- [10] Moritz Bächer, Benjamin Hepp, Fabrizio Pece, Paul G Kry, Bernd Bickel, Bernhard Thomaszewski, and Otmar Hilliges. Defense: Computational design of customized deformable input devices. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 3806–3816. ACM, 2016.

- [11] Moritz Bächer, Emily Whiting, Bernd Bickel, and Olga Sorkine-Hornung. Spin-it: Optimizing moment of inertia for spinnable objects. *ACM Transactions on Graphics (TOG)*, 33(4):1–10, 2014.
- [12] Jernej Barbič, Marco da Silva, and Jovan Popović. Deformable object animation using reduced optimal control. *ACM Transactions on Graphics (TOG)*, 28(3):53, 2009.
- [13] Jernej Barbič and Jovan Popović. Real-time control of physically based simulations using gentle forces. In *ACM transactions on graphics (TOG)*, volume 27, page 163. ACM, 2008.
- [14] James M Bern, Kai-Hung Chang, and Stelian Coros. Interactive design of animated plushies. *ACM Transactions on Graphics (TOG)*, 36(4):1–11, 2017.
- [15] Javier Bonet and Richard D Wood. *Nonlinear continuum mechanics for finite element analysis*. Cambridge university press, 1997.
- [16] Christof Büskens and Dennis Wassel. The esa nlp solver worhp. In *Modeling and optimization in space engineering*, pages 85–110. Springer, 2012.
- [17] Jacques Calì, Dan A Calian, Cristina Amati, Rebecca Kleinberger, Anthony Steed, Jan Kautz, and Tim Weyrich. 3d-printing of non-assembly, articulated models. *ACM Transactions on Graphics (TOG)*, 31(6):1–8, 2012.
- [18] A. Censi. A class of co-design problems with cyclic constraints and their solution. *IEEE Robotics and Automation Letters*, 2(1):96–103, Jan 2017.
- [19] Duygu Ceylan, Wilmot Li, Niloy J Mitra, Maneesh Agrawala, and Mark Pauly. Designing and fabricating mechanical automata from mocap sequences. *ACM Transactions on Graphics (TOG)*, 32(6):1–11, 2013.
- [20] Desai Chen, David I. W. Levin, Wojciech Matusik, and Danny M. Kaufman. Dynamics-aware numerical coarsening for fabrication design. *ACM Trans. Graph.*, 36(4):84:1–84:15, July 2017.
- [21] Desai Chen, Pitchaya Sitthi-amorn, Justin T Lan, and Wojciech Matusik. Computing and fabricating multiplanar models. In *Computer graphics forum*, volume 32, pages 305–315. Wiley Online Library, 2013.
- [22] Nick Cheney, Josh Bongard, Vytas SunSpiral, and Hod Lipson. Scalable co-optimization of morphology and control in embodied machines. *Journal of The Royal Society Interface*, 15(143):20170937, 2018.
- [23] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling evolution: Evolving soft robots with multiple materials and a powerful generative encoding. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, GECCO '13*, pages 167–174, 2013.

- [24] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 167–174, 2013.
- [25] Jingyuan Cheng, Oliver Amft, and Paul Lukowicz. Active capacitive sensing: Exploring a new wearable sensing modality for activity recognition. In *International conference on pervasive computing*, pages 319–336. Springer, 2010.
- [26] Alex Chortos, Jia Liu, and Zhenan Bao. Pursuing prosthetic electronic skin. *Nature materials*, 15(9):937–950, 2016.
- [27] Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel. Computational design of mechanical characters. *ACM Transactions on Graphics*, 32(4):83:1–83:12, July 2013.
- [28] Francesco Corucci, Nick Cheney, Hod Lipson, Cecilia Laschi, and Josh Bongard. Evolving swimming soft-bodied creatures. In *ALIFE XV, The Fifteenth International Conference on the Synthesis and Simulation of Living Systems, Late Breaking Proceedings*, volume 6, 2016.
- [29] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.
- [30] Utku Culha, Surya Nurzaman, Frank Clemens, and Fumiya Iida. Svas3: strain vector aided sensorization of soft structures. *Sensors*, 14(7):12748–12770, 2014.
- [31] David W Currier. Automation of sheet metal design and manufacturing. In *Proceedings of the 17th Design Automation Conference*, pages 134–138, 1980.
- [32] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. In *Advances in Neural Information Processing Systems*, pages 7178–7189, 2018.
- [33] Jonas Degraeve, Michiel Hermans, Joni Dambre, et al. A differentiable physics engine for deep learning in robotics. *arXiv preprint arXiv:1611.01652*, 2016.
- [34] Jonas Degraeve, Michiel Hermans, Joni Dambre, et al. A differentiable physics engine for deep learning in robotics. *Frontiers in neurorobotics*, 13:6, 2019.
- [35] Cosimo Della Santina, Robert K Katzschmann, Antonio Biechi, and Daniela Rus. Dynamic control of soft robots interacting with the environment. In *2018 IEEE International Conference on Soft Robotics (RoboSoft)*, pages 46–53. IEEE, 2018.
- [36] Mario Deuss, Daniele Panozzo, Emily Whiting, Yang Liu, Philippe Block, Olga Sorkine-Hornung, and Mark Pauly. Assembling self-supporting structures. *ACM Trans. Graph.*, 33(6):214–1, 2014.

- [37] PaGMO Developers. Pagmo and pygmo, 2019.
- [38] KM Digumarti, C Gehring, S Coros, J Hwangbo, and R Siegwart. Concurrent optimization of mechanical design and locomotion control of a legged robot. In *17th International conference on climbing and walking robots (CLAWAR 2014)*, 2014.
- [39] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.
- [40] Tao Du, Kui Wu, Pingchuan Ma, Sebastien Wah, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Diffpd: Differentiable projective dynamics with contact. *Transactions on Graphics*, 2021.
- [41] Tao Du, Kui Wu, Andrew Spielberg, Wojciech Matusik, Bo Zhu, and Eftychios Sifakis. Functional optimization of fluidic devices with differentiable stokes flow. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.
- [42] Erwin Coumans Yizhou Sheng Gaurav S. Sukhatme Eric Heiden, David Millard. Neursim: Augmenting differentiable simulators with neural networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021.
- [43] Yu Fang, Yuanming Hu, Shi-min Hu, and Chenfanfu Jiang. A temporally adaptive material point method with regional time stepping. *Computer graphics forum*, 37, 2018.
- [44] R Featherstone, DE Orin, B Siciliano, O Khatib, et al. Handbook of robotics. *chapter Dynamics*, pages 35–65, 2008.
- [45] Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.
- [46] Samuel Felton, Michael Tolley, Erik Demaine, Daniela Rus, and Robert Wood. A method for building self-folding machines. *Science*, 345(6197):644–646, 2014.
- [47] Ian Fitzner, Yue Sun, Vikram Sachdeva, and Shai Revzen. Rapidly prototyping robots: Using plates and reinforced flexures. *IEEE Robotics & Automation Magazine*, 24(1):41–47, 2017.
- [48] Mark Fuge, Greg Carmean, Jessica Cornelius, and Ryan Elder. The mechprocessor: Helping novices design printable mechanisms across different printers. *Journal of Mechanical Design*, 137(11), 2015.
- [49] Thomas Funkhouser, Michael Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David Dobkin. Modeling by example. *ACM transactions on graphics (TOG)*, 23(3):652–663, 2004.
- [50] Jakub Gafka, Mariusz Mąsior, Mateusz Zaborski, and Katarzyna Barczewska. Inertial motion sensing glove for sign language gesture acquisition and recognition. *IEEE Sensors Journal*, 16(16):6310–6316, 2016.

- [51] Akash Garg, Andrew O Sageman-Furnas, Bailin Deng, Yonghao Yue, Eitan Grinspun, Mark Pauly, and Max Wardetzky. Wire mesh design. *ACM Transactions on Graphics*, 33(4), 2014.
- [52] Markus Gifftthaler, Michael Neunert, Markus Stäuble, and Jonas Buchli. The control toolbox — an open-source c++ library for robotics, optimal and model predictive control. *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, pages 123–129, 2018.
- [53] Philip E. Gill, Walter Murray, and Michael A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization, 2002.
- [54] Oliver Glauser, Daniele Panozzo, Otmar Hilliges, and Olga Sorkine-Hornung. Deformation capture via soft and stretchable sensor arrays. *ACM Transactions on Graphics (TOG)*, 38(2):16, 2019.
- [55] Oliver Glauser, Shihao Wu, Daniele Panozzo, Otmar Hilliges, and Olga Sorkine-Hornung. Interactive hand pose estimation using a stretch-sensing soft glove. *ACM Transactions on Graphics (TOG)*, 38(4):41, 2019.
- [56] Jiangtao Gong, Jue Wang, and Yingqing Xu. Paperlego: Component-based papercraft designing tool for children. In *SIGGRAPH Asia 2014 Designing Tools For Crafting Interactive Artifacts*, pages 1–4. 2014.
- [57] Satyandra K Gupta and Dana S Nau. Systematic approach to analysing the manufacturability of machined parts. *Computer-Aided Design*, 27(5):323–342, 1995.
- [58] David Ha. Reinforcement learning for improving agent design. *Artificial life*, 25(4):352–365, 2019.
- [59] Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim, and Katsu Yamane. Joint optimization of robot design and motion parameters using the implicit function theorem. In *Robotics: Science and systems*, 2017.
- [60] Susumu Hara, Tetsuji Zama, Wataru Takashima, and Keiichi Kaneto. Artificial muscles based on polypyrrole actuators with large strain and stress induced electrically. *Polymer journal*, 36(2):151, 2004.
- [61] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments. *CoRR*, abs/1707.02286, 2017.
- [62] Aaron M Hoover, Erik Steltz, and Ronald S Fearing. Roach: An autonomous 2.4 g crawling hexapod robot. In *2008 IEEE/RSJ international conference on intelligent robots and systems*, pages 26–33. IEEE, 2008.

- [63] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Fredo Durand. DiffTaichi: Differentiable programming for physical simulation. In *International Conference on Learning Representations*, 2019.
- [64] 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):150, 2018.
- [65] 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. *ICRA*, 2019.
- [66] 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 *2019 International Conference on Robotics and Automation (ICRA)*, pages 6265–6271. IEEE, 2019.
- [67] Yuanming Hu, Xinxin Zhang, Ming Gao, and Chenfanfu Jiang. On hybrid lagrangian-eulerian simulation methods: practical notes and high-performance aspects. In *ACM SIGGRAPH 2019 Courses*, pages 1–246. 2019.
- [68] Josie Hughes and Fumiya Iida. Multi-functional soft strain sensors for wearable physiological monitoring. *Sensors*, 18(11):3822, 2018.
- [69] Andrew Ilyas, Logan Engstrom, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. A closer look at deep policy gradients. In *International Conference on Learning Representations*, 2019.
- [70] Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. The affine particle-in-cell method. *ACM Trans. Graph.*, 34(4), July 2015.
- [71] Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. The material point method for simulating continuum materials. In *ACM SIGGRAPH 2016 Courses*, pages 1–52. 2016.
- [72] VV Kamat. A survey of techniques for simulation of dynamic collision detection and response. *Computers & graphics*, 17(4):379–385, 1993.
- [73] Wisama Khalil. Dynamic modeling of robots using recursive newton-euler techniques. In *ICINCO2010*, 2010.
- [74] Rahul Khardekar, Greg Burton, and Sara McMains. Finding feasible mold parting directions using graphics hardware. *Computer-Aided Design*, 38(4):327–341, 2006.
- [75] Beomkeun Kim, Seong Beom Lee, Jayone Lee, Sehyun Cho, Hyungmin Park, Sanghoon Yeom, and Sung Han Park. A comparison among neo-hookean model, mooney-rivlin model, and ogden model for chloroprene rubber. *International Journal of Precision Engineering and Manufacturing*, 13(5):759–764, 2012.

- [76] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [77] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *stat*, 1050:10, 2014.
- [78] Bongjin Koo, Wilmot Li, JiaXian Yao, Maneesh Agrawala, and Niloy J Mitra. Creating works-like prototypes of mechanical objects. *ACM Transactions on Graphics*, 33(6), 2014.
- [79] Andreas Krause, Jure Leskovec, Carlos Guestrin, Jeanne VanBriesen, and Christos Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management*, 134(6):516–526, 2008.
- [80] Vikash Kumar, Emanuel Todorov, and Sergey Levine. Optimal control with learned local models: Application to dexterous manipulation. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 378–383. IEEE, 2016.
- [81] Manfred Lau, Akira Ohgawara, Jun Mitani, and Takeo Igarashi. Converting 3d furniture models to fabricatable parts and connectors. In *ACM Transactions on Graphics (TOG)*, volume 30, page 85. ACM, 2011.
- [82] Steven M LaValle and James J Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. 2000.
- [83] Q Li, WJ Zhang, and Li Chen. Design for control-a concurrent engineering approach for mechatronic systems design. *IEEE/ASME transactions on mechatronics*, 6(2):161–169, 2001.
- [84] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pages 222–229. Citeseer, 2004.
- [85] Xian-Ying Li, Chao-Hui Shen, Shi-Sheng Huang, Tao Ju, and Shi-Min Hu. Popup: automatic paper architectures from 3d models. *ACM Trans. Graph.*, 29(4):111–1, 2010.
- [86] Junbang Liang, Ming Lin, and Vladlen Koltun. Differentiable cloth simulation for inverse problems. In *Advances in Neural Information Processing Systems*, pages 771–780, 2019.
- [87] Xinqin Liao, Zheng Zhang, Zhuo Kang, Fangfang Gao, Qingliang Liao, and Yue Zhang. Ultrasensitive and stretchable resistive strain sensors designed for wearable electronics. *Materials Horizons*, 4(3):502–510, 2017.
- [88] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR (Poster)*, 2016.

- [89] Hod Lipson and Jordan B Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, 2000.
- [90] C Karen Liu and Sumit Jain. A quick tutorial on multibody dynamics. *Online tutorial*, June, page 7, 2012.
- [91] Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. Quasi-newton methods for real-time simulation of hyperelastic materials. *ACM Transactions on Graphics (TOG)*, 36(3):1–16, 2017.
- [92] Yuhao Liu, Matt Pharr, and Giovanni Antonio Salvatore. Lab-on-skin: a review of flexible and stretchable electronics for wearable health monitoring. *ACS nano*, 11(10):9614–9635, 2017.
- [93] Pingchuan Ma, Tao Du, John Z Zhang, Kui Wu, Andrew Spielberg, Robert K Katzschmann, and Wojciech Matusik. Diffaqua: A differentiable computational design pipeline for soft underwater swimmers with shape interpolation. *arXiv preprint arXiv:2104.00837*, 2021.
- [94] Andrew D Marchese, Robert K Katzschmann, and Daniela Rus. A recipe for soft fluidic elastomer robots. *Soft Robotics*, 2(1):7–25, 2015.
- [95] Constantinos Mavroidis, Kathryn J DeLaurentis, Jey Won, and Munshi Alam. Fabrication of non-assembly mechanisms and robotic systems using rapid prototyping. *J. Mech. Des.*, 123(4):516–524, 2001.
- [96] Vittorio Megaro*, Espen Knoop*, Andrew Spielberg*, David IW Levin, Wojciech Matusik, Markus Gross, Bernhard Thomaszewski, and Moritz Bächer. Designing cable-driven actuation networks for kinematic chains and trees. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 15. ACM, 2017.
- [97] Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus Gross, and Stelian Coros. Interactive design of 3D-printable robotic creatures. *ACM Transactions on Graphics*, 34(6):216:1–216:9, October 2015.
- [98] Ankur Mehta, Joseph DelPreto, and Daniela Rus. Integrated codesign of printable robots. *Journal of Mechanisms and Robotics*, 7(2), 2015.
- [99] Ankur Mehta, Joseph DelPreto, Kai Weng Wong, Scott Hamill, Hadas Kress-Gazit, and Daniela Rus. Robot creation from functional specifications. In *The International Symposium on Robotics Research (ISRR)*, Sestri Levante, Italy, Sep 2015.
- [100] Ankur M Mehta and Daniela Rus. An end-to-end system for designing mechanical structures for print-and-fold robots. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1460–1465. IEEE, 2014.

- [101] Jun Mitani and Hiromasa Suzuki. Making papercraft toys from meshes using strip-based approximate unfolding. *ACM transactions on graphics (TOG)*, 23(3):259–263, 2004.
- [102] 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.
- [103] Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel V Todorov. Interactive control of diverse complex characters with neural networks. In *Advances in Neural Information Processing Systems*, pages 3114–3122, 2015.
- [104] Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics (TOG)*, 31(4):43, 2012.
- [105] Yuki Mori and Takeo Igarashi. Plushie: an interactive design system for plush toys. In *ACM SIGGRAPH 2007 papers*, pages 45–es. 2007.
- [106] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li F Fei-Fei, Josh Tenenbaum, and Daniel L Yamins. Flexible neural representation for physics prediction. In *Advances in Neural Information Processing Systems*, pages 8799–8810, 2018.
- [107] Shuntaro Okazaki, Yoshihiro Muraoka, and Risa Suzuki. Validity and reliability of leap motion controller for assessing grasping and releasing finger movements. *Journal of Ergonomic Technology*, 17:32–42, 2017.
- [108] Gustavo Olague and Roger Mohr. Optimal camera placement for accurate reconstruction. *Pattern recognition*, 35(4):927–944, 2002.
- [109] M Pacelli, G Loriga, N Taccini, and R Paradiso. Sensing fabrics for monitoring physiological and biomechanical variables: E-textile solutions. In *2006 3rd IEEE/EMBS International Summer School on Medical Devices and Biosensors*, pages 1–4. IEEE, 2006.
- [110] Jahng-Hyon Park and Haruhiko Asada. Concurrent design optimization of mechanical structure and control for high speed robots. 1994.
- [111] Jahng-Hyon Park and Haruhiko Asada. Concurrent design optimization of mechanical structure and control for high speed robots. *Journal of dynamic systems, measurement, and control*, 116(3):344–356, 1994.
- [112] 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.

- [113] Jay Patel and Matthew I Campbell. An approach to automate and optimize concept generation of sheet metal parts by topological and parametric decoupling. *Journal of Mechanical Design*, 132(5), 2010.
- [114] John Kangchun Perng, Brian Fisher, Seth Hollar, and Kristofer SJ Pister. Acceleration sensing glove (asg). In *Digest of Papers. Third International Symposium on Wearable Computers*, pages 178–180. IEEE, 1999.
- [115] Samuel Pfrommer, Mathew Halm, and Michael Posa. Contactnets: Learning of discontinuous contact dynamics with smooth, implicit representations. *arXiv preprint arXiv:2009.11193*, 2020.
- [116] A.C. Pil and H.H. Asada. Integrated structure/control design of mechatronic systems using a recursive experimental optimization method. *IEEE/ASME Transactions on Mechatronics*, 1(3):191–203, Sept 1996.
- [117] Michael Posa and Russ Tedrake. Direct trajectory optimization of rigid body dynamical systems through contact. In *Algorithmic Foundations of Robotics X*, volume 86, pages 527–542. 2013.
- [118] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017.
- [119] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming Lin. Scalable differentiable physics for learning and control. In *International Conference on Machine Learning*, pages 7847–7856. PMLR, 2020.
- [120] Benjamin Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2:253–279, 2019.
- [121] Julie A. Reyer and Panos Y. Papalambros. Combined optimal design and control with application to an electric dc motor. *Journal of Mechanical Design*, 124:183–191, 2002.
- [122] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic back-propagation and approximate inference in deep generative models. *International Conference on Machine Learning (ICML)*, 2014.
- [123] Greg Saul, Manfred Lau, Jun Mitani, and Takeo Igarashi. Sketchchair: an all-in-one chair design system for end users. In *Proceedings of the fifth international conference on Tangible, embedded, and embodied interaction*, pages 73–80, 2010.
- [124] Charles Schaff, David Yunis, Ayan Chakrabarti, and Matthew R Walter. Jointly learning to construct and control agents using deep reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9798–9805. IEEE, 2019.

- [125] Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks. In *Conference on Robot Learning*, pages 317–335, 2018.
- [126] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.
- [127] Adriana Schulz, Ariel Shamir, David I. W. Levin, Pitchaya Sitthi-amorn, and Wojciech Matusik. Design and fabrication by example. *ACM Transactions on Graphics*, 33(4):62:1–62:11, July 2014.
- [128] 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*, SIGGRAPH ’12, pages 20:1–20:50, New York, NY, USA, 2012. ACM.
- [129] Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM, 1994.
- [130] Mélina Skouras, Bernhard Thomaszewski, Peter Kaufmann, Akash Garg, Bernd Bickel, Eitan Grinspun, and Markus Gross. Designing inflatable structures. *ACM Transactions on Graphics (TOG)*, 33(4):1–10, 2014.
- [131] Breannan Smith, Fernando De Goes, and Theodore Kim. Stable neo-hookean flesh simulation. *ACM Transactions on Graphics (TOG)*, 37(2):1–15, 2018.
- [132] Seungmoon Song, Joohyung Kim, and Katsu Yamane. Development of a bipedal robot that walks like an animation character. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3596–3602. IEEE, 2015.
- [133] Andrew Spielberg*, Alexander Amini*, Lillian Chin, Wojciech Matusik, and Daniela Rus. Clasp: Co-learning and sensor placement (for soft robot modeling, classification, and control).
- [134] Andrew Spielberg, Brandon Araki, Cynthia Sung, Russ Tedrake, and Daniela Rus. Functional co-optimization of articulated robots. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5035–5042. IEEE, 2017.
- [135] Andrew Spielberg, Tao Du, Yuanming Hu, Daniela Rus, and Wojciech Matusik. Advanced soft robot modeling in chainqueen. *Robotica*, 2021.
- [136] Andrew Spielberg, Allan Zhao, Tao Du, Yuanming Hu, Daniela Rus, and Wojciech Matusik. Learning-in-the-loop optimization: End-to-end control and co-design of soft robots through learned deep latent representations. In *Advances in Neural Information Processing Systems*, pages 8282–8292, 2019.
- [137] Alexey Stomakhin, Russell Howes, Craig Schroeder, and Joseph M Teran. Energetically consistent invertible elasticity. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 25–32. Eurographics Association, 2012.

- [138] Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. A material point method for snow simulation. *ACM Transactions on Graphics (TOG)*, 32(4):102, 2013.
- [139] Timothy Sun and Changxi Zheng. Computational design of twisty joints and puzzles. *ACM Transactions on Graphics (TOG)*, 34(4):1–11, 2015.
- [140] Yi Sun, Shuang Song, Xinquan Liang, and Hongliang Ren. A miniature soft robotic manipulator based on novel fabrication methods. *IEEE Robotics and Automation Letters*, 1(2):617–623, 2016.
- [141] Cynthia Sung and Daniela Rus. Foldable joints for foldable robots. *Journal of Mechanisms and Robotics*, 7(2), 2015.
- [142] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Robotics: Science and Systems*, 2018.
- [143] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4906–4913, 2012.
- [144] Russ Tedrake. Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for mit 6.832). *Downloaded in Fall*, 2014.
- [145] Russ Tedrake and the Drake Development Team. Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems, 2016.
- [146] Maxime Thieffry, Alexandre Kruszewski, Thierry-Marie Guerra, and Christian Duriez. Reduced order control of soft robots with guaranteed stability. In *European Control Conference ECC18*, 2018.
- [147] Bernhard Thomaszewski, Stelian Coros, Damien Gauge, Vittorio Megaro, Eitan Grinspun, and Markus Gross. Computational design of linkage-based characters. *ACM Transactions on Graphics (TOG)*, 33(4):1–9, 2014.
- [148] Thomas George Thuruthel, Benjamin Shih, Cecilia Laschi, and Michael Thomas Tolley. Soft robot perception using embedded soft sensors and recurrent neural networks. *Science Robotics*, 4(26), 2019.
- [149] Anastasia Tkach, Andrea Tagliasacchi, Edoardo Remelli, Mark Pauly, and Andrew Fitzgibbon. Online generative model personalization for hand tracking. *ACM Trans. Graph.*, 36(6), November 2017.
- [150] E. Todorov. A convex, smooth and invertible contact model for trajectory optimization. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1071–1076, May 2011.

- [151] Nobuyuki Umetani, Takeo Igarashi, and Niloy J Mitra. Guided exploration of physically valid shapes for furniture design. *ACM Trans. Graph.*, 31(4):86–1, 2012.
- [152] Nobuyuki Umetani, Danny M Kaufman, Takeo Igarashi, and Eitan Grinspun. Sensitive couture for interactive garment modeling and editing. *ACM Trans. Graph.*, 30(4):90, 2011.
- [153] Nobuyuki Umetani, Yuki Koyama, Ryan Schmidt, and Takeo Igarashi. Pteromys: Interactive design and optimization of free-formed free-flight model airplanes. *ACM Transactions on Graphics (TOG)*, 33(4):1–10, 2014.
- [154] Richard Vallett, Ryan Young, Chelsea Knittel, Youngmoo Kim, and Genevieve Dion. Development of a carbon fiber knitted capacitive touch sensor. *MRS Advances*, 1(38):2641–2651, 2016.
- [155] Jeroen van den Brand, Margreet de Kok, Marc Koetse, Maarten Cauwe, Rik Verplancke, Frederick Bossuyt, Michael Jablonski, and Jan Vanfleteren. Flexible and stretchable electronics for wearable health devices. *Solid-State Electronics*, 113:116–120, 2015.
- [156] Kevin Wampler, Jovan Popović, and Zoran Popović. Animal locomotion controllers from scratch. In *Computer Graphics Forum*, volume 32, pages 153–162. Wiley Online Library, 2013.
- [157] Kevin Wampler and Zoran Popović. Optimal gait and form for animal locomotion. *ACM Trans. Graph.*, 28(3):60:1–60:8, July 2009.
- [158] Cheng-Hua Wang and Robert H Sturges. Bendcad: a design system for concurrent multiple representations of parts. *Journal of Intelligent Manufacturing*, 7(2):133–144, 1996.
- [159] Tingwu Wang, Yuhao Zhou, Sanja Fidler, and Jimmy Ba. Neural graph evolution: Towards efficient automatic robot design. *arXiv preprint arXiv:1906.05370*, 2019.
- [160] Frank Weichert, Daniel Bachmann, Bartholomäus Rudak, and Denis Fisseler. Analysis of the accuracy and robustness of the leap motion controller. *Sensors*, 13(5):6380–6393, 2013.
- [161] Keenon Werling, Dalton Omens, Jeongseok Lee, Ionnis Exarchos, and C Karen Liu. Fast and feature-complete differentiable physics for articulated rigid bodies with contact. *arXiv preprint arXiv:2103.16021*, 2021.
- [162] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440. IEEE, 2016.

- [163] Jie Xu, Andrew Spielberg, Allan Zhao, Daniela Rus, and Wojciech Matusik. Multi-objective graph heuristic search for terrestrial robot design. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.
- [164] Shanshan Yao and Yong Zhu. Wearable multifunctional sensors using printed stretchable conductors made of silver nanowires. *Nanoscale*, 6(4):2345–2352, 2014.
- [165] Sehyuk Yim, Shuhei Miyashita, Daniela Rus, and Sangbae Kim. Teleoperated micro-manipulation system manufactured by cut-and-fold techniques. *IEEE Transactions on Robotics*, 33(2):456–467, 2017.
- [166] Hironori Yoshida, Takeo Igarashi, Yusuke Obuchi, Yosuke Takami, Jun Sato, Mika Araki, Masaaki Miki, Kosuke Nagata, Kazuhide Sakai, and Syunsuke Igarashi. Architecture-scale human-assisted additive manufacturing. *ACM Transactions on Graphics (TOG)*, 34(4):1–8, 2015.
- [167] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*, 2020.
- [168] Allan Zhao, Jie Xu, Mina Konaković Luković, Josephine Hughes, Andrew Spielberg, Rus Daniela, and Wojciech Matusik. Robogrammar: Graph grammar for terrain-optimized robot design. *ACM Transactions on Graphics (TOG)*, 39(6), 2020.
- [169] Lifeng Zhu, Weiwei Xu, John Snyder, Yang Liu, Guoping Wang, and Baining Guo. Motion-guided mechanical toy modeling. *ACM Transactions on Graphics (TOG)*, 31(6):1–10, 2012.