



POZNAN UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATIONS

Institute of Computing Science

DOCTORAL THESIS

Composite neurosymbolic architectures for
learning and reasoning from real and abstract
data

mgr inż. Jakub Bednarek

Supervisor
prof. dr hab. inż. Krzysztof Krawiec

Poznań, 2026

*Niniejszą pracę dedykuję przyszłym pokoleniom, dla których sztuczna
inteligencja będzie codziennością.*

*Chciałbym złożyć podziękowania osobom, bez których ta praca nigdy by nie
powstała.*

*Przede wszystkim prof. dr. hab. inż. Krzysztofowi Krawcowi, za nieocenione
wsparcie w trakcie prowadzenia wspólnych badań, wnikliwe dyskusje i
wskazówki, które pozwoliły mi rozwijać się zawodowo i osobiście.*

*Mojej ukochanej żonie Monice, za oparcie w trudnych chwilach oraz za
pokazanie mi świata, który inspirowuje i motywuje mnie do rozwoju oraz
realizacji celów.*

Mojej rodzinie i najbliższym, za niezachwiane wsparcie i wiarę we mnie.

*I dedicate this thesis to future generations, for whom artificial intelligence will
be an everyday reality.*

*I would like to express my gratitude to the people without whom this work
would never have been possible.*

*First and foremost, to Professor Krzysztof Krawiec, for his invaluable support
during our collaborative research, for the insightful discussions and guidance
that have allowed me to grow both professionally and personally.*

*To my beloved wife Monika, for support in difficult times, and for showing me
a world that inspires and motivates me to grow and pursue my goals.*

To my family and loved ones, for their unwavering support and belief in me.

Abstract

The scientific pursuit of artificial intelligence has historically alternated between two distinct paradigms: connectionist, which excels at intuitive pattern recognition but lacks casual transparency, and classical symbolic reasoning, which provides rigorous logical consistency but struggles with the stochasticity of the real world. To resolve this dichotomy, the thesis evaluates composite **neuro-symbolic architectures** designed to learn and reason in both empirical (real-world) and generative (abstract) domains.

At the core of this research is the hypothesis of **structural isomorphism**—the proposition that the underlying mechanisms required to transform physical signals into embodied actions are conceptually and structurally identical to those required to transform visual abstractions into logical programs. To validate this hypothesis, the thesis proposes a unified framework structured around three fundamental cognitive pillars: Sensing, Acting, and Problem Solving.

- **Sensing:** This domain frames Sensing as an active source of discrete symbolic data rather than a passive measurement tool. Deep learning models—employing convolutional and recurrent neural networks—are first introduced to classify and cluster terrain semantics (such as gravel, sand, and asphalt) directly from raw 6D force-torque time-series signals collected during legged locomotion. This perception layer is subsequently integrated into a real-time control loop through a haptic sequential Monte Carlo localization framework. Tested on the ANYmal robot platform, this haptic particle filter fuses symbolic terrain classes with geometric contact constraints against a prior map to bound odometry and yaw drift in vision-denied environments.
- **Acting:** To address the execution bottlenecks and sample inefficiency of traditional analytical solvers and reinforcement learning, a dual-agent **Oracle–Apprentice knowledge distillation framework** is developed. First applied to a quadruped equipped with rigid foot, a neural network (the Apprentice) is trained on synthetic elevation maps to map terrain geometry directly to foothold selection scores, internalizing the non-linear kinematic and collision constraints of a computationally intensive analytical planner (the Oracle). This architecture is then extended to a mechanically adaptive soft foot morphology, where the learned policy captures the non-linear soft-body deformation dynamics. This adaptation reduces planning times from seconds to milliseconds, demonstrating zero-shot transfer from synthetic data to physical hardware.
- **Problem Solving:** Transposing these principles to higher cognitive functions, the sensorimotor loop is applied directly to abstract workspaces. The technical baseline is established by proposing the Structure-Aware Program Synthesis framework, showing that a connectionist architecture can map natural language specifications into syntactically correct abstract syntax tree tokens without rule-based post-processing. This approach is then expanded into the TransCoder architecture to address the heterogeneous visual grid puzzles of the Abstract Reasoning Corpus. TransCoder couples raster self-attention encoders with a variational program generator operating over a rigid Domain Specific Language. This process is driven by a **task generation mechanism** utilizing a “learning from mistakes” paradigm. By re-contextualizing failed execution traces as successful solutions to newly synthesized tasks, the system establishes a learning gradient through self-simulation, serving as a cognitive counterpart to physical sensorimotor feedback.

This thesis validates **generative simulation** as a scalable substitute for large, human-annotated datasets. Integrating neural learning within structured symbolic boundaries—ranging from physical equations of motion to formal grammars—provides a pathway toward self-correcting agents capable of zero-shot generalization in real and abstract worlds.

More broadly, this work addresses the limitations of purely data-driven end-to-end learning. The resulting architecture grounds high-dimensional statistical pattern matching (System 1) within the symbolic structures of human reason (System 2). Because logical simulation yields a learning gradient comparable to physical feedback, reasoning functions here as a generalized form of action. These insights provide the motivation to surpass static training distributions, allowing autonomous neuro-symbolic agents to learn dynamically from execution errors across both physical and logical domains.

Streszczenie

Badania nad sztuczną inteligencją historycznie oscylowały między dwoma odrębnymi paradygmatami: koneksjonistycznym głębokim uczeniem, które doskonale radzi sobie z intuicyjnym rozpoznawaniem wzorców, lecz cierpi na brak przejrzystości przyczynowo-skutkowej, oraz klasycznym wnioskowaniem symbolicznym, zapewniającym spójność logiczną, lecz słabo radzącym sobie ze stochastyczną naturą świata rzeczywistego. Aby przełamać tę dychotomię, w niniejszej rozprawie poddano ocenie złożone **architektury neurosymboliczne**, zaprojektowane do jednoczesnego uczenia się i wnioskowania zarówno w domenach empirycznych (rzeczywistych), jak i generatywnych (abstrakcyjnych).

U podstaw podjętych badań leży hipoteza „**structural isomorphism**” (izomorfizmu strukturalnego)—założenie, że mechanizmy do transformacji sygnałów fizycznych w rzeczywiste akcje są koncepcyjnie i strukturalnie tożsame z mechanizmami mapowania wizualnych abstrakcji na programy logiczne. W celu weryfikacji tej hipotezy zaproponowano spójną strukturę systemu zorganizowaną wokół trzech fundamentalnych filarów kognitywnych: Sensing, Acting oraz Problem Solving.

- **Sensing:** W tej domenie Sensing przyjmuje funkcję źródła dyskretnych danych symbolicznych, zamiast pasywnego narzędzia pomiarowego. W pierwszej kolejności zostały zaprezentowane modele głębokiego uczenia, wykorzystujące konwolucyjne oraz rekurencyjne sieci neuronowe, w zadaniu klasyfikacji i klastrowania typów podłoża (takich jak żwir, piasek czy asfalt) bezpośrednio na podstawie surowych sześciowymiarowych szeregów czasowych sił i momentów obrotowych, rejestrowanych podczas ruchu robota krocącego. Ta warstwa percepcji następnie zintegrowana została z pętlą sterowania w czasie rzeczywistym poprzez haptyczny algorytm lokalizacji oparty na sekwencyjnej metodzie Monte Carlo. Haptyczny filtr cząsteczkowy, przetestowany na platformie robotycznej ANYmal, integruje przewidywane klasy terenu z geometrycznymi ograniczeniami kontaktu w odniesieniu do znanej mapy terenów otoczenia, co pozwala skutecznie ograniczyć dryf odometrii oraz obrotu w warunkach całkowitego braku widoczności.
- **Acting:** Aby wyeliminować ograniczenia obliczeniowe oraz niską efektywność próbkowania tradycyjnych solverów analitycznych i uczenia ze wzmocnieniem, opracowano dwuagentowy schemat destylacji wiedzy **Oracle–Apprentice**. Początkowo—dla robota wyposażonego w sztywną stopę—sieć neuronowa (Apprentice) jest trenowana na syntetycznych mapach wysokości, ucząc się bezpośredniego mapowania geometrii terenu na natychmiastowy (ang. *single-shot*) wybór punktów podparcia. Proces ten pozwala na internalizację nieliniowych ograniczeń kinematycznych i kolizyjnych wymagającego obliczeniowo planisty analitycznego (Oracle). Architektura tę rozszerzono następnie na adaptacyjną morfologię deformowalnych stóp, gdzie wyuczona polityka sterowania skutecznie odtwarza złożoną, nieliniową dynamikę odkształceń. Podejście to redukuje czas planowania z rzędu sekund do poziomu milisekund, demonstrując transfer typu *zero-shot* z danych syntetycznych bezpośrednio na fizyczny sprzęt.
- **Problem Solving:** Przenosząc te zasady na wyższe funkcje kognitywne, pętla sensoryczno-motoryczna została zaadaptowana bezpośrednio do przestrzeni abstrakcyjnej. Początkowo, zaproponowano architekturę Structure-Aware Program Synthesis, pokazując, że czysto koneksjonistyczna architektura jest w stanie mapować specyfikacje wyrażone w języku naturalnym na strukturalnie poprawne drzewa składniowe bez potrzeby stosowania dodatkowych heurystyk naprawiających. Podejście to rozwinięto następnie w architekturze TransCoder, stworzonej z myślą o rozwiązywaniu heterogenicznych, wizualnych

zadań abstrakcyjnych, pochodzących ze zbioru Abstract Reasoning Corpus. TransCoder łączy rastrowe enkodery wykorzystujące mechanizm *self-attention* z wariacyjnym generatorem programów opierającym działanie o język dziedzinowy. Proces ten napędzany jest przez **mechanizm generowania zadań**, wykorzystujący paradygmat „uczenia się na błędach”. Poprzez zmianę kontekstu, błędnie rozwiązane zadania przedstawione zostały jako nowe syntetyczne zadania, za pomocą których system buduje gradient uczenia w drodze automatycznej symulacji—pełniąc rolę kognitywnego odpowiednika fizycznego sprzężenia zwrotnego.

Niniejsza praca weryfikuje **symulację generatywną** jako skalowalną alternatywę dla dużych, anotowanych przez człowieka zbiorów danych. Osadzenie uczenia neuronowego w sztywnych ramach symbolicznych—od fizycznych równań ruchu po gramatyki formalne—wytycza ścieżkę rozwoju w kierunku samokorygujących się agentów zdolnych do uogólniania wiedzy w świecie rzeczywistym i abstrakcyjnym.

W szerszym kontekście praca ta mierzy się z ograniczeniami czysto statystycznego uczenia w paradygmacie *end-to-end*. Wypracowana architektura osadza wysokowymiarowe, statystyczne rozpoznawanie wzorców (System 1) w strukturach symbolicznych ludzkiego rozumowania (System 2). Ponieważ symulacja logiczna generuje gradient uczenia porównywalny z fizycznym sprzężeniem zwrotnym, wnioskowanie funkcjonuje tutaj jako uogólniona forma podejmowania akcji. Wnioski te stanowią motywację pod przekroczenie ograniczeń statycznych rozkładów danych treningowych, umożliwiając autonomicznym agentom neurosymbolicznym dynamiczną naukę na błędach zarówno w domenie fizycznej, jak i logicznej.

Contents

Abstract	v
Streszczenie	vii
Glossary	xvii
1 Introduction	1
1.1 The evolution of AI: learning, reasoning, and cognition	1
1.1.1 The shift to learning models	1
1.1.2 What is reasoning in AI?	1
1.1.3 The cognitive science perspective	2
1.2 The data dichotomy: real vs. abstract domains	3
1.2.1 The historical focus on real data	3
1.2.2 The bottleneck of real data	3
1.2.3 The rise of abstract data	4
1.3 Embodiment and intelligence: robotics as the foundation	4
1.3.1 The motivation for embodied AI	5
1.3.2 The evolution of AI in robotics	5
1.3.3 The physical world as a starting point	6
1.3.4 The friction of reality: limitations of physical exploration	7
1.4 Bridging the gap: the neuro-symbolic hypothesis	8
1.4.1 Intelligence as search: the problem space	8
1.4.2 The need for meaningful tools: the symbol grounding problem	8
1.4.3 The neuro-symbolic synthesis: Sensing, Acting, and Problem Solving	8
1.5 Overarching research objectives and hypothesis	9
1.6 Organization of the thesis	10
1.6.1 The structural blueprint	10
1.6.2 Thesis guidebook	12
I Sensing: from physical interaction to symbolic information	15
2 Preliminaries	17
2.1 Introduction to Sensing	17
2.1.1 Sensing in embodied systems	17
2.1.2 Haptic sensing as the basis of a cognitive system	19
2.2 Key challenges in robotic sensing	19
2.2.1 Data acquisition	19
2.2.2 Task engineering	19
2.3 Sensing in the context of a neuro-symbolic system	20
2.3.1 A bridge for a human-neural network interaction	21
3 What am I touching? Learning to classify terrain via haptic sensing	23
3.1 Motivation	23
3.1.1 Related work	23
3.2 The environments	24
3.2.1 The ETH dataset and robot	25

3.2.2	The PUT dataset and robot	25
3.3	Methods	26
3.3.1	Fixed-length step model	26
3.3.2	Variable-length step model	27
3.3.3	Training classification models	28
3.3.4	Signals clustering	28
3.3.5	Training the clustering model	29
3.4	Results	30
3.4.1	Conclusions	31
3.5	Implications	32
4	Navigating by touch: haptic Monte Carlo localization via geometric sensing and terrain classification	33
4.1	Motivation	33
4.1.1	Related work	34
4.2	Details	36
4.2.1	Problem statement	36
4.2.2	Proposed method	37
4.2.3	Implementation	40
4.3	Results	41
4.3.1	Discussion	48
4.3.2	Conclusions	52
4.4	Implications	52
5	Conclusions	53
5.1	Summary of contributions	53
5.2	Implications and broader context	53
II	Acting: learning physical policies from synthetic data	55
6	Preliminaries	57
6.1	Introduction to Acting	57
6.1.1	Acting in embodied systems	58
6.1.2	The Oracle and the Apprentice: a paradigm for learning	58
6.1.3	Objective: learning from synthetic data	60
6.2	Key challenges in learning robot actions	61
7	Single-shot foothold selection and constraint evaluation for quadruped locomotion	63
7.1	Motivation	63
7.1.1	Related work	64
7.1.2	Approach and contribution	65
7.2	Details	65
7.2.1	Inference procedure	67
7.2.2	Dataset	68
7.2.3	Convolutional Neural Network	70
7.2.4	Training the network	71
7.3	Results	71
7.3.1	Conclusions and future work	74
7.4	Implications	74

8	CNN-based foothold selection for mechanically adaptive soft foot	75
8.1	Motivation	75
8.1.1	Related Work	76
8.1.2	Approach and Contribution	77
8.2	Details	77
8.2.1	General architecture	77
8.2.2	Mechanically adaptive soft foot	78
8.2.3	Foothold selection with polynomial fitting	78
8.2.4	Convolutional Neural Network	80
8.3	Results	81
8.3.1	Conclusions and Future Work	85
8.4	Implications	85
9	Conclusions	87
9.1	Summary of contributions	87
9.2	Implications and broader context	87
III	Problem Solving: grounding abstract reasoning in logical execution	89
10	Preliminaries	91
10.1	Introduction to Problem Solving	91
10.1.1	The nature of abstract problems	92
10.1.2	Adaptation of symbolism to Problem Solving	92
10.1.3	The objective: automatic construction of the problem space	94
10.2	Key challenges in abstract Problem Solving	94
11	Ain't nobody got time for coding: structure-aware program synthesis from natural language	97
11.1	Motivation	97
11.1.1	Related work	97
11.2	Details	98
11.3	Results	102
11.3.1	Discussion	105
11.3.2	Conclusion	107
11.4	Implications	107
12	Solving abstract reasoning problems with neuro-symbolic program synthesis and task generation	109
12.1	Motivation	109
12.1.1	Abstract Reasoning Corpus	110
12.1.2	Related work	111
12.2	Details	113
12.2.1	The perception module	113
12.2.2	Solver	115
12.2.3	Workspace	116
12.2.4	The symbolic interface	117
12.2.5	Program generator	120
12.2.6	Training	121
12.2.7	Learning from mistakes	122

12.3 Results	123
12.3.1 Conclusions and future work	128
12.4 Appendix	130
12.4.1 Specification of the DSL	130
12.4.2 Examples of synthetic data	133
12.5 Implications	133
13 Conclusions	135
13.1 Summary of contributions	135
13.2 Implications and broader context	135
IV Thesis summary	137
14 Discussion	139
14.1 Summary of key findings	139
14.1.1 Validation of symbolic sensing (Part I)	139
14.1.2 Validation of abstract learning for physical policies (Part II)	140
14.1.3 Validation of structural isomorphism in reasoning (Part III)	140
14.1.4 Cross-cutting finding: the meta-structural role of constraints	141
14.2 Interpretation of results	142
14.2.1 The isomorphism of embodied and cognitive navigation	142
14.2.2 The efficacy of symbolic interface as inductive biases	142
14.2.3 Generative simulation as a substitute for experience	143
14.3 Discussion of implications	143
14.3.1 Implications for neuro-symbolic AI: the role of structure	143
14.3.2 Implications for robotics: autonomy in extreme environments	144
14.3.3 Theoretical implications: grounding symbols in action	145
14.4 Strengths and limitations of the study	145
14.4.1 Strengths of the research design	146
14.4.2 Limitations and boundaries	146
14.5 Directions for future research	147
14.5.1 Automated discovery of symbolic primitives (DSL learning)	147
14.5.2 Online adaptation and continual learning	148
14.5.3 Closing the loop: from abstract to physical and back	148
15 Conclusions	151
15.1 Restatement of the research problem and aims	151
15.2 Synthesis of the thesis	152
15.3 Statement of contribution and significance	152
15.3.1 A framework for symbolic haptic perception (Part I)	153
15.3.2 The Oracle-Apprentice framework distillation method for soft robotics (Part II)	153
15.3.3 The TransCoder and logical proprioception architecture (Part III)	153
15.3.4 Validation of generative simulation as a data substitute	153
15.4 Final concluding remarks	154

List of Figures

1.1	A conceptual comparison between purely data-driven learning models and rule-based reasoning systems.	2
1.2	The dichotomy of data in AI.	5
1.3	The evolution of robotic control paradigms.	6
2.1	Decision-making loop	18
2.2	The experimental setup for ANYmal robot.	20
3.1	ANYmal and Messor robots.	23
3.2	Terrain types in ETH dataset.	24
3.3	Terrain types in PUT dataset.	25
3.4	Recurrent neural network for terrain classification.	26
3.5	Convolutional neural network for terrain classification.	27
3.6	Clustering architecture for terrain identification.	29
3.7	Visualization of clustered terrain samples.	30
4.1	ANYmal robot in a demanding environment.	33
4.2	Terrain map for ANYmal localization.	36
4.3	Comparison between contact measurements and terrain class map representations.	38
4.4	Localization flow diagram.	39
4.5	Neural network architecture for terrain classification in localization problem.	42
4.6	Localization experiment 1: setup.	43
4.7	Localization experiment 1: recorded trajectories.	44
4.8	Localization experiment 1: estimated errors.	45
4.9	Localization experiment 2: overview.	46
4.10	ANYmal haptic localization experiment	47
4.11	Localization experiment 3: recorded trajectories.	49
4.12	Localization experiment 3: estimated errors.	50
4.13	Distribution of localization particles across experiments.	51
6.1	Oracle vs Apprentice	59
7.1	The foothold selection problem.	64
7.2	The local map during foothold selection.	66
7.3	Foothold selection procedure with CNN.	66
7.4	Local elevation map available during foothold selection.	67
7.5	The foothold selection training example.	68
7.6	ERF network for foothold selection.	69
7.7	The foothold selection in different scenarios.	69
7.8	Model inference results for foothold selection.	72
7.9	The foothold selection experiment in real-world scenario.	73
8.1	The foothold selection with CNN method.	75
8.2	The foothold selection CNN-based system.	77
8.3	CAD of the main components and dimensions of SoftFoot-Q.	79
8.4	Parameters of the model and corresponding shape of the foot.	79
8.5	Example soft foot configurations.	80

8.6	Illustration of the convex hull computation.	80
8.7	Comparison between two foothold selection methods.	82
8.8	Obstacles and feet trajectories during experiments with the CNN-based foothold selection for the adaptive soft foot.	83
8.9	The foothold selection on irregular terrains.	84
10.1	The problem space conceptualization	93
11.1	Overall architecture of SAPS.	99
11.2	The modified DRNN cell.	100
11.3	Propagation of the hidden states and the latent vector during decoding in DRNN.	101
11.4	The grammar of the AlgoLisp DSL.	102
12.1	Examples from the Abstract Reasoning Corpus Dataset.	110
12.2	The overall architecture of TransCoder.	112
12.3	The autoencoder architecture used to pre-train the raster encoder in Perception.	114
12.4	Execution process of an example program applied to an input raster (scene).	120
12.5	Symbols selection during program generation with TransCoder.	121
12.6	The state diagram of TransCoder’s training, including learning from mistakes.	123
12.7	The dynamics of the TransCoder experiments.	127
12.8	Examples of (task, program) pairs synthesized by TransCoder.	129

List of Tables

1.1	The diagram summarizing the key contributions of this thesis and its organization, with emphasis on the neuro-symbolic integration.	13
3.1	The fully-connected neural model for processing extracted FFT features. . .	28
3.2	Terrain classification results.	31
4.1	Experiment 1: Estimation performance.	43
4.2	Experiment 3: Estimation performance.	48
7.1	Model performance for foothold selection.	71
8.1	Foothold selection models benchmark.	82
8.2	Performance details of ERF model.	83
11.1	Percentage of perfectly synthesized programs for SAPS configurations trained from scratch.	103
11.2	Percentage of perfectly synthesized programs for pretrained SAPS configurations.	104
11.3	Percentage of programs passing all tests for comparing models.	104
11.4	Percentage of programs passing all tests for SAPS model.	104
11.5	The effects of modifications of NL specification on SAPS outcomes.	106
12.1	The list of types available in the TransCoder’s DSL.	118
12.2	Definitions of the operations available in the TransCoder’s DSL.	119
12.3	TransCoder’s blacklisting rules.	125
12.4	TransCoder’s normalization rules.	125
12.5	SynthRate and ARCRate achieved by TransCoder.	127
12.6	Performance of TransCoder snapshots in cycles.	128
12.7	The list of predefined symbol keys available in the TransCoder’s DSL. . . .	130
12.8	Definitions of the operations available in the TransCoder’s DSL.	131
12.9	The examples of synthesized programs across successive exploration phases for Run 1.	134

Glossary

Acting The structural component responsible for selecting or generating grounded interventions that change an environment’s state. This execution framework encompasses physical locomotion policies and symbolic program generation.

Apprentice A reactive learned policy—typically implemented as a neural network—trained to approximate Oracle decision boundaries for real-time inference.

Oracle An analytical model or explicit policy that provides high-fidelity, computationally intensive evaluation of candidate actions or states. This component provides the supervision required to distill a lower-latency learned policy.

Oracle-Apprentice framework A dual-agent optimization framework where an analytical Oracle provides high-fidelity training supervision and a neural Apprentice compresses this behavior into a reactive policy. This paradigm enables lower-latency execution suitable for real-time deployment.

Problem Solving The component representing the closed-loop integration of sensing and acting within a structured problem space, aiming to solve a predefined task.

Problem space The structured representation of a task environment. Within the problem solving framework, it covers the initial state, goal state, available operators, and all intermediate states (Newell, Simon, et al., 1972).

Sensing The structural component responsible for transforming high-dimensional input into task-relevant symbolic state information. This process is applied first to haptic perception and subsequently extended to the symbolic perception of abstract problem states.

Structural isomorphism The hypothesis that the architecture required to ground physical signals into embodied action is structurally equivalent to that required to ground abstract syntax into logical reasoning.

Symbol grounding problem The problem of how abstract syntactic symbols acquire semantic meaning. This definition focuses on the mechanism by which formal entities link to action, perception, and physical or logical domains (Harnad, 1999; Searle, 1990).

Symbolic interface The formal, structured layer that constrains and organizes an agent’s interaction with a domain. These interfaces encompass terrain semantics, analytical physics models, and domain-specific languages, establishing the structural framework that bounds neural inference.

TransCoder An architecture designed for abstract reasoning that integrates neural perception with symbolic program synthesis over a domain-specific language. The system optimizes parameters by training directly on program execution traces (J. Bednarek & Krawiec, 2026).

Chapter 1

Introduction

1.1 The evolution of AI: learning, reasoning, and cognition

Artificial intelligence (AI) research has historically shifted between two philosophies: engineering explicit logical rules and identifying implicit statistical patterns. Contextualizing the neuro-symbolic architectures proposed in this thesis requires examining how the field arrived at this state. Crucially, this requires distinguishing between the capacity to learn from data and the capacity to reason.

1.1.1 The shift to learning models

In its nascent decades, AI research was dominated by what is colloquially termed Good Old-Fashioned AI (GOFAI). These classical systems relied on symbolic logic, explicit knowledge representations, and rule-based search algorithms (Russell et al., 2020). GOFAI achieved notable success in highly constrained, deterministic environments, such as chess or automated theorem proving. However, these systems struggled when confronted with the noisy, continuous nature of real-world data. The bottleneck lay in manual knowledge engineering: human experts had to anticipate and codify every environmental variation, a task that quickly led to a combinatorial explosion.

This paradigm shifted with the rise of modern machine learning (ML) and deep learning (DL) (LeCun et al., 2015). Instead of relying on explicit rules, these architectures leverage backpropagation and increased computational power to learn hierarchical representations directly from unstructured data. The rapid expansion of DL—particularly within computer vision (CV), natural language processing (NLP), and speech recognition—was largely driven by this capacity to bypass manual feature extraction.

However, this success in pattern recognition obscures a fundamental limitation. Deep neural networks operate as function approximators that interpolate across high-dimensional data manifolds. Crucially, interpolation is not synonymous with comprehension. As these models scale, researchers increasingly question whether mapping inputs to outputs based on historical statistical correlations enables genuine generalization, or simply sophisticated memorization (Marcus, 2018).

1.1.2 What is reasoning in AI?

Formalizing this critique requires separating *learning* from *reasoning*. In modern AI, learning generally refers to optimizing parameters to minimize a loss function over a data distribution—a process suited for capturing surface-level perceptual patterns. By contrast, reasoning involves the algebraic manipulation of structured knowledge. This capacity allows a system to infer causality, answer novel questions, or plan actions toward a goal (Frappier, 2018).

Cognitive psychology models this dichotomy using “dual process theory,” which distinguishes between System 1 and System 2 cognition (Kahneman, 2011). System 1 processes

information rapidly, automatically, and intuitively, mirroring a forward pass in a deep neural network during image classification. By contrast, System 2 is slow, deliberate and analytical, akin to an algorithm executing a multi-step mathematical proof. As illustrated in Figure 1.1, modern DL remains largely confined to System 1 execution.

Although large language models (LLMs) can generate text that mimics logical deduction, critics argue they operate primarily as “stochastic parrots”—probabilistically chaining words together without access to an underlying causal model of the phenomena they describe (Bender et al., 2021). This limitation reflects the classic symbol grounding problem (Harnad, 1999; Searle, 1990): the philosophical observation that a system manipulating meaningless syntactic symbols cannot inherently possess semantic understanding. Genuine reasoning requires moving beyond statistical correlation. It demands grounding abstract tokens in actionable meaning, performing counterfactual thinking (“What would happen if?”), adhering to logical constraints, and manipulating variables within a robust internal model of the world.

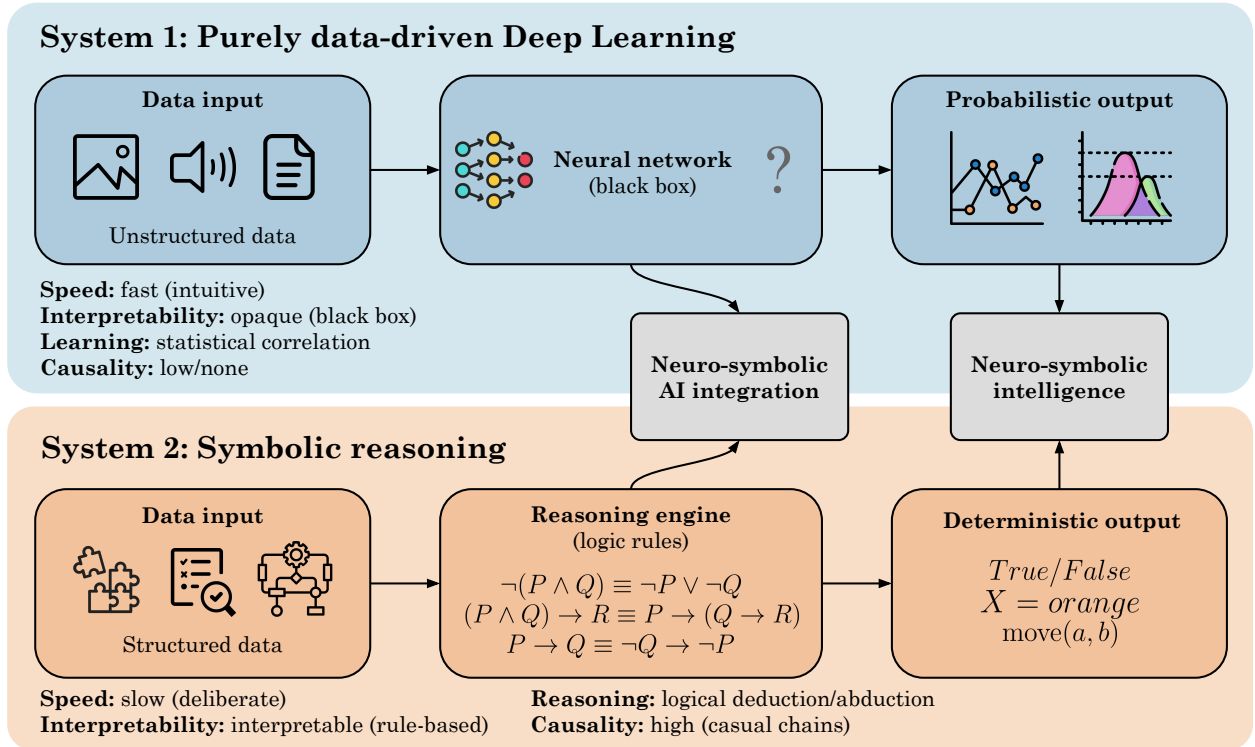


Figure 1.1: A conceptual comparison between purely data-driven learning models (analogous to fast, intuitive System 1 cognition) and rule-based reasoning systems (analogous to deliberate System 2 cognition). As illustrated, neuro-symbolic AI integration is not merely a late-stage fusion of final answers. Rather, it actively bridges the two pipelines so that each paradigm steers the other during inference—neural intuition guides the logical search, while symbolic rules constrain statistical predictions. Neuro-symbolic integration thus operates not only at the convergence of probabilistic and deterministic outputs, but as a direct result of this joint processing.

1.1.3 The cognitive science perspective

If current AI models lack the architecture for true reasoning, where should the field look for a blueprint? This thesis turns to psychology and cognitive science, which demonstrate that human intelligence does not stem solely from large-scale data absorption. Instead, humans possess and develop structured “core knowledge”—including intuitive physics, basic geometry, and an understanding of intentional agents (Spelke & Kinzler, 2007). Building upon this

foundation, human cognition is characterized by the capacity to reason by analogy (Gentner, 1983). By mapping relational structures from familiar experiences to novel domains, individuals interpret and construct abstract concepts. This analogical capacity enables rapid generalization and problem-solving in novel situations without exhaustive, trial-and-error learning.

Cognitive scientists argue that to build machines that learn and think like people, AI must move beyond pattern recognition and incorporate model-building capabilities (Lake et al., 2017). Human intelligence relies on constructing generative, causal models of the environment that can be mentally simulated to predict action consequences. This capacity is deeply rooted in embodiment. Concepts and abstractions do not exist in isolation—they are directly grounded in sensorimotor experiences (Barsalou, 1999).

This cognitive perspective forms the core premise of this thesis. It suggests that scalable, robust AI cannot be achieved solely by scaling data and network parameters. Instead, architectures must separate the “intuitive” perception of the world from the “logical” manipulation of its rules. This distinction requires a structural integration of neural learning and symbolic reasoning.

1.2 The data dichotomy: real vs. abstract domains

As discussed in Section 1.1, the transition from classical, rule-based systems to modern learning algorithms altered how AI models acquire knowledge. Instead of executing explicitly programmed logic, these models derive their behavior from empirical data. The composition and structure of this training data ultimately define the operational boundaries of the resulting system. Analyzing the need for integrated reasoning capabilities thus requires examining how data origins shape these constraints. An illustrative contrast emerges between learning from the noisy observations of the physical world and learning from the rigorous, generative rules of abstract domains.

1.2.1 The historical focus on real data

For the past decade, the dominant paradigm in AI has centered on scaling the volume of real-world data. This data—comprising images, audio, and text—serves as an unstructured representation of the observable world. As outlined in Section 1.1.1, deep learning progress was largely driven by the availability of large-scale datasets like ImageNet (Deng et al., 2009) alongside significant increases in computational power.

This focus was primarily pragmatic. To build systems capable of operating in human-centric environments, researchers required models that could handle real-world ambiguity and variance. By training on billions of tokens of text or millions of annotated images, models learned to interpolate across the manifold of human experience, leading to high accuracy in CV and NLP (Devlin et al., 2018). Within this data-driven paradigm, however, intelligence was implicitly defined as the capacity to match or surpass human-level performance on highly specific tasks (He et al., 2015; Stallkamp et al., 2012). This focus has shifted research priorities. Significant computational and financial resources are now expended striving for marginal, state-of-the-art improvements on well-established leaderboards—an optimization process that often takes precedence over developing genuine algorithmic breakthroughs (Schlangen, 2020).

1.2.2 The bottleneck of real data

However, relying exclusively on empirical, real-world data presents distinct scaling and epistemic limits. The AI community has begun hitting what is termed the “data wall.”

Gathering standard, average-case data remains cost-effective; however, capturing the “long tail” of edge cases required for robust generalization is highly costly and logistically prohibitive.

Real data are also inherently observational rather than causal. For example, an image of a shattered glass does not explicitly contain the physical laws of momentum and fragility. Instead, it captures the statistical correlation between pixel arrangements and the underlying event. As argued in Section 1.1.2, this limitation prevents purely data-driven models from inferring underlying causal mechanisms.

This bottleneck is most acute in embodied domains such as robotics. In these environments, data collection cannot be accelerated beyond real-time constraints. Physical trial-and-error is inherently bounded by execution time, hardware degradation, and safety risks. Although deep reinforcement learning (RL) has shown promise, high sample inefficiency complicates training robust policies directly in the real world (Dulac-Arnold et al., 2019). Relying solely on empirical data thus limits the development of causal generalization.

1.2.3 The rise of abstract data

To address the limitations of real-world datasets, researchers have increasingly turned to *abstract data*. Unlike empirical data, which reflect observations of the physical world, abstract data—such as mathematics, formal logic, computer code, and Domain Specific Languages (DSLs)—structurally define the rules of an environment.

The rise of LLMs trained on code, such as Codex (M. Chen et al., 2021), illustrates this development. Code is not merely text; it represents an executable graph of causal relationships. When an agent synthesizes a program, that program can be compiled, executed, and evaluated. If it fails, the execution trace provides an unambiguous error signal. This capacity for self-verification allows AI to generate synthetic, abstract data through internal simulation, bypassing the physical limits of real-world collection. As shown in Figure 1.2, this contrast between noisy, observational empirical data and structured, executable abstract domains forms the core premise for the neuro-symbolic architectures developed in this thesis.

While real data capture the noise of the physical world, abstract data provide the logical structure required for reasoning. Once the rules of a DSL or a physics simulator are defined, an agent can evaluate diverse permutations of a problem and simulate counterfactual scenarios (Silver et al., 2017). This thesis posits that bridging these two domains—grounding the explicit causality of abstract rules within noisy perceptual inputs—is necessary for developing adaptable autonomous agents.

1.3 Embodiment and intelligence: robotics as the foundation

Having established the limitations of relying solely on unstructured data and ungrounded learning algorithms, we must identify an environment where the mechanisms of intelligence can be rigorously tested. Although this thesis explores both physical and abstract domains, it posits that physical embodiment offers a concrete environment for investigating these foundational mechanisms. This requirement draws attention toward robotics, which bridges abstract computation and real-world physical constraints. This field functions not just as an application domain, but as a baseline for developing the proposed neuro-symbolic architectures.

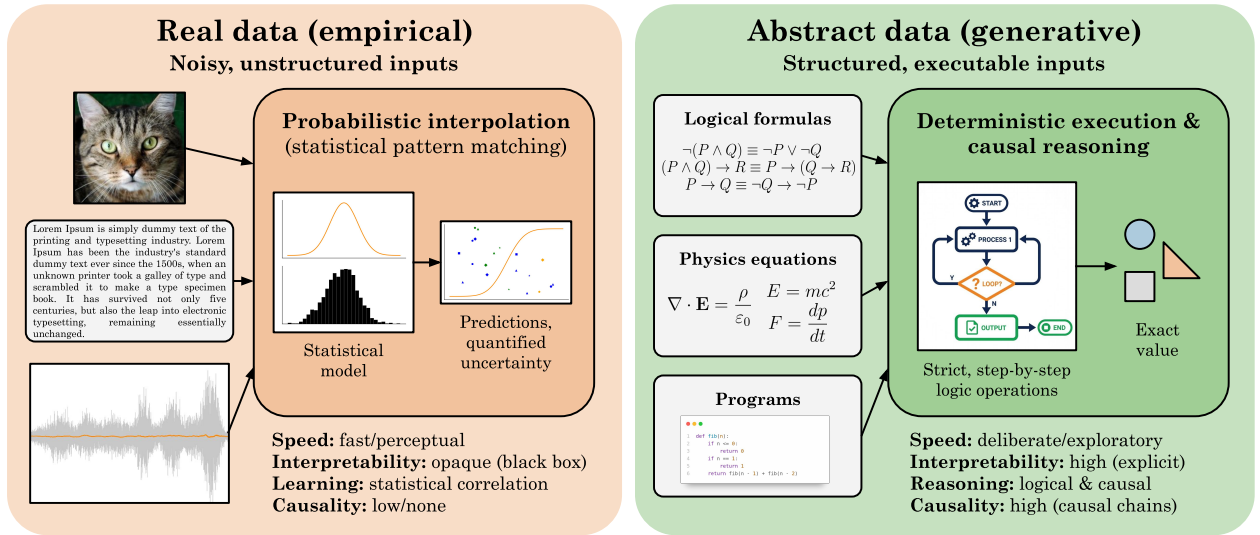


Figure 1.2: The dichotomy of data in AI. Real data facilitates perceptual pattern matching but lacks causal transparency. Abstract data, such as code and formal logic, provides executable rules and unambiguous feedback, enabling self-supervised exploration and logical reasoning.

1.3.1 The motivation for embodied AI

During the early decades of AI, researchers identified a fundamental contradiction now known as Moravec’s paradox (Moravec, 1988). This thesis notes that high-level cognitive tasks, such as logical deduction or chess, require minimal computation. Conversely, low-level sensorimotor skills such as walking or object recognition demand immense computational resources. This observation upends the conventional assumption that abstract reasoning is inherently more difficult than sensory perception.

This paradox illustrates a profound aspect of intelligence: abstract reasoning is an evolutionary development built upon millions of years of sensorimotor development. The evolutionary recency of abstract logic suggests its foundations lie in primitive processes—layers of intelligence that remain difficult to replicate in robotic systems. A purely cognitive model lacks this physical context, which requires spatial reasoning, intuitive physics, and causal interaction (Pfeifer & Bongard, 2006). Embodied AI demonstrates that an agent’s morphology and physical interactions provide the structural context for these cognitive processes. In robotics, systems must operate within physical constraints, making the domain a clear baseline for observing how computational models translate into physical consequence.

1.3.2 The evolution of AI in robotics

The history of robotic control mirrors the broader evolution of AI discussed in Section 1.1, although it is bounded directly by the physical limits of hardware. Traditionally, these systems relied on classical, analytical control approaches. These frameworks utilized mathematical models—such as inverse kinematics and optimal control theory—to calculate trajectory execution (Siciliano et al., 2010). Classical control provides formal safety guarantees and high precision in structured environments. However, these models degrade when deployed in unmodeled real-world settings.

Conversely, recent works have shifted toward learning-based frameworks, particularly deep RL. These methods map raw sensor data directly to actuator torques using end-to-end neural networks, bypassing explicit analytical models (Levine et al., 2015). Although deep RL provides high flexibility, it shares the epistemic deficiencies outlined in Section 1.2.2. These policies remain sample-inefficient, lack interpretability, and struggle to generalize outside their training distributions. As illustrated in Figure 1.3, robotics operates between the rigidity

of analytical symbols (classical control) and the opacity of neural intuition (pure learning). This tension motivates a neuro-symbolic synthesis.

Evolution of robotic control paradigms

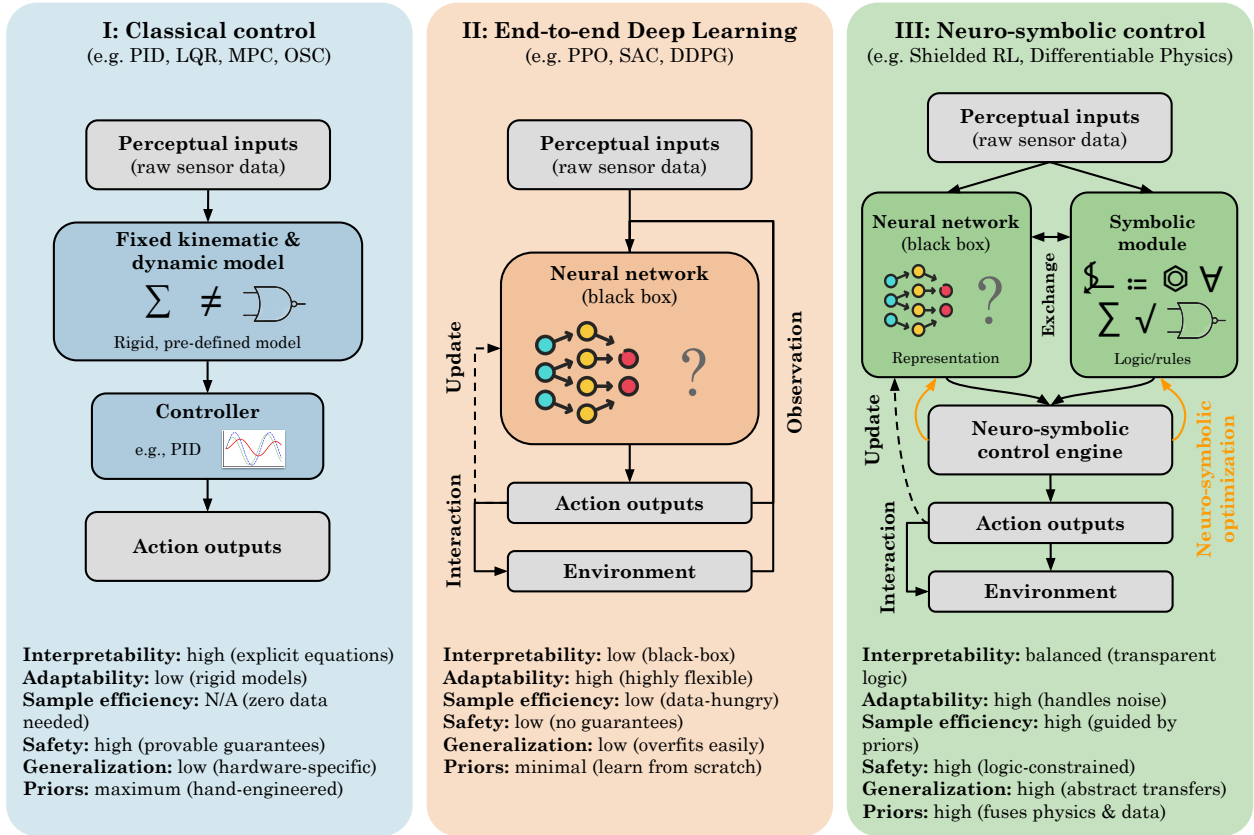


Figure 1.3: The evolution of robotic control paradigms, categorized by architectural configurations and core operational properties (interpretability, adaptability, sample efficiency, safety, generalization, and priors). **(I) Classical control** operates via a feed-forward pipeline where perceptual inputs are processed through fixed kinematic and dynamic models to a traditional controller (e.g., PID). This architecture ensures formal safety guarantees but limits environmental adaptability. **(II) End-to-end deep learning** maps inputs directly to actions via a neural network. The network interacts with the environment using closed-loop feedback, optionally updating its internal state, offering high adaptability at the expense of transparency and formal safety. **(III) Neuro-symbolic control** represents the proposed integrated framework, processing perceptual inputs concurrently through neural networks and symbolic modules. These components exchange representations to balance probabilistic optimization against deterministic constraints. A neuro-symbolic control engine balances data-driven task objectives against hard physical rules and logical constraints to formulate safe actions. This design aims to combine the adaptability and generalization of DL with the sample efficiency, interpretability, and formal safety guarantees inherent to classical priors.

1.3.3 The physical world as a starting point

Given that the ultimate aim of this thesis spans both physical perception and abstract reasoning (as detailed in Part III), why does the investigation explicitly begin in the physical realm of robotics?

The answer lies in a fundamental, yet often overlooked, advantage of reality: the physical world releases us from the need to manually construct a complete world model. When devel-

oping an abstract environment—such as a DSL or a virtual game—researchers must explicitly define every rule, logic gate, and boundary condition. If a rule is missing or contradictory, the abstract environment degrades, allowing the agent to exploit these architectural gaps.

The physical world, conversely, is invariant and self-consistent. Gravity operates without software errors, and physical phenomena such as friction exist independent of an agent’s internal model. Consequently, structural collisions yield immediate, deterministic feedback. Reality functions as an implicit, “pre-compiled” testbed.

Because it is an invariant and self-consistent system, the physical world serves as a baseline environment for developing the cognitive mechanisms required for complex problem-solving. Although abstract environments are highly varied, they remain difficult to define as mathematically concise and complete domains without introducing logical loopholes or omitted constraints. The physical world, by contrast, operates under invariant rules. When a robot interacts with this environment, the sensory feedback and the resulting consequences are absolute. Starting with robotics allows us to observe how an agent learns to ground symbols (e.g., the concept of a “slippery surface”) within the physical consequences of action to navigate its problem space. Demonstrating how neuro-symbolic architectures map physical signals into grounded symbolic representations is a necessary first step. Once verified in the physical domain, this structural mechanism can be mathematically extended to navigate the constructed realities of abstract logic.

1.3.4 The friction of reality: limitations of physical exploration

Although the consistency of the embodied environment makes it a reliable baseline for studying symbol grounding, it simultaneously introduces distinct practical constraints. The physical world defines the clear consequences of actions, but it limits open-ended exploration and large-scale problem-solving. This creates an operational tension in embodied AI. The environment that provides the most reliable ground truth is also the most restrictive to traverse.

The primary limitation stems from real-time execution constraints. Unlike an abstract simulation or a DSL—where an agent can execute thousands of logical operations or code compilations per second—a physical robot is bound by the clock of the universe. Every physical trial, whether it is a step on uneven terrain or grasping an object, consumes real time and physical energy. Consequently, relying on the physical world for the massive trial-and-error exploration demanded by modern RL algorithms results in prohibitive sample inefficiency (Kober et al., 2013).

Furthermore, the “friction of reality” introduces hardware degradation and safety constraints. In an abstract environment, a critical failure simply returns an error code or a negative reward. In the real world, a failure often means stripped gears, burnt actuators, or catastrophic damage to the robot and its surroundings (Garcia & Fernández, 2015). This forces engineers to constrain the agent’s problem space (Newell, Simon, et al., 1972) through safety protocols, preventing the agent from discovering novel, out-of-the-box solutions that might require high-risk physical exploration.

Finally, while the underlying rules of reality are consistent, the agent’s perception of them is not. As explored in Part I, robots face a state observability problem. They only have access to noisy, continuous, and often unreliable sensor streams, making it highly complex to extract discrete symbols required for high-level reasoning. Therefore, while physical embodiment is necessary to prove the foundational mechanics of neuro-symbolic grounding, scaling an agent’s problem-solving capacity requires abstracting these mechanics into virtual, generative spaces (as demonstrated in Part III), where the constraints of hardware, time, and physical risk are removed.

1.4 Bridging the gap: the neuro-symbolic hypothesis

The preceding sections outline a fundamental divergence in modern AI. Pure learning models excel at intuitive perception but lack the structural capacity for causal reasoning. Conversely, classical symbolic models possess rigorous causality but fail to adapt to unmodeled physical environments. This thesis hypothesizes that autonomous agency requires a structural synthesis of both paradigms. Understanding this integration requires formalizing the mechanics of intelligence through cognitive science.

1.4.1 Intelligence as search: the problem space

Newell and Simon proposed the *physical symbol system hypothesis*, establishing that human problem-solving is fundamentally framed as navigation through a problem space (Newell, Simon, et al., 1972). This space is defined by three core components: an initial state, a desired goal state, and a discrete set of *operators* (actions or rules) that transform one state into another. Intelligence, in this context, is the ability to successfully search for a valid trajectory of operators that bridges the initial state to the goal.

This structured model of cognition illustrates the limitations of pure DL frameworks. Standard neural networks typically map the initial state directly to the goal state through a single continuous function. Because they lack explicit, discrete operators, these networks lack the structural mechanism required to execute sequential search, verify intermediate states, or retrace a path when an execution error occurs. Traversal of complex, combinatorial problem spaces requires an agent equipped with such structured operators.

1.4.2 The need for meaningful tools: the symbol grounding problem

However, simply providing an agent with a set of operators is insufficient. To successfully navigate a problem space, an agent must treat these operators as valid, predictable “tools.” This introduces one of the most profound challenges in AI: the *symbol grounding problem*.

If these operators function as empty syntactic tokens—akin to the ungrounded characters in Searle’s Chinese Room argument (Searle, 1990)—the agent is essentially guessing. It cannot perform a deliberate search because it does not understand what the tools do. This thesis argues that symbols acquire meaning, and thus become useful tools for traversal, only when they are grounded in the *predictive consequences of action*. An agent understands the symbol “mud” not by matching its spelling, but by predicting that it causes physical slippage. Similarly, an agent understands the abstract coding operator `reverse()` not through statistical correlation, but by predicting that its execution flips an array. Only when symbols are grounded in actionable consequences can an agent confidently navigate a problem space.

1.4.3 The neuro-symbolic synthesis: Sensing, Acting, and Problem Solving

To implement this cognitive theory, this thesis proposes a composite neuro-symbolic architecture that operationalizes the dual-process framework introduced in Section 1.1.2. This synthesis relies on a functional division of labor between two distinct computational engines. The *symbolic interface* acts as the deliberate System 2; it provides the formal definitions and constraints of the problem space (e.g. the physics engine, the DSL, the rules of the environment) alongside discrete, executable operators. Conversely, the neural network functions as the System 1, serving as a heuristic function. It provides the “intuition” required

to select the right grounded tools, efficiently guiding the logical search through massive combinatorial spaces to avoid brute-force calculation.

This synthesis is formalized through three core capabilities: Sensing, Acting, and Problem Solving. Rather than treating these as isolated tasks, this work demonstrates how Sensing and Acting are first grounded in physical reality, and subsequently elevated to form the closed-loop engine of abstract Problem Solving.

Sensing: from physical perception to state extraction.

Sensing is the capacity to extract discrete, meaningful state representations from a complex environment for a defined task. Investigated in Part I (the physical realm), this capacity demonstrates how neural perception can extract symbolic terrain classes from raw haptic data. In Part III, this mechanism extends to the abstract realm: Sensing becomes the agent’s ability to interpret the symbolic state of a programmatic workspace and perceive the execution trace of a logic puzzle.

Acting: from physical locomotion to operator execution.

Acting is the functional ability to translate rules and symbols into a policy that modifies the environment state. Investigated in Part II (the physical realm), this capacity leverages the abstract “symbols” of analytical physics (Oracles) to train a neural network how to generate stable locomotion policies. In the abstract realm (Part III), acting transitions from commanding physical guides to synthesizing discrete code operations, using a DSL to actively manipulate the abstract environment.

Problem Solving: the synthesis in the abstract realm.

Problem Solving operates as the integration of Sensing and Acting within a closed cognitive loop. Analyzed in Part III, this capability couples continuous action generation—such as writing code—with the sensing of execution traces to traverse an abstract problem space. In this setting, symbols are grounded by the deterministic feedback of code execution rather than physical interaction. This loop allows the system to evaluate intermediate states, explore alternative trajectories, and solve combinatorial tasks within its internal model.

Structuring the research across these three capabilities allows this thesis to evaluate how integrating neural networks with symbolic rigor supports grounded machine intelligence. Under this framework, components verified within the physical domain are mathematically extended to abstract logic.

1.5 Overarching research objectives and hypothesis

The primary objective of this thesis is to design, implement, and validate composite neuro-symbolic architectures capable of robust learning and reasoning across both empirical (real-world) and generative (abstract) domains. As established in the preceding sections, a fundamental limitation persists in modern AI. Pure neural approaches lack the structural causality required for multi-step problem-solving, whereas pure symbolic approaches lack the perceptual adaptability to interface with continuous environments. Integrating neural networks with symbolic constraints—specifically through the capabilities of Sensing, Acting, and Problem Solving—addresses this gap.

This objective rests on the hypothesis that a *structural isomorphism* connects physical embodiment and abstract reasoning. In cognitive science and mathematics, an isomorphism

refers to a structural mapping where the fundamental relationships and operations between elements are preserved, even when the underlying domains are entirely distinct (Gentner, 1983). This thesis hypothesizes that the underlying computational architecture required to ground physical signals (e.g. translating haptic feedback into discrete terrain symbols) is conceptually and structurally identical to the architecture required to ground abstract syntax (e.g., translating visual problem states into executable logical programs).

To validate this hypothesis, this work evaluates whether a shared architectural framework—wrapping a neural heuristic function around a deterministic symbolic evaluator—can address core AI limitations across distinct domains.

- In the **physical realm** (Parts I & II), the symbolic evaluator is the unyielding physics of the real world (or its analytical surrogate). It grounds the neural network’s perception and actions through physical feedback, circumventing the bottleneck of real-world sample inefficiency and state observability.
- In the **abstract realm** (Part III), the symbolic evaluator is a DSL engine. It grounds the neural network’s generated problem-solving solutions through exact execution traces, mitigating the bottleneck of combinatorial search and logical hallucination.

If integrating these components yields consistent improvements in out-of-distribution generalization, self-supervised learning, and sample efficiency across both physical robotics and virtual logic puzzles, the hypothesis of structural isomorphism is supported. The core objective of this work extends beyond engineering an optimized robotic controller or a more capable code synthesizer. Instead, this research validates a unified framework for grounded machine cognition.

1.6 Organization of the thesis

To effectively navigate the intersection of robotics, ML, and cognitive science, this thesis follows the structural progression of the neuro-symbolic framework introduced in Section 1.4. The document is divided into three primary parts, each representing one of the core cognitive pillars: Sensing, Acting, and Problem Solving.

1.6.1 The structural blueprint

For consistency, each of the three main parts follows an identical structure:

- **Preliminaries:** Each part opens with a “Preliminaries” chapter to establish the domain-specific theoretical background and introduce key computational challenges, such as state observability in robotics or combinatorial explosion in abstract search. These chapters provide the cognitive science and psychological context motivating the localized technical solutions, grounding the contributing publications within the broader scope of the thesis.
- **Contribution Chapters:** The core chapters of each part present the technical contributions. These chapters are directly derived from publications produced during this doctoral research. **They contain the actual text of the articles, adapted for the needs of the dissertation, and all articles incorporated into this work were authored or co-authored by the writer of this thesis.**

- **Conclusions:** Each part concludes with a summary chapter that evaluates the findings against the neuro-symbolic challenges introduced in the preliminaries. This section also outlines the structural transition toward the next capability framework.

Following the three capabilities, the thesis concludes with a **Summary** (Part IV), synthesizing the research results across two chapters:

- **Discussion:** Chapter 14 evaluates the hypothesis of *structural isomorphism* by synthesizing empirical evidence from the physical and abstract domains. This evaluation determines whether a unified framework is supported, while addressing the limitations of the current approach and outlining potential avenues for future research.
- **Conclusions:** Chapter 15 concludes the thesis by situating technical validation within the broader development of AI. This section explores the necessity of grounding modern foundation models within structured computational methodologies. This framework frames the development of AI beyond statistical modeling, focusing on grounded agents capable of symbol comprehension, problem space traversal, and interaction layouts familiar to human cognition.

The following list details the publications authored or co-authored during this doctoral research. Each entry is annotated with its evaluation metrics, including the points awarded by the Polish Ministry of Science and Higher Education (Ministry points) and, where applicable, the CORE ranking classification.

The CORE (Computing Research and Education Association of Australasia) ranking is a widely recognized, premier evaluation metric for computer science conferences, categorizing venues into tiers (e.g., A*, A, B, and C) based on their prestige and impact. In the Polish academic evaluation system, this international ranking is directly translated into a standardized point scale, reflecting the quality of the publication venue.

The core publications upon which the contribution chapters of this thesis are explicitly built are emphasized in **bold**:

1. *Methods of Enriching the Flow of Information in the Real-Time Semantic Segmentation Using Deep Neural Networks* J. Bednarek, Piaskowski, and Bednarek, 2018; Ministry points: 20
2. *Ain't Nobody Got Time for Coding: Structure-Aware Program Synthesis from Natural Language* J. Bednarek, Piaskowski, and Krawiec, 2018; Ministry points: 0
3. *What Am I Touching? Learning to Classify Terrain Via Haptic Sensing* J. Bednarek, Bednarek, Wellhausen, et al., 2019; Ministry points: 20, CORE points: 70
4. *Robotic Touch: Classification of Materials for Manipulation and Walking* J. Bednarek, Bednarek, Kicki, and Walas, 2019; Ministry points: 20
5. *Single-Shot Foothold Selection and Constraint Evaluation for Quadruped Locomotion* Belter et al., 2019; Ministry points: 20, CORE points: 70
6. *Cnn-Based Foothold Selection for Mechanically Adaptive Soft Foot* J. Bednarek et al., 2020; Ministry points: 20; CORE points: 140
7. *Navigating by Touch: Haptic Monte Carlo Localization Via Geometric Sensing and Terrain Classification* Buchanan et al., 2021; Ministry points: 100

8. *Fast Haptic Terrain Classification for Legged Robots Using Transformer* M. Bednarek, Łysakowski, et al., 2021; Ministry points: 20
9. *Gaining a Sense of Touch Object Stiffness Estimation Using a Soft Gripper and Neural Networks* M. Bednarek, Kicki, et al., 2021; Ministry points: 100
10. *Learning to Solve Abstract Reasoning Problems with Neurosymbolic Program Synthesis and Task Generation* J. Bednarek and Krawiec, 2024; Ministry points: 20
11. *Learning-Based Foot-Shape-Aware Foothold Selection for Quadrupedal Robots* Tolomei et al., 2025; Ministry points: 200
12. *Solving Abstract Reasoning Problems with Neurosymbolic Program Synthesis and Task Generation* **J. Bednarek and Krawiec, 2026**; Ministry points: 5

1.6.2 Thesis guidebook

Because this research spans distinct domains—from physical legged robots operating in real-time to neural networks synthesizing programmatic code—Table 1.1 is provided as a structural guidebook. This matrix summarizes the domains, learning paradigms, and specific neuro-symbolic representations utilized across the three parts, serving as a quick-reference map for the reader to contextualize the overarching themes of the work.

Realm	Physical		Abstract
Domain	Robotics		Abstract reasoning
Thesis part	Part I: Sensing	Part II: Acting	Part III: Problem Solving
Class of problems	Haptic localization	Locomotion planning (foothold selection)	Abstract problem solving, analogical reasoning, program synthesis
Key components	<i>Sensing</i>	Hybrid (symbolic odometry + neural touch sensor)	—
	<i>Acting</i>	—	Neural foothold selection (Apprentice) based on symbolic human-designed model (Oracle)
	<i>Problem Solving</i>	Monte Carlo-based algorithmic search (particle filter)	—
Key advantages	Pioneering demonstration of purely haptic-based navigation	Fast neural model enabled online deployment and real-time querying	Novel hybrid perception, self-supervised learning, problem synthesis
Engaged ML paradigms	DL	DL	DL, RL
Modes of learning	Offline	Offline	Offline and online (learning from mistakes)
Engaged representations	Neural features & symbolic terrain classes	Neural cost maps & symbolic kinematic models	Neural latent vectors & symbolic workspace
Physical deployment	Yes (ANYmal, Messor robots)	Yes (ANYmal robot)	No (logical interpreter)

Table 1.1: The diagram summarizing the key contributions of this thesis and its organization, with emphasis on the **neuro-symbolic** integration.

Part I

Sensing: from physical interaction to
symbolic information

Chapter 2

Preliminaries

This part establishes the haptic foundations of the decision-making loop, focusing on environmental Sensing through direct physical interaction. The subsequent chapters detail the progression from classifying raw force-torque signals into semantic terrain categories to fusing these attributes with contact geometry within a probabilistic, purely proprioceptive localization framework optimized for vision-denied environments.

Chapter 3 is based on the article *What Am I Touching? Learning to Classify Terrain Via Haptic Sensing* (J. Bednarek, Bednarek, Wellhausen, et al., 2019) and outlines methods for converting raw force/torque data into terrain classifications. This analysis uses deep learning (DL) models to address noisy, variable-length data, indicating that haptic signals can provide reliable perceptual information.

Chapter 4 expands this framework based on the article *Navigating by Touch: Haptic Monte Carlo Localization Via Geometric Sensing and Terrain Classification* (Buchanan et al., 2021), integrating the haptic classification module into a broader control architecture. This configuration demonstrates the fusion of symbolic data from the Sensing module with geometric information. The resulting proprioceptive localization system operates reliably in vision-denied deployments.

2.1 Introduction to Sensing

A fundamental requirement for any system that interacts with an environment is a mechanism for gathering information that describes its current state, surroundings, conditions, and potential interactions. This process involves the system perceiving the incoming raw data through an interface, processing that signal, and ultimately transforming it into task-relevant information. We refer to this entire component as *sensing* (Bajcsy, 1988). The *Sensing component* comprises interfaces for receiving signals from the environment and is tailored to the higher-order task the system is aiming to accomplish. Its goal is to provide this specific and task-relevant information to other components of the system, making the output useful within a broader context. Crucially, the output of a Sensing component is a **task-relevant** information, distinguishing it from Acting component (see Part II), which provides a **goal-oriented** action. This configuration is illustrated in Figure 2.1.

2.1.1 Sensing in embodied systems

Sensing is an essential capability for autonomous robots, enabling them to perceive and interact with their surroundings. Review literature outlines the historical development and integration of these sensory systems (Siciliano & Khatib, 2008; Thrun et al., 2005). Traditional *exteroceptive sensing* modalities, such as vision-based systems using cameras and geometric sensors such as LiDAR, have been widely adopted for their ability to provide rich, high-dimensional data about the environment. These sensors have driven significant advances in fields such as autonomous driving, mapping, and navigation. However, their performance is often limited by environmental conditions; they can be severely impaired by factors such as fog, rain, dust, the absence of light, and are vulnerable to sensor damage. This dependence

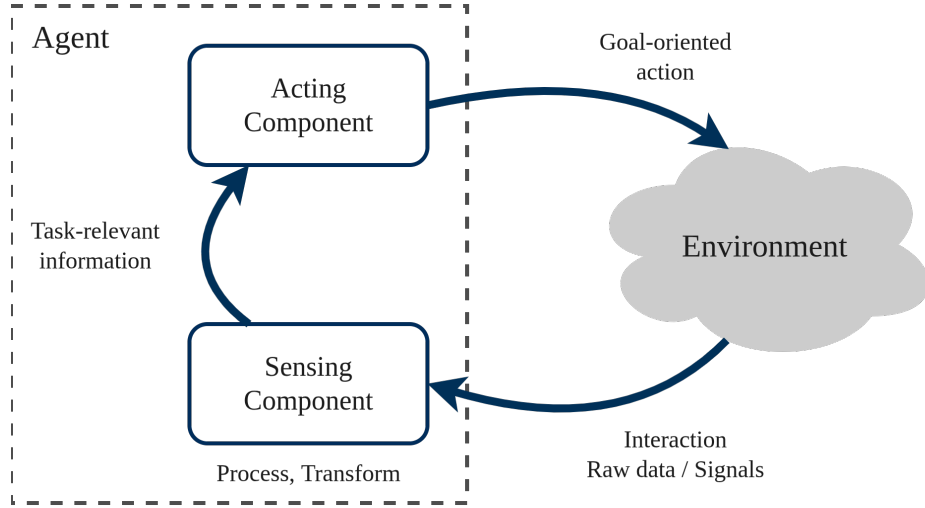


Figure 2.1: The agent-environment interaction loop, illustrating how the sensing component transforms raw signals into task-relevant information for decision-making.

on external, often fragile, sensors poses a significant challenge for robots operating in extreme or unstructured environments such as underground mines or sewage systems.

In contrast, *proprioceptive sensors*, which measure a robot’s internal state such as joint positions, forces, and torques, offer more robust information as they are less susceptible to external environmental interference and degradation of signal quality. This part focuses on a specific type of active proprioceptive sensing: *haptic perception*. Extensive reviews in the field of artificial touch emphasize that while vision provides global spatial context, tactile and haptic modalities are leading for understanding immediate physical contact dynamics, friction, and material properties (Dahiya et al., 2010; Luo et al., 2017). We define haptic perception as the process of converting raw physical signals that capture the robot’s internal state and its changes triggered by environmental interaction via the end-effector (the device at the end of the robotic arm or leg designed to interact with its surroundings) into meaningful, task-relevant information (Lederman & Klatzky, 1987). This allows the robot to “feel” and interpret the world through direct contact, providing a resilient means of perception even in the absence of exteroceptive information.

The ability to sense physical interaction plays a crucial role in adapting the sensing component to the larger system capable of deciding about the action that the robot should take to achieve a goal. This builds a continuous decision-making loop: while the sensing component quantifies the physical interaction, Acting component (see Part II) decides on the next interaction with the environment, and the outcomes are visible back in the newly sensed data. This intertwining of Sensing and Acting components allows them to benefit from each other, continuously improving the system’s ability to navigate and perform tasks (as presented in Part III). We find it essential to emphasize that the sensing process, as understood in this thesis, goes beyond the pure measurement and transformation of signals and provides tools for a more broader and free definition of tasks.

H1. *We hypothesize that a specialized perceptive module can successfully abstract raw, low-level sensory data acquired during robot-environment interaction into high-level, quantifiable features (e.g. symbolic properties) that are sufficiently meaningful and accurate to be directly utilized by the upstream decision-making and control systems to improve overall task performance.*

2.1.2 Haptic sensing as the basis of a cognitive system

The primary objective of this part is to demonstrate that haptic sensing can provide valuable, high-level, qualitative information about the environment. Specifically, we show that force and torque signals from a robot’s feet can be used to classify terrain types, such as asphalt, grass, or gravel. This capability elevates sensing from a simple measurement task to a core component of a larger, possibly neuro-symbolic system, where the symbolic output (e.g. “ground is gravel”) can be integrated with other cognitive processes, such as localization.

2.2 Key challenges in robotic sensing

The haptic data used throughout this chapter originate from force and torque sensors, which are specifically designed to measure the mechanical interaction between the robot and its environment. The sensors are mounted on the robot’s feet providing a direct physical measurement of the forces and torques experienced during contact. The data provided by a single sensor of this kind take the form of *6D time-series* signals, each time step capturing three values for force along the x , y , and z axes, and three for torque around those same axes. This continuous stream of physical measurements provides a detailed, albeit raw, account of the robot’s interaction dynamics as it touches and moves across different surfaces.

Unlike typical exteroceptive perception, such as vision or LiDAR, which can passively scan the environment, haptic sensing requires actual “playouts” in the environment. This necessity makes data acquisition inherently demanding, technically difficult, and often expensive, as it requires other complex components, such as motion planning, manual teleoperation, or automated controllers, to be fully operational. Furthermore, the complexity of the robotic setup introduces noise and inaccuracy to the collected data; for instance, the robot’s motors may produce subtle vibrations that cause minor mechanical “looseness” affecting stability. All these factors contribute to making raw haptic data an incredibly rich yet highly expensive data source to obtain and process effectively.

2.2.1 Data acquisition

A primary challenge in developing a robust haptic sensing system is the demanding nature of data acquisition. Creating a comprehensive and diverse dataset is an extremely difficult, time-consuming, and expensive. It requires a specific robot, often in a controlled laboratory setting, with precise measurements and careful preparation to limit external environmental factors. Furthermore, the data collection process must be highly repetitive to ensure the collection of a broad range of interaction dynamics and to enable proper annotation for supervised learning. The example setup for experiments conducted in this part is shown in Figure 2.2.

2.2.2 Task engineering

Haptic sensing is quite often reduced to the task of terrain classification. However, this view is misleading, as its practical use (integration into a larger cognitive or robotic system) introduces significant engineering complexities. Although a simple laboratory classification pipeline involves a well-defined process of data acquisition, annotation, and model training, this simplicity is immediately lost when the objective shifts to providing truly task-relevant information from low-level raw sensory data.

The fundamental difficulty lies in task engineering itself: the sensing component must anticipate the requirements of subsequent, potentially undefined, downstream components,



Figure 2.2: The experimental setup featuring the ANYmal robot in a custom-built environment equipped with an OptiTrack motion capture system built for the purpose of evaluating the localization system (Chapter 4). The large-scale scene contains diverse terrain types to facilitate repetitive haptic experiments.

such as the localization or navigation systems. This forces researchers to determine the utility of the output a priori—what information will be valuable and what will not—even if those utilizing components are out-of-scope for the current phase.

Transitioning the sensing proof-of-concept to a functioning system for a robot has several major practical consequences:

- **Experimental rigor:** the experimental environment must be meticulously engineered, often requiring the construction of large-scale terrain courses with various surfaces and transitions to provide a realistic and diverse testbed for robot operation. This makes the already demanding data acquisition process far more rigorous, complicated, and expensive (e.g. Figure 2.2).
- **System adaptation:** downstream robotic components must be either specifically adapted or built from the ground up to handle the unique format and constraints of the haptic sensor data.
- **A priori requirements:** this broader task definition often imposes external requirements, such as the necessity for a precise, a priori known map of the environment for systems like haptic localization, a consequence demanding more than simple classification.

2.3 Sensing in the context of a neuro-symbolic system

A modern robotic system is structured as a stack of complex components, each responsible for specialized tasks such as steering motors, planning movements, or managing control signals. These components communicate through predefined interfaces, exchanging both low-level physical measurements and higher-level symbolic properties, such as specific steering commands or the detection of a certain state of the system. These components usually remain

entirely encapsulated within the system and communicate via these established protocols, they can interact directly. The recent proliferation of deep learning has demonstrated exceptional capabilities in handling raw, high-dimensional perceptual data; however, reviews of the field highlight that its standalone integration into reliable, safety-critical robotic control architectures remains fundamentally limited by a lack of causal reasoning and formal guarantees (Sünderhauf et al., 2018). A profoundly different situation arises when a component interfaces directly with the environment. Here, the sensing component must employ a methodology to transform the stream of incoming raw sensory data into high-level, usable, and usually more compact properties that can be passed to other components.

This necessity motivates the participation of neural-based approaches for perception and symbolic-based components for interpretation and integration (Garcez & Lamb, 2023). Neural components process raw, high-dimensional sensory data to identify patterns and features, while symbolic components perform logical reasoning and planning based on discrete, high-level representations. Our work on haptic sensing based localization for legged robots is a direct application of this scenario. The neural network components developed in this part act as the perceptive layer, learning to extract patterns from noisy force/torque signals and converting these low-level data into a high-level symbolic representation, such as terrain types (e.g., “sand” or “gravel”). This symbolic output can then be directly utilized by other modules within a larger robotic control system accepting a symbolic property of a terrain class.

The true value of converting haptic data into symbolic information is realized when it is integrated into a broader cognitive system. A critical application demonstrated in this part is *probabilistic localization*. In many environments, particularly those with featureless geometry like flat floors, traditional geometric-based localization methods can suffer from unbounded drift. However, by fusing the symbolic terrain class information with geometric measurements, a robot can significantly improve its pose estimation. For example, knowing that a robot is on a specific type of terrain, as indicated by a symbolic label from the haptic sensor, provides a powerful constraint that can correct for odometry drift and help determine its location within a prior map. This integration elevates the role of sensing from a passive data-collection task to an active information-providing component that is essential for robust and reliable robot operation in complex, real-world conditions.

2.3.1 A bridge for a human-neural network interaction

Historically, robotic communication has often been bifurcated: systems typically relied on either bottom-up approaches—leveraging low-level raw data streams—or top-down ones, which utilized predefined high-level commands that were frequently decoupled from the raw sensory input. Although traditional feature engineering has long attempted to bridge this gap, the integration of neural network components into the sensing pipeline significantly enhances this capability. By learning to extract complex, high-level features directly from raw input, such models streamline the interpretation of environmental data, providing a more robust and nuanced representation of the robot’s state than was previously feasible with manual heuristics.

This technological step creates a critical interface layer—a bridge—between the robot’s perception system and the human operator. Such new features are still symbolic, like traditional control commands, but are now neural-based, meaning that they result from complex pattern recognition that neither low-level filtering nor manual programming could achieve. This enables a powerful new form of human interaction and intervention in the process of training and tuning of an autonomous system: operators can take advantage of the nuanced perception of the robot to define tasks or interpret its state. For example, it becomes possible to build a predefined environment map (see Figure 4.2) that incorporates

the classified terrain types identified by the neural network (e.g. “sand” or “gravel”). This in turn opens the possibility for a new set of semantically rich tasks and commands provided by humans, such as: “Find the spot with grass”, “Recover to solid terrain”, or “Avoid areas classified as loose gravel”. This shift enriches operability by allowing humans to command and interpret the robot using concepts grounded in the robot’s enhanced perception of the physical world.

Chapter 3

What am I touching? Learning to classify terrain via haptic sensing

3.1 Motivation

Today, autonomous mobile robots start venturing into challenging outdoor environments. They are supposed to operate in deserts, forests, underground and extraterrestrial missions, to name only a few. Locomotion in such environments poses many challenges from both a control and a perception perspective. One of the main problems is the assessment of the terrain type the robot is traversing. This information is used to adjust the control strategy of traversing a given terrain.

Terrain identification is present in different domains: autonomous driving, precision agriculture, mobile robotics, remote sensing, to name a few. Most of the approaches are based on pure vision (Valada et al., 2016). However, less attention is paid to the identification of terrain from the mechanical interaction of robots with the environment (Walas, 2015).

In this chapter, we focus on terrain classification and clustering for the legged robot, based on force/torque readings from the sensor mounted in the robot foot. The contributions include: a) a new classification method based on raw readings from the sensor using recurrent neural networks, b) a new fully connected neural model for classification of the terrain type for different walking speeds and directions, and c) a new clustering technique which allows the robot to not only recognize differences between terrain but also which leg is interacting with the terrain.

3.1.1 Related work

Terrain classification methods A significant part of the research in the field of terrain classification is focused on the use of visual information. The more traditional approaches are based on hand-crafted features (Filitchkin & Byl, 2012; Namin et al., 2015; Walas, 2015), in this group, the methods based on infrared images can be distinguished (Lenskiy & Lee, 2010). This classical vision based terrain type prediction applied to the agriculture scenario

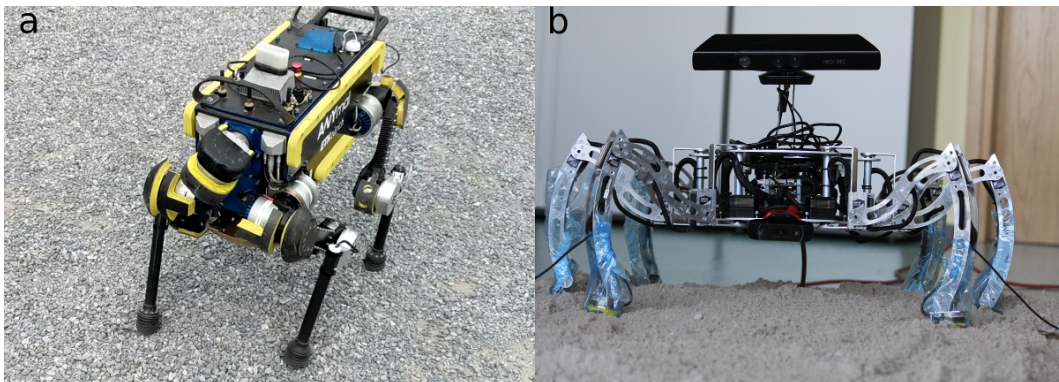


Figure 3.1: ANYmal (a) and Messor (b) robots collecting data from foot-ground interaction.



Figure 3.2: Example images of different terrain types in ETH dataset: *asphalt* (a), *gravel path*, *grass* (b), *dirt* (c), *sand* (d), *gravel* (e)

is presented in (Narváez et al., 2018). Robot motion planning based on semantic terrain labels obtained from vision data was described in Belter et al., 2018. An interesting approach to using vibration data for self-supervision of terrain classification from images is presented in (Otsu et al., 2016).

More recent research is based on the use of deep neural architectures. A learning process based on multi-spectral images is described in Valada et al., 2016. In another approach, the use cases of a deep neural network (DNN) for forest trail following based on visual input are presented (Giusti et al., 2016). Additionally, in the work Brandão et al., 2016 estimation of terrain friction parameters is proposed using CNN and vision input.

Other sensing modalities include Laser Range Finders, which have been used for terrain classification based on infrared intensity signals (Walas & Nowicki, 2014). A distinctly different method involves the use of acoustic signals for terrain identification (Christe & Kottege, 2016; Valada et al., 2018).

Tactile information in material classification In this chapter, we focus on approaches based on a direct mechanical interaction between the robot and the terrain. An example of such an approach is Giguère et al., 2006, where the authors propose a method of identifying terrain parameters using a single leg hitting the ground substrate. Similarly, using the single leg, classification of natural (M. Hoepflinger et al., 2010) and artificial terrains (Höpflinger et al., 2010) is achieved. Another approach is to fuse RGB-D and tactile data for terrain classification (M. A. Hoepflinger et al., 2013; 2014). In Kertesz, 2016, the quadruped robot based on the fused accelerometer and the ground contact force data classified indoor surface types. In Hoffmann et al., 2014 the authors present a quadruped robot equipped with proprioceptive sensors and research on the influence of different gaits on classification performance. In Mrva and Faigl, 2015, a six-legged walking robot was identifying terrains, using feedback from a robot actuators and the desired trajectory.

The majority of terrain classification work is performed in a highly controlled laboratory environment. Only Bosworth et al., 2016 describes a feasibility study of in-situ terrain modeling for a legged robot using force data. Finally, most of the work mentioned above is based on various supervised learning methods. However, as presented in Dallaire et al., 2015, unsupervised learning based on the Pitman-Yor (Pitman & Yor, 1997) process is also possible.

3.2 The environments

The research conducted in this chapter is based on two sets of data. Both were recorded while the robots were walking with a normal gait used for locomotion. No special exploratory movements were required. In case of the ETH dataset, the robot was using a fixed step-size gait while walking forward. In case of PUT dataset the gathered data was even more challenging

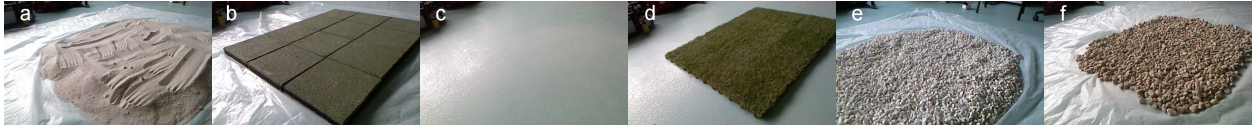


Figure 3.3: Terrains in the PUT dataset: *sand* (a), *rubber* (b), *concrete floor* (c), *artificial grass* (d), *chipping* (e), *gravel* (f).

as the robot was walking with different walking speeds and directions of movement.

3.2.1 The ETH dataset and robot

Robot and sensor The experimental data were collected using the ANYmal quadrupedal robot (Hutter et al., 2016). During operation, the platform performed simultaneous localization and mapping (SLAM) to simultaneously construct an environmental map and maintain its global localization. Concurrently, the system synchronized the acquisition of visual data (via RGB imagery) and tactile information (represented as 3D coordinates of footsteps within the spatial map). Given that the robot maintains precise extrinsics for both the visual sensors and end-effectors, and can determine its pose within the global frame, individual footholds can be retrospectively projected onto the collected RGB images. Subsequently, utilizing a weakly-supervised methodology (Wellhausen et al., 2019), the framework assigns semantic class labels to these footholds by mapping image-space pixel coordinates to their corresponding physical contact points.

ANYmal (shown in Figure 3.1 a) has a weight of around 30 kg and is fully torque-controlled with 12 series-elastic actuators. Ground interactions occur through four point-feet, which consist of an elliptical-shaped rubber sole, supported by 15 mm of memory foam to dampen impacts. Each of the four shanks is equipped with *BOTA Rokubi Mini 1.00 USB* six-axis force/torque sensors. They provide force measurements up to 500 N, torque measurements up to 5 Nm with typical measurement error smaller than 2% and noise level lower than 0.05% of the measurement range, at a sampling frequency of 100 Hz.

Environment Data were recorded in and around Zürich, Switzerland, over five days in late April and early May (2018) under different weather conditions. The environments in which data collection was carried out are an urban park, a forest, and a farmland. We categorized the data from these areas into six human-defined terrain classes, shown in 3.2. They are *asphalt*, gravel path (hereafter referred to as just *path*), *grass*, loose gravel (hereafter simply called *gravel*), *dirt* as found on farmland and dry *sand*. The crucial distinction between *path* and *gravel* is that the former is compacted gravel mixed with dirt, while the latter consists of loose layers of pebbles. The dataset was randomly divided into training and test sets.

3.2.2 The PUT dataset and robot

Robot and sensor For these experiments, the six-legged platform called Messor (shown in Figure 3.1 b) was used (Walas & Belter, 2011). It weighs ~ 5 kg and uses position controlled servos in the joints. The robot is equipped with a force/torque sensor (ATI Mini45 ATI Industrial Automation, 2026) in one of its feet. The 6-DOF signals were sampled at 200 Hz. The robot was moving in different directions with different walking speeds. That is, it was walking back and forth, sideways left and right, diagonally, and around its center of mass. The change in walking speed was in percent of the maximum speed: 30, 55, 80, where the maximum speed is 0.025 m/s.

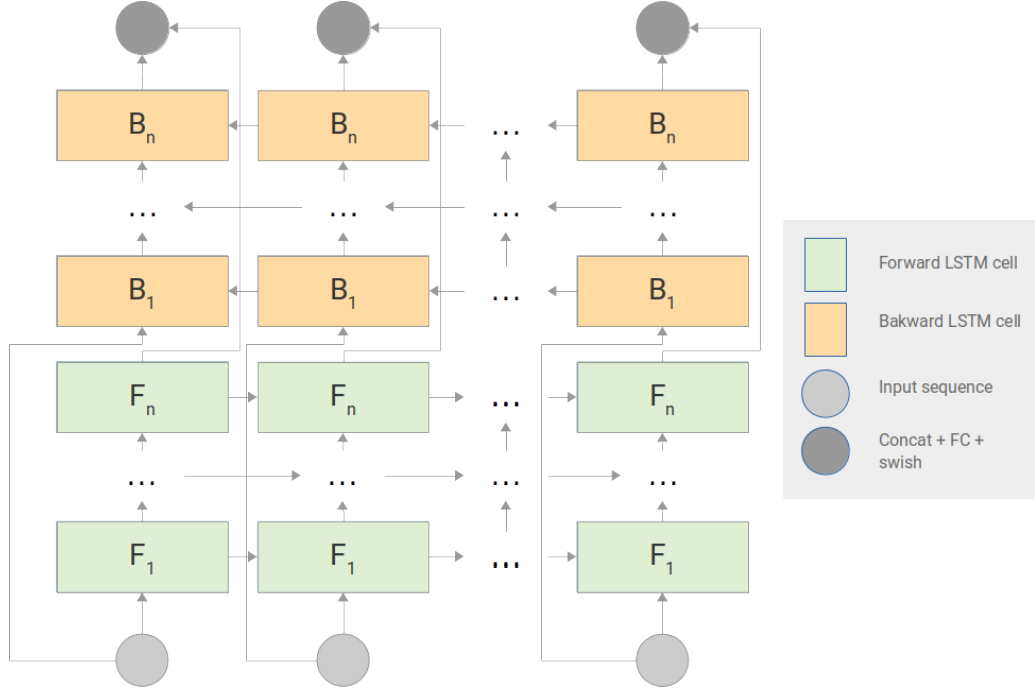


Figure 3.4: recurrent neural network architecture with forward and backward long short-term memory cells.

Environment The PUT environment is a synthetic setup developed under controlled laboratory conditions. There are six different terrain types: sand, rubber, concrete floor, artificial grass, chipping, and gravel. The terrain samples were placed in the laboratory and the robot was able to perform ten gait cycles on each terrain substrate. In total, the experiment gathered 8640 steps: 6 terrain types $\times$ 3 different walking speeds $\times$ 6 walking directions $\times$ 80 steps, which were then randomly divided into training and test sets. The terrain types are shown in Figure 3.3.

3.3 Methods

3.3.1 Fixed-length step model

The first presented model is based on the recurrent neural network (RNN) architecture, and its learning is performed directly on the signal in the time domain (force or torque; 6 values in total per timestep). The architecture is shown in Figure 3.4. The network contains two long short-term memory (LSTM) cells, one for the forward pass and a second for the backward pass. Every vertical block consists of three cells, with 128 hidden units, connected sequentially in such a way that the output from the previous one is passed both to the next cell and the next step of processing. The results at time t of both processing paths, backward and forward, are considered hidden states $h_f(t)$ and $h_b(t)$ respectively. They are obtained after processing the first element from the backward pass and the last one from the forward pass. The total result of the two-cell state is the concatenation of $h_f(t)$ and $h_b(t)$. Afterwards, this result is processed by an additional multilayer perceptron layer into higher dimensional data. Terrain classification is performed by two fully-connected layers after the projection module, where the first one consists of 100 neurons, and the second one of as many units as the number of terrain classes.

Another proposed model is based on a convolutional neural network (CNN) architecture. In this case, we also perform learning directly from the signal in the time domain. The

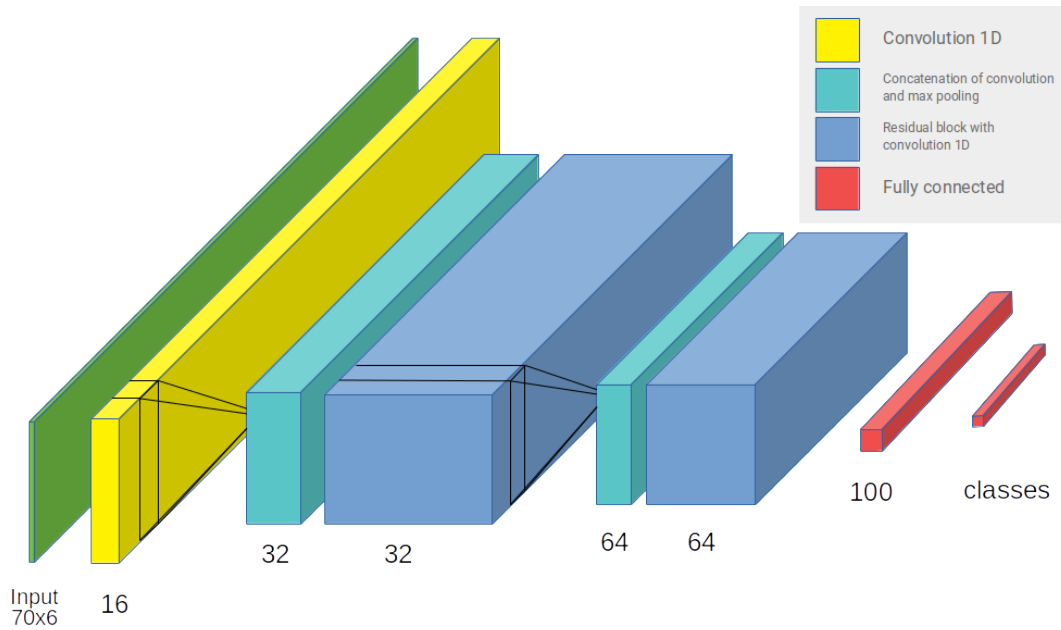


Figure 3.5: convolutional neural network architecture with 1D convolution for signals in time domain classification. The number of filters used by convolutions or neurons in fully-connected layers is shown under the corresponding blocks.

architecture is shown in Figure 3.5. It consists of a 70×6 input layer (6 force/torque signals with a length of 70 samples). After that, we convolve the signal in 1D to expand the dimensionality of the data. Next, we perform *down-sampling*, which consists of separate max pooling and 1D convolution, both reducing the width by two, followed by concatenation and an activation function. Thus, data are passed to the residual block, built with multiple *layers of residual 1D*. Given that the convolutional residual layers proposed in He et al., 2016 have become one of the most powerful elements used in convolutional networks to extract features, we used a modified version of such a layer by replacing 2D by 1D convolutions. The first residual block in our model is a sequence of three *1D residual layers* with 32 filters. Then, similar down-sampling and feature extraction is performed with five *1D residual layers* with 64 filters each. Finally, we obtained a fully-connected layer, containing 100 neurons, and a classification layer.

3.3.2 Variable-length step model

In the case of the PUT dataset, the proposed CNN method was unsuitable for this task, as its architecture could not accommodate signals of variable length, which is the case when the robot changes its walking speed. However, the signal length for the two data sets considered in this work is diametrically different. The shortest recorded signals from the PUT dataset ranged from 100 to 200 samples, which is still about three times more than the examples from the ETH dataset. Moreover, extreme cases from the PUT dataset are signals that reach more than even 1200 samples. The application of the RNN model on such data resulted in a well-known problem with long sequences – vanishing gradient – causing the results to be obtained at the level of a random classifier.

Thus, we move from a time domain to a frequency domain. To do so, we propose an FFT-based descriptor that limits the length of the input signal to some arbitrary number. In our case, we extract the three strongest (w.r.t. module) spectral components from the Fast Fourier Transform of a signal. After that, we compute the differences between them, taking into account their corresponding frequencies and modules. The formula for this is shown in

Equation 3.1, where variables $v1, v2$ and $v3$ refer to the frequencies or modules of the signal.

$$values = [abs(v1 - v2), abs(v1 - v3), abs(v2 - v3)] \quad (3.1)$$

After that, the descriptor is prepared as a sequence of 9 values – three highest amplitudes of the FFT signals, three differences between them according to Equation 3.1, and then an analogous set of three differences calculated from the frequencies of those strongest spectral components. For processing of extracted features, we proposed a fully-connected neural (FCN) model. The network is shown in Table 3.1. Each layer, except the last one, contains batch normalization and dropout.

Layer No.	1	2	3	4	5	6
Size	256	512	256	128	64	C
Dropout	0.2	0.4	0.4	0.4	0.4	0

Table 3.1: The fully-connected neural model for processing extracted FFT features.

3.3.3 Training classification models

The loss function used for all classification models is the weighted cross entropy between one-hot ground truth data and the probability distribution predicted by the network.

$$\mathcal{L}_{class} = - \sum_k^K w_k y_k^{true} \log y_k^{predict} \quad (3.2)$$

The ground truth one-hot encoded labels are denoted as y_k^{true} , whereas $y_k^{predict}$ is a predicted probability distribution. To overcome the problem of class imbalance, we calculate the coefficients w_k for each class as proposed in (Paszke et al., 2016). Specifically, the weight for a given class k is determined by:

$$w_k = \frac{1}{\ln(c + p_k)} \quad (3.3)$$

where p_k is the frequency of class k in the training set and c is a constant used to bound the resulting weights (set to 1.02). Finally, we also add the L_2 regularization term to the objective of the model.

$$\mathcal{L}_{reg} = \frac{\lambda}{2M} \sum_m^M \theta_m^2 \quad (3.4)$$

$$\mathcal{L}_{total} = \mathcal{L}_{class} + \mathcal{L}_{reg}, \quad (3.5)$$

where M is the total number of trainable parameters of the neural model and θ_m^2 indicates the square of the m th parameter.

3.3.4 Signals clustering

Since data labeling is a costly and cumbersome process and also incurs the risk of bias from the human perspective, we performed clustering to check what kind of information is available in the data alone. As the method used for this analysis is based on a CNN, which is limited to the use of fixed-length signals, for this purpose, we used the ETH dataset with fixed window size.

The proposed clustering model comprises a CNN-based autoencoder (similar to the one we used for classification), and an additional decoder part built with the same *1D residual layers*

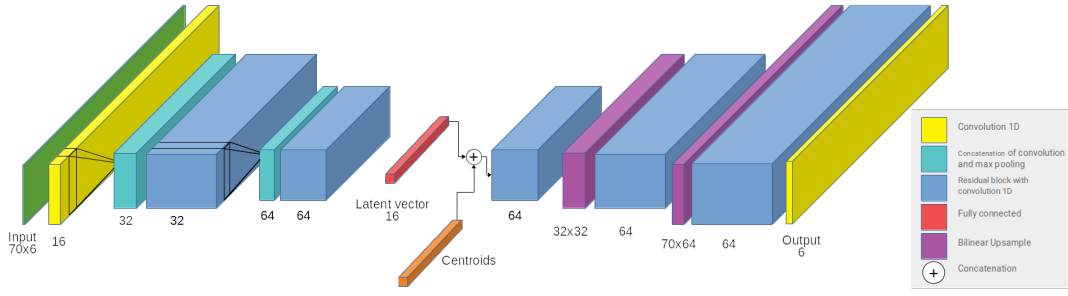


Figure 3.6: The Encoder-Decoder architecture employed in the clustering task. Representation obtained in latent vector used for clustering. The number of filters used by convolutions, size of the fully-connected layer and output sizes of the bilinear up-samplings are described in the below blocks.

as in the CNN classification model and bilinear upsampling. The architecture is presented in Figure 3.6. The task performed by the model is auto-association. The first step of processing involves encoding the input data into a denser, latent representation. The decoder then reconstructs the input data from the latent representation.

To discover what information is contained in the latent representation, an approach based on the classic k-means clustering algorithm was applied, according to the methodology defined in (Aljalbout et al., 2018). Similarly to k-means, the proposed method requires the number of centroids to be determined in advance. The components of the centroids vector are trainable parameters initialized with random values. To classify the samples, we calculate the nearest centroid for each of them. The distance metric used by the method is the Euclidean distance.

3.3.5 Training the clustering model

The optimization objective of the clustering model consists of three parts. To create a faithful representation of the input data, the model needs a constraint that would ensure the reproducibility of the input data from its latent representation (G. Hinton & Salakhutdinov, 2006). Such a constraint is the reconstruction loss function which has been implemented as a mean square error. The second term minimizes the distance between the decoder’s output fed by the centroids vector and the all input data for the current batch. The loss is additionally weighted by a similarity function between the input latent representations and the centroids vector to preserve the properties of the k-means algorithm. Our proposed clustering loss causes the result of the inference of the decoder on the vector of centroids to be close to the average value of the input elements of the cluster. In addition, this formulation of the iteration of the k-means algorithm allowed us to obtain a much better representation of the input data by the autoencoder since we do not explicitly minimize the distance between latent representations and centroid vectors. The last term is the L2 regularization loss.

$$\mathcal{L}_{total} = \alpha * \mathcal{L}_{recon} + (1 - \alpha) * \mathcal{L}_{clustering} + \mathcal{L}_{reg} \quad (3.6)$$

As in (Aljalbout et al., 2018), the balance between the loss functions was manipulated by the factor α . First, the model was trained with $\alpha = 0.75$ to focus on reconstruction loss. Then, additional training with $\alpha = 0.2$ was performed focusing the optimizer’s attention more on clustering than on reconstruction.

Let N denote the size of the current batch, x_i the i -th example in the batch, l_i the latent representation obtained from processing x_i by the encoder, and g the decoder function that

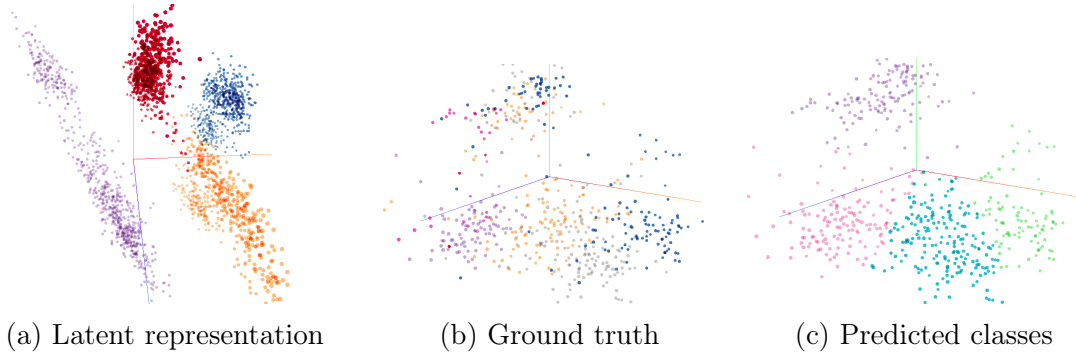


Figure 3.7: Visualization of the examples’ latent representations in the 3D space with the usage of principal component analysis method: (a) depicts distinction of the latent representations according to the leg from which they originate. Front left and right and rear left and right legs are denoted by the corresponding colors: red, orange, blue, violet. Total variance described by the highest three components is equal to the 82.3% (1st - 48.1%, 2nd - 17.7%, 3rd - 16.5%). (b) & (c) show the same latent representations for one selected leg. The colors represent their ground truth and predicted classes.

produces data of the same shape as x_i . We define the reconstruction loss as follows:

$$\mathcal{L}_{recon} = \frac{1}{N} \sum_i^N \|x_i - g(l_i)\|_2 \quad (3.7)$$

Unlike the classic k-means method, ours takes into account all combinations of centroids and the latent representations inside a batch. The distance between each centroid and the latent representation is weighted by the similarity function.

$$\mathcal{L}_{clustering} = \frac{1}{C + N} \sum_c^C \sum_i^N a_{ci} \|x_i - g(z_c)\|_2 \quad (3.8)$$

The symbol C denotes the number of centroids in the batch, while z_c is a vector of centroids, and a_{ci} is a similarity coefficient between the c -th centroid and the i -th latent representation. The similarity coefficient is defined as follows:

$$s_{ck} = \frac{1}{1 + \|z_c - l_k\|_2^2} \quad (3.9)$$

$$a_{ck} = \frac{s_{ck}}{\sum_i^C s_{ik}} \quad (3.10)$$

3.4 Results

Each introduced model was examined with a single experiment. The aggregated results in all models are presented in Table 3.2. In each case, the training process lasted 1000 epochs, during which the best results were taken as final results.

Fixed-length step model As can be observed in Table 3.2, our models outperformed traditional methods, while working on the data consists of sensor readings from the fixed-length robot gait. For example, comparing our results with the best SVM method (Gaussian

Model	Dataset	Accuracy [%]
CNN	ETHZ (raw)	93.59
RNN	ETHZ (raw)	92.69
RNN	PUT (raw)	18.49
FCN	ETHZ (FFT extracted)	79.53
FCN	PUT (FFT extracted)	80.39
SVM	ETHZ (Gaussian kernel)	75.00
SVM	PUT (Gaussian kernel)	67.00

Table 3.2: Results obtained by our models.

kernel), we improved the classification accuracy by 18%. Relatively short data samples allow us to use the powerful recurrent neural networks without any major gradient vanishing problems, while the constant length of steps made it possible to implement the 1D CNN architecture. Usage of our FFT descriptor connected with fully-connected network did not achieve such accuracy but still performs better than traditional methods.

Variable-length step model While fixed-length data samples were convenient to use in case of deep architectures (CNN, RNN), the variable-length samples cause their direct use to be problematic or even infeasible. As was discussed earlier, there is no possibility to use the CNN architecture on such structured raw data, while RNNs are exposed to the harmful effect of gradient vanishing in long sequences that spoils the final result. RNN achieved only 18% accuracy, and this score is entirely unreliable. However, preprocessing of variable-length data with our proposed FFT-based descriptor lets us perform classification using neural networks. Again, both of our fully-connected networks outperformed traditional methods by 5%.

Clustering of step signals The analysis of the data revealed a relationship between samples belonging to the same robot foot. The results of the principal component analysis (PCA) on latent representation are shown in Figure 3.7a. The elements represent the latent vector in the 3D space, whereas the color indicates different legs. The front legs are colored red and orange, while the rear legs are colored blue and violet. As is visible, the signals from the front legs are very similar. On the other hand, latent representation of signals from rear legs differs significantly from each other.

A noteworthy observation is that the model prefers to cluster signals based on which leg they originate from, rather than the terrain type. Nevertheless, when analyzing individual legs, we can find that the model tends to make additional clustering related to the terrain type individually for each leg. The PCA results for one single front leg are presented in Figure 3.7. The results obtained for the front legs are very similar according to the number of classes found and the structure of the groups, opposite to the rear legs, where it is hard to find similarities between both groups.

3.4.1 Conclusions

In this chapter, we proposed methods for terrain classification using deep learning techniques based on readings from force/torque sensors of quadruped and hexapod robots. In addition, we compared them with variations of the traditional method (SVM) and showed a significant improvement in performance. Additionally, while CNN and RNN architectures did not find use in the processing of variable-length signal samples, we proposed the method to overcome this issue using the FFT-based descriptor to preprocess such data. In addition,

we present the clustering technique, which enables the robot to differentiate between types of terrain and recognize which leg touches which type of it.

3.5 Implications

The research presented in this chapter applies deep neural networks to raw haptic data, diverging from traditional robotic sensing methods. By validating that deep learning architectures can outperform classical methods in interpreting physical interactions, this work establishes the neural component as a critical provider of task-relevant information. The neural classifier developed here is not just a standalone tool, but the essential “neural component” that allows the broader system to operate and reason in demanding environments. The success of this classification module provides the necessary foundation for the larger cognitive system presented in Chapter 4.

This chapter transforms the role of the Sensing component from a simple data collector to an active generator of symbolic information. As defined in the preliminaries, the goal of sensing is to transform raw signals into “a task-relevant information”. This work confirms that a neural component can successfully abstract low-level, continuous signals (6D force/torque readings) into high-level, discrete symbols (e.g. “gravel”, “sand”). This capability is critical in the context of bridging the gap between the processing of signals that reflect the interaction with an environment and logical reasoning. This specific property of sensing is an important foundation for the research presented in Part III.

Chapter 4

Navigating by touch: haptic Monte Carlo localization via geometric sensing and terrain classification

4.1 Motivation

Recent advances in the maturity and robustness of quadrupedal robots have made them appealing for dull and dirty industrial operations, such as routine inspection and monitoring. Automating these operations in underground mines and sewers is particularly challenging due to darkness, in-air dust, dirt, and water vapor, which can significantly impair a robot’s vision system (Kolvenbach et al., 2020). Additionally, camera or laser sensor failure may leave only proprioceptive sensors (i.e., inertial measurement unit (IMU) and joint encoders) at the robot’s disposal.



Figure 4.1: An ANYmal robot (Hutter et al., 2016) in a sewer with two feet in a slippery, wet depression and two feet in a dry, elevated area. With prior information about terrain type and geometry, it is possible for the robot to localized in the world using only touch. This would be extremely useful in dark and foggy environment (Image courtesy of RSL/ETH; Kolvenbach et al., 2020).

Blind quadrupedal locomotion has achieved impressive levels of reactive robustness without requiring vision sensors (Focchi et al., 2018; Lee et al., 2020). However, without the ability to also *localize* proprioceptively, a robot would still be incapable of completing missions or inspections.

The remainder of this chapter is structured as follows: Section **Related work** summarizes relevant research in the fields of terrain classification, in-hand tactile localization, and legged

haptic localization; Section **Problem statement** defines the mathematical background of the legged haptic localization problem; Section **Proposed method** describes our proposed haptic localization algorithm; Section **Implementation** describes the implementation details to deploy our algorithm on a quadruped platform; Section **Results** presents the experimental results collected using the ANYmal robot; Section **Discussion** provides an interpretation of the results and discusses the limitations of the approach; finally, Section **Conclusions** concludes with final remarks.

4.1.1 Related work

Tactile terrain classification The first tactile terrain classification method for walking robots was presented by M. Hoepflinger et al., 2010 and concerned experiments with a single leg detached from a robot’s body. In this work, force measurements and motor currents were used to successfully distinguish between four types of terrain. These results paved the way for the application of terrain classification methods on complete legged robots.

More recently, Kolvenbach et al., 2019 used a single leg on a real, standing ANYmal robot to differentiate between four different types of soil. Two types of feet (point and planar) were used to collect force, torque, and IMU measurements that were processed by an SVM classifier. The system showed that tactile information could be used to differentiate between visually similar soils. However, their method required a pre-determined probing action which is impractical as it forces the robot to stop walking.

A terrain classification system that could operate during locomotion was presented by Wellhausen et al., 2019. At start, their legged robot was trained to assign a terrain negotiation cost based on force/torque sensors. Once operating, their system assigned a terrain negotiation cost from images based on previous feet-to-image correspondences and terrain classification based on proprioceptive sensors. The ability to predict the terrain negotiation based on images was then used to plan the robot’s motion and avoid high cost terrains.

In complete darkness, which is our intended domain, vision-based sensors are of limited use for terrain classification, so we focus on purely proprioceptive sensing. In Chapter 3, we showed how deep learning models can be used to increase the accuracy of terrain classification. The system showed 98% classification accuracy from force-torque measurements against six terrain classes during a statically stable walk. However, this approach was limited to fixed-length input signals and thus could not generalize to aperiodic gaits, different speeds, or uneven terrain. In our work, we overcome these limitations with a novel masking mechanism in the convolutional layer, which allows us to process variable length signals.

More recently, Lee et al., 2020 showed an end-to-end approach to terrain classification for locomotion. Their deep learning controller was based on proprioceptive signals to adapt the gait to rough terrains. Even though tactile terrain classification was not explicitly performed, an internal representation of the terrain type was implicitly stored inside the network’s memory. In our work, we opt for a modular approach that explicitly returns a terrain class, which can then be used by the localization estimator.

Tactile localization in manipulation Tactile localization involves the estimation of the 6-Degree of Freedom (DoF) pose of an object (of known shape) in the robot’s base frame by means of kinematics of the robot’s fingers and its tactile sensors. Since the object can have any shape, the probability distribution of its pose given tactile measurements can be multimodal. For this reason, tactile localization has typically been addressed using Sequential Monte Carlo (SMC) methods, a subfamily of which is called *particle filters* (Fox et al., 2001). SMC methods are sensitive to the dimension of the state space, which should be low enough to avoid combinatorial explosions or particle depletion. State-of-the-art methods aim to

reduce this dimensionality and also to sample the state space in an efficient manner. For example, (Koval et al., 2016) reduced the state space of the pose of an in-hand object to the observable contact manifold.

Chalon et al., 2013 proposed a particle filtering method for online in-hand object localization that tracks the pose of an object while it is being manipulated by a fixed base arm. The estimated pose was subsequently used to improve the performance of pick and place tasks. The particle weights were updated by penalizing the finger/object co-penetration and the distance between the object and the fingertip in contact.

Manuelli and Tedrake, 2016 approached a slightly different problem, using a particle filter to estimate external contacts on a rigid body robot using only force/torque sensors in the joints of the robot. The particles were distributed around the robot’s body and the particle weights were computed from how well the contact point explained an external torque.

Vezzani et al., 2017 proposed an algorithm for tactile localization using the Unscented Particle Filter (UPF) on the iCub robot with sensorized fingertips to localize four different objects in the robot’s reference frame. The algorithm was recursive and could process data in real-time. The object and the robot’s base were assumed to be static, allowing the pose to be estimated as a fixed parameter. For legged haptic localization, the assumption of both a static robot and terrain does not always hold, and more general methods are required.

Haptic localization of legged robots The first example of haptic localization applied to legged robots is from Chitta et al., 2007. In their work, they presented a proprioceptive localization algorithm based on a particle filter for LittleDog, a small electric quadruped. The robot was commanded to perform a statically stable gait over a known irregular terrain course, using a motion capture system to feed the controller. While walking, the algorithm approximated the probability distribution of the base state with a set of particles. The state included three DoF: linear position on the xy -plane and yaw. Each particle was sampled from the uncertainty of the odometry, while the weight of a particle was determined by the L2 norm of the height error between the map and the contact location of the feet. The algorithm was run offline on eight logged trials of 50 s each.

Schwendner et al., 2014 demonstrated haptic localization on a wheeled robot with protruding spikes. The spikes detected contacts with the ground, which were compared to a prior 2.5D elevation map. Each wheel enabled multiple contact measurements, which they used to perform a plane fitting against the prior map and improve localization over larger, flatter terrain. They also performed terrain classification, but with a camera, which we do not require in our proposed work. They demonstrated an average position error 39 cm in five experiments, of approximately 100 m each.

In Buchanan et al., 2020, an SMC method was presented that estimated the previous trajectory (instead of the latest pose) at each step. Furthermore, the localization was performed for the full 6-DoF of the robot, instead of just the x , y and yaw dimensions as in Chitta et al., 2007 and Schwendner et al., 2014. The localization system was experimentally demonstrated online and onboard an ANYmal robot and used in a closed loop navigation system. When walking on flat areas, the localization uncertainty increased due to the lack of constraints on the xy -plane. Therefore, we are motivated to use the terrain classification techniques described in Chapter 3 to incorporate more information into the SMC.

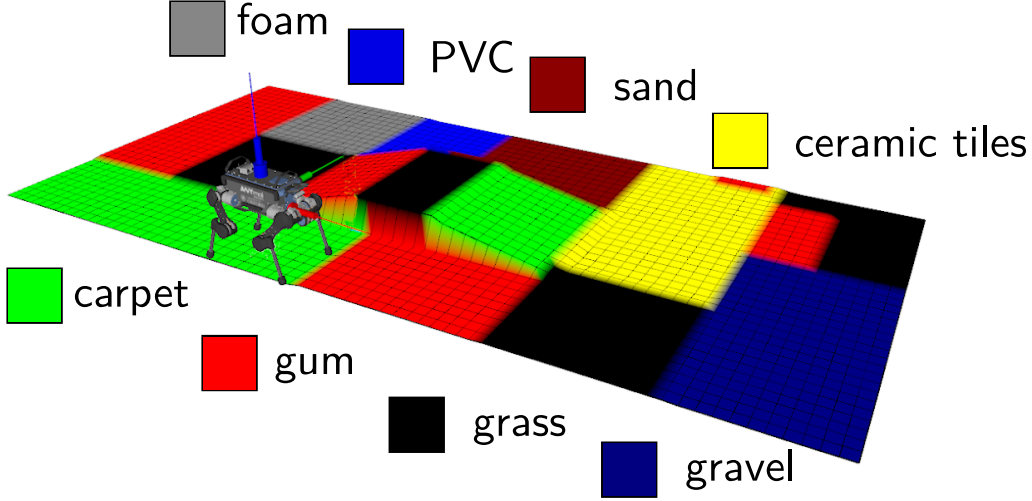


Figure 4.2: Terrain map used in Experiment 3, showing terrain categories.

4.2 Details

4.2.1 Problem statement

Let $\mathbf{x}_k \in SE(3)$ be the pose of a robot in time k . We use the notation $\tilde{\mathbf{x}}$ to represent a pose estimate from an external estimator, and $\mathbf{x}^*$ to represent its most likely estimate from our SMC filter.

Quadruped state definition We assume that for each timestep k , an estimate of the robot pose $\tilde{\mathbf{x}}_k$ and its covariance $\Sigma_k \in \mathbb{R}^{6 \times 6}$ are available from an inertial-legged odometric estimator, such as Bloesch et al., 2018; Fink and Semini, 2020; Hartley et al., 2020. The uncertainties for the rotation manifold are maintained in the Lie tangent space, as in (Forster et al., 2017). We also assume that the locations of the robot’s end effectors in the base frame $\mathcal{D}_k = (\mathbf{d}_{LF}, \mathbf{d}_{RF}, \mathbf{d}_{LH}, \mathbf{d}_{RH}) \in \mathbb{R}^{3 \times 4}$ are known from forward kinematics. The forces acting on each foot $\mathcal{F}_k = (\mathbf{f}_{LF}, \mathbf{f}_{RF}, \mathbf{f}_{LH}, \mathbf{f}_{RH}) \in \mathbb{R}^{3 \times 4}$ are measured by foot sensors (when available) or inferred from the inverse dynamics. Finally, the binary contact states $\mathcal{K}_k = (\kappa_{LF}, \kappa_{RF}, \kappa_{LH}, \kappa_{RH}) \in \mathbb{B}^4$ are inferred from $\mathcal{F}_k$.

For simplicity, we assume that errors due to joint encoder noise or limb flexibility are negligible. Therefore, the propagation of the uncertainty from the base to the end effectors is straightforward to compute. For brevity, the union of the aforementioned states (pose, contacts, and forces) at time k will be referred to as the *quadruped state* $\mathcal{Q}_k = \{\tilde{\mathbf{x}}_k, \Sigma_k, \mathcal{D}_k, \mathcal{K}_k, \mathcal{F}_k\}$.

Prior map Our approach can localize against 2.5D terrain elevation maps as well as full 3D maps. Terrain classification is meant to be carried out while the robot is walking, with no dedicated probing actions, therefore 2.5D maps are augmented with a terrain class category for each cell. This enables our method to overcome the degeneracy caused by featureless geometries (e.g., flat grounds, which are uninformative about the robot position on the xy -plane). Point clouds are used for 3D maps and only contain geometric information. To distinguish between the three types of map, we will refer to $\mathcal{M}$ when 2.5D only, $\mathcal{M}_3$ when 3D, and $\mathcal{M}_c$ when 2.5D augmented with class information, respectively. An example of a map $\mathcal{M}_c$ colorized by terrain class is shown in Figure 4.2.

Estimation objective Our goal is to use a sequence of quadruped states to estimate the most likely sequence of robot poses up to time k :

$$\mathcal{X}_k^* = [\mathbf{x}_0^*, \mathbf{x}_1^*, \dots, \mathbf{x}_k^*] \quad (4.1)$$

such that the likelihood of the contact points being on the map is maximized. Additionally, we assume that $\mathbf{x}_0^*$ is known.

4.2.2 Proposed method

To perform localization, we sample a predefined number of particles at regular intervals from the pose distribution provided by the odometry (as described in Section **Locomotion control and sampling strategy**) and compute the likelihoods of the measurements by comparing each particle with the prior map, in order to update the weights of the particle estimator (see Section **Measurement likelihood model for 2.5D data**). For convenience, we give a brief summary of SMC theory in Section **Sequential Monte Carlo localization**.

Sequential Monte Carlo localization In SMC Localization, the objective is to approximate the posterior distribution of the state $\mathbf{x}_k$ given a history of measurements $\mathbf{z}_0, \dots, \mathbf{z}_k = \mathbf{z}_{0:k}$ as follows:

$$p(\mathbf{x}_k | \mathbf{z}_{0:k}) = \sum_i w_{k-1}^i p(\mathbf{z}_k | \mathbf{x}_k) p(\mathbf{x}_k | \mathbf{x}_{k-1}^i) \quad (4.2)$$

where w^i is the *importance weight* of the i -th particle; $p(\mathbf{z}_k | \mathbf{x}_k)$ is the measurement likelihood function and $p(\mathbf{x}_k | \mathbf{x}_{k-1}^i)$ is the motion model for the i -th particle state. Since $p(\mathbf{x}_k | \mathbf{z}_{0:k})$ is typically unknown, the state $\mathbf{x}_k$ is typically sampled from $p(\mathbf{x}_k | \mathbf{x}_{k-1}^i)$, yielding:

$$p(\mathbf{x}_k | \mathbf{z}_{0:k}) = \sum_i w_{k-1}^i p(\mathbf{z}_k | \mathbf{x}_{k-1}^i) \delta(\mathbf{x}_k - \mathbf{x}_{k-1}^i) \quad (4.3)$$

where $\delta(\cdot)$ is the Dirac delta function. Over repeated sampling steps, the particles will spread out across the state space with weights approaching zero. To avoid this “impoverishment” of the particles, an additional re-sampling from the mostly likely state is used. When the re-sampling is done at every step, the method is known as particle filtering. We use a different strategy that merges likelihoods from a history of states and so refer to our method with the more general term SMC.

Locomotion control and sampling strategy Without a very robust reactive controller, blind locomotion requires conservative footstep placement, hence we opt for a statically stable gait, which guaranties stability at all times even when the motion is stopped mid flight phase. Since only one leg can be moved at a time, as soon as the swing leg touches the ground, the robot enters a four-support phase; at this time, the quadruped state estimate $\mathbf{Q}_k$ and the estimated terrain class $\tilde{c}$ are collected for a given foot position. Then, a new set of particles is sampled in a manner similar to Chitta et al., 2007:

$$p(\mathbf{x}_k | \mathbf{x}_{k-1}^i) = \mathcal{N}(\mathbf{x}_k, \Delta \tilde{\mathbf{x}}_k \mathbf{x}_{k-1}^i, \Sigma_k) \quad (4.4)$$

where $\Delta \tilde{\mathbf{x}}_k = \tilde{\mathbf{x}}_{k-1}^{-1} \tilde{\mathbf{x}}_k$ is the pose increase measured by the onboard state estimator at times $k-1$ and k . At time k , the new particles $\mathbf{x}_k^i$ are thus sampled from a normal distribution centered at the pose estimated from the odometry with its corresponding covariance. Since roll and pitch angles are observable from inertial-legged estimators, their estimates have low uncertainty. In practice, this allows us to reduce the number of necessary particles along these two dimensions, while still retaining the ability to compensate for the errors of the state estimator due to IMU biases, which are observable by exteroceptive sensors only.

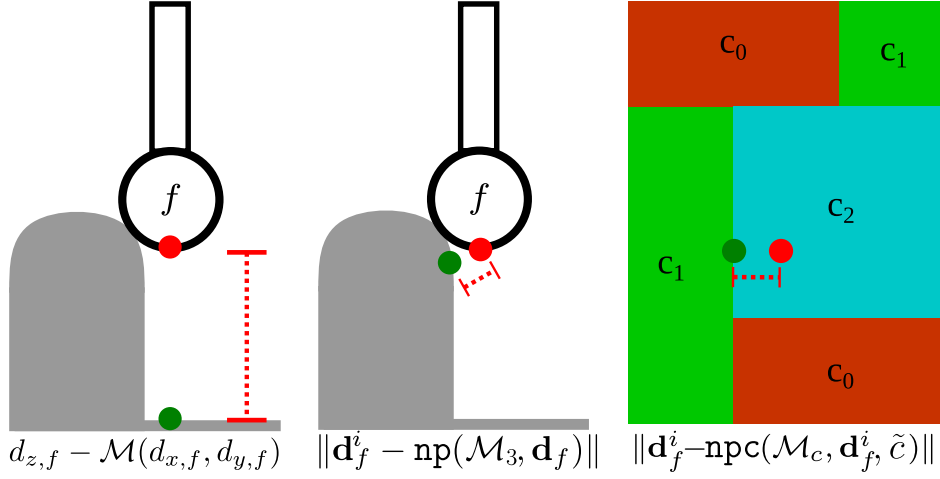


Figure 4.3: Comparison between contact measurements for 2.5D (left), 3D (middle) and terrain class (right) map representations. The red dots indicate the contact point as sensed by the robot, while the green dots indicate the corresponding location returned by the map. The red line shows the magnitude of the measurement. In the terrain classification case, the view is from a top down perspective. The robot’s foot sensed a contact in the c_2 region but a classification of c_1 was detected. The nearest c_1 point was returned by the map.

Measurement likelihood model for 2.5D data The likelihood of measurement is modeled as a univariate Gaussian centered at the local elevation of each cell, as done in (Buchanan et al., 2020). The variance σ_z was manually set to 1 cm. Given a particle state $\mathbf{x}_k^i$, the estimated position of a contact in the world coordinates for an individual foot f is defined as the concatenation of the estimated robot base pose and the location of the end effector, in the base coordinates:

$$\mathbf{d}_f^i = (d_{xf}^i, d_{yf}^i, d_{zf}^i) = \mathbf{x}_k^i \mathbf{d}_f \quad (4.5)$$

Thus, the measurements and their relative likelihood functions for the i -th particle and a specific foot f are (Figure 4.3, left):

$$z_k = d_{zf}^i - \mathcal{M}(d_{xf}^i, d_{yf}^i) \quad (4.6)$$

$$p(z_k | \mathbf{x}_k^i) = \mathcal{N}(z_k, 0, \sigma_z) \quad (4.7)$$

where: d_{zf}^i is the vertical component of the estimated location of the contact point in the world coordinates of foot f , according to the i -th particle; $\mathcal{M}(d_{xf}^i, d_{yf}^i)$ is the corresponding map elevation at the xy coordinates of $\mathbf{d}_f^i$.

Measurement likelihood model for 3D data Our method can incorporate contact events from 3D probing. This is useful for areas where the floor does not provide enough information to localize. In this case, the robot can probe walls and 3D objects with its feet. To better model this situation, we represent the prior map $\mathcal{M}_3 \in \mathbb{R}^{3 \times N}$ by a 3D point cloud with N points. The likelihood of a particular contact point is computed using the Euclidean distance between the foot and the nearest point on the map. This likelihood is again modeled as a zero-mean Gaussian evaluated at the Euclidean distance between the estimated contact point $\mathbf{d}_f^i$ and its nearest neighbor on the map, with variance σ_z :

$$z_k = \|\mathbf{d}_f^i - \text{np}(\mathcal{M}_3, \mathbf{d}_f^i)\| \quad (4.8)$$

$$p(z_k | \mathbf{x}_k^i) = \mathcal{N}(z_k, 0, \sigma_z) \quad (4.9)$$

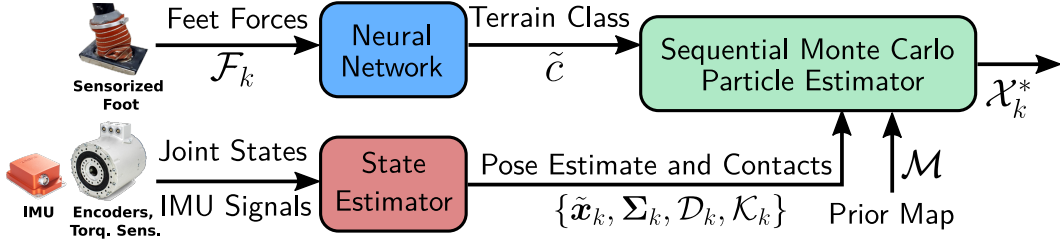


Figure 4.4: Information flow diagram of the overall system.

where $\text{np}(\mathcal{M}_3, \mathbf{d}_f^i)$ is the function that returns the nearest point of $\mathbf{d}_f^i$ on the map $\mathcal{M}_3$, computed from its k-d tree (Figure 4.3, middle). In our tests, point clouds were sufficiently small to make the k-d tree search time negligible. For larger scale environments, more compact representations based on Truncated Sign Distance Fields (Oleynikova et al., 2017) or Octrees (Vespa et al., 2018) could be used to allow faster search and less memory usage.

Terrain classification Let $f: \mathcal{S} \mapsto \mathcal{C}$ be the haptic terrain classification function that associates an element from the signal domain $\mathcal{S}$ to an integer from the class counter-domain $\mathcal{C}$. The set $\mathcal{S} : \{s \in \mathbb{R}^{l(s) \times 6}\}$ includes sequences of variable length force and torque signals s (of length $l(s)$) generated by a foot touchdown event. The set $\mathcal{C}$ is defined as the integers from 0 to $n - 1$, where n is the total number of terrain classes that the robot is expected to be walking on (in our case, $n = 8$). The terrain classes were chosen to have a different haptic response, ensuring the existence of the function f . Given the above definition of the problem, we introduce the following.

- a classification method $f': \mathcal{S} \mapsto \mathcal{C}$, which approximates the function f . As an implementation of f' we used a neural network;
- a dataset consisting of a list of pairs $d: [(s, c)]$, where $s \in \mathcal{S}$, $c \in \mathcal{C}$. This data set was divided into two subsets, training and validation, with a ratio of 80 : 20;
- a training process formulated as an approximation of the function f using the function f' by the minimization of cross-entropy between the probability distributions generated by these functions.

Terrain class measurement likelihood model The prior map is represented as a 2-dimensional grid whose cells are associated with a terrain class (an example is provided by Figure 4.2). Measurements are represented as a piece-wise cost function. If the estimated terrain class $\tilde{c}$ for a given foot position $\mathbf{d}_f^i$ matches the class c , at that location on the prior map $c = \mathcal{M}_c(d_{xf}^i, d_{yf}^i)$ the probability is a constant value corresponding to the maximum value of a zero-mean univariate Gaussian with manually selected variance σ_c of 5 cm (which is the width of the foot).

If the estimated class does not match the expected class in the map, the same Gaussian distribution is used to model the likelihood, where the input z_k^c is the distance to the closest point in the map with the expected class. This function is shown as

$$p(z_k | \mathbf{x}_k^i) = \begin{cases} \frac{1}{\sigma_c \sqrt{2\pi}} & \tilde{c} = c \\ \mathcal{N}(z_k, 0, \sigma) & \tilde{c} \neq c, \end{cases} \quad (4.10)$$

where

$$z_k^c = \|\mathbf{d}_{xyf}^i - \text{npc}(\mathcal{M}_c, \mathbf{d}_{xyf}^i, \tilde{c})\|_2. \quad (4.11)$$

The function $\text{npc}(\mathcal{M}_c, \mathbf{d}_{xyf}^i, \tilde{c})$ returns the nearest 2D point with class $\tilde{c}$ to the 2D foot position $\mathbf{d}_{xyf}^i$ on the map $\mathcal{M}_c$. The last case is shown in Figure 4.3, right.

We assume that elevation and terrain class measurements (z_k, z_k^c) are conditionally independent. Therefore, their joint probability can be computed as

$$p(z_k, z_k^c | \mathbf{x}_k^i) = p(z_k | \mathbf{x}_k^i) p(z_k^c | \mathbf{x}_k^i). \quad (4.12)$$

4.2.3 Implementation

The block diagram of our system is shown in Figure 4.4. The internal estimator on the ANYmal robot, TSIF (Bloesch et al., 2018), provides the odometry for the particle estimator, while the neural network estimates the class. This information is compared with the previous map to provide an estimate of the robot’s trajectory $\mathcal{X}_k^*$.

The pseudocode for the particle estimator (green block in Figure 4.4) is listed in Algorithm 1. At time k , the estimates of the terrain class $\tilde{c}$ and the pose of the robot $\tilde{\mathbf{x}}_k$ are collected. The pose estimate is used to compute the relative motion $\Delta \tilde{\mathbf{x}}_{k-1:k}$, propagate the state of each particle $\mathbf{x}_k^i$, and draw a sample from the distribution centered on $\Delta \tilde{\mathbf{x}}_k \mathbf{x}_{k-1}^i$ with covariance $\Sigma_k = (\sigma_{x,k}, \sigma_{y,k}, \sigma_{z,k})$.

The weight of a particle w^i is then updated by multiplying it by the likelihood that each foot is in contact with the map and the terrain class. In our implementation, we modify the likelihood functions from Equation 4.7 as follows:

$$p(z_k | \mathbf{x}_k^i) = \min(\rho, \mathcal{N}(z_k, 0, \sigma_z)) \quad (4.13)$$

where ρ is a minimum weight threshold, so that outlier contact measurements do not immediately lead to degeneracy.

Re-sampling is triggered when the variance of the weights rises above a certain threshold. This is necessary to avoid dispersion of the particle set across the state space with many particles of low weight. By triggering this process when the variance of the weights increases, the particles can first track the dominant modes of the underlying distribution.

Particle statistics The pose estimate for the k -th iteration, $\mathbf{x}_k^*$, is calculated from the weighted mean of all the particle poses. However, as we showed in our previous work, the particle distribution is often multi-modal. This motivated us to selectively update different dimensions of the robot pose. If the variance of the particle positions in the x and y axes are low (i.e., $\ll \sigma_{x,k}, \sigma_{y,k}$), we assume a well defined estimate and update the robot’s full pose. However, if they are high, we update only the z component of the robot’s location, which is always low because the robot keeps in contact with the ground.

In practice, with the terrain classification we found the terrain course was sufficiently detailed to keep particle position variance in the xy -plane low; therefore, z -only updates were rare. The threshold we used was a standard deviation of 10 cm, which corresponds to the typical uncertainty in our experiments.

To avoid particle degeneracy, importance sampling can be done in areas with higher likelihood. For example, if a grass terrain is detected, some particles could be injected into every grass terrain on the map. Further investigation on the benefits of importance sampling is left for future work.

Terrain classifier network Our neural network architecture is shown in Figure 4.5 and consists of three components: convolutional, recurrent, and predictive. Both the convolutional and the recurrent components must process variable-length data. Therefore, for the

```

 $\mathbf{x}_0^i \sim \mathcal{N}(\mathbf{x}_0, \tilde{\mathbf{x}}_0, \Sigma_0) \quad \forall i \in N$ 
foreach four-support phase  $k$  do
     $\Delta \tilde{\mathbf{x}}_k \leftarrow \tilde{\mathbf{x}}_{k-1}^{-1} \tilde{\mathbf{x}}_k$ 
    foreach particle  $i \in N$  do
         $\mathbf{x}_k^i \sim \mathcal{N}(\mathbf{x}_k, \Delta \tilde{\mathbf{x}}_k \mathbf{x}_{k-1}^i, \Sigma_k)$ 
         $w_k^i \leftarrow w_{k-1}^i$ 
        foreach foot  $f$  do
            if 3D then
                 $z_k \leftarrow \|\text{np}(\mathcal{M}_3, \mathbf{d}_f^i) - \mathbf{d}_f^i\| \quad w_k^i \leftarrow w_k^i p(z_k | \mathbf{x}_k^i)$ 
            else
                 $z_k \leftarrow d_{zf}^i - \mathcal{M}(d_{xf}^i, d_{yf}^i) \quad z_k^c \leftarrow \|\mathbf{d}_{xyf}^i - \text{npc}(\mathcal{M}_c, \mathbf{d}_{xyf}^i, \tilde{c})\|$ 
                 $w_k^i \leftarrow w_k^i p(z_k, z_k^c | \mathbf{x}_k^i)$ 
            end
        end
         $\mathbf{x}_k^* \leftarrow \text{WeightedMean}(\mathbf{x}_k^0 \dots \mathbf{x}_k^N, w_k^0, \dots, w_k^N) \quad \mathcal{X}_k^* \leftarrow [\mathbf{x}_0^j, \dots, \mathbf{x}_k^j] \text{ if } \text{Var}(w_k^i) > \text{threshold} \text{ then}$ 
         $\text{resample}(\mathbf{x}_k^i)$ 
    end
end

```

Algorithm 1: Haptic Sequential Monte Carlo Localization

convolution part, masking of the signal is required to prevent padded values from affecting the forward-pass result.

The first component of our network consists of two residual layers (ResLay). The ResLay used in our work is an adaptation of the one of He et al., 2016 with 2D convolutions replaced by 1D and masking support. The recurrent component uses two bidirectional layers (Bidir) with two Gated Recurrent Units (GRU) (Cho et al., 2014) in each. The output of the recurrent component is an average of two resulting hidden states of the last Bidir. The final output of the neural network is produced by the predictive component, which takes the recurrent component’s output, and using two fully connected (FC) layers produces a probability distribution from which the terrain class is inferred. The consecutive layers of the model are presented in Figure 4.5.

4.3 Results

We extensively evaluated the performance of our algorithm in three different experiments, each one targeting a different type of localization. These are described in more detail in Sections **Experiment 1: 2.5D terrain course**, **Experiment 2: online haptic exploration on vertical surfaces**, and **Experiment 3: terrain classification**.

Evaluation protocol There are three different modalities in our algorithm: HL-G (Haptic Localization Geometric), which uses only geometric information, HL-C (Haptic Localization Class), using only class information, and HL-GC (Haptic Localization Geometric and Class), which uses both geometric and terrain class information. HL-G was tested in Experiments 1 and 2 using 2.5D and 3D prior maps, respectively. HL-G, HL-C, and HL-GC were tested in Experiment 3 with 2.5D maps augmented with terrain class information.

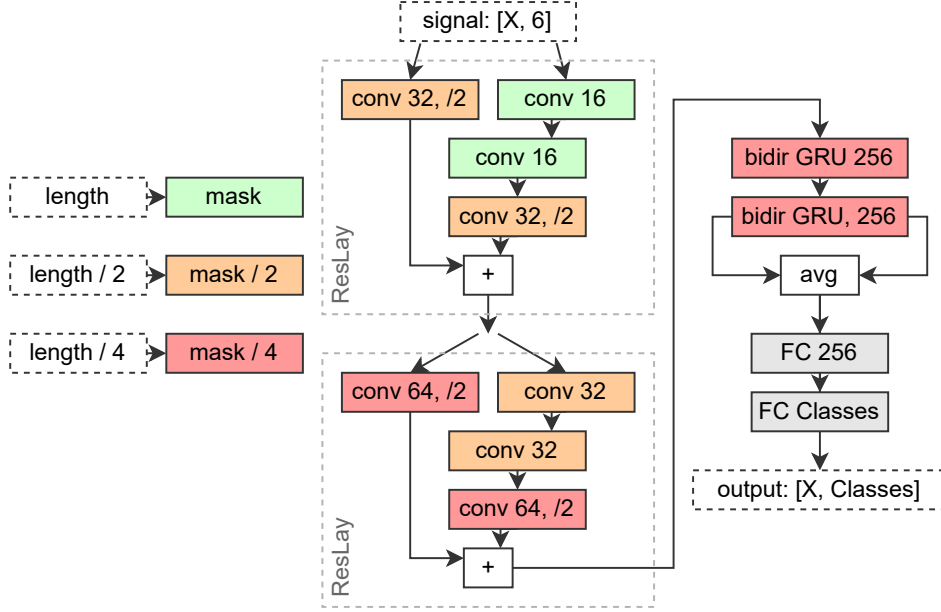


Figure 4.5: Neural Network Structure used for Terrain Classification. The main blocks of neural architecture are convolutional (conv), recursive (GRU bidir), and fully connected (FC) blocks. The masking mechanism used in variable-length signal processing blocks takes appropriate masks, marked in green, orange, and red. These masks correspond to the individual signal lengths, taking into account the initial length and the reduction of size by convolution with stride equal to 2.

Evaluation metrics We quantitatively evaluated localization performance by computing the mean of the Absolute Translation Error (ATE) as described by Sturm et al., 2012:

$$\frac{1}{n} \sum_{i=1}^n \|\text{trans}(\mathbf{T}_i^{-1} \hat{\mathbf{T}}_i)\| \quad (4.14)$$

where $\mathbf{T}_i$ and $\hat{\mathbf{T}}_i$ are the ground truth of the robot and the estimated poses, respectively. In contrast to Sturm et al., 2012, we do not perform the alignment of trajectories, as ground truth and estimated poses are represented in the same coordinate system.

A qualitative evaluation was also performed for Experiments 1 and 2 by assessing the robot’s ability to reach its planned goals or end effector targets while using the online localization. This demonstrated the benefit of localization when used in the loop with the onboard motion planner.

Ground truth and prior map The ground truth trajectories were collected by motion capture systems at 100 Hz. The pose of both the robot and the terrain course reference frame were accurately measured with mm accuracy via reflective markers installed on them.

At the start of the experiment, the relative position of the robot within the map was measured using ground truth and used for initialization only. Thereafter, the pose of the robot was estimated using the particle filter. To account for initial errors, the particles at the beginning were sampled from a Gaussian centered at the initial robot pose with a covariance of 20 cm.

The prior maps were captured with survey grade laser scanners (Leica BLK-360 and SURPHASER 100HSX) which provided point clouds with sub-centimeter accuracy.

Mean Absolute Translation Error (ATE)				
Trial	Dist. [m]	Time [s]	TSIF [m]	HL-G [m]
1	66.42	525	0.63	0.13
2	145.31	1097	2.57	0.40
3	55.67	557	0.52	0.19
4	68.71	604	0.65	0.32
5	172.65	1606	2.00	0.61

Table 4.1: Experiment 1: Estimation performance. TSIF = Two-State Implicit Filter (Bloesch et al., 2018); HL-G = Haptic Localization with Geometry only.

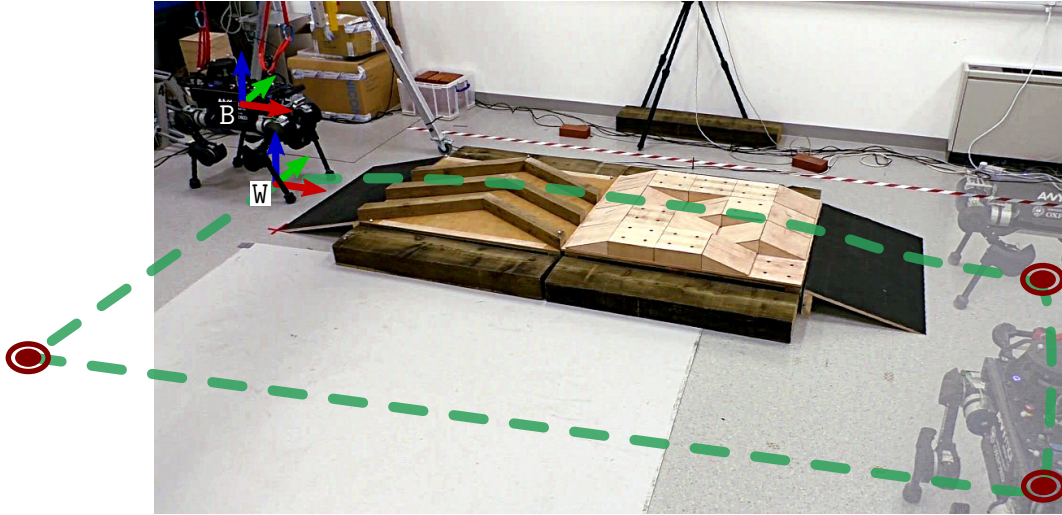


Figure 4.6: Experiment 1: ANYmal haptic localization experiments. The robot traverses the terrain, turns 90 deg right and comes back to the initial position passing through the flat area. The goals given to the planner are marked by the dark red disks, while the planned route is a dashed green line (one goal is out of the camera field of view). The world frame W is fixed to the ground, while the base frame B is rigidly attached to the robot’s chassis. The mutual pose between the robot and the terrain course is bootstrapped with the motion capture.

Experiment 1: 2.5D terrain course In this experiment, the robot was commanded to navigate between four walking goals at the corners of a rectangle. One of the edges required crossing a 4.2m terrain course composed of a 12° ascending ramp, a 13 cm high chevron pattern, an asymmetric composition of uneven square blocks and a 12° descending ramp (Figure 4.6). After crossing the wooden course, the robot returned to the starting position across a portion of flat ground, so as to test the system behavior in feature-deprived conditions.

While blind reactive locomotion has been developed by several groups, including Di Carlo et al., 2018 and Focchi et al., 2018, unfortunately our blind controller was not sufficiently reliable to cross this terrain course, so we resorted to use of the statically stable gait from Fankhauser et al., 2018 which used a depth camera to aid in footstep planning. However, pose estimation was performed without access to any camera information.

To demonstrate repeatability, we performed five trials of this experiment, for a total distance traveled of more than 0.5 km and 1 h 13 min duration. A summary of the results is presented in Table 4.1, where HL-G shows an overall improvement between 50 % to 85 % in the ATE compared to the onboard state estimator. ATE is 33 cm on average, which reduces to 10 cm when evaluating only the feature-rich portion of the experiments (i.e., the terrain course traversal).

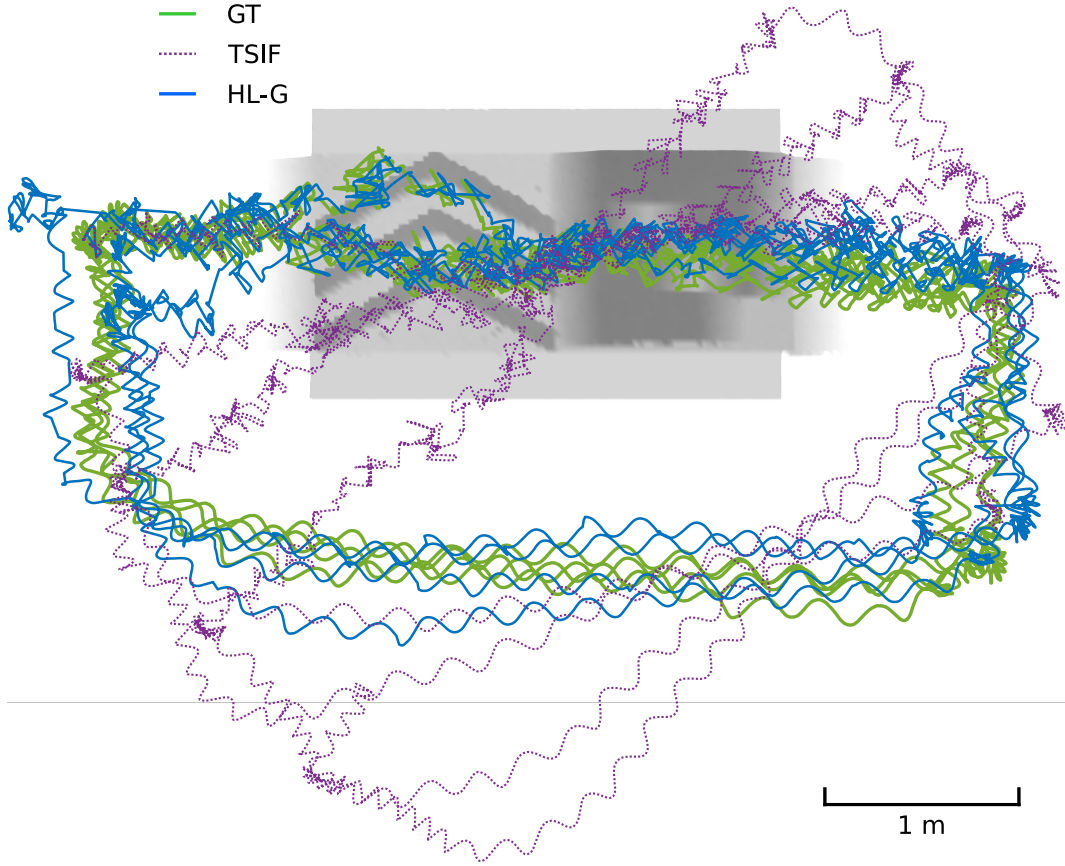


Figure 4.7: Experiment 1, Trial 2: Top view of the estimated trajectories from TSIF (dashed purple), haptic localization (blue) and ground truth (green).

For trials 1 and 2, the robot was manually operated to traverse the terrain course, completing two and four loops, respectively. In trials 3–5, the robot was commanded to follow the rectangular path autonomously. In these trials, the haptic localization algorithm was run online in closed-loop and effectively guided the robot towards the goals (Figure 4.7). Using only the prior map and contact events only, the robot stayed localized in all the runs and successfully tracked the planned goals. This can be seen in Figure 4.7, where the estimated trajectory (in blue) diverges from the ground truth on the xy -plane when the robot is walking on flat ground. This is due to growing uncertainty from lack of geometric information; however, the covariance reduces significantly and the cluster mean re-aligns with the ground truth when the robot reaches the terrain.

Figure 4.8 shows in detail the performance of the estimator for each of the three linear dimensions and yaw. Since position and yaw are unobservable, the drift on these states is unbounded. In particular, the error on the odometry filter (TSIF (Bloesch et al., 2018), purple dashed line) is dominated by upward drift (due to kinematics errors and impact nonlinearities, see third plot) and yaw drift (due to IMU gyro bias, see bottom plot). This drift is estimated and compensated for by the haptic localization (blue solid line), allowing for accurate tracking of the ground truth (green solid line) in all dimensions. This can be noted particularly at the four peaks in the z -axis plot, where the estimated trajectory and ground truth overlap. These times coincide with the robot is at the top of the terrain course.

Experiment 2: online haptic exploration on vertical surfaces The second experiment involved a haptic wall following task with the robot starting in front of a wall but with an uncertain location. The particles were again initialized with 20 cm position covariance. To test the capability to recover from an initial error, we applied a 10 cm offset in both x and y

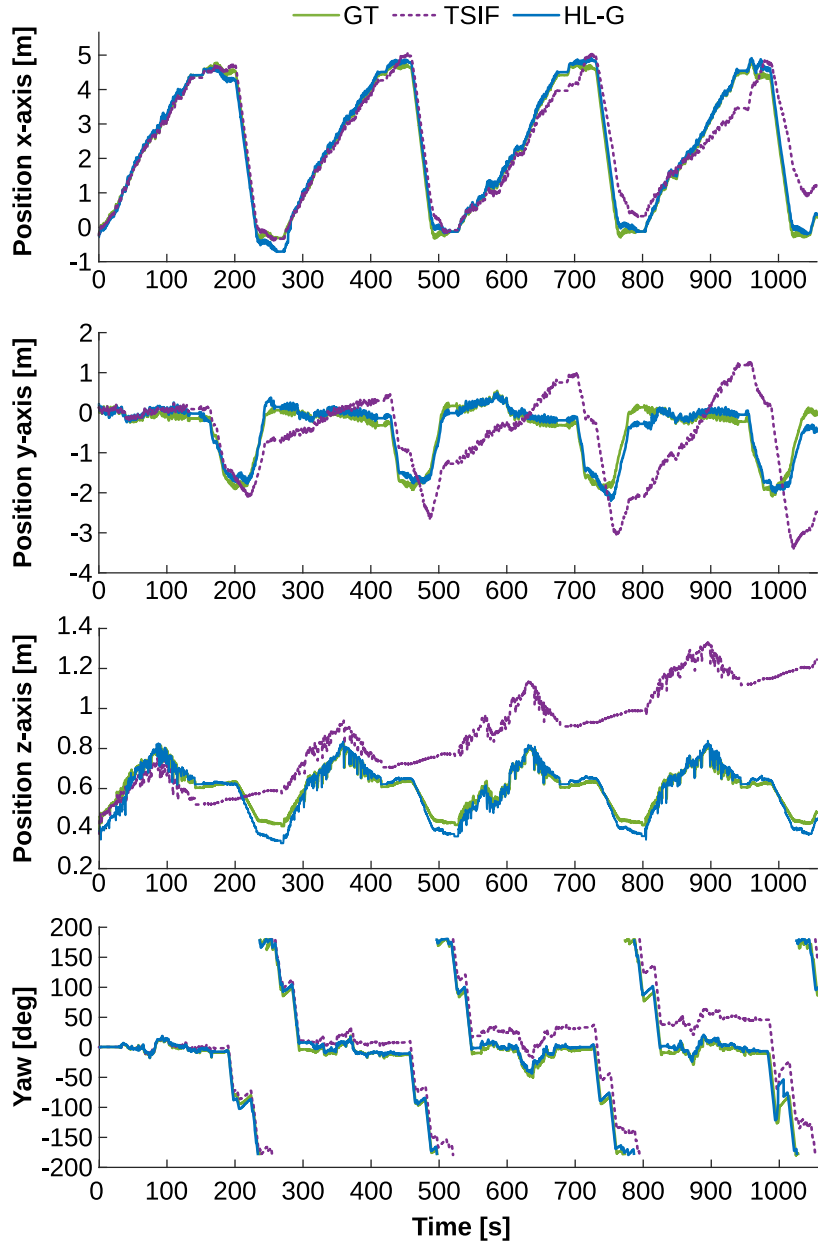


Figure 4.8: Experiment 1, Trial 2: Comparison between the estimated position from TSIF (dashed purple) and haptic localization (blue) against ground truth (green). After 200s, the estimation error in TSIF has drifted significantly upward and in yaw. In particular, the upward drift is noticeable in the third plot, where the values grow linearly. The drift is eliminated by the re-localization against the prior map.

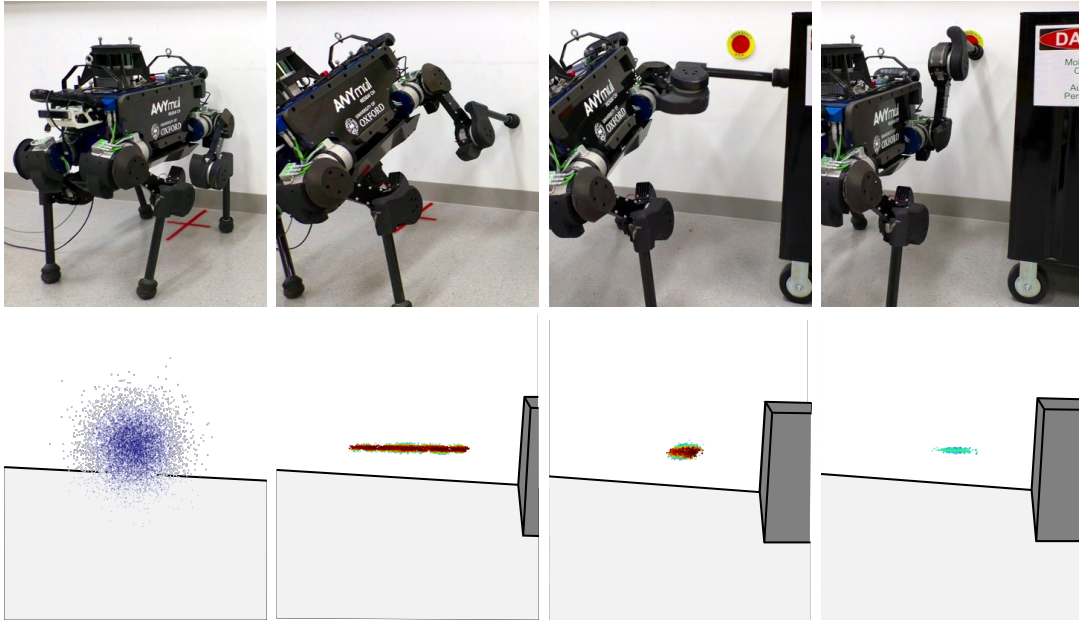


Figure 4.9: Experiment 2: Haptic probing experiment in 3-dimensions. The top row shows the robot performing the experiment while the bottom row shows the particle distribution of position estimates. The particle set is colored by normalized weight according to the jet colormap (i.e., dark blue = lowest weight, dark red = highest weight). First, on the bottom left, the robot has an initial distribution with poses equally weighed. The robot makes a forward probe and then moves to the right. Now the particles are distributed as an ellipse with high uncertainty to the left and right of the robot. Then, the robot makes a probe to the right and touches an obstacle; the particle cloud collapses into a tight cluster. Since the robot is now localized, it is able to complete the task of pressing the button on the wall.

from the robot’s true position on the map. At start, the robot was commanded to walk 1 m to the right (negative y direction) and press a button on the wall, whose location in the prior map was known. To accomplish the task, the robot needed to “feel its way” by alternating probing motions with its right front foot and walking laterally to locate itself inside the room. The fixed number of probing motions was pre-scripted so with each step to the right, the robot probed both in front and to its right. The whole experiment was executed blindly with the static controller from (Fankhauser et al., 2018).

As shown in Figure 4.9 the robot was able to correct its localization and complete the task of touching the button. The initial probe to the front reduced the uncertainty in the robot’s x and z directions, which reduces the particles to an ellipsoidal elongated along y . As the robot moves, uncertainty in the x direction increases slightly. By touching the wall on the right, the robot re-localized in all three dimensions in much the same way as a human following a wall in the dark would. The re-localization allows the robot to press the button, demonstrating the generalization of our algorithm to 3D. This would enable a robot to localize by probing a known piece of mining machinery, allowing it to perform maintenance tasks. The final position error was: $[7.7, -3.7, -0.2]$ centimeters in the x , y and z directions.

Experiment 3: terrain classification In the third experiment, we demonstrate the localization using terrain semantic information, which has been tested on a custom designed terrain course. Multiple 1m x 1m tiles of different terrain materials were placed on a 3.5m x 7m area. The course includes a 20cm high platform with two ramps with different terrain materials, as shown in Fig 4.10. The different terrain types used were: gum, carpet, PVC, foam, sand, artificial grass, ceramic and gravel.

For training, we gathered an additional dataset of the robot walking on the different



Figure 4.10: **Top:** Examples of foot positioning when collecting data on different terrains under controlled laboratory conditions. **Bottom:** Experiment 3: ANYmal with sensorized flat feet standing on the multiple terrain type course. Close up of the foot is provided. An IMU and fore/torque sensor is located in the sole with coordinate frame shown.

patches consisting of 8773 steps with a quasi-static walk gait. Examples of data collection are shown in Figure 4.10 Top. During data collection, maximum base displacement and rotation were enforced at 0.21 m and 0.23 rad, respectively. These limits ensured a stable walk at all times. The dataset was split into 7018 training and 1775 testing samples. To minimize the impact of unbalanced data on the learning process (e.g., more steps on a specific class), the loss function was weighted based on the number of steps taken on each terrain. The mean and standard deviation of the accuracy achieved by the network was estimated from k-fold cross-validation to be 94 % and 0.09, respectively.

The robot was equipped with sensorized feet which feature high quality 6-axis force/torque sensors (Valsecchi et al., 2020). These feet are necessary to provide the signals for terrain classification as in J. Bednarek, Bednarek, Wellhausen, et al., 2019. The robot autonomously

Mean Absolute Translation Error (ATE)						
Trial	Dist. [m]	Time [s]	TSIF [m]	HL-G [m]	HL-C [m]	HL-GC [m]
1	191	1114	0.64	0.23	0.63	0.14
2	331	1850	1.28	0.25	0.73	0.11
3	193	1090	0.72	0.21	0.61	0.18

Table 4.2: Experiment 3: HL-G = Haptic Localization with only geometry; HL-C = Haptic Localization with only terrain class; HL-GC = Haptic Localization with both geometry and terrain class.

walked between pre-programmed waypoints placed over the entire course, including several passes over the ramp. Large sections of the trajectory were only on the flat terrain tiles, forcing the algorithm to rely mostly on terrain classification for localization. Unlike Experiment 1, the robot was able to walk completely blind and no exteroception was used for footstep planning. A statically stable gait was used such that one foot was in the air at a time.

To demonstrate repeatability, we have performed three trials of this type, for a total distance traveled of more than 0.7 km and 1 h 7 min duration. We compare results produced using HL-G (Geometry), HL-C (Terrain Class), and HL-GC (Geometry and Terrain Classification). As the majority of the terrain course is flat, there is not enough information for geometry only localization to be continuously accurate. Only when using terrain class information as well as geometry can the robot localize in all parts of the terrain course.

A summary of the experiments is presented in Table 4.2, where HL-GC shows an overall improvement between 14% to 56% in the ATE compared to HL-G. Using only the prior knowledge of the terrain geometry and class, the robot stayed localized in all the runs and bounded the linearly growing drift of the state estimator. This can be seen in Figure 4.11, where the estimated trajectory (in red) is able to stay near the ground truth trajectory (green). In areas where there are large patches of the same material, such as gravel (dark blue) and ceramic (yellow), there is not enough information to localize in the xy -plane and the pose estimate drifts. When the robot crosses the boundary into a new terrain type, the localization is able to correct it.

Figure 4.12 shows in detail the performance of the estimator for each of the three linear dimensions and yaw. As in Experiment 1, the error on the odometry filter (TSIF, purple dashed line) of the robot is dominated by upward and yaw drift. This drift is estimated and compensated for by the haptic localization (red solid line), allowing accurate tracking of the ground truth (green solid line) in all dimensions¹.

4.3.1 Discussion

The results presented in Sections **Experiment 1: 2.5D terrain course** and **Experiment 2: online haptic exploration on vertical surfaces** demonstrate that terrain with a moderate degree of geometric complexity (such as Figure 4.6) already provides enough information to bound the uncertainty of the robot’s location. The effectiveness of a purely geometric approach is obviously limited by the actual terrain morphology in a real world situation, which would need to contain enough features such that all the DoF of the robot are constrained once the robot has touched them.

In the case where there is not enough geometric information, we have shown in Section **Experiment 3: terrain classification** that terrain semantics can be used to localize. With sufficiently diverse terrain types (as shown in Figure 4.10), boundary crossings from one

¹A video showing all of these experiments is attached as supplementary material

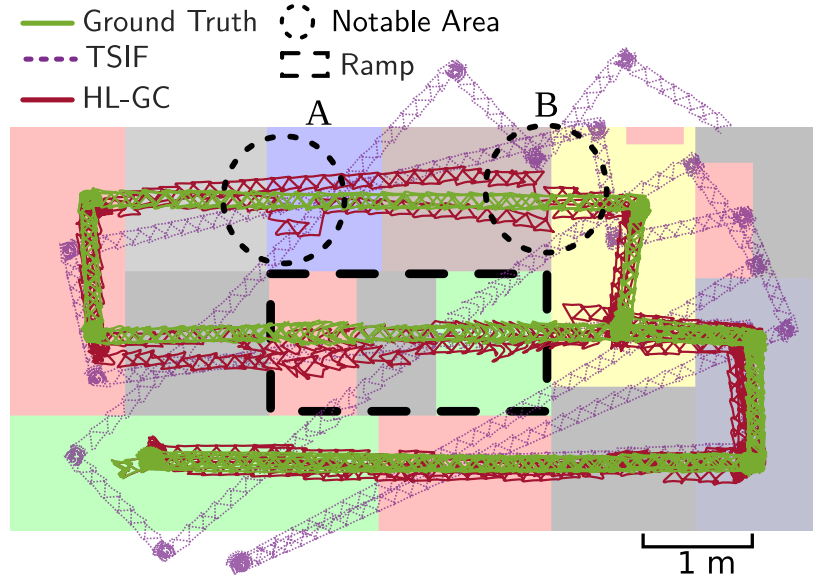


Figure 4.11: Experiment 3, Trial 2: Top down view of the state estimator (dashed purple), HL-GC estimated trajectory (red) and ground truth (green). Trajectories are overlaid on terrain map with ramp shown in a dashed box. Two dotted circles show notable areas in the trajectory. **A:** At a boundary crossing, the particle mean diverges from the ground truth. However, as the particle cloud nears the ramp, the geometric information gives higher likelihood to the particles in the center of the terrain. **B:** Another boundary crossing which in two separated crossings triggers good localization updates.

terrain to another provide enough information to correct for drift in the xy -plane.

Analysis of particle distribution on geometric terrain Figure 4.13 top shows the evolution of the particles up to the first half of the terrain course for Experiment 1, Trial 2. As the robot walks through, the particle cluster becomes concentrated, indicating a good convergence to the most likely robot pose.

In the third subfigure, it can be noted how the probability distribution over the robot’s pose follows a bimodal distribution, which is visible as two distinct clusters of particles. This situation justifies the use of particle filters, as they are able to model non-Gaussian distributions, which can arise from a particular terrain morphology. In this case, the bimodal distribution is caused by the two identical gaps in between the chevrons. In such situations, a weighted average of the particle cluster would lead to a poor approximation of the true pose distribution. Therefore, the evolution of the particles illustrated in Section **Particle statistics** is crucial to reject such an update.

Figure 4.13 Middle shows the particle distribution over flat ground. While not transitioning between terrain types, our method must rely on geometric information only and therefore is constrained in z but not x and y . In these cases, we only update the drift estimate in the z direction which keeps the particle distribution near the ground but spread out.

Analysis of particle distribution on terrain class We compare results presented in Figure 4.12 from HL-G, HL-GC and HL-C. We can see that even with only class information, this method is able to keep pose estimation error bounded in the x - y plane (mean ATE for the class only trajectory was 0.63 m). In the third subplot from the top, the class only trajectory drifts upward. This is because of the absence of any measurement in the z , hence our method relies on the proprioceptive state estimator.

Further analysis of the effect of terrain class on localization is shown in Figure 4.13 middle. Here, we show the evolution of particles from HL-GC in Experiment 3, Trial 3. The particles,

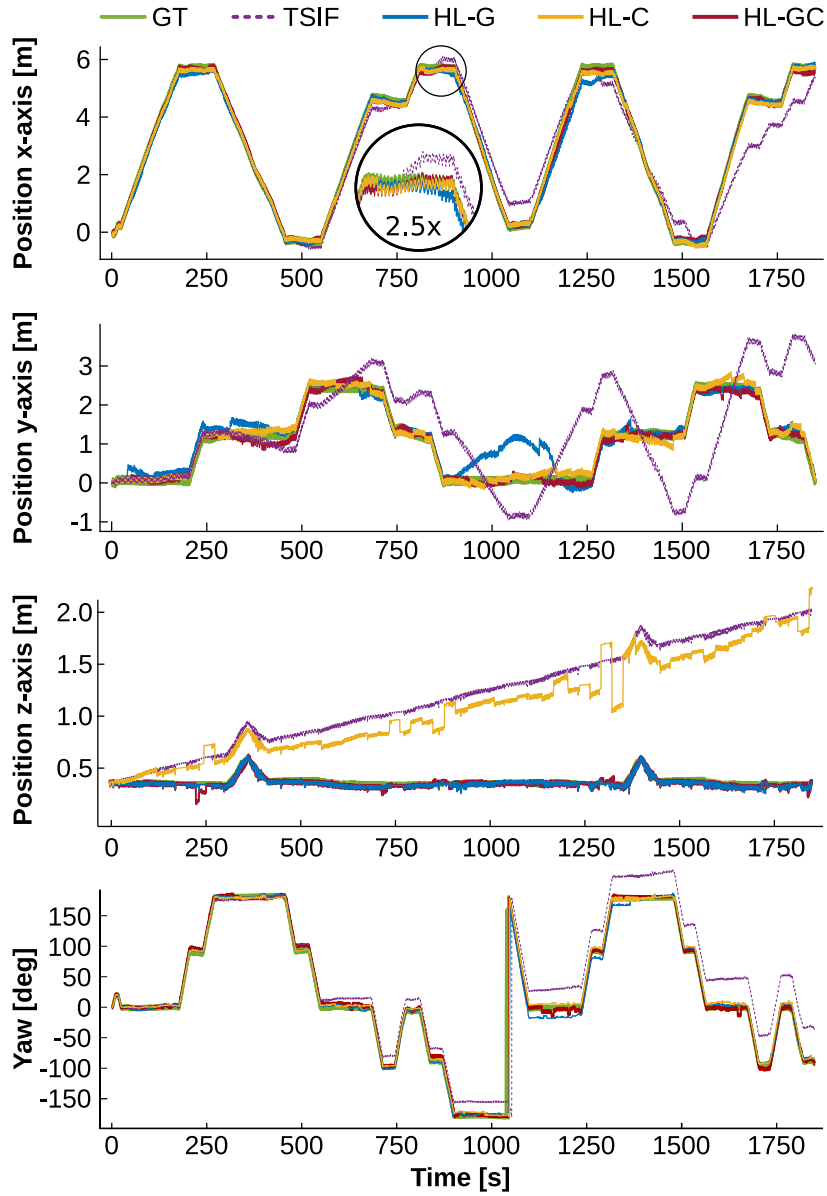


Figure 4.12: Experiment 3, Trial 2: Here we show the difference in results when using only geometric information HL-G (blue), only terrain class information HL-C (yellow) and both HL-GC (red). We compare these to the state estimator (dashed purple) and ground truth (green).

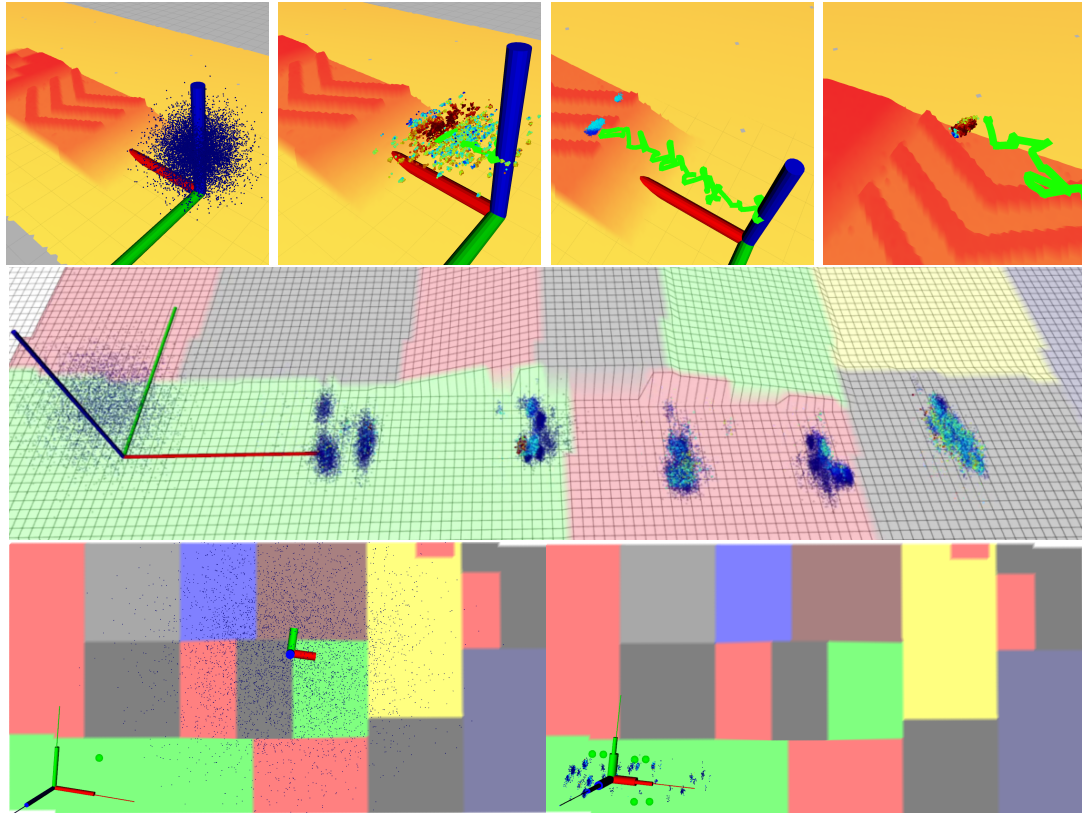


Figure 4.13: Evolution of particle distributions during experiments. Particles are colorized by normalized weight according to the jet colormap (i.e., dark blue = lowest weight, dark red = highest weight). **Top (Experiment 1):** The green line indicates the ground truth trajectory. *A)* At start, all the particles have the same weight and are normally distributed at the starting position. *B)* After a few steps on the ramp, the robot pose is well estimated on x and z directions, but there is uncertainty on y . *C)* When the robot approaches the chevron the particle set divides into two clusters, indicating two strong hypotheses as to the robot pose. *D)* After a few more steps on the chevron, the robot is fully localized and the particles are tightly clustered. **Middle (Experiment 3):** As the robot walks from left to right, the particle cloud makes two terrain class transitions. As the robot crosses the first transition, the cloud becomes more narrow in the x direction as error along this axis is corrected. **Bottom (Experiment 3):** Top-down perspective with the robot estimate initialized in the middle of the map and the initial particle distribution spread out over the whole map. Within 5 steps the particle distribution has re-sampled over the correct green terrain patch. The green circles are the classified footsteps.

which are initially normally distributed around the starting position, quickly converge along the z axis as floor elevation information is used. Once the boundary transition from green to red occurs, the particles correct for drift in the x direction.

Finally, in Figure 4.13 bottom we show the behavior of HL-GC when the particles are initialized evenly across the entire map. Within 5 footsteps the distribution has converged to the green section in the bottom left. This is because there are only two green terrains, and the bottom left is the only one on flat ground.

4.3.2 Conclusions

We have presented a haptic localization algorithm for quadrupedal robots based on Sequential Monte Carlo methods. The algorithm can fuse geometric information (in 2.5D or 3D) as well as terrain semantics to localize against a prior map. We have demonstrated that even using only geometric information, walking over a non-degenerate terrain course containing slopes and interested geometry can reduce localization error to 10 cm. Our method also works if the robot probes vertical surfaces, measuring its environment in full 3D. Finally, we have shown how in areas of even sparser geometric information, terrain semantics can be used to augment this geometry.

The proposed approach demonstrated an average of 20 cm position error over all areas of a terrain course with different terrain classes and geometries. The ability to localize purely proprioceptively is valuable for repetitive autonomous tasks under vision-denied conditions, such as inspections of sewage systems. This method could also serve as a backup localization system in case of sensor failure, enabling a robot to complete its task and return to base.

4.4 Implications

The work presented in this chapter serves as the experimental validation of the core hypothesis introduced in Section 2.1: that a specialized Sensing module can abstract raw sensory data into high-level symbolic properties that are directly utilized by upstream decision-making systems to complete the task. By successfully employing the neural terrain classifier as an information provider for probabilistic localization, this research demonstrates the transition of haptic sensing from a passive data collection mechanism to a critical driver of cognitive robot behavior.

This research highlights the necessity of “task-relevant information” over simple geometric measurement. In scenarios where geometric data were featureless (e.g., flat floors), traditional geometric localization suffered from unbounded drift. By integrating the neural component, the system gained access to a new dimension of information—terrain semantics—which provided the necessary constraints to disambiguate the robot’s location. This confirms that the utility of a sensing component is defined by its ability to provide information specifically tailored to the needs of the system, effectively overcoming the limitations of purely geometric perception.

Chapter 5

Conclusions

5.1 Summary of contributions

This part of the thesis explored the use of haptic Sensing as a robust alternative to traditional perception methods in challenging, vision-denied environments. The initial study (J. Bednarek, Bednarek, Wellhausen, et al., 2019; Chapter 3) established that haptic signals from sensors mounted on the feet of a robot contain information about the terrain. That information is encoded in characteristics of the interaction with the environment and can be robustly decoded using deep learning techniques. Our contributions include the development of machine learning models—specifically CNN and RNN architectures—that effectively process noisy and variable-length force/torque data without relying on handcrafted features. These methods achieved high classification accuracy on complex datasets, proving the viability of haptic perception for terrain classification. The culmination of these efforts was the integration of this sensing capability into a purely proprioceptive real-time localization system presented by Buchanan et al., 2021. The results of the experiments (Section 4.3) show that providing task-relevant information, derived from the neural component, allows achieving good localization quality in a demanding environment. The experimental results successfully validated the hypothesis introduced in Section 2.1.1 (**H1**). We demonstrated that a Sensing component capable of processing complex physical interaction data can be successfully incorporated into a larger decision-making system.

5.2 Implications and broader context

The contributing articles focused on the foundational component of any autonomous system: sensing. It provided both a conceptual introduction to the role of sensing in decision-making and a practical implementation of this concept within an embodied system. By demonstrating that a neural component can effectively transform raw, low-level sensory data from physical interactions into high-level symbolic representations (e.g., classifying terrain as “sand” or “gravel”), we have established a baseline for how information is extracted from the environment.

The experiments show how a specialized Sensing component can be successfully integrated to supply the larger system with critical symbolic outputs, functionally transforming raw data into the necessary task-related intelligence, successfully bridging the gap between perception and reasoning. The presented findings have significant implications for real-world robotics. The ability to localize using only haptic sensing is invaluable for autonomous tasks in extreme conditions, such as inspections in sewers or underground mines, where cameras and LiDAR are often ineffective.

However, the rigor and expense of acquiring real-world data from physical robots pose a significant challenge, especially when the task changes from observation to reaction. Transitioning from perception to intervention, Part II addresses two practical concerns (see Figure 2.1): first, how to circumvent the high cost of real-world data acquisition; and second, can acting of the robot be upgraded with the same neural principles applied here to sensing.

Further elaboration on the versatility of the Sensing component (Part III) prompts the fundamental question for the later chapters: if a Sensing module can successfully convert complex, unstructured haptic data into meaningful symbols in the physical world, can this mechanism be generalized to process abstract, cognitive inputs? Thus, the “sensing” framework established here is not merely a solution for robotic localization, but the prototype for the *symbolic sensing* mechanism that will later enable abstract reasoning.

Having successfully introduced a neural mechanism into the Sensing component, the narrative now progresses to the complementary half of the decision loop. The forthcoming part of the thesis shifts focus to Acting. We move from interpreting the world to changing it, setting the stage for the final integration of these two components in the context of abstract problem solving.

Part II

Acting: learning physical policies from
synthetic data

Chapter 6

Preliminaries

The forthcoming chapters integrate the core contributions of two co-authored articles. Both works present a direct realization of the paradigm described in Section 6.1.2, effectively implementing the separation between the computationally expensive *slow Oracle* and the efficient, learned *fast Apprentice*.

Chapter 7 is based on the article *Single-Shot Foothold Selection and Constraint Evaluation for Quadruped Locomotion* (Belter et al., 2019). This research addresses the core challenge of real-time decision-making by demonstrating a method that uses a convolutional neural network (CNN) to evaluate the quality of a foothold in a single rapid inference step. This initial work focused on a quadruped robot equipped with a rigid foot. Training on synthetic data allowed CNN to implicitly learn the complex relationships between terrain geometry, kinematic limits, and potential collisions, effectively serving as a fast heuristic for standard locomotion tasks.

Chapter 8 draws from *Cnn-Based Foothold Selection for Mechanically Adaptive Soft Foot* (J. Bednarek et al., 2020). This study introduces more complex dynamics by equipping the robot with a soft, mechanically adaptive foot. The inclusion of a compliant foot presents substantial modeling challenges, as the definition of an optimal foothold shifts to depend on how the foot deforms to fit the terrain. Our research demonstrated that the same neural network framework could successfully learn a robust policy for this setup. This result highlights the network’s capacity to model the intricate dynamics of soft-body interactions, proving the versatility of the fast Apprentice approach and its ability to generalize to complex physical environments.

6.1 Introduction to Acting

The operation of an intelligent agent is fundamentally grounded in the continuous cycle of interaction between the actor and its environment (Russell et al., 2020; Sutton & Barto, 2018). This relationship is not static; rather, it is characterized by a dynamic decision-making loop in which the agent perceives the state of the world and acts to alter it (see Figure 2.1). To operate autonomously, this interaction cannot be random. It must be driven by a defined goal—a desired state or objective that the system seeks to achieve. As established in the related literature on robotics and artificial intelligence (Russell et al., 2020; Siciliano & Khatib, 2008), a goal can range from a discrete objective, such as navigating to a specific coordinate, to a continuous maximization of a utility or reward function, such as maintaining stability while traversing rough terrain (Sutton & Barto, 2018).

The presence of a goal requires a strategy for achieving it. This strategy is formalized as an acting policy. In its canonical definition, a policy is a mapping from the perceived state of the environment to an action. In the reinforcement learning (RL) literature (Sutton & Barto, 2018), this policy serves as the decision-making mechanism that determines an agent’s behavior to maximize a cumulative reward signal. The structural composition of this policy fundamentally depends on the nature and objective of the task at hand. While highly structured, classical control tasks often rely on explicit heuristics composed of well-defined sub-goals or programmatic “if-then” rules, modern embodied systems facing unstructured

environments—particularly in the domain of deep visuomotor control (Levine et al., 2015)—increasingly shift away from such discrete formulations. In these continuous, end-to-end paradigms, the policy is predominantly implicit and partial: rather than executing formalized specifications or explicitly defined problem-solving steps, the agent’s behavior is directly guided by learned internal representations and the objective (e.g. continuous optimization of reward signals). For instance, a locomotion policy might implicitly estimate friction coefficients to prevent slipping and stabilize a robot’s mass distribution by simply providing steering commands, without explicitly solving equations of motion or computing a geometric path to the target.

Consequently, such an acting policy does not necessarily represent an optimal control law derived from first principles, but rather a functional approximation. It achieves the goal not by conscientiously solving every subsequent sub-problem analytically, but by internalizing a strategy that yields the same practical outcome. This shift from explicit definitions to implicit behaviors directly motivates the application of neural networks. Because specifying exact heuristics for complex environments is often intractable, neural networks serve as ideal function approximators. They excel at learning “good enough” heuristics—robust, non-linear mappings that can solve high-dimensional tasks without requiring a mathematically precise description of the solution space.

This formulation of Acting is pivotal when compared with Sensing component discussed in Part I. As defined in Chapter 2, the primary role of Sensing is to abstract raw signals into task-relevant information, essentially a process of interpretation and extraction. Acting, on the contrary, operates as an intervention process that converts task-relevant information into a goal-oriented decision—an action.

6.1.1 Acting in embodied systems

In robotics, Acting presents a significantly higher degree of difficulty compared to Sensing. Sensing is primarily an observational process: the system perceives raw signals and processes them to extract task-relevant information. Although challenging, Sensing is essentially a challenge-suited information provider, determining “what is currently happening” or “what is the state of the observable part of the environment?” In contrast, Acting is an interventionist process with an ultimate goal that must be reached to succeed. It is rarely a discrete classification problem but rather a continuous optimization process determining “how well can we follow a policy given physical constraints?” The complexity of execution is compounded by hardware limitations; not every desired action is feasible due to kinematic constraints, self-collisions, or torque limits.

This distinction introduces a critical data acquisition bottleneck. In Part I we observed that, while collecting haptic data is demanding, it can often be performed passively while the robot operates safely. However, acquiring data to learn complex action policies in the real world is exponentially more expensive and risky. To learn a robust policy, an agent must explore the boundaries of failure. In scenarios such as the “crossing the stream” problem, where a robot must navigate slippery stones to reach the other bank, a failure in the acting policy results in a catastrophic fall. Therefore, relying solely on real-world trial-and-error to learn acting policies is often infeasible.

6.1.2 The Oracle and the Apprentice: a paradigm for learning

The central theme of this part of the thesis operates on a dichotomy that parallels the dual-process theory of cognition described by Kahneman, 2011. The author distinguishes between two modes of thought: “System 1”, which is fast, instinctive, and automatic and

“System 2”, which is slow, deliberative, and logical. A fundamental challenge in robotics is replicating the balance between them to resolve the trade-off between decision-making accuracy and computational speed.

In embodied systems, a complete and explicit acting policy serves as the *slow Oracle*, analogous to Kahneman’s System 2. This component is able to perform a comprehensive analysis of walking scenarios, meticulously evaluating complex dynamics, kinematic limits, and environmental constraints to determine the most efficient course of action. However, this analytical rigor comes at a high computational cost. A robot relying exclusively on this “deliberate” mode for every footstep would be forced to pause and plan extensively, making fluid and dynamic locomotion impossible. Furthermore, the inherent latency of such comprehensive planning creates a critical vulnerability in dynamic environments. Because the real world is continuously evolving, a detailed, long-term action plan may become obsolete before it can be executed; by the time the system finishes computing the optimal sequence of actions, the underlying environmental state may have fundamentally changed, resulting with the calculated trajectory invalid.

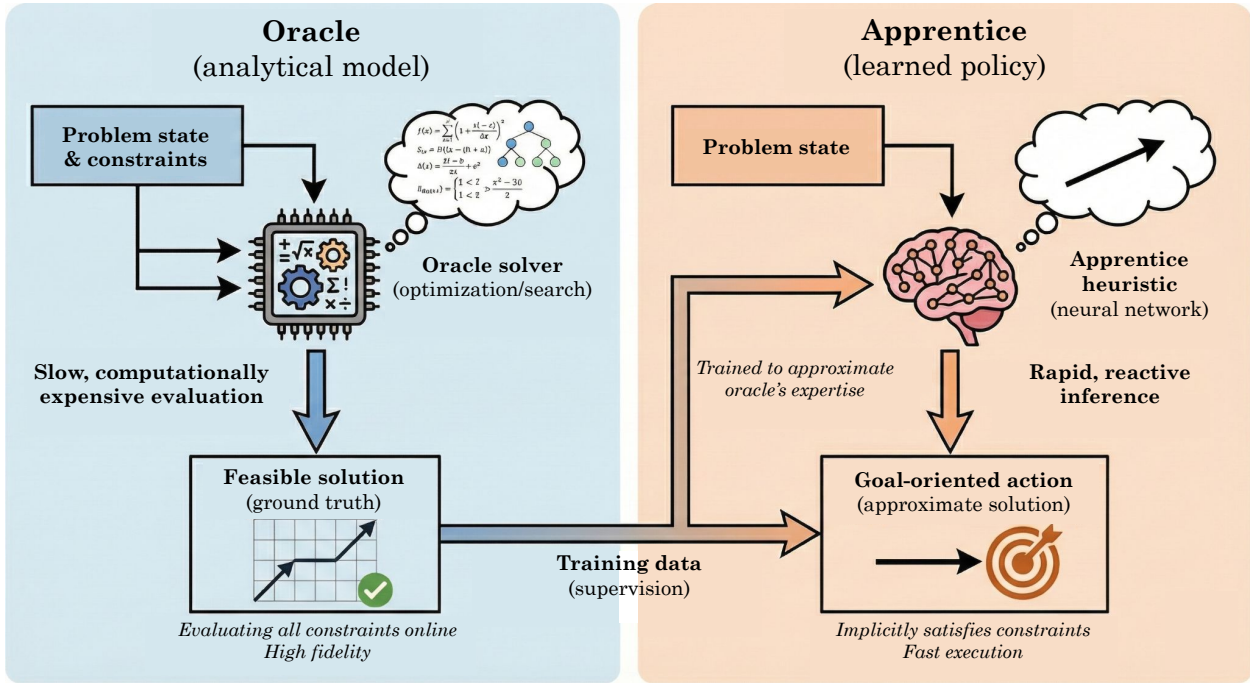


Figure 6.1: Conceptual comparison between the Oracle and the Apprentice paradigms. The Oracle acts as a rigorous solver, conscientiously evaluating the system dynamics and environmental constraints to derive a formally grounded, feasible solution. While this process provides high-fidelity signals, it is computationally intensive and characterized by high latency. In contrast, the Apprentice is a learned policy trained to approximate the expertise. It replaces explicit optimization with rapid, reactive inference, producing a goal-oriented intervention (an action) that implicitly satisfies constraints without the need for online numerical solving.

To overcome this limitation, we propose an architectural mapping that realizes these cognitive roles through an Oracle-Apprentice framework. Specifically, we construct an implicit Acting policy with a neural network that serves as the *fast Apprentice*, representing a functional implementation of System 1. The objective is to distill the knowledge of the slow and analytical model into a fast and reflexive policy. This approach extends the concept of *Knowledge Distillation* (G. E. Hinton et al., 2015) to the dual-process paradigm, where the complex, computationally expensive Oracle provides high-fidelity supervision to a lightweight Apprentice network optimized for real-time inference. A visual comparison of these two

paradigms is presented in Figure 6.1. The completeness of the Oracle makes it possible to provide expected input signals from a simulated environment and synthesize outcomes. By training on the Oracle-generated data the Apprentice learns to approximate the feasible decision-making policy but executes it in real-time. Although the Apprentice network may not retain the absolute precision of the analytical Oracle, it is designed to be sufficiently accurate to enable the robot to operate dynamically with satisfying performance, effectively bridging the gap between rigorous analysis and reactive agility.

The choice of the terms *Oracle* and *Apprentice* in this work, rather than the more conventional *teacher* and *student* prevalent in machine learning literature, is deliberate and reflects a fundamental distinction in the components’ roles. The Oracle is not designed with the inherent intent of instruction; its primary purpose is the rigorous, explicit, and accurate generation of feasible solutions within its formal domain. It is an authority on the physical or logical constraints it navigates, operating independently of the learning process. Conversely, the Apprentice is not a passive recipient of knowledge, but a component defined by its focus on practice and imitation. Much like a medical intern or a craft apprentice, this network observes the high-fidelity outputs of the Oracle and seeks to internalize those complex behaviors into its own reactive repertoire. This nomenclature emphasizes the relationship of observation and mimicry rather than explicit pedagogical interaction, effectively capturing the transition from deliberate, analytical computation to rapid, practitioner-like execution.

6.1.3 Objective: learning from synthetic data

The prohibitive cost and risk associated with acquiring real-world interaction data require a fundamental shift in the learning paradigm for embodied agents. If the physical world is too dangerous or slow for the initial phases of policy acquisition, the learning process must be decoupled from physical execution. The primary objective of this part of the thesis is to investigate the efficacy of *synthetic data*—synthetic information generated by analytical models—as a sufficient proxy to learn robust physical policies.

In this context, we define synthetic data not only as simulated sensor readings but as the codified output of the Oracle. These data represent the idealized physics, kinematic constraints, and geometric relationships of the robot-environment **interaction**. The core premise is that a problem, if defined at a sufficiently high level of abstraction, can be fully captured by a synthetic model. By stripping away the aleatoric noise¹ of the real world and focusing on the fundamental principles of the task (e.g., stability, reachability, collision avoidance), we can generate a training distribution that is mathematically precise and covers the entire manifold of the robot’s state space—including hazardous configurations impossible to explore physically. This approach is grounded in the following hypothesis:

H2. *We hypothesize that a neural network acting as an “Apprentice” can distill the complex, non-linear decision boundaries of an analytical “Oracle” solely from synthetic data. Furthermore, we posit that a policy learned from this high-level abstract representation captures the essential dynamics of the interaction sufficiently well to allow for direct, zero-shot transfer to the physical world, effectively bridging the gap between analytical rigor and real-time execution.*

¹In the context of robotic perception and learning, a distinction is made between aleatoric and epistemic uncertainty. Aleatoric uncertainty refers to the stochasticity or “noise” in the environment—such as measurement quality or unpredictable physical interactions—which persists regardless of data quantity. In contrast, epistemic (or systemic) uncertainty reflects a lack of knowledge about the model’s parameters or structure and can be reduced through additional training or exposure to diverse data (Hüllermeier & Waegeman, 2019).

6.2 Key challenges in learning robot actions

Developing effective and robust physical policies for legged robots is fundamentally constrained by three interconnected factors: the scarcity of training data, the latency of decision-making, and the fidelity of environmental modeling. To achieve the goal of dynamic, autonomous operation, the following challenges must be overcome.

The sim-to-real gap. To avoid data acquisition bottleneck, it is necessary to shift training to a simulated environment. This introduces the challenge of the “sim-to-real gap”, the discrepancy between the synthetic training environment and the physical world. A policy optimized in a simulation may exploit numerical inaccuracies or simplified physics, leading to poor performance when transferred to the real robot. This challenge is particularly acute for complex mechanical systems such as soft feet, where accurately modeling material compliance and friction in a simulator is difficult and computationally slow, complicating the generation of high-fidelity synthetic data.

The data acquisition bottleneck. Learning sophisticated acting policies typically requires massive datasets that cover a wide distribution of state-action pairs. For embodied agents, the acquisition of these data in the physical world is severely strained by the cost of operation and the risk of hardware damage. Unlike sensing, active policy learning requires the agent to explore the limits of its capabilities, often leading to failures. In high-stakes scenarios, such as traversing disjointed terrain, a single failure can be catastrophic. Consequently, gathering a dataset large enough to train a robust deep learning model solely through physical interaction is often practically infeasible.

Computational speed and policy complexity. Dynamic locomotion requires a control loop capable of making decisions at high frequencies. However, the analytical models required to determine feasible actions—such as foothold selection—are computationally intensive. Rigorous evaluation of potential actions involves checking the kinematic reachability, collision avoidance, and stability margins against complex terrain geometries. This computational bottleneck increases with the integration of mechanically adaptive soft feet, where contact dynamics requires complex deformation modeling. Traditional analytical solvers cannot meet real-time deployment constraints. This latency forces a direct trade-off between model accuracy and execution speed.

Chapter 7

Single-shot foothold selection and constraint evaluation for quadruped locomotion

7.1 Motivation

Locomotion on challenging terrain requires careful selection of footholds. The robot should select the locations that provide stable support for each foot to avoid slippages and falls. This strategy is crucial when the robot deals with highly irregular terrain. Another challenging example is the problem of “crossing the stream” when the robot has to reach another bank of the river on the stones which are the only acceptable footholds.

In contrast, other types of locomotion assume that the robot walks dynamically on rough terrain and stabilizes its posture using fast control algorithms and compliant legs (Hutter et al., 2016; Hyun et al., 2014). This approach is efficient on moderately rough terrain. In this case, the robot does not have to build the precise model of the terrain. The problem of stable locomotion is solved by the controller, which reacts to disturbances such as slippages resulting from unstable footholds. To efficiently navigate in an unknown environment, this kind of robot needs the perception system which allows one to detect high obstacles and avoid them while walking (Wooden et al., 2010). However, this type of locomotion will fail when the robot has to face the “crossing the stream” problem.

The deliberative approach to rough terrain locomotion requires a good model of the terrain. In this chapter, we focus on the use of elevation map to find the best foothold for each leg. The elevation map is efficient when the height of the obstacles is less than the height of the robot (Belter, Labecki, et al., 2016). To obtain a complete 3D model of the environment, OctoMap (Hornung et al., 2013) or Normal Distribution Transform Occupancy Maps (NDT-OM) (Saarinen et al., 2013) can be used. However, computation and memory requirements can grow substantially as the resolution of the map is increased. In our case, we are looking for computationally efficient solutions. To select the foothold for a single leg, the controller has to assess 400 (the usual resolution of a workspace for the next step) potential footholds, and only the elevation map guarantees constant and quick access to every piece of ground. In addition, the elevation map can be directly transformed to the grayscale image to feed the convolutional neural network (CNN, Krizhevsky et al., 2012) that we use in this research to select the best foothold.

The problem of foothold selection is presented in Figure 7.1. To select the best foothold, the robot evaluates the region (elevation map) for the i -th leg. The robot should avoid sharp edges and footholds that are located on steep slopes because they are potentially risky. The local shape of the elevation map is not the only factor that should be taken into account when the robot selects footholds. The selected foothold should be within the kinematic range of the robot (inside the leg workspace). The robot should also consider collisions with the terrain and self-collisions and exclude potential footholds that do not fulfill these requirements. In the classical approach, all constraints are verified sequentially by the robot controller (Belter & Skrzypczyński, 2011). In this research, we propose the neural network that evaluates the properties of local terrain to determine the quality of potential footholds in a single model forward pass. The neural network provides the results which take into account all previously

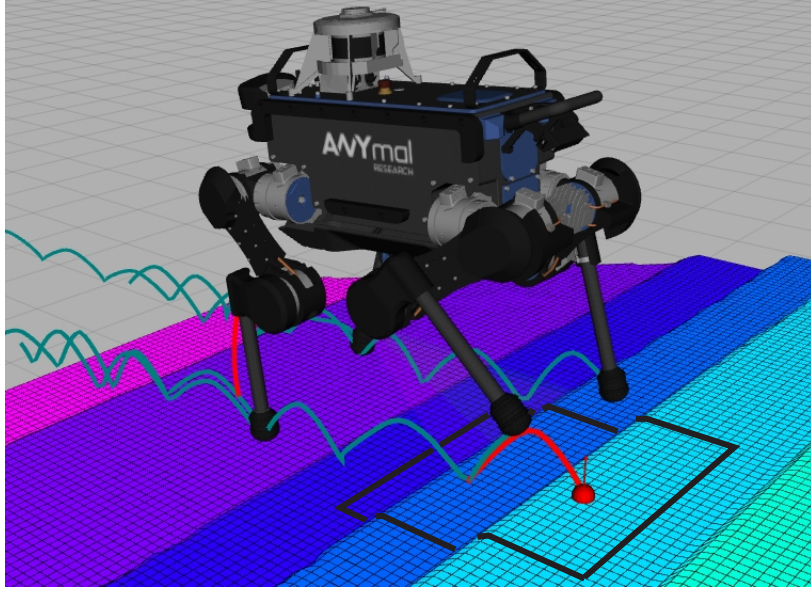


Figure 7.1: The foothold selection problem for a quadruped robot. The region of the elevation map below the i -th leg is evaluated to find the best foothold. Each candidate position of the foot has to be kinematically feasible and collision-free. All coordinate systems are defined in the world frame W .

mentioned constraints.

7.1.1 Related work

The problem of foothold selection is similar to the problem of multi-finger grasping and was studied widely by the robotics community. Recent developments in this field include the method that uses local geometrical properties of objects to find acceptable positions of the fingertips on the surface of the object (Kopicki et al., 2015). The grasp configurations are trained from the real examples. The collision and kinematic constraints are taken into account during the inference procedure. Recently, CNNs have gained great popularity in robotics applications. Also, for grasp detection, CNN is applied to select feasible grasp and robotics finger positions on the object's surface using point cloud (Mahler et al., 2017) or depth images (Gualtieri et al., 2016).

Before the broad application of deep neural networks, most approaches were based on local features computed for the surface of the terrain. The Ambler robot computes the inclination of the terrain, the roughness, and the local curvature of the elevation maps (Krotkov & Simmons, 1996). These features are provided to the input of the simple neural network which was trained on the data provided by the human expert. In contrast to this approach, the CNN-based method does not depend on the manually computed features, but extracts features automatically from the data. Another method that uses the elevation map was proposed by Chun-Hung Chen and Kumar, 1996. The method returns the probability map which is related to the capability of each cell to provide stable support for the robot's feet. The unique method for foothold assessment is proposed by M. A. Hoepflinger et al., 2013. The robot equipped with the haptic device at the tip of the leg can explore and assign the cost to the potential foothold of the terrain without human supervision. The HyQ robot controller is more focused on the reflexes that stabilize the robot (Focchi et al., 2018). Visual information about the terrain is used to place the foot on the surface of the terrain without avoiding risky footholds. The approach that is more similar to our work is presented on the same HyQ robot (Barasuol et al., 2015). The robot corrects the nominal foothold position according to the output from the visual pattern classifier applied on the terrain patches. The

map is built on-line using the RGB-D sensor.

Great progress in the field of autonomous legged locomotion on rough terrain was made with the quadruped robot LittleDog. The authors Rebula et al., 2007 proposed a terrain scorer that assesses potential footholds. The scorer computes the spatial relation between the considered point and neighboring points and rejects points which are located on edges, on the too large slope, on the base of the cliff or inside a hole. The learning based method for the LittleDog robot was proposed by Kalakrishnan et al., 2009. The robot learns to evaluate terrain templates based on human demonstration. The terrain scorer is also trained in another method proposed for the LittleDog robot (Kolter et al., 2008). The weights in the linear combination of geometric features of the terrain are obtained during training and used to plan footsteps on rough terrain.

The foothold selection method for a six-legged robot is represented by the method implemented on the Lauron IV robot (Roennau et al., 2009). The foothold selection module considers points around the initial foothold and takes into account the credibility of the elevation, the mean height, and the variance in height of the cells. The other learning-based method was presented on the six-legged robot Messor (Belter & Skrzypczyński, 2011). The robot learns in the simulation which points on the elevation map described by local geometrical features provide stable support for the robot. Then, the trained Gaussian Mixture is used to select the footholds in the RRT-based motion planner (Belter, Łabźcki, & Skrzypczyński, 2016). The kinematic and self-collision constraints are also taken into account but the constraints checking significantly slow-down the foothold selection process.

7.1.2 Approach and contribution

In this chapter, we propose a new approach to the problem of evaluating potential footholds for the quadruped robot. The method uses CNN that performs evaluation for all potential footholds in a single forward pass of the model. We collect data to train the network using the robot kinematic model and the rough terrain map. The method takes into account the local geometrical properties of the elevation map. The network at the same time takes into account kinematic and collision constraints. The network implicitly learns the kinematic model of the leg and can detect footholds that are outside the workspace. The trained model also rejects footholds that are not collision-free. We are the first who show that the CNN can be used to evaluate geometrical properties of the potential positions (footholds) on the map and simultaneously consider all kinematic constraints which are related to the model of the robot. It means that we can evaluate 400 potential footholds and check constraints in a single inference step which takes about 10 ms on the CPU.

7.2 Details

The proposed foothold selection module evaluates the potential footholds that are within the local map extracted from the elevation map built by the robot (Fankhauser et al., 2014). The center of the global map is always below the center of the robot. The size of the map built by the robot is 6×6 m with a cell size of 2×2 cm. The local map is presented in Figure 7.2. The center of the map M_i is always below the joint related to the considered leg. The size of the local map is 40×40 cells. The size of the local map is set to cover the maximal kinematic range of the robot's leg. To select the optimal foothold, we also consider the nominal foothold N_i given by the locomotion controller. The nominal foothold is computed for the desired motion of the robot assuming that the robot is walking on a flat terrain.

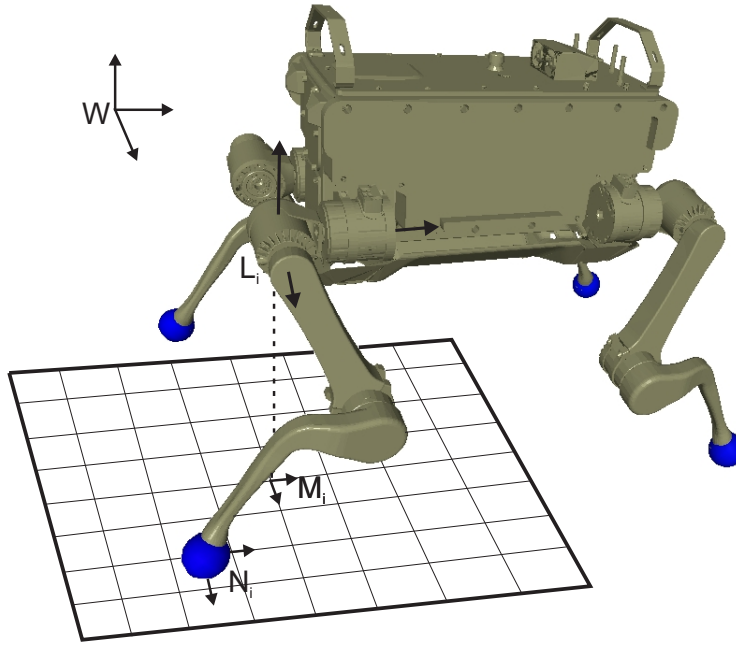


Figure 7.2: Center of the local map M_i used for foothold selection is located below the i -th leg joint L_i . The foothold selection algorithm considers also the nominal position of the foot N_i .

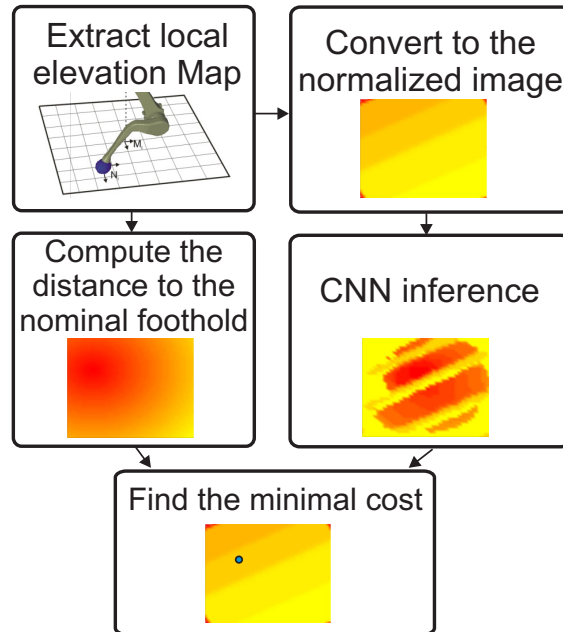


Figure 7.3: Foothold selection procedure on the local elevation map with the convolutional neural network

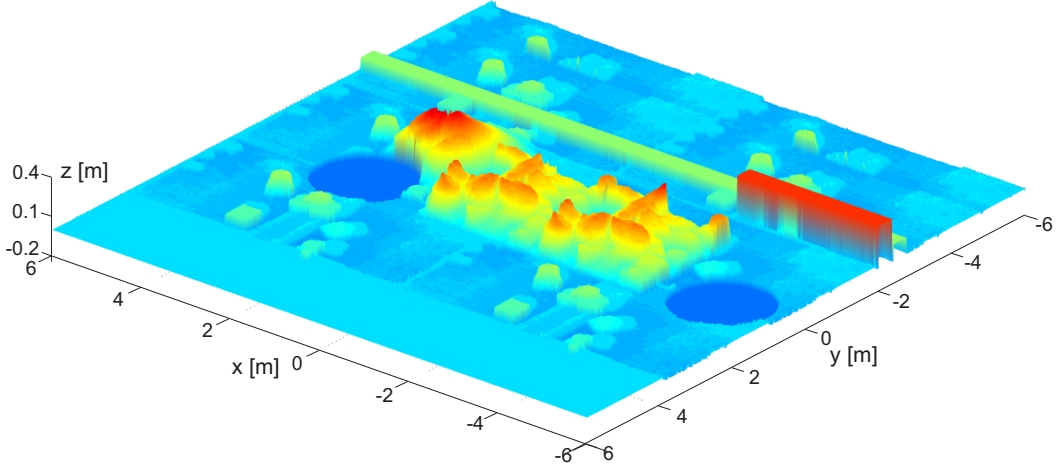


Figure 7.4: Elevation map used to collect data for training.

7.2.1 Inference procedure

The inference procedure is presented in Figure 7.3. In the first step, we crop the global map built by the robot to the area of potential footholds. The map obtained aligns with the world coordinate system W , but our neural network uses the elevation map that aligns with the robot coordinate system. Thus, we rotate the obtained local map by the current orientation of the robot on the horizontal plane (yaw angle). Because during rotation of the elevation map we lose some information about cells that are located in the corners, we took the map that is larger than the final sub-map. Before rotation, the size of the local map is 51×51 cells and after rotation, we crop again the map to size 40×40 cells.

In the next step, we convert the sub-map to the image. To this end, we compute the distance between the i -th leg coordinate system L_i and each cell of the map. We use 8-bit grayscale images as an input to the network, so the obtained distance values are fitted into the range 0–255. We use a constant normalization factor (0.85 m) for each leg of the robot. The distance that is smaller than the normalization factor is represented by the minimal value in the image (0) and the distance that is larger than the normalization factor is represented by the maximal value in the image (255).

The resulting image that represents the terrain patch around the considered leg is provided to the input of the neural network. The network classifies each pixel on the image. The returned id of the class Z_c represents the cost associated with each class. The lower the id of the class, the lower the cost of the foothold. The example inference results for the input image that represents stairs are presented in Figure 7.3. The pixels that are located on the stairs' edges (between treads) are brighter, which means that the robot should avoid these footholds. At the same stage of the inference procedure, we compute the distance from the nominal foothold d_n . Then, we compute the final cost c_{final} for each pixel (foothold):

$$c_{\text{final}} = Z_c + k \cdot d_n, \quad (7.1)$$

where k is the constant value that determines the influence of the distance from the nominal foothold on the final cost of the potential foothold. In the experiments presented in Section **Results** the value k is set to 160. We compute the final cost c_{final} for each pixel in the image (c.f. Figure 7.3) and find the minimal value. Then the image coordinates of the pixel with the minimal cost is converted to 3D coordinates in the world coordinate system. The calculated coordinate is sent to the controller that executes the motion to the given foothold.

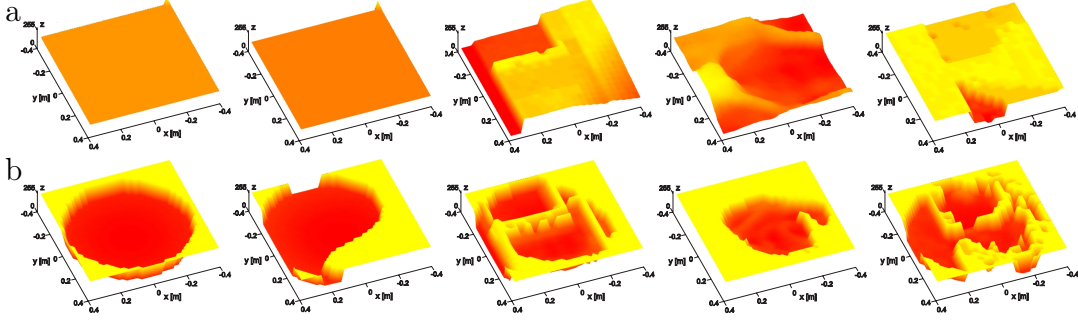


Figure 7.5: Example training data: local elevation maps (a), and corresponding terrain cost (b)

7.2.2 Dataset

To train the neural network, we gather the samples on the elevation map presented in Figure 7.5. We collected data for two legs only (one front and one rear) and trained two separate models. The ANYmal robot used in this research is symmetrical and we can use the same model to evaluate foothold for the right and left legs. To gain from that property, we flip horizontally the input terrain map before the model inference, and once again, flip horizontally the resulting cost map.

The environment for data collection is a simulation built with Gazebo engine. Thus, we can freely move the robot model on the map and start the experiment. To generate data for training, we randomly select the position of the robot on the map (horizontal position and distance to the ground). The orientation of the robot on the horizontal plane (yaw angle) is randomly selected from four main orientations: $n \cdot \frac{\pi}{2}$, for $n = 0, 1, 2, 3$. For the pose obtained from the robot, we compute the pose of the i -th leg and extract a local elevation map of 40×40 (an input map).

The analytical model responsible for the evaluation of an input map is composed of five subsequent steps that check different constraints. The output of that model is used as a target for training the neural network. The constraints that are checked by the analytical model include:

- kinematic range,
- self-collision,
- collisions with the ground,
- kinematic margin,
- cost related to the local shape of the terrain.

First, we check the kinematic range. If the given position of the foot is outside the workspace of the considered leg, we set the cost at the maximal value (255). Then we check for self-collisions and collisions with the ground. To determine whether parts of the robot collide with each other, we create a mesh model of each part and use the Flexible Collision Library to determine collisions (Pan et al., 2012). The same procedure is performed to check for a collision with the ground. If we detect a collision, the cost of the foothold is set to 255 and we do not check other constraints.

If the given position of the foothold is collision-free and the foot is within the workspace, we compute the kinematic margin c_k . The kinematic margin is the distance between the current position of the foot and the border of the workspace. If the kinematic margin is equal to 0 the leg cannot be moved in one direction. The maximal value of the kinematic margin

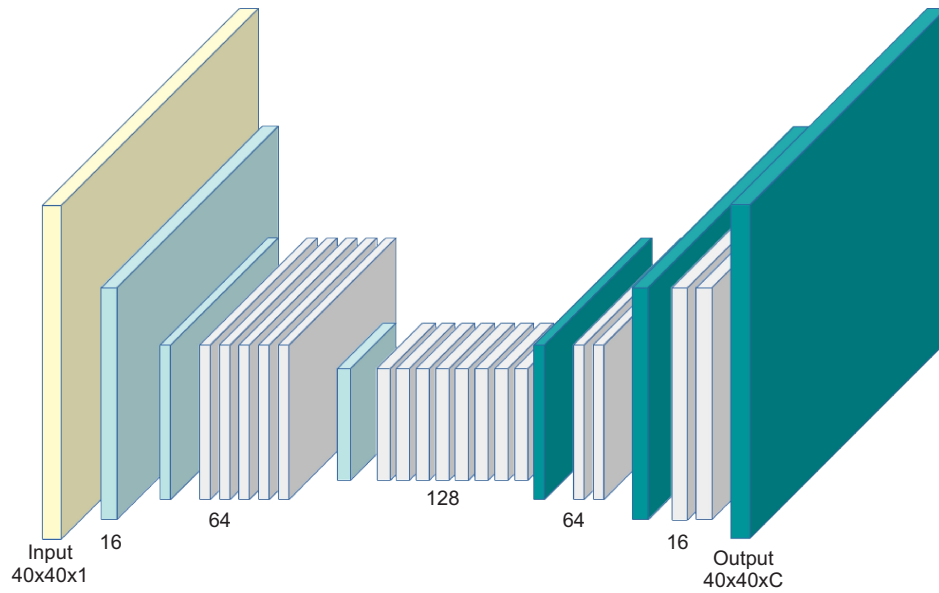


Figure 7.6: The model of ERF network. Light blue blocks represent downsampling, dark blue - upsampling by transposed convolution and while blocks show residual layers. Numbers below blocks describes the number of feature maps used in specific levels. C denotes the number of classes.

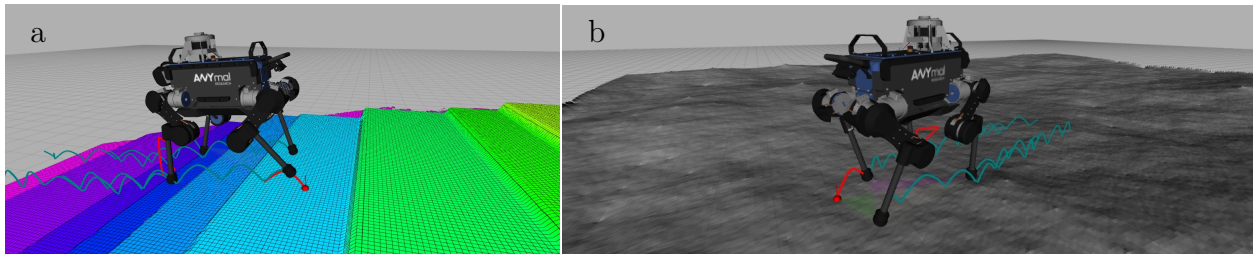


Figure 7.7: Experiment with the ANYmal robot on stairs (a) and on rough terrain (b) in the Gazebo simulator

means that the leg has the maximal motion range. Finally, we compute the cost which depends on the properties of the local map. To calculate the cost c_m we use the function originally used for the hexapod robot Messor (Belter & Skrzypczyński, 2011). Both robots are equipped with the same hemispherical feet and the cost function does not depend on the robot kinematic model, thus the method can be easily adapted for the ANYmal robot. Finally, we compute the ultimate cost of the potential foothold c_f and scale the cost to the range $<0,255>$:

$$c_f = \frac{c_k + 2 \cdot c_m}{3} \cdot 255. \quad (7.2)$$

We repeat the procedure for each cell (potential foothold) of the input elevation map resulting with a foothold cost map. Then, we save input (elevation map) and output (cost map) as 8-bit images. Example training data are presented in Figure 7.5. The first two columns present the computed cost map for the flat terrain. In this case, the output depends on the leg’s workspace and the kinematic margin. The distance between the terrain and the robot is greater on the map in the first column of Figure 7.5a than in the second column of Figure 7.5a. The output cost map also differs. The obtained cost map represents the horizontal cross-section over the workspace of the robot’s leg. The yellow cells represent positions of the foot that are outside the workspace and are inaccessible to the robot. The red cells are correlated with the value of the kinematic margin. In the following examples in Figure 7.5 the terrain is irregular and we can observe how the workspace of the robot is limited by the terrain shape. The edges of obstacles significantly increase the cost of footholds.

7.2.3 Convolutional Neural Network

The neural network is constructed to process the input images represented by the local elevation map and produce the evaluation of potential footholds, emulating the analytical model. We selected the architecture that can be run in real-time on the machine without GPGPUs. Because the initial architecture search suggested that neural networks are much more efficient in solving the classification problem than the regression problem, we discretize the input cost and divide the training examples into classes. The neural network returns the ID of the class that is related to the predicted terrain cost. The discretization process was automated using an equal-frequency binning strategy. Cells were sorted by cost and partitioned into consecutive intervals, where each class was dynamically expanded until it contained at least 5% of the total dataset.

The proposed CNN architecture is an Efficient Residual Factorized ConvNet (ERF) first introduced in Romera et al., 2017. The characteristic property of this model is related to the modification of the *residual layer* (Romera et al., 2017) called *the 1D non-bottleneck residual layer*. The 2D convolution with filter shape 3×3 is replaced by two 2D convolutions with filter shapes 3×1 and 1×3 (which actually perform the 1D convolution operation). This approach reduces the number of variables and a computational complexity. The ERF model is shown in Figure 7.6. First, the input data are processed by two downsampling blocks. The downsampling blocks are created from the concatenation of the max pooling layer and the 2D convolution with filter shape 3×3 , and the stride set to 2. The concatenation is followed by the activation function. Then, five residual layers and another downsample block are added. The output of the encoder part is processed by eight residual layers, which are interwoven with different dilation rates applied to the convolutions. The decoder part of the model consists of two series of convolutional upsamples and two residual layers. The upsampling is performed by transposing convolution with the stride set to 2. The output of the model is produced by upsampling convolution with filter shape 2×2 and the number of filters equal to

Leg	Accuracy [%]	IoU
front leg	82.61	49.9
rear leg	82.61	49.88

Table 7.1: Accuracy and Intersection over Union (IoU) obtained on validation set for front and rear leg models

the number of classes and stride 2. The activation function used in each nonlinear layer is *ReLU* (Nair & Hinton, 2010).

The optimized objective of the model is composed of cross-entropy loss and regularization loss. The cross-entropy is additionally weighted according to the ground truth number of examples for each class label in the training dataset. We computed statistics for the training datasets to find weights related to each class such as in (Paszke et al., 2016).

$$w_i = \frac{1}{\log(c + p_i)} \quad (7.3)$$

Where w_i is the weight of the i -th class, c is a constant, and p_i is a probability of the occurrence of the i -th class based on the share of examples marked as the i -th in the entire training dataset. In this application, we used $c = 1.08$. The weighting of the cross-entropy allows the handling of unbalanced data. In the training set, the majority of examples are provided for the class that represents footholds inaccessible for the robot (cost equal to 255). Without balancing the training data, the network learns to recognize the predominant class and very often misclassifies the remaining classes.

7.2.4 Training the network

The optimization method used to train the model is Adam (Kingma & Ba, 2014) with an initial learning rate of $5e-4$. Additionally, the exponential decay was applied after each epoch to the learning rate by a factor of 0.98. Due to the nature of the training examples, we cannot use any of the known data augmentation methods. To measure the quality of a model, the accuracy and intersection over union (IoU) metrics were monitored. The learning process lasts 500 epochs. In Table 7.1 we show the results obtained by two ERF models trained separately for front and rear legs.

7.3 Results

The first stage of evaluation the proposed foothold selection method was performed in the Gazebo simulator with the model of ANYmal in a scenario of walking on stairs (Figure 7.7a) and rough terrain (Figure 7.7b). The robot uses a simulated model of Intel RealSense D435 RGB-D sensor to build a map of the environment (Fankhauser et al., 2014). We use the controller presented in Fankhauser et al., 2018 to plan the foot trajectories above the obstacles, estimate the state of the robot, and execute the planned trajectories. We only replaced the foothold selection model.

The example inference results are presented in Figure 7.8. Each row represents a separate example composed of a terrain patch extracted from the global elevation map (an input map), the distance between potential and nominal footholds, the output from the CNN and the ultimate cost map with a selected foothold. The examples a, b, and c (walking on stairs experiment) provide an intuition behind the trained foothold selection policy to avoid placing feet on the edges of stairs. These regions are classified by the neural network as risky (bright regions) and rejected by the foothold selection module. Similar behavior can be observed

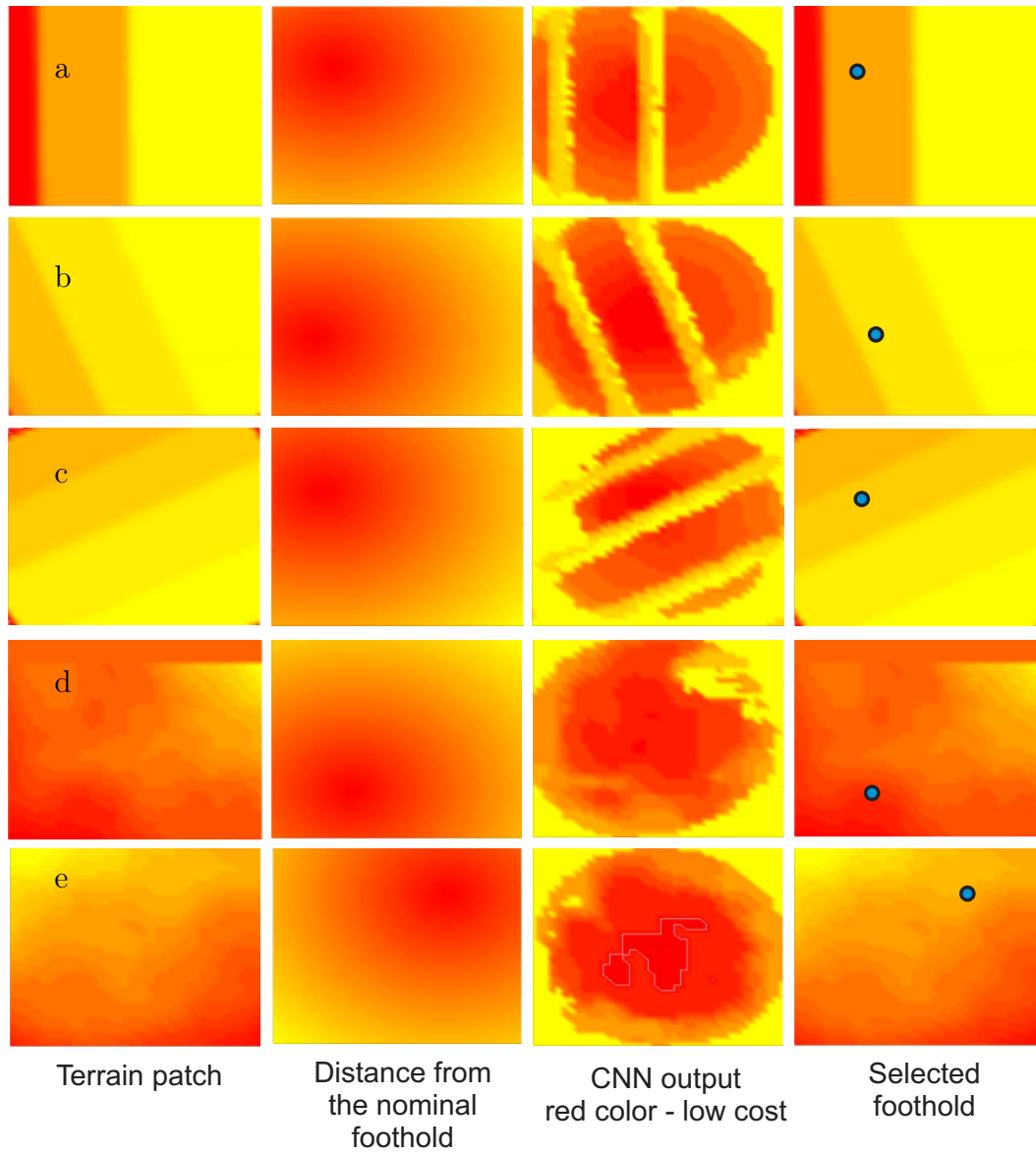


Figure 7.8: Example inference results obtained during the experiment on stairs (a,b,c) and on rough terrain (d,e)

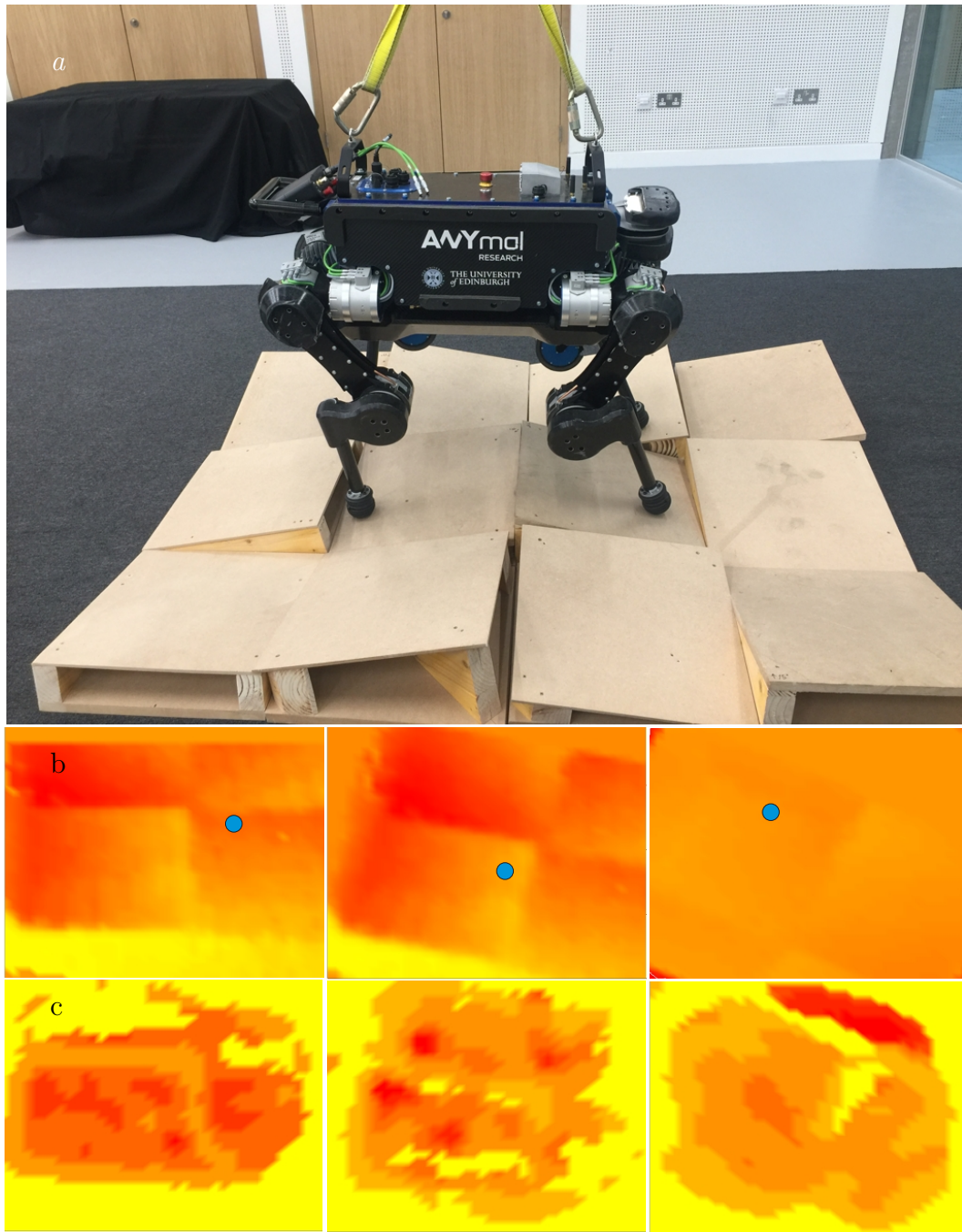


Figure 7.9: Experiment with the ANYmal robot on the rough terrain mockup (a): example terrain patches (b) and CNN output

in the results obtained on rough terrain (examples d and c). In those cases, the obstacles are more irregular. For both patches obtained on rough terrain, the region in the center of the workspace has a similar cost. In this case, the distance from the nominal foothold plays an important role. The selected footholds are close to the nominal foothold, but still in the position with acceptable foothold cost predicted by the CNN.

Finally, we performed the experiments on the real robot. Example results are presented in Figure 7.9. The input elevation map (Figure 7.9b) is less accurate than the map from the simulation, but the robot can still identify risky edges and place its feet in stable positions, ultimately leading to a successful passage through the irregular terrain shown in Figure 7.9a.

7.3.1 Conclusions and future work

In this chapter, we propose a new foothold selection method for the quadruped walking robot. The developed method is based on an attempt to emulate an accurate, but slow, analytical model by a neural network. In contrast to methods known from the literature, the proposed neural network evaluates the terrain patches and checks all constraints in a single step. The proposed procedure significantly reduces the time required for inference and imitates the analytical model well enough for the robot to operate effectively in real time and overcome obstacles, taking steps in safe locations. With the proposed method, the robot avoids placing its feet on the edges or steep slopes. The neural network also implicitly takes into account the kinematic range of the leg and detects self-collisions and collisions with the ground. The proposed foothold selection module is integrated with the robot controller. In the simulation and experiments on the real robot, we present the properties and efficiency of the proposed method.

7.4 Implications

The results presented in this chapter attempt to validate the hypothesis posited in Section 6.1.3: that a neural network, acting as an Apprentice, can successfully distill the complex decision boundaries of an analytical Oracle solely from synthetic data. By training the CNN on procedurally generated elevation maps, we demonstrated that the network could implicitly internalize the robot’s physical constraints—such as workspace reachability and collision geometries—without ever observing a real robot. This successful transfer from abstract simulation to real-world execution confirms that high-fidelity synthetic data is a sufficient proxy for learning robust physical policies, effectively bypassing the data acquisition bottleneck inherent in embodied robotics.

Furthermore, this “single-shot” approach fundamentally alters the computational paradigm of the foothold selection approaches. Traditional methods rely on sequential, computationally expensive checks for every potential foothold, often creating a latency that hinders dynamic motion. By condensing these evaluations into a single rapid inference step (~ 10 ms), the neural component transforms Acting policy from a slow deliberative process into a fast reactive reflex. This shift not only enables real-time adaptability on challenging terrain, but also establishes the architectural precedent for the fast Apprentice model, proving that neural approximations can deliver the necessary precision for mechanical interaction while maintaining the speed required for agile locomotion.

Chapter 8

CNN-based foothold selection for mechanically adaptive soft foot

8.1 Motivation

Walking robots, like animals, should adapt to terrain when walking on rough terrains. Robots can deal with small irregularities without careful foothold selection (Wooden et al., 2010) and react dynamically to unexpected slippages. However, this approach is risky when the robot is climbing a rough and challenging environment. Additional feedback from the perception system allows us to analyze the terrain profile and select the suboptimal position for the foot (Belter et al., 2019; Villarreal et al., 2019). In this scenario, the robot considers the elevation map of the terrain and modifies the nominal position of the foot taking into account the slope of the terrain and the risky edges of the obstacles. Despite visual feedback, the deliberative approach to footstep planning can be applied together with dynamic gaits (Villarreal et al., 2019).

Dynamic walking (Wooden et al., 2010) and foothold adaptation based on visual feedback can also be supported by the mechanical adaptation of the robot. Robotic manipulators use hands with compliant fingers that can adapt to the shape of various objects (Catalano et al., 2014). This approach improves the stability of the grasp only with mechanical feedback. A similar method can be applied when designing the foot for a walking robot (Piazza et al., 2016). The mechanical design of the foot used in this research improves the properties of contact between the robot and the terrain and increases the stability of the robot on rough terrain.

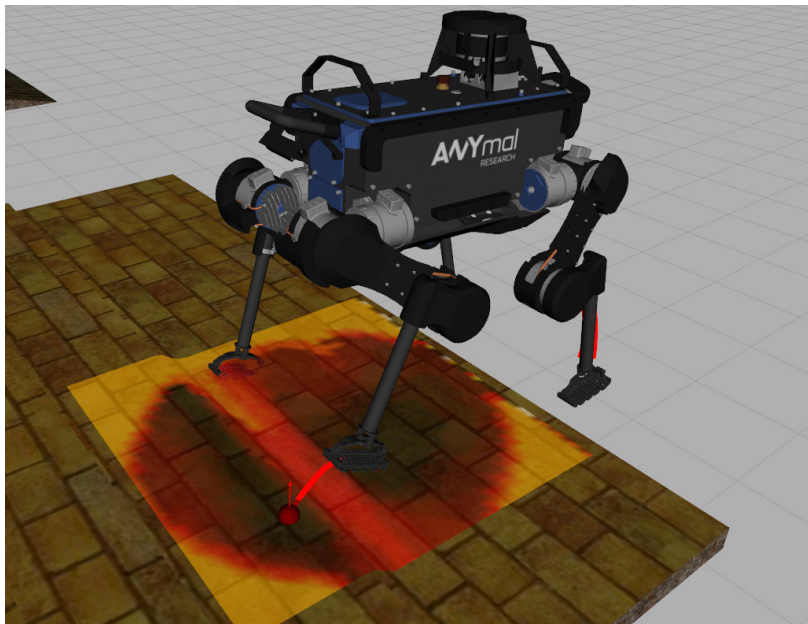


Figure 8.1: Foothold selection procedure on the ANYmal robot: the convolutional neural network provides the cost of each foothold on the local patch of the terrain taking into account properties of the compliant foot and robot constraints

In this chapter, we combine two approaches and take advantage of mechanical and visual feedback. We employ the SoftFoot-Q adaptive foot on the ANYmal quadruped robot. SoftFoot-Q is a mechanically adaptive foot for quadrupedal robots inspired by Piazza et al., 2016. These compliant feet increase the stability of the robot by improving grip and traction. The mechanical design compensates small irregularities of the terrain and the robot can walk on moderately rough terrains without visual feedback. However, when walking in dangerous and rough terrains, the robot should still avoid regions outside the workspace of the legs, avoid high slopes, collisions, and select footholds on obstacles that provide a better grip of the foot. To this end, in this chapter, we also utilize visual feedback to select footholds.

The ANYmal robot and the example scenario considered in this research are presented in Figure 8.1. In contrast to Chapter 7, the robot is equipped with an adaptive soft foot (see Figure 8.5). During the planning of the next step of the robot, the nominal position of the foot is computed, which results from the desired length of the step and direction of motion. The nominal foothold is modified according to the local elevation map below the hip joint of the leg. The local elevation map is provided to the input of the neural network (convolutional neural network; CNN) that computes the cost map that is later used to find the best foothold (red sphere in Figure 8.1) taking into account the shape of the terrain, the motion constraints and the position of the nominal foothold.

8.1.1 Related Work

Foothold selection plays an integral part in quadrupedal robot locomotion, especially on challenging terrains. Most foothold cost evaluation methods utilize terrain properties such as height-maps, slopes, and slippage (Winkler et al., 2014). The authors Barasuol et al., 2015 evaluate candidate footholds through visual pattern classification based on height-maps. The algorithm is tested on the quadruped HyQ on rough terrains with obstacles. The 3D map is built using a motion capture system and an RGB-D sensor. During locomotion, the robot requests a local height-map around the nominal foothold and evaluates this candidate foothold through visual pattern classification.

Visual-based foothold selection requires long processing time; therefore, most evaluation algorithms resort to other sensors such as LiDARs and force-torque sensors. In Kalakrishnan et al., 2009, footholds are evaluated solely on height-maps. The costs are computed through matching with pre-selected terrain templates. The research in Kolter et al., 2008 presents a foothold evaluation method based on terrain elevations and slopes for the quadrupedal robot “Little Dog”. The algorithm consists of a two-level control strategy. The high-level controller evaluates foothold costs, while the low-level controller is responsible for calculating foot trajectories to reach the desired footholds. The controller aims at providing proper footholds for traversing rough terrains.

Real-time foothold evaluation is designed to traverse challenging terrains in the absence of prior information about the terrain properties. The authors Fankhauser et al., 2018 present a real-time foothold planning for quadruped ANYmal that aims to traverse rough terrains with stairs. The updated elevation map is used to assess the traversability of nominal footholds, while the trajectory planner guides the foot to the selected safe location (Fankhauser et al., 2018). The real-time dynamic foothold adaptation algorithm based on self-supervised classification is introduced in Villarreal et al., 2019. The proposed strategy enables the integration of visual information in terrain perception whilst maintaining online foothold evaluation.

All of the approaches covered in this section evaluate candidate footholds for point-type feet. In this chapter, we present a foothold selection algorithm for passive adaptive feet. In particular, we used the SoftFoot-Q, a compliant foot for quadrupedal robots, which is loosely

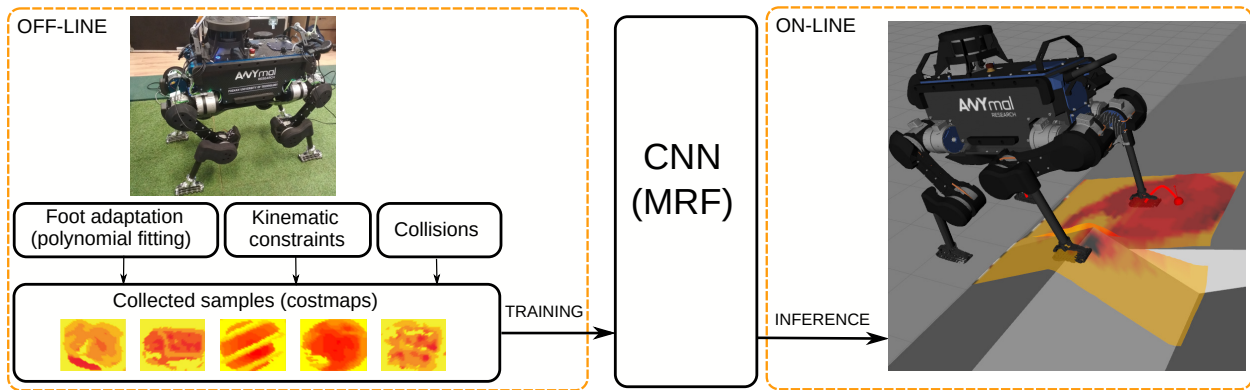


Figure 8.2: Architecture of the proposed system: data collection and CNN training are performed off-line but inference is performed on-line when the robot walks on rough terrain

inspired by Piazza et al., 2016. These feet are mounted on the quadruped ANYmal to provide increased compliance on different terrain types. The presented approach utilizes geometric information on the foot in addition to terrain properties to select the most compatible candidate foothold.

8.1.2 Approach and Contribution

In this chapter, we extend our previous approach(Chapter 7, (Belter et al., 2019)) to planning footsteps for a ball-like foot. We show that the proposed framework can also be applied to various feet models other than round-shaped feet. In addition, we demonstrate how to design the function that evaluates each potential foothold for the soft adaptive foot. Moreover, we improve the segmentation results provided by the neural network and provide a comparison to the other state-of-the-art results.

The main contributions of this chapter include the following:

1. To our knowledge, CNN has not been demonstrated before that evaluates potential footholds for compliant feet that can mechanically adapt to the terrain profile.
2. Analytical evaluation of the foothold cost that corresponds to the adaptation to the terrain patch and allows generating training data on the terrain map without the need to sample the terrain using the real robot.
3. Improved segmentation-based evaluation of potential footholds on the elevation map using a CNN.

8.2 Details

8.2.1 General architecture

The architecture of the proposed system is presented in Figure 8.2. The core component of the system is the CNN neural network that evaluates the elevation map below the hip joint of the considered leg. Data for training the neural network are collected off-line. After training, the CNN is used to infer about the cost of potential footholds. We train two neural networks: one for the right front leg and one for the right hind leg. Because the mechanical design of the robot is symmetrical, we use the obtained neural network to find footholds for the left legs by flipping the local elevation maps and transforming the results into the frame of the left leg.

Neural networks need many training examples to find a proper relationship between input and output data and generalize the results. Collecting data from the real robot is difficult and time-consuming. It requires measuring the quality of the selected foothold (e.g. by measuring slippage). This problem can be solved by simulation (Belter & Skrzypczyński, 2011), but the soft foot simulation used in this research is also slow due to its mechanical complexity. Instead, we propose the analytical function that describes how the feet adapt to the terrain patch considered and how far the obtained configuration of the foot is from the preferred one. In addition, we check the leg workspace and collisions with the terrain to avoid constrained configurations of the robot when selecting footholds. We use the proposed method to generate pairs of input elevation maps and the corresponding cost maps (Figure 8.2).

During inference, we provide the local map of the terrain below the hip joint to the input of the neural network. The size of the local map is 40×40 cells and the size of each cell is 2×2 cm. CNN provides the cost map, which is used to modify the nominal foothold – the desired position of the foot for a given step length and assuming that the robot is walking on flat terrain. To find the best foothold we consider the cost c_{final} taking into account the output of the neural network c_f and the distance from the nominal foothold d_n :

$$c_{\text{final}} = c_f + k \cdot d_n. \quad (8.1)$$

8.2.2 Mechanically adaptive soft foot

The SoftFoot-Q is a novel passive robotic foot designed specifically for quadrupedal robots, loosely inspired by Piazza et al., 2016. The core idea behind its design is that a mechanically adaptive sole can increase the support area, conform to the shape of the ground, and thus aid in reducing slippage and enhance the stability of locomotion.

Figure 8.3 highlights the main components and dimensions of the foot, which is a closed kinematic chain composed of the following components: i) an *ankle base* (A) meant to be connected to the robot leg; ii) two *arch links* (B) connected to the ankle by means of a revolute joint, which provides pitching movements on the longitudinal plane; iii) two *roll links* (C) connected to the extremities of the arches through two revolute joints that make rolling motions possible along the frontal plane; iv) four *paddled chains* (D) connecting the two roll links and creating an adaptable sole presenting a stiffening by compression behavior: these flexible chains deform as they contact the terrain, conform to its shape and gradually become rigid in extension.

The roll joints are designed to be positioned as low as possible, to increase the static stability of the foot. Indeed, since all forces applied by the leg to the foot must pass through the roll rotation axis, a higher position of the roll joint would increase the chance that the force applied by the foot would lie outside the support area. This problem is exacerbated by the small width of the foot, which must comply with specifications for its use on the quadrupedal robot ANYmal (Hutter et al., 2016). Finally, although the relative motion between the two arches tends to misalign the two roll links, the roll motion is still allowed by the play between the chain links. Play is also of extreme importance in guaranteeing adaptiveness to variations in the terrain profile in the lateral direction.

8.2.3 Foothold selection with polynomial fitting

The Pisa foot can adapt to the terrain profile. In the approach to foothold selection, we compute how the foot can fit the local shape of the terrain and how far the obtained configuration is from the preferred one. Example soft foot configurations are presented in Figure 8.5. The preferred foot configuration is presented in Figure 8.5b. as the mechanically adaptive sole has a stiffening behavior by compression and tends to act like a rope in a

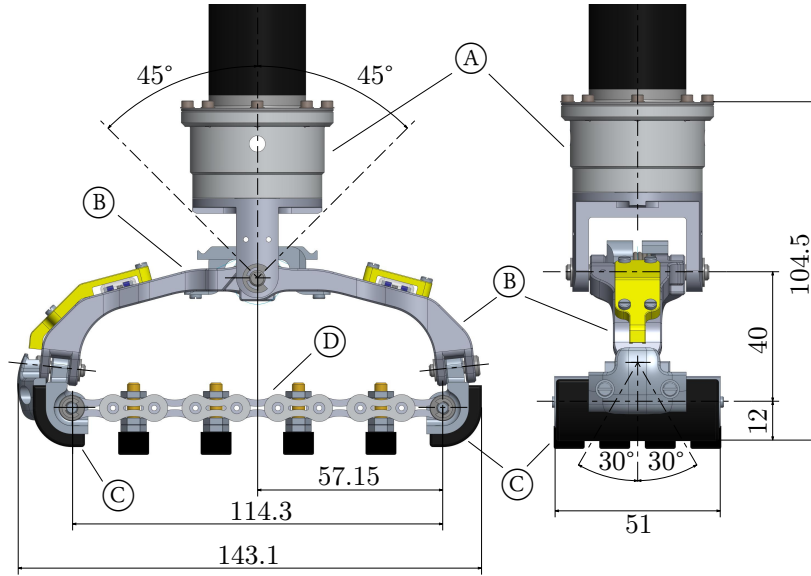


Figure 8.3: CAD of the main components and dimensions of SoftFoot-Q.

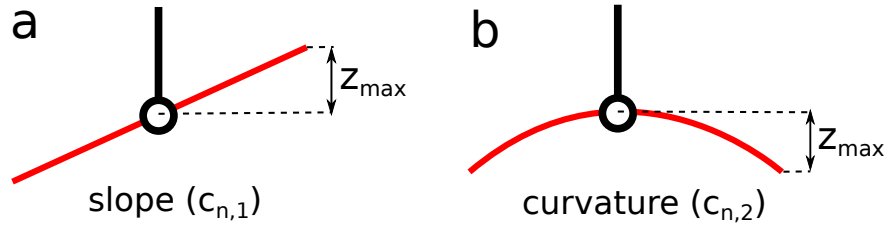


Figure 8.4: Parameters of the model and corresponding shape of the foot: slope (a), curvature (b)

tension state when approaching a convex hull of the soil. We model the shape of the foot using a second-order polynomial:

$$y = \sum_{n=1}^N \sum_{m=0}^M c_{n,m} \cdot x_n^m, \quad (8.2)$$

where x_n is the input coordinate (x_1 corresponds to the x axis and x_2 corresponds to y), $c_{n,m}$ is the constant value, $N = 2$, and $M = 2$.

During polynomial fitting, we search for the values of the parameters $c_{n,m}$ that fit the given terrain patch. Example polynomials and the corresponding foot shapes are presented in Figure 8.5. We limit the values of the polynomial parameters $c_{n,m}$ to better represent the shape and constraints of the real foot. The parameters of the polynomial have a physical meaning, and these values can easily be adjusted (Figure 8.4). $c_{n,0}$ corresponds to the offset, $c_{n,1}$ corresponds to the slope value, and $c_{n,2}$ represents the curvature along the n -th dimension. Finally, the cost of foothold is evaluated using the following formula.

$$p_p = \alpha_R c_R + \sum_{n=1}^2 \sum_{m=1}^2 \frac{\alpha_{n,m}}{c_{n,m}^{\max} - c_{n,m}^{\min}} \cdot |c_{n,m}^{\text{pref}} - c_{n,m}| \quad (8.3)$$

where $\alpha_{n,m}$ are constants and $c_{n,m}^{\text{pref}}$ is the preferred foot configuration. The preferred foot configuration is presented in Figure 8.5b. The value c_R represents the polynomial fitting to the terrain patch:

$$c_R = \sum_{x_1=L/2}^{L/2} \sum_{x_2=-W/2}^{W/2} |p(x_1, x_2) - \text{elev}(x_1, x_2)|, \quad (8.4)$$

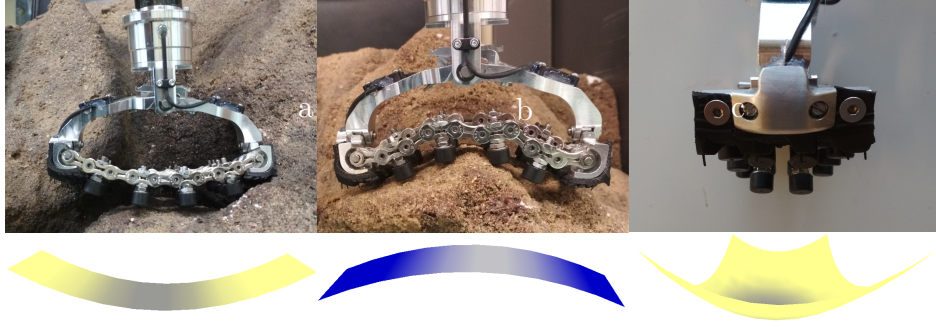


Figure 8.5: Example foot configurations and corresponding surfaces for $c_{1,2} = 2.4$ (a), $c_{1,2} = -2.4$ (b), $c_{1,2} = -2.4$ and $c_{2,2} = 12$ (c)

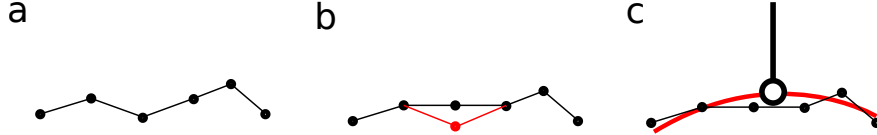


Figure 8.6: Illustration of the convex hull computation: initial set of points representing the terrain surface (a), computation of the convex hull (b), and regression result (c)

where W is the width of the foot L is the length of the foot, and $p(x_1, x_2) - elev(x_1, x_2)$ is the difference between the polynomial and the elevation of the map for the coordinates (x_1, x_2) .

The soft foot effectively compensates for small irregularities in the terrain. Thus, we close small holes in the elevation map before computing the cost c_R (8.4). To this end, we compute the convex hull on the 3D point cloud created by the local elevation map. We close holes smaller than two cells only. The procedure is illustrated in Figure 8.6.

8.2.4 Convolutional Neural Network

To estimate the cost of selecting the foothold, we initially engaged the solution proposed in Belter et al., 2019 (Chapter 7). Similarly, the elevation map (40×40 cells) is provided to the CNN input that returns the cost for each cell. As in the mentioned chapter, the regression problem was converted into a classification problem by foothold cost discretization. The discretization process was streamlined by automatically aligning histograms composed of cost ranges and the number of pixels that belong to each of them. The ranges were created by iteratively adding pixels to the current range in ascending order (in terms of their cost value) until its size exceeds 5% of the total data set. The left end of the current range is initially defined at point 0, the right end is defined as the value of the last added pixel. For the next interval, the left end is the right end of the previous range. The schema is duplicated until the entire collection is divided into classes. Similarly, as in Belter et al., 2019, additional classes weighting was performed. In this way, we obtained a balanced data set in terms of the number of occurrences of each class, which was used to teach the prediction model.

The Efficient Residual Factorized ConvNet (ERF) originally proposed in Romera et al., 2017 was used to learn the foothold cost from the analytical model. Despite the conversion of the estimation problem to the classification problem, the goal is to minimize the error in the output of the network. Due to the nature of the data, the situation in which the model confuses adjacent classes (adjacent classes represent a similar cost) is preferred over the situation in which distant classes are confused. For this purpose, we propose an additional weighting of the loss function, which multiplies the error by the absolute distance between the predicted and the ground truth class.

Because the identifier of the class is proportional to the cost, we can model the co-occurrence relations using known probabilistic mechanisms. Hence, we propose using Markov

Random Fields (MRF, Li and Wand, 2016) on top of the ERF model. The following formula defines the MRF operating scheme:

$$P(x) = \frac{1}{Z} \exp(-E(x))$$

$$E(x) = \sum_i \Phi(y_i) + \sum_{i,j} \Psi(y_i, y_j)$$

where P is a probability function of image segmentation, x is the input data (grayscale

```

 $q = \text{ERF}(\text{Image});$ 
 $\Phi = -\log(q);$ 
while  $q$  not converged do
     $\Psi = \text{conv}(q, \text{kernel});$ 
     $U = -(\Phi + \Psi) * s;$ 
     $q = \text{softmax}(U);$ 
end
return  $q$ 

```

Algorithm 2: Mean field approximation is used on top of the ERF model to estimate probability according to the Markov Random Field. The *Kernel* is a randomly initialized trainable tensor, with zero values in diagonal (non-trainable), whereas s is a trainable scaling factor. During training, only one iteration of the loop is performed to propagate the error.

image), Z is a normalization term and E refers to *the energy function*. The *energy function* is composed of Φ unary and Ψ pairwise costs. To adapt MRF into our model we used *mean field approximation* method in which we treat negative log of the output from the ERF model (probability distribution for each pixel) as Ψ . The MRF output is estimated using the algorithm 2.

8.3 Results

In the first experiment, we compute the cost function for the elevation map presented in Figure 8.7. We use (8.3) to determine the cost map. The defined function is anisotropic, so the cost value for each cell depends on the orientation of the foot. In Figure 8.7c and Figure 8.7d we show how the cost map computed using (8.3) changes when the orientation of the foot changes by $\pi/2$. We also compare the proposed method with the method from Belter et al., 2019. The results are shown in Figure 8.7b. When the method from Belter et al., 2019 is applied, the robot avoids small bumps and selects flat patches or small concavities. In contrast, the robot equipped with soft feet prefers small bumps, but also accepts flat terrain.

In order to justify the application of the proposed neural networks, we present qualitative results of the ablation study. We compare the following three neural network configurations:

- ERF with default segmentation loss function used in Belter et al., 2019,
- ERF with difference weighting,
- MRF on top of ERF with difference weighting.

As in Chapter 7, we note the metrics: accuracy and intersection over union (IoU). Both metrics were performed between the discretized ground-truth cost assignments and the neural network classification result. The metrics provide information about the accuracy of the

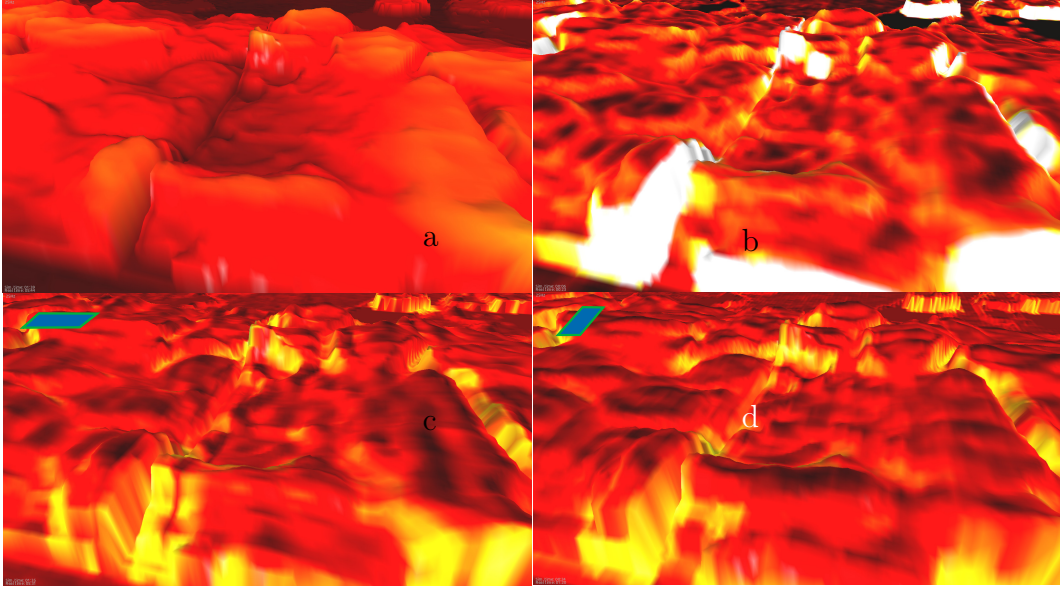


Figure 8.7: Comparison between two foothold selection methods: example terrain (a), foothold selection method for a ball-like foot (Belter et al., 2019) (b), and polynomial fitting method for example orientations of the foot (c,d). Dark red color represent good footholds, yellow weak footholds

	Accuracy [%]	IoU
ERF basic	88.85	58.32
ERF diff weighting	89.23	60.59
MRF over ERF	89.89	62.22

Table 8.1: Comparison between neural network architectures on the validation dataset.

predictions of each class separately and how different areas (in terms of class) are confused with each other. The results are presented in Table 8.1. MRF has significantly improved the results, with an increase in the IoU metric to 66.22.

Detailed results for the MRF and ERF models are presented in Table 8.2. The summary shows a clear improvement for the MRF model.

We verified the foothold selection algorithm on the various obstacle types in the Gazebo simulation environment which has the same control interface as the real ANYmal robot¹. The experimental sets of examples are presented in Figure 8.8. We checked the algorithm on stairs, rough terrains, bumps, and various geometric structures. The robot uses a locomotion planner proposed by Fankhauser et al., 2018. We replaced the module for foothold selection. The direction of the robot’s motion is given by the human operator. Then, the robot adapts the position of the feet and optimizes the pose of the robot’s body to climb obstacles. This adaptation strategy slightly modifies the direction of motion given by the operator. It should be noted that at the present stage the compliance of the feet is not taken into account in the simulation because of its mechanical complexity, which makes it difficult to simulate contacts for the numerous links each foot is made of.

Example elevation maps, estimated cost maps, and selected footholds are presented in Figure 8.9. The preferred footholds are represented on the cost maps by dark red cells. The selected foothold (red sphere) is always a compromise between the preferences given by the neural network and the position of the nominal foothold. Thus, the robot does not select the cell with the lowest value in the cost map.

¹short video from experiments is available at <https://youtu.be/p6CvUnqtqkQ>

	0	1	2	3	4		0	1	2	3	4
0	0.916	0.058	0.002	0.001	0.000		0.923	0.059	0.002	0.001	0.001
1	0.043	0.869	0.070	0.001	0.001		0.055	0.870	0.062	0.001	0.001
2	0.001	0.050	0.867	0.065	0.001		0.002	0.068	0.857	0.057	0.001
3	0.000	0.001	0.058	0.873	0.043		0.000	0.001	0.078	0.851	0.048
4	0.000	0.000	0.002	0.085	0.858		0.000	0.000	0.002	0.100	0.838
5	0.005	0.001	0.001	0.002	0.072		0.008	0.001	0.001	0.002	0.090
6	0.001	0.004	0.050	0.049	0.012		0.002	0.006	0.057	0.050	0.014
7	0.001	0.001	0.004	0.006	0.047		0.002	0.001	0.005	0.005	0.052
8	0.001	0.001	0.008	0.011	0.014		0.001	0.002	0.011	0.012	0.016
9	0.002	0.001	0.014	0.022	0.014		0.002	0.002	0.020	0.022	0.016
10	0.000	0.000	0.001	0.003	0.004		0.000	0.000	0.002	0.002	0.005
11	0.000	0.000	0.001	0.001	0.001		0.000	0.000	0.001	0.001	0.000
12	0.000	0.000	0.000	0.000	0.000		0.000	0.000	0.000	0.000	0.000
13	0.000	0.000	0.000	0.000	0.000		0.000	0.000	0.000	0.000	0.000
MRF						ERF					

Table 8.2: Slice of the confusion matrix for classes 0 to 4. The results show a comparison of the ERF model and the ERF with MRF module built on it. Bold numbers indicate a better result.

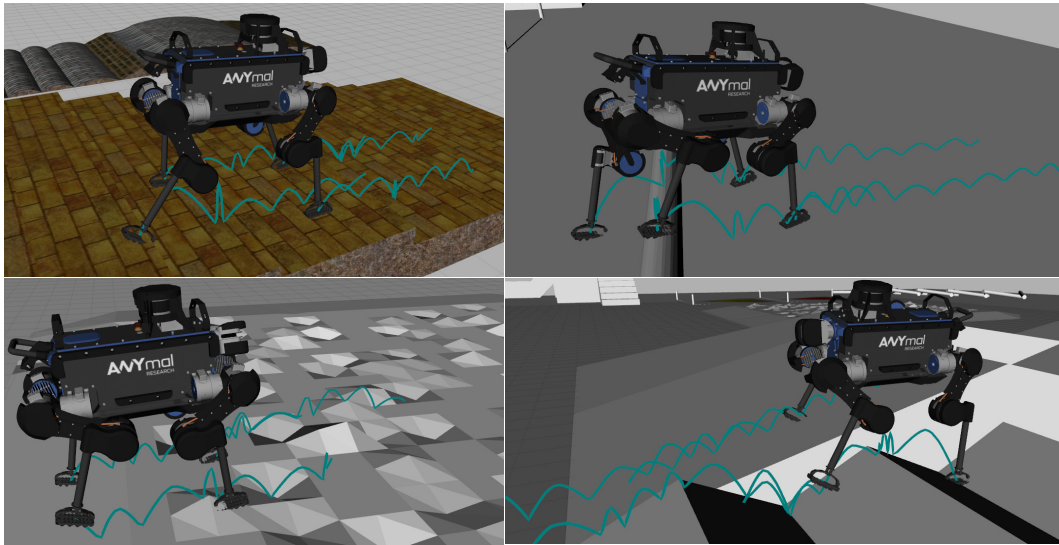


Figure 8.8: Obstacles and feet trajectories during experiments with the CNN-based foothold selection for the adaptive soft foot.

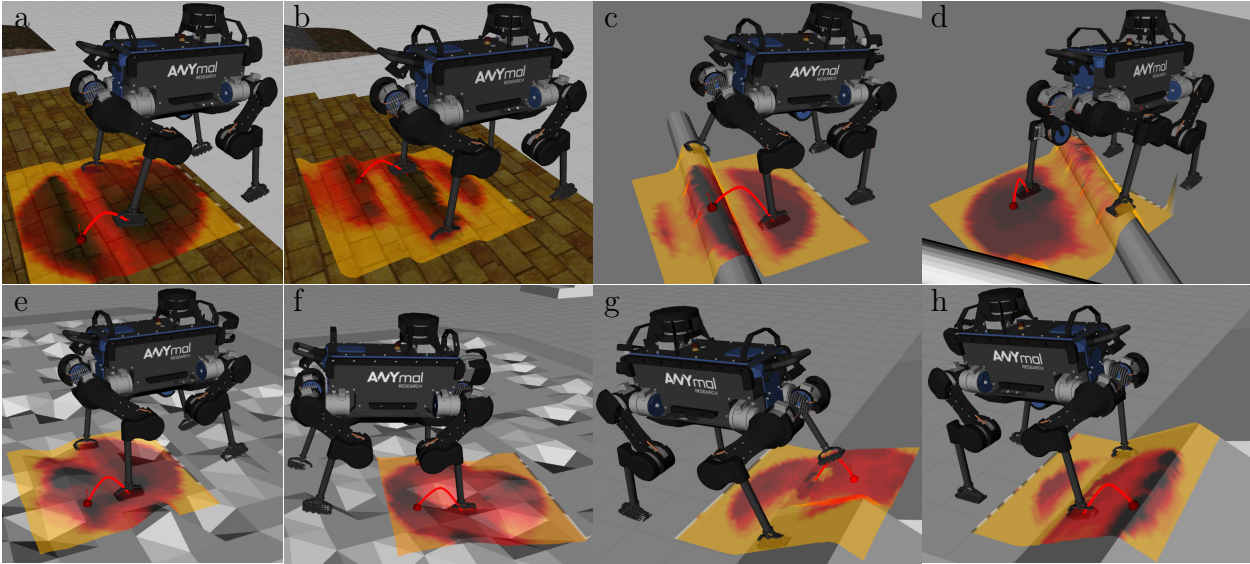


Figure 8.9: Example footholds during walking on irregular terrain: hot color map patch represents evaluated region below hip joint (dark red – good foothold, yellow – weak foothold, red sphere – selected foothold)

The first two examples show the behavior of the robot on the stairs (Figure 8.9a and Figure 8.9b). The robot avoids the bases of the step due to collisions. In contrast to Chapter 7, when we considered a ball-like foot (Belter et al., 2019), the robot prefers positions that are closer to the edges. In this case, the robot remains stable even if the adaptive foot is placed on the edge of the step. Similar behavior can be observed when the robot deals with a single step. The results are presented in Figure 8.9c and Figure 8.9d. In contrast to the robot equipped with a ball-like foot, the current planner has a larger set of foot positions on the same terrain that are acceptable and stable for the robot.

The example cost maps obtained during the experiment on rough terrain presented in Figures 8.9e and 8.9f show that the robot prefers small bumps and avoids concavities. This behavior is expected due to the parameters of the preferred foot configuration defined in (8.3). The soft foot better adapts to small bumps which provide better support for the robot. In the examples presented in Figure 8.9g the robot selects the foothold on a slope even though the cost of the foothold in this region is high. This happens because all footholds in this area provide weak support. As a result, the robot selects the foothold that is close to the nominal position of the foot. In the last example in Figure 8.9h, the robot selects the preferred foothold on the edge because it is in the range of the considered leg. In all examples presented in Figure 8.9 the low-cost footholds are within the circular region. This shape corresponds to the robot workspace that is used to define motion constraints while training the neural network.

In addition, we compare the operating times of the previous model (ERF) and the current one (MRF). Two tests were carried out, for the GPU and CPU. The GPU used is NVIDIA GTX 1660, while the CPU is Intel Core i7-9750H CPU with 2.60GHz processor clock. For the purposes of the test, in both cases, the models received batch data with a size of 32. The ERF model achieved a result of 33.82 ms on the CPU and 5.76 ms on the GPU, while the MRF reached 34.43 ms and 5.98 ms, respectively. The tests were repeated 100 times. The difference between the models is very small and allows for real-time operation in both cases.

8.3.1 Conclusions and Future Work

In this chapter, we propose a novel foothold selection method for legged robots equipped with passive compliant feet. The proposed method uses the neural network that emulates the analytical model that considers a set of physical constraints to produce the cost map, given only the elevation map. The neural network implicitly learns to analyze the local shape of the terrain along with the parameters and the kinematic model of the foot and predict the target cost map of potential footholds taking into account collisions between the leg and the terrain.

To avoid time-consuming data collection, we propose the cost function, which computes how the foot can adapt to the given terrain patch. We also take into account the distance from the desired configuration that guarantees the best grip. The proposed cost function allows one to generate training data for the neural network.

Finally, the obtained neural network is used on-line to select footholds for the quadruped robot ANYmal on various obstacles. With the proposed method, the robot avoids risky configurations and selects footholds that provide stable support. In the future, we are going to define the CNN-based method that selects the footholds and the configuration of the robot in a single step. Moreover, we will also study and validate through rigorous experiments the impact of our approach on the performance of the robot on rough terrain.

8.4 Implications

The results of this study provide an extension to the validation of the hypothesis from Section 6.1.3—that a neural Apprentice can effectively approximate the decision boundaries of a complex Oracle. While Chapter 7 established this capability for rigid geometric constraints, this chapter demonstrates the framework’s versatility in handling the non-linear dynamics of soft-body interactions. By successfully encoding the behavior of a mechanically adaptive foot, the neural network proved that it is not limited to static spatial reasoning but can also implicitly model complex physical phenomena, such as material deformation and compliance, solely from data.

Crucially, this chapter illustrates the scalability of the paradigm in terms of computational efficiency. In analytical approaches, introducing a compliant foot exponentially increases the cost of evaluating a foothold, often requiring expensive contact modeling or finite element analysis. However, in the proposed framework, the computational cost of the Apprentice (the CNN) remains constant regardless of the physical complexity it approximates. This effectively compresses the heavy computational burden of soft-body simulation into the fixed and rapid inference time of the neural network. Consequently, this confirms that Acting policy can leverage sophisticated morphological adaptations—which are typically computationally prohibitive to model online—without sacrificing the real-time performance required for dynamic locomotion.

Chapter 9

Conclusions

9.1 Summary of contributions

The primary contribution of this part is the successful demonstration of a training methodology that addresses the data and speed bottlenecks inherent in traditional robotics. This approach, framed as “the slow Oracle and the fast Apprentice,” allowed for the creation of a robust real-time action policy by training a neural network (the Apprentice) on synthetic data generated by a slow but accurate analytical model (the Oracle). We successfully demonstrated that a neural network can learn from a high-fidelity analytical model, effectively overcoming the limitations of real-world data collection and computational complexity.

We developed a computationally efficient method for the selection of a foothold in real-time using a CNN. The approach presented in Chapter 7 showed that a neural network can implicitly learn and evaluate complex physical and kinematic constraints in a single, fast inference step, a significant improvement over traditional sequential methods. The versatility of this approach was further demonstrated in Chapter 8 by extending the foothold selection policy to a more complex robot platform equipped with a soft, mechanically adaptive foot. The evaluation of the neural network demonstrated that a learned policy can adapt to advanced hardware and complex physical interactions.

Ultimately, this part demonstrates that an effective physical policy can be learned from synthetic data, which is a crucial capability for building intelligent, autonomous systems. The successful integration of these principles with knowledge distillation provides a powerful perspective for developing advanced robotic skills, paving the way for more sophisticated behaviors in unstructured and unpredictable environments.

9.2 Implications and broader context

This part illustrates the integration of robotic Acting into broader control architectures rather than treating execution as an isolated physical skill. Learning real-time behaviors—such as foothold selection—provides the baseline capacity required for complex locomotion. This physical competence establishes the operational foundation for high-level environmental reasoning.

Looking ahead, this research opens several avenues for future work. One such avenue involves developing a more integrated, CNN-based method that optimizes both foothold position and the robot’s overall body configuration in a single, unified step. Additionally, future research may involve conducting rigorous, real-world experiments to validate the long-term impact of these advanced action policies on a robot’s performance over challenging, complex terrain.

Acting aligns with the paradigm of neuro-symbolic AI (Garcez & Lamb, 2023), which advocates systems in which the neural components responsible for learning are seamlessly integrated with the symbolic components responsible for reasoning. Within this framework, Acting module operates as a specialized neural component. It uses deep learning to generate an instantaneous abstract representation of Acting policy—for example, a judgment of “steppability” that encodes the robot’s goals and kinematic constraints. That representations

serve as an indicator for something inherently artificial and abstract: a possible decision. This outcome is subsequently utilized by the reasoning process, which may range from a simple heuristic (e.g., “take an action with the highest score”) to a complex high-level deduction mechanism.

This part of the thesis serves as a point of reference for the further extension of Acting component in Part III. By proving that complex physical actions can be derived from learned cost maps rather than hard-coded heuristics, we establish the blueprint for *symbolic acting*. This prepares the ground for the next stage of the research, where the output of the system shifts from selecting discrete geometric points (footholds) to generating discrete logical tokens (code).

The validation of the hypothesis stated in Section 6.1.3 (**H2**) leads us to a more profound question: “What are the limits of abstraction in modeling decision processes?” If we can successfully model complex physical interactions like foothold selection using synthetic data, does this principle extend to cognitive tasks? Can we define *any* problem—regardless of its complexity—at a level of abstraction that allows us to generate a solution synthetically? This question serves as the central link to Part III, where we explore the generation of abstract programs to solve general reasoning tasks.

Part III

Problem Solving: grounding abstract reasoning in logical execution

Chapter 10

Preliminaries

This part synthesizes the findings from two publications developed during this doctoral studies, which collectively narrate the evolution of the proposed Problem Solving framework. These works demonstrate how the architectural principles of the decision-making loop—originally established in the contexts of embodied Sensing and Acting—can be formalized and subsequently elevated to the cognitive domain.

Chapter 11, based on *Ain't Nobody Got Time for Coding: Structure-Aware Program Synthesis from Natural Language* (J. Bednarek, Piaskowski, & Krawiec, 2018), serves as the initial formalization of our general approach. In this work, we introduce a baseline architecture composed of “simple” Sensing and “simple” Acting components, embedded in the non-embodied environment, designed to map natural language (NL) specifications to executable source code. Although this study has not yet fully addressed the complexities of abstract reasoning, it establishes the essential structural precedent: it demonstrates that the components of a Problem Solving framework can be constructed to operate on symbolic logic rather than physical forces. By defining Sensing as the interpretation of a problem description and Acting as the synthesis of a solution program, this chapter provides the architectural skeleton upon which the subsequent, more advanced framework is built.

Chapter 12, based on *Solving Abstract Reasoning Problems with Neurosymbolic Program Synthesis and Task Generation* (J. Bednarek & Krawiec, 2026), expands this methodology to address the rigorous demands of Abstract Reasoning Corpus (ARC). This work provides the empirical verification of the central hypothesis posited in Section 10.1.3. By transforming the components into their advanced neuro-symbolic versions—upgrading perception to encode visual grid-based demonstrations and latent representations of problem’s symbols; and acting to the generative mechanism operating over a Domain Specific Language—we successfully implement the proposed structural isomorphism in a challenging cognitive domain. Crucially, this work introduces the concept of task generation, a mechanism that allows the system to actively build and explore its own problem space by learning from execution traces. This transition from a static supervised mapping to a dynamic, self-improving loop marks the complete realization of the neuro-symbolic Problem Solving framework.

10.1 Introduction to Problem Solving

The preceding parts of this thesis, Part I and Part II, established a foundational framework for an autonomous agent operating within a physical environment. These embodied applications demonstrated clear and pragmatic objectives: Part I focused on the use of proprioceptive Sensing to localize in vision-denied conditions, while Part II addressed the challenge of traversing demanding terrains with safe and fluent locomotion. In both instances, the agent’s behavior was governed by a decision-making loop, binding two fundamental components: the perception of the state of the world and the execution of actions to alter it. However, implicit in these operations is a higher-order structure; the decision-making loop is not merely a mechanism for movement but an engine aiming to achieve pre-established objectives—either to recover to safe spot with enhanced localization or reach a target with a sense of reliability. By defining Problem Solving as a framework that integrates Sensing and

Acting to resolve a specific discrepancy between a current state and a desired goal, we can elevate these architectural principles from the physical domain to a purely cognitive level.

10.1.1 The nature of abstract problems

When detached from the constraints of physical embodiment, the definition of a problem shifts from a navigational or manipulation task to a purely cognitive process of information transformation. In the classical context of artificial intelligence (AI), problem solving is rigorously defined not by physical obstacles but by the existence of a gap between an *initial state* and a desired *goal state*, with no immediately apparent operator to bridge them. As formalized in the foundational work on *problem space* (Newell, Simon, et al., 1972), this cognitive framework outlines the topology of all possible states and the transitions between them. Solving a problem at this level is conceptualized as the process of discovering a valid trajectory through this topology. Consequently, the agent must explore a distinct *solution space*—defined as the set of all possible candidate sequences of known operators or actions—to successfully transform the initial state into the goal state.

Building upon this foundation, *abstract Problem Solving* elevates this process to generative (synthetic) domains. Here, rather than navigating a fully observable space of predefined state transitions, the agent is required to deduce and synthesize latent rules. In the context of modern neuro-symbolic AI and benchmarks such as ARC (Chollet, 2019), this definition evolves further. Although solving ARC does not necessarily require a specific computational methodology, a prominent and highly effective approach frames the challenge as a task of *program synthesis*. In this paradigm, the “solution” is an algorithmic abstraction capable of generalizing from a few examples to a latent rule. Here, the physical feedback loop is replaced by a logical validation loop, yet the core architecture—perceiving a state (Sensing; a tool for observing a problem space) and applying a transformation (Acting; a transition between states)—remains intact. The conceptualization of the problem space in the context of Problem Solving framework is presented in Figure 10.1.

This definition of Problem Solving as a traverse in a problem space highlights a critical distinction between the physical and abstract domains: the heterogeneity of representation. Although physical tasks are grounded in the consistent laws of physics—sharing a universal “language” of forces, velocities, and geometry—abstract problems exhibit extreme diversity in their encoding. A problem instance may be expressed as a set of rigid logical constraints, an ambiguous natural language specification, or a visual grid-based puzzle such as those found in ARC benchmark. This diversity introduces a structural challenge: the absence of a universal “common language” for the definition of the problem. Each domain typically requires a bespoke formalism—a custom language—to adequately capture its specific constraints and objectives. Consequently, this fragmentation extends to the solution space; since there is no unified syntax for describing every possible abstract task, there exists no corresponding universal language for their solutions, forcing solvers to adapt to the idiosyncratic syntax and semantics of each unique problem domain.

10.1.2 Adaptation of symbolism to Problem Solving

In the context of neuro-symbolic AI, where both Sensing and Acting can be built with neural mechanisms, a fundamental challenge emerges in the construction of a problem space. Unlike traditional symbolic approaches where the environment is usually a priori defined, here the Sensing component learns from scratch to organize and represent the problem states within a structured latent manifold. Simultaneously, the Acting component acquires strategies necessary to navigate this evolving space toward a target configuration.

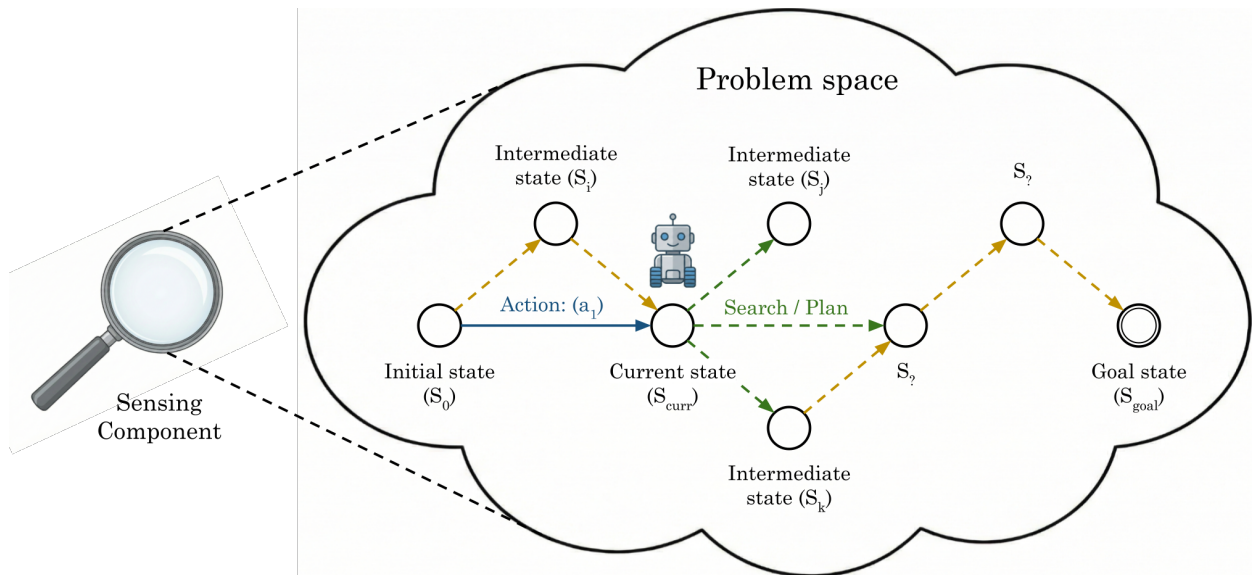


Figure 10.1: Conceptualization of the problem space within a neuro-symbolic framework. The Sensing component (depicted as a magnifier) is not a passive observer but a learned mechanism that organizes raw information into a structured problem space through training, and allows the agent to observe the space (states). The agent traverses this space to reach the goal state. The blue arrow represents the action that the agent has successfully executed to reach the current state (S_{curr}). From this state, the agent engages in a search and planning process, evaluating potential future operators (green arrows) to bridge the gap to the goal. The other (yellow) arrows represent other possible actions that are not currently considered by the agent.

Notably, these capabilities—describing the space and operating within it—do not develop in isolation; rather, they are co-dependent processes of discovery that advance in tandem. This simultaneous optimization allows the components to leverage each other, ensuring that the learned representation is inherently actionable and that the synthesized actions are grounded in a coherent perception of the problem.

To operationalize this dynamic construction of the problem space, we align our established decision-making framework with the principles of neuro-symbolic AI—a convergence of connectionist statistical learning and symbolic discrete reasoning. Although the preceding parts of this thesis demonstrated the efficacy of learning-based control in physical environments, the move to abstract Problem Solving necessitates a framework that can not only perceive patterns, but also reason about them explicitly. Consequently, we elevate the “simple” components of physical interaction into their “symbolic” counterparts, creating a composite architecture capable of the simultaneous optimization described above.

The core adaptation begins with the Sensing component, which is responsible for the construction of the latent space. In the physical domain, Sensing involved mapping noisy sensor data (haptic feedback) to a state estimate. In the neuro-symbolic domain, we define *symbolic sensing* as the process of mapping a raw problem specification—whether it be a NL description or a visual grid—into a structured, yet symbolic, latent representation. The result of this module is not only the bound neural and symbolic representation, but also the provision of a problem space for Acting module.

Conversely, the Acting component is tasked to handle navigation within this learned space. We define *symbolic acting* as the generation of executable tokens (programs) within a formal language and representation from a symbolic sensing. Here, the role of the physical “actuator” is assumed by the program executor. While the Acting component synthesizes the sequence of instructions (the plan), it is the executor that applies these symbolic operations

to the current context. Just as a selected foothold allows a robot to leverage physical laws to traverse a terrain, an executed program token leverages logical rules to transform the state of the problem, bridging the gap between the initial configuration and the solution.

10.1.3 The objective: automatic construction of the problem space

The dichotomy between current computational paradigms and human-like reasoning motivates the essential objective of this work. On the one hand, deep neural networks excel in domains characterized by noisy high-dimensional data—such as raw sensory inputs encountered in Sensing and Acting parts—by extracting statistical correlations that form an implicit, often opaque, latent structure. On the other hand, human reasoning operates on a high level of abstraction, utilizing concise discrete languages (e.g., mathematics, code) to define problem spaces with infinite compositional variety. The objective of this framework is to merge these two approaches: to leverage the learnable, continuous nature of neural networks to construct the problem space while imposing the precise, structural constraints provided by symbolic descriptions. By doing so, we aim to create a system that does not merely memorize solutions to fixed tasks but automatically builds an internal representation that is reflective of the abstract structure of the problem.

Consequently, we formalize the central hypothesis of this dissertation. Drawing from the architectural validations in the preceding parts, we propose that the solution to this objective lies in preserving the functional topology of the robotic framework while transposing its components to the abstract domain.

H3. *We hypothesize that the challenge of automatic fitting to abstract problems can be resolved by establishing a structural isomorphism between the embodied decision-making loop and the cognitive reasoning process. By elevating mechanical Sensing to symbolic perception and physical Acting to program synthesis, we postulate that a unified neuro-symbolic architecture can utilize program execution traces as a training signal analogous to physical feedback, enabling the joint optimization of problem perception and solution strategy.*

This pursuit of “automatic construction” directly addresses the fundamental question raised in Section 9.2: “*What are the limits of abstraction in modeling decision processes?*” In Part II we demonstrated that a robot could master complex locomotion by training on abstract geometric representations rather than raw physical experience alone. Here, we push this hypothesis to its logical conclusion by exploring the generation of synthetic data as a mechanism for cognitive reasoning. We propose that if a problem can be described formally, we can utilize that symbolic description to generate diverse curriculum of synthetic tasks. This effectively allows the system to sample the theoretical problem space, aligning the decision boundaries derived from the Sensing component with the logical operators of the Acting component. Through this exposure to generated examples, the system bridges the gap between the abstract definition of a problem and its practical, executable solution.

10.2 Key challenges in abstract Problem Solving

Transitioning from physical to abstract Problem Solving introduces unique difficulties. Unlike robotic actuation, where challenges are often physical or practical (e.g. friction, motor limits, high cost of experiments), the challenges in abstract reasoning are primarily representational and algorithmic. We identify three key hurdles that an abstract Problem Solving framework must address.

The challenge of formal definition. The first and perhaps most formidable challenge is the formalization of intuitive tasks. Problems that appear self-evident to human observers—such as visual puzzles in ARC—are notoriously difficult to describe in strict formal language. A human glancing at a grid-based puzzle immediately perceives high-level concepts such as “symmetry”, “gravity”, or “repetition” without needing them explicitly defined. However, for a computational system, every facet of the problem must be grounded in a specific syntax. The “definition gap” lies in translating these rich and intuitive concepts into a constrained formal logic without losing the essence of the task. If the formal description is too loose, the problem space becomes intractable; if it is too rigid, it may exclude the intended solution entirely.

Reachability and completeness. Even if a problem is well-defined, there is no guarantee that the tools provided to the agent (e.g., a Domain Specific Language or a set of neural operators) are sufficient to construct the solution. This is a question of completeness: does the solution space defined by the agent’s atomic actions actually contain the target program? In physical robotics, a target location might be unreachable due to kinematic constraints; in abstract reasoning, a solution is unreachable if the agent’s “mental vocabulary” lacks the expressivity to describe the necessary transformation. Designing a solver therefore requires a delicate balance: the language must be expressive enough to cover the problem space (ensuring reachability), but constrained enough to make the search efficient.

The illusion of simplicity. Finally, abstract problems suffer from an unknown level of difficulty. In physical domains, energy expenditure or distance often serves as a proxy for difficulty. In abstract domains, such metrics are misleading. A problem that appears visually simple—containing few objects or a small grid size—may require a profoundly complex chain of reasoning to solve (e.g., recursive logic or conditional branching). Conversely, a visually chaotic input might be solvable by a single trivial operation. This disconnect between “perceived complexity” and “algorithmic complexity” (often related to *Kolmogorov complexity*) makes it difficult to estimate the computational resources required for a given task, complicating the design of curriculum learning or heuristic guidance systems.

Chapter 11

Ain't nobody got time for coding: structure-aware program synthesis from natural language

11.1 Motivation

Program synthesis, meant as automatic or semi-automatic generation of programs from specifications, can be posed in several ways. It is most common to assume that specification has the form of input-output pairs (tests), in which case synthesis resembles learning from examples. This approach is limited by its inductive nature: even if a program passes all provided tests, little can be said about generalization for other inputs. This is one of the rationales for synthesis from formal specifications, which are typically expressed as a pair of logical clauses (a *contract*) comprising a *precondition* and a *postcondition*. Programs so synthesized are correct by construction, but the task is NP-hard, so producing programs longer than a few dozen of instructions becomes computationally challenging. Synthesis from formal specifications is also difficult for programmers because writing a correct and complete specification of a nontrivial program can be actually *harder* than its implementation.

From a practical perspective, the most intuitive and convenient way of specifying programs is NL. This way of formulating synthesis tasks has rarely been studied in the past, but recent progress in natural language processing (NLP) and deep neural networks made it more realistic, as confirmed by a few studies we review in Section 11.1.1. Here, we propose Structure-Aware Program Synthesis (SAPS), an end-to-end approach to program synthesis from NL. SAPS receives a short NL description of a requested functionality and produces in response an executable snippet of code. We combine generic word and sequence embeddings with doubly-recurrent decoders trained on abstract syntax tree (abstract syntax trees (ASTs)), with the following contributions: (i) folding the entire NL specification into a fixed-dimensional latent space, which is then mapped by the decoder onto the AST of a code snippet; (ii) modular architecture that facilitates usage of pretrained components; (iii) new signal propagation strategies in the decoder; and (iv) a ‘soft attention’ mechanism over the latent space.

11.1.1 Related work

Program synthesis from examples Balog et al., 2016 applied a neural network to guide a search algorithm and predict the most probable code tokens, greatly reducing the number of solutions that needed to be evaluated during the search. Krawiec et al., 2017 used genetic programming to synthesize programs from formal specifications and counterexamples resulting from the iterative formal verification process. Devlin et al., 2017 create programs using noisy input-output pairs.

Differentiable compilers Another approach is to, rather than synthesizing human readable code, embed the whole process implicitly in the neural substrate. These methods often utilize neural networks and could be seen as differentiable compilers. Graves et al., 2014 implement

a Turing machine using a neural network, endowing it with differentiable memory. A similar approach was proposed by Sukhbaatar et al., 2015, where memory was also used.

Synthesis from natural language Several approaches utilizing NL descriptions for code synthesis were introduced. Rabinovich et al., 2017 and Yin and Neubig, 2017 generate AST rather than plain source code, using specialized recursive neural decoders for the generation of ASTs. Ling et al., 2016 use a set of neural modules to generate source code from multimodal inputs (text and quantitative features) in a trading card game. Another related topic is *semantic parsing* (see, e.g. Rabinovich et al., 2017), which puts particular emphasis on synthesis from NL and other informal specifications.

Last but not least, the main reference approach for this work is the neuro-guided approach proposed by Polosukhin and Skidanov, 2018, where a neural network prioritizes a Tree-Beam search algorithm. A sequential encoder folds the NL description, followed by a tree decoder that generates a probability distribution over the nodes in the AST. For each node, node labels are evaluated on a set of input-output tests. The proceeds recursively for consecutive nodes of the AST tree. The authors used an attention mechanism (Bahdanau et al., 2014) on the entire input sequence – contrary to our approach, where attention concerns only the latent vector.

11.2 Details

Once trained, SAPS accepts a short NL specification as input and produces in response an AST. The method pipeline comprises three main components (Figure 11.1): (i) a word embedding for preprocessing of the specification, (ii) a mapping of the embedded word sequence tokens to a fixed-size latent representation $h^{(latent)}$, and (iii) a decoder that unfolds $h^{(latent)}$ into an AST.

Word embedding The NL specifications proposed in Polosukhin and Skidanov, 2018, which we use in the experimental part, include only the standard English vocabulary. Given that, we rely on Common Crawl, the smallest pretrained GloVe embedding that covers all terms occurring in the considered dataset. It has been trained on generic NL database comprising 42B tokens by (Pennington et al., 2014), has a vocabulary size of 1.9M tokens, and embeds the words in a 300-dimensional space. Given an NL query phrase of n tokens, this module produces thus a sequence of n 300-dimensional vectors $(q_1, \dots, q_n)$. We pad each sequence in a mini-batch with a special out-of-vocabulary value.

Sequence embedding To map the sequences of word embeddings of the NL specification (300-dimensional vectors $(q_1, \dots, q_n)$) to the latent space $h^{(latent)}$, we employ a multilayer bidirectional long short-term memory (LSTM) (Graves et al., 2005; Schuster & Paliwal, 1997). Because some variants of SAPS engage pretraining, the concatenated outputs of the both vertical and horizontal LSTMs is passed through a tanh layer in order to match the $h^{(latent)}$ dimensionality and the $(-1, 1)$ range of values produced by the tree encoder. There are four forward LSTM cells stacked upon each other, and analogously a stack of four backward cells. The final state of the topmost forward cell and the final state of the topmost backward cell are concatenated to form the final output.

Tree decoder The decoder can be trained from scratch or pretrained within an autoencoder (dashed part of Figure 11.1). In the latter variant, we rely on the *tree2tree* autoencoder architecture based on *TreeLSTM* (Tai et al., 2015) for encoding and doubly recurrent neural

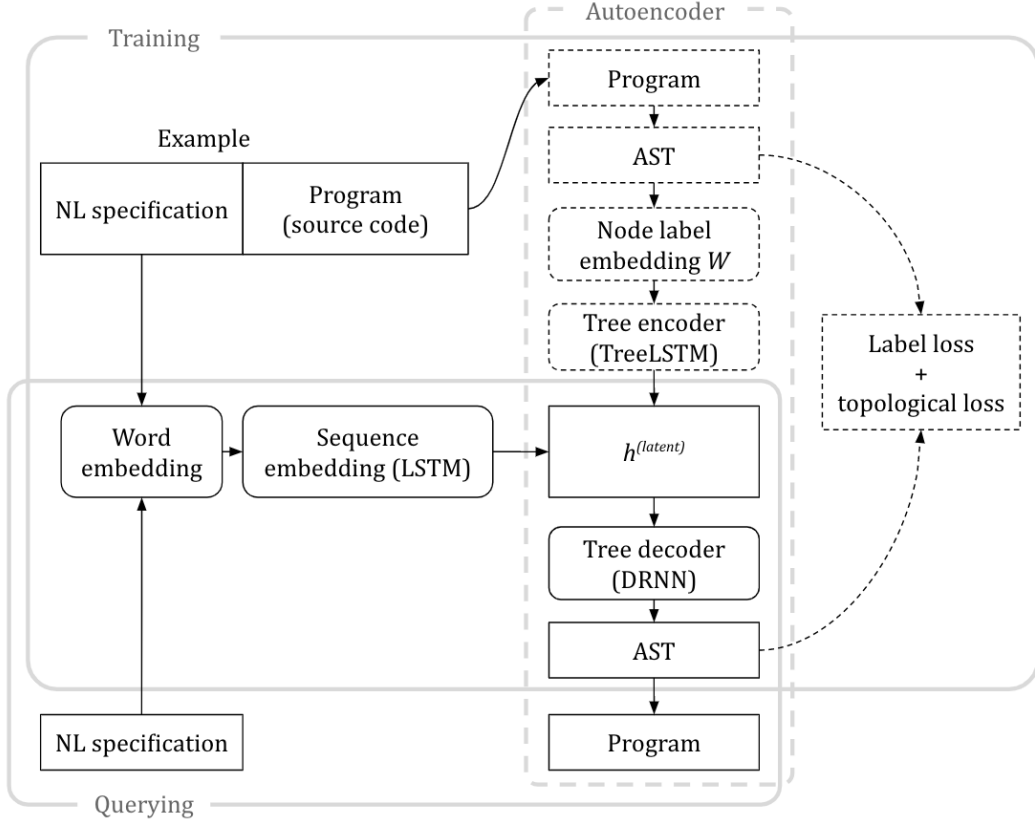


Figure 11.1: Overall architecture of SAPS. Rectangles mark data entities, rounded boxes are operations.

network (DRNN), Alvarez-Melis and Jaakkola, 2017 for decoding. It diverges from the latter work in (i) using only the latent vector for passing the information between encoder and decoder, and (ii) in new ways in which decoder’s state is being maintained when generating trees.

Encoder A single training example is $(x, x) \in X_{AST} \times X_{AST}$, where x is an AST of a snippet of code. Given (x, x) , the encoder starts by encoding the label of every node x_j in x using one-hot encoding $v_j = V(x_j)$; for the experiments presented in Section 11.3, there are 72 node labels, so $|v_j| = 72$. Then, following Mikolov et al., 2013, we map v_j s, for each node in x independently, to a lower-dimensional space, using a learnable embedding matrix W : $y_j = Wv_j$. The result is a tree y of node label embeddings y_j , isomorphic to x . The encoder network folds the tree bottom-up, starting from the tree leaves, aggregating the hidden states h_j , and ending up at the root node. As ASTs can have arbitrary arity, we apply the recurrent TreeLSTM (Tai et al., 2015).

The hidden states and cell states are initialized with zeroes. For each node, we first compute the sum of the hidden states of its children $C(j)$, $\tilde{h}_j = \sum_{k \in C(j)} h_k$. For leaf nodes, $\tilde{h}_j = 0$. Then we compute the values of input, forget and output gates:

$$\begin{aligned} i_j &= \sigma(W_i y_j + U_i \tilde{h}_j + b_i) & f_{jk} &= \sigma(W_f y_j + U_f h_k + b_f) \\ o_j &= \sigma(W_o y_j + U_o \tilde{h}_j + b_o) & u_j &= \tanh(W_u y_j + U_u \tilde{h}_j + b_u) \end{aligned} \quad (11.1)$$

where σ is the sigmoid function, and W_i, W_f, W_o and W_u and b_i, b_f, b_o and b_u are, respectively, learnable weights and biases. Note that i_j, o_j and u_j depend on the aggregated children’s states ($\tilde{h}_j$), whereas f_{jk} is computed child-wise, yielding a separate output f_{jk} of forgetting gate for each child of the current node. We then update the state c_j of the memory cell of

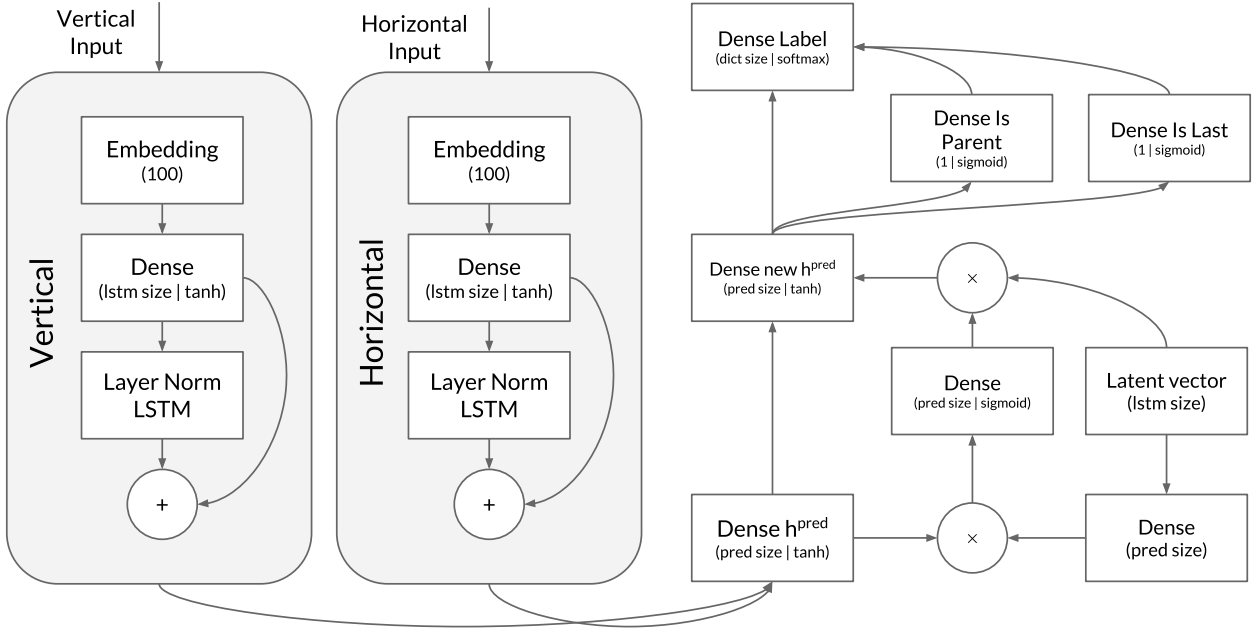


Figure 11.2: The modified DRNN cell consists of two LSTM blocks, each comprising embedding, input projection layer, layer normalization LSTM cell, and residual skip connection. It uses latent vector to enrich state with information that might have been lost during hierarchical processing.

the current node:

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k \quad (11.2)$$

where $\odot$ stands for elementwise multiplication, and where we assume that the initial states of c_j s of non-existent children of leaves are zero. Next, we compute the hidden state of the current node, $h_j = o_j \odot \tanh c_j$, which is then recursively aggregated by the above procedure. Once this process reaches the root node, the obtained state h_j becomes the latent vector, i.e. $h^{(latent)}(x)$. Additionally, we have applied *layer normalization* (Ba et al., 2016) in the same manner as to traditional LSTM.

Decoder Our decoder is a variant of the doubly recurrent neural network (DRNN, Alvarez-Melis and Jaakkola, 2017), comprising two LSTM cells that separately capture the *vertical* and *horizontal* traversal of the resulting AST tree (Figure 11.2). The state of the former (h_i^a) is passed vertically downwards from parents to children, and of the latter (h_i^f) horizontally left-to-right to consecutive siblings. The state of the horizontal LSTM is reset before processing the children of each node, but preliminary experiments with Python ASTs have shown that discarding this resetting step, i.e. allowing the state to propagate between children of different parent nodes, may bring significant improvements. We hypothesize that ASTs capture only program *syntax*, while pieces of code can be *semantically* related even if they do not reside in a common part of an AST. By not resetting the state of the horizontal LSTM, we provide it with more context that would be otherwise unavailable.

This observation inclined us to consider three variants of DRNN, denoted in the following as: (i) H , in which only the state of the horizontal LSTM is propagated (alongside the entire depth level of a tree), (ii) V , in which only the state of the vertical LSTM is propagated, (iii) VH , in which states of both LSTMs are propagated. These variants vary only in the above aspect of propagation (or, conversely, state resetting). We start by initializing both states of the vertical and the horizontal LSTM cells h_i^a, h_i^f with $h^{(latent)}(x)$. Alvarez-Melis and Jaakkola, 2017 suggest to initialize only vertical hidden state with latent vector, however,

our modification concerning the transfer of horizontal states in breadth-first order implies the same initialization approach also to the horizontal state. The DRNN merges both states of LSTMs in a *prediction state*:

$$h_i^{(pred)} = \tanh(U^f h_i^f + U^a h_i^a) \quad (11.3)$$

where U^f and U^a are learnable. $h_i^{(pred)}$ is then mapped to the probability distribution of node labels

$$l_i = \text{softmax}(W h_i^{(pred)} + \alpha_i v^a + \varphi_i v^f) \quad (11.4)$$

where $|l_i| = |v_i|$ (see the description of the encoder), and v^a and v^f are learnable parameters applied to *is-parent* flag (α_i) and *is-last-child* flag (φ_i). In training, these are used for *teacher forcing*, i.e. replaced with ground truth data according to the shape of the input tree x ; in inference mode, the flags are calculated as (u^a and u^f are learnable parameters):

$$\alpha_i^{(pred)} = \sigma(u^a h_i^{(pred)}) \quad \varphi_i^{(pred)} = \sigma(u^f h_i^{(pred)}) \quad (11.5)$$

In training, we minimize the sum of *label loss* and *topological loss*:

$$L(x) = \sum_i L^{xent}(l_i, V(x_i)) + L^{bxent}(\alpha_i^{(pred)}, \alpha_i) + L^{bxent}(\varphi_i^{(pred)}, \varphi_i) \quad (11.6)$$

where $L^{xent}(l_i, V(x_i))$ is the categorical cross-entropy loss between the ground truth label $V(x_i)$ and the label l_i produced by the DRNN, and L^{bxent} is the binary cross-entropy, applied separately to the *is-parent* flag and *is-last-child* flag. *Teacher forcing* makes the result tree structure always equal to the target one allowing us to calculate the loss function for each node.

Soft attention mechanism Because successive iterations of the DRNN may arbitrarily modify its prediction state $h_i^{(pred)}$, the information from the encoder may fade away with time, making it harder to reproduce the input tree x . A range of works, most relevantly X. Chen

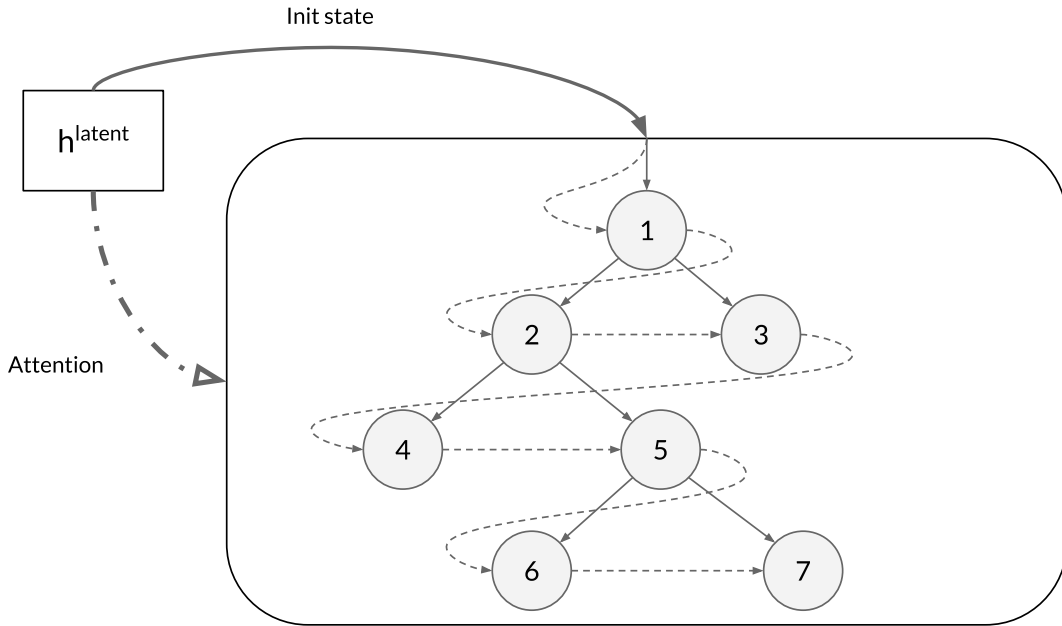


Figure 11.3: Propagation of the hidden states and the latent vector during decoding. The dashed line illustrates the traversal of the horizontal state, the solid line propagation of the vertical state. The latent vector is used to initialize both states, but has also impact on subsequent decoding steps via an attention mechanism.

```

program      ::= symbol
symbol       ::= constant | argument | function_call | function | lambda
constant     ::= number | string | True | False
function_call ::= (function_name arguments)
function     ::= function_name
arguments    ::= symbol | arguments , symbol
function_name ::= Reduce | Filter | Hap | Head | + | - | ...
lambda      ::= lambda function_call

```

Figure 11.4: The grammar of the AlgoLisp DSL from (Polosukhin & Skidanov, 2018).

et al., 2018, have shown that this loss of information can be supplemented with an *attention mechanism*. However, the specific attention architecture proposed there assumes that the window of attention scans the corresponding nodes of the input tree x , which implies that x has to be available also during decoding. We cannot assume this in our setting, where the decoder is intended to be ultimately detached from the encoder and receive only the latent vector h^{latent} as input (Figure 11.1). Therefore, we come up with an alternative attention mechanism that relies only on h^{latent} . We define a *soft attention window* as

$$A = \sigma((h^{(latent)} U^{(A)} \odot h_i^{(pred)}) W^{(A)}) \quad (11.7)$$

and then use it to update the prediction state (compare to the attention-less update formula (11.3)):

$$h_i^{(pred)} = \tanh((A \odot h^{(latent)}) C^{(A)} + h_i^{(pred)} C^{(pred)}) \quad (11.8)$$

where $U^{(A)}$, $W^{(A)}$, $C^{(A)}$ and $C^{(pred)}$ are learnable parameters. We expect this mechanism to learn focusing on the parts of $h^{(latent)}$ that carry the information that is relevant for reproduction of tree structure, particularly at the deeper levels. This mechanism *scales* the original tensor, rather than explicitly *selecting* some of its elements, and as such can be likened to a *gating function*; nevertheless, we refer to it as ‘soft attention’ by analogy to models used in visual information processing.

11.3 Results

We assess the performance of SAPS on the set of NL-based program synthesis tasks proposed in Polosukhin and Skidanov, 2018. The domain-specific language used in this suite is Lisp-based AlgoLisp, defined by the grammar in Figure 11.4. A single program is composed as a nested list of instructions and operands of three types: **string**, **Boolean** and **function**. The dataset comprises 99506 examples, each being a pair of NL specification (e.g., *You are given an array a ; find the smallest element in a , which is strictly greater than the minimum element in a .*), and the corresponding AlgoLisp program (respectively `(reduce (filter a (partial0 (reduce a inf) <)) inf min)`). To handle variables (like a in the above example), each occurrence of a new variable allocates a new *placeholder*, and the placeholders extend the vocabulary of tokens used in the NL embedding. Placeholders are shared between examples. As in the cited work, we split the data into training set (79214 examples), validation set (9352) and test set (10940).

Dataset inconsistency After closer examination, we identified inconsistency in the dataset: for about 10% of tasks (in the test set), some tests are not passed even by the ground truth program (we verified that using the AlgoLisp interpreted published in Polosukhin and Skidanov, 2018). To address this issue, we decided to use two performance measures: one

Model	Validation	Test	Number of parameters [M]
SAPS V	0%	0%	5.06
SAPS V Att	0%	0%	5.17
SAPS H	84.31%	78.92%	5.06
SAPS H Att	89.54%	86.20%	5.17
SAPS VH	93.65%	89.10%	5.62
SAPS VH Att	93.73%	92.36%	5.73

Table 11.1: Percentage of perfectly synthesized programs for SAPS configurations **trained from scratch** and latent vector of length 256.

based on the unmodified test and validation sets, in order to maintain comparability to results presented in that study, and another one using a filtered test set, containing only the tasks that were free from the above problem. Note, however, that this issue does not affect the training of SAPS, as it does not involve tests.

Experiment dimensions We examine a range of SAPS configurations that vary along (i) decoder state propagation schemes: vertical (V), horizontal (H), and both (VH), (ii) presence/absence of the soft attention mechanism (Att), (iii) pretraining the decoder (pre) or training it from scratch (pre). Atop of that, we consider also a range of sizes of $h^{(latent)}$ (64, 128, and 256). We train all components using Adam optimizer (Kingma & Ba, 2014) with initial learning rate set to 5×10^{-4} and then reduced exponentially at rate 0.975 per every two epochs. We applied L_2 regularization to all trainable parameters, except for biases and layer normalization.

Training SAPS from scratch In Table 11.1, we present the predictive accuracy of SAPS trained end-to-end from scratch in six configurations that vary on the first two of above-mentioned dimensions, i.e. propagation scheme and presence of attention mechanisms, with latent vector of size 256. Model’s output is considered correct if the produced AST tree exactly matches the syntax of the target program. The synergy between the vertical and horizontal propagation schemes is quite evident. The effect of the attention mechanism is also overwhelmingly positive, particularly on the test set. Absence of propagation of the horizontal state substantially cripples SAPS, as it demands the network to produce different subtrees from the same vertically propagated state.

Pretraining of decoder In the second experiment, we ask whether the best SAPS configuration identified above (SAPS VH Att with latent representation of size 256) can benefit from pretraining of the DRNN decoder in the autoencoder architecture (Figure 11.1). To this aim, we first train the autoencoder to reproduce AST trees on all programs from the training set until saturation, then combine the trained DRNN decoder obtained in this way with the remaining modules to form the SAPS pipeline, which is then trained end-to-end on the (NL specification, program) pairs. Comparing the first row of Table 11.2 with Table 11.1 suggests that pretraining may be indeed helpful: although the pretrained model attains marginally worse accuracy on the test set, it beats the trained from scratch on the validation set by almost one percent point. We examined also the capabilities of pretrained models with smaller latent size (128 and 64). Their performances (Table 11.2) were significantly worse, so we keep the latent size of 256 throughout the rest of this study.

In the above configurations, the decoder (taken from the trained autoencoder) learns alongside with the sequence embedding LSTM. For comparison, we consider also a variant

Model	Validation	Test	AC-Validation	AC-Test
SAPSPre VH Att (256)	94.59%	92.14%	72.98%	68.42%
SAPSPre VH Att (128)	82.59%	77.07%	67.06%	60.18%
SAPSPre VH Att (64)	58.65%	52.22%	50.24%	39.36%
SAPSPre VH Att (256) Fixed	14.91%	14.39%		

Table 11.2: Percentage of perfectly synthesized programs for **pretrained** SAPS configurations. Size of the latent vector given in parentheses. The rightmost two columns show the percentage of programs perfectly reproduced by the corresponding autoencoder (AC) used for pretraining.

Model	Validation	Test
Attentional Seq2Seq	54.4%	54.1%
Seq2Tree	61.2%	61.0%
SAPSPre VH Att (256)	86.67%	83.80%
Seq2Tree + Search	86.1%	85.8%

Table 11.3: Percentage of programs passing all tests; non-SAPS results from Polosukhin and Skidanov, 2018.

dubbed SAPSPre VH Att (256) Fixed, where the latent vectors from the pretrained autoencoder model serve as (fixed) target (desired output) for the sequence embedding LSTM. In this configuration the pretrained decoder remains fixed. As expected, the obtained accuracy is inferior to other configurations, which clearly indicates that mutual co-adaptation of sequence embedding and pretrained decoder is essential. Nevertheless, SAPSPre VH Att (256) Fixed still outperforms the approach based only on input-output pairs from Polosukhin and Skidanov, 2018, which achieved 13.3% of accuracy in the best case, despite relying on an external search mechanism.

Behavior of synthesized programs on input-output tests In Table 11.3 we juxtapose our best configuration, SAPSPre VH Att (256), with the results reported by Polosukhin and Skidanov, 2018. This time the comparison metric is the percentage of programs that pass all tests provided (alongside with NL specifications and target programs) in AlgoLips database prepared by the authors of that study, using their implementation of evaluation metric. Due to the inconsistencies in of tests in about 10% of examples in that dataset (Section 11.3), the numbers for our best SAPS configuration are smaller than those in Table 11.2, which normally would not be possible. Nevertheless, the comparison is fair, as all methods are tested on the same (faulty) tests and thus can be expected to be affected to the same extent. SAPS clearly outperforms the models based on Seq2Seq and Seq2Tree architectures. More importantly however, it performs on par with the Seq2Tree combined with external search algorithm, which was the best approach reported in the cited work (last row of the table).

Model	All tests				Only passable tests			
	Validation		Test		Validation		Test	
	Accuracy	50-Accuracy	Accuracy	50-Accuracy	Accuracy	50-Accuracy	Accuracy	50-Accuracy
SAPSPre VH Att (256)	86.67%	89.64%	83.80%	87.45%	95.02%	95.81%	92.98%	94.15%
SAPSPre VH Att (128)	76.62%	80.92%	70.91%	75.89%	83.90%	86.41%	78.56%	81.54%
SAPSPre VH Att (64)	55.08%	60.35%	49.80%	56.31%	60.19%	64.31%	55.17%	60.56%

Table 11.4: Percentage of programs passing all tests, using all examples (*All tests*), and using only the tests that are consistent with the target program (*Only passable tests*).

Though SAPS does not match its test-set performance, i.e. 85.8%, it is worse only by 2 percent point (and slightly better on the validation set), while being trained end-to-end using gradient and without access to tests, rather than with the help of a sophisticated search mechanism and input-output tests.

In Table 11.4 we compare the results of the best configurations of SAPS tested only on the non-faulty examples (right columns) to their performance on the original, unmodified *AlgoLisp* dataset (left columns). For better insight, we report not only the percentage of programs that pass all tests (*Accuracy*), but also an analogous number for programs that pass at least 50% of tests (*50-Accuracy*). Expectedly, networks obtain higher scores, and those scores are also higher than the percentages of programs that were syntactically identical to the target (Table 11.2). *None* of the trained configurations of SAPS produced any syntactically incorrect programs.

Illustration of SAPS response to modifications of specification In Table 11.5 we present examples of programs synthesized by SAPS. The first row illustrates how the generated source code changes as the specification is being simplified. Remarkably, the network produces correct output even when the input is very simplified and laconic. We evaluated also generalization by replacing operations (*multiplied* $\rightarrow$ *minimum*), applying different ranges of arrays, and complex modifications affecting multiple parts of NL specification simultaneously.

11.3.1 Discussion

SAPS manages to achieve state-of-the-art test-set accuracy, on par with that of Polosukhin and Skidanov, 2018, and does so with a bare neural model, without any search, additional postprocessing or other forms of guidance. This remains in stark contrast to that study, where network was queried repeatedly in a Tree-Beam search heuristics to produce the target program step by step, testing the candidate programs on provided tests. There are also many differences with respect to Balog et al., 2016, where a network was used to prioritize search conducted by an external algorithm. SAPS’s architecture can provide similar accuracy with purely neural mechanisms, with fixed-dimensionality latent layer $h^{(latent)}$ that implements a complete embedding for a large number of programs.

The fact that none of SAPS configurations ever produced a syntactically incorrect program (including the testing phase) suggests that SAPS captured well the *syntax* of *AutoLisp* and is likely to produce well-formed programs for other NL specifications – which was also corroborated by the effects of manipulations shown in Table 11.5. Moreover, it is also likely that SAPS has rather good insight into program *semantics*, which can be concluded from its behavior on the passable tests, summarized in the right part of Table 11.4. Even when a synthesized program is not identical to the target one, there is still a chance for it to be correct. For instance, for the test set, SAPSpre VH Att (256) produces a perfect copy of the target program in 92.14% of test examples (Table 11.2), while in total 92.98% all programs it synthesizes pass all passable tests (Table 11.4). Therefore, 0.84% of programs pass all tests despite being syntactically different from the target program (which translates into roughly one in 9 of such programs), i.e. the network came up with an alternative implementation of the target concept expressed by the NL specification.

Making SAPS to scale well for much longer NL specifications and/or target programs (the median length of the former was 36 words, and of the latter 20 tokens) would arguably require augmenting the method with additional modules, for instance an attention mechanism for interpretation of the NL specification, as used by Polosukhin and Skidanov, 2018. Note, however, that any additional mechanisms that circumvents the latent layer makes this architecture less modular, which may be undesirable, particularly for pretraining (note that

Specification	Synthesized program
<i>you are given numbers a and b, your task is to find $a + b$</i>	<code>(+, a, b)</code>
<i>you given numbers a b, your is find $a + b$</i>	<code>(+, a, b)</code>
<i>given a numbers b, find $a + b$</i>	<code>(+, a, b)</code>
<i>given a numbers b, $a + b$</i>	<code>(+, a, b)</code>
<i>a b, $a + b$</i>	<code>(+, (+, a, b), c)</code>
<i>you are given numbers a and b, your task is to find a multiplied by b</i>	<code>(*, a, b)</code>
<i>you are given numbers a and b, your task is to find minimum a and b</i>	<code>(min, a, b)</code>
<i>given a number a and an array of numbers b, find the length of the longest subsequence of range from 0 to a inclusive that is a prefix of b</i>	<code>(reduce, (range, 0, (+, a, 1)), 0, (lambda2, (if, (==, arg2, (if, (<, arg1, (len, b)), (deref, b, arg1), 0)), (+, arg1, 1), arg1)))</code>
<i>given a number a and an array of numbers b, find the length of the longest subsequence of range from 1 to a exclusive that is a prefix of b</i>	<code>(reduce, (range, 1, a), 0, (lambda2, (if, (==, arg2, (if, (<, arg1, (len, b)), (deref, b, arg1), 0)), (+, arg1, 1), arg1)))</code>
<i>given an array of numbers a, find median of values in a after only keeping first half</i>	<code>(deref, (sort, (slice, a, 0, (/ , (len, a), 2))), (/ , (len, (slice, a, 0, (/ , (len, a), 2))), 2))</code>
<i>given an array of numbers a, find mean of values in a after only keeping second half</i>	<code>(/ , (reduce, (slice, a, (/ , (len, a), 2), (len, a)), 0, +), (len, (slice, a, (/ , (len, a), 2), (len, a))))</code>

Table 11.5: The effects of modifications of NL specification. The first specification in each group is an original task from the validation set, and those that follow are its modified variants (modifications marked in bold). Except for a b , $a + b$, all synthesized programs are consistent with specification.

our autoencoder can be now potentially pretrained on a (possibly large) collection of programs without NL specifications).

We attribute the accuracy of SAPS mainly to the quite sophisticated design of our decoder, which is suggested by the comparison of SAPSpre VH Att (256) to SAPSpre VH Att (256) Fixed (Table 11.2), where the latter used a fixed decoder, pretrained in the autoassociative mode. The huge gap between the performances of these architectures clearly indicates that it was essential to allow the decoder to adapt in the end-to-end learning spanning from NL to AST trees. The experiment with different signal propagation schemes (Table 11.1) suggests that the decoder owes its malleability in great part to DRNN being allowed to propagate its state both horizontally and vertically. Last but not least, the proposed soft attention mechanism also contributes significantly to the performance of SAPS, even though it has access only to the latent vector.

11.3.2 Conclusion

We have shown that reliable end-to-end synthesis of nontrivial programs from NL is plausible with exclusively neural means and gradient descent as the learning mechanism. We find it likely that methods like SAPS may be successfully applied for practical synthesis of short code snippets, for instance in end-user programming. In future works, we plan to devise means to address the composite character of both specifications and program code.

11.4 Implications

The research presented in this chapter provides the proof-of-concept for the problem solving framework, demonstrating that the architectural topology of the sensing-acting is not exclusive to physical agents. By defining the interpretation of natural language as a mechanism of *sensing* and the generation of tokens of Abstract Syntax Trees as a mechanism of *acting*, we have established that the cycle of perception and intervention can be effectively transposed to navigate the constraints of a formal language.

This work transforms the perspective on program synthesis. It shifts the burden of logic construction from rigid, rule-based derivation to a flexible, learnable mapping. The success of the SAPS architecture implies that the “structure” of a solution—typically the domain of discrete logic—can be implicitly learned and generated by a neural component without explicit symbolic reasoning modules. This establishes a critical precedent for the thesis: that a neural network is not just a statistical pattern matcher for noisy data, but a system capable of learning to understand the rules of a structured output space.

However, while this chapter validates the *structural feasibility* of the framework, it operates within a supervised regime, reliant on aligned pairs of specifications and code. It maps input directly to output without constructing a navigable problem space. This finding lays the necessary groundwork for the next stage of the research, where these components must be elevated to their “symbolic” versions to enable the active algorithmic exploration required for abstract reasoning.

Chapter 12

Solving abstract reasoning problems with neuro-symbolic program synthesis and task generation

12.1 Motivation

Abstract reasoning tasks have a long-standing history in AI, as epitomized with Bongard’s problems (Bongard, 1967), Hofstadter’s analogies (Hofstadter, 1995), and more recently the Abstract Reasoning Corpus (Chollet, 2019), Figure 12.1. They reach beyond the input-output mapping tasks typical for contemporary machine learning (e.g. classification and regression) and thus pose several nontrivial challenges. The suits of such tasks tend to be very diverse, each engaging a different ‘conceptual apparatus’ and thus requiring a bespoke solution strategy. This, in turn, implies the necessity of devising sophisticated features that are highly task-specific and need to be “invented on the spot”, like the presence/absence of previously unseen objects in the visual input, or the number of tokens in a sequence, or a specific pattern thereof. Another challenge is the staged nature of reasoning, which often needs to comprise multiple steps of inference, thus requiring some notion of working memory for storing and manipulating diverse concepts.

These and other characteristics of abstract reasoning tasks become particularly challenging when one attempts to *learn* to solve them, rather than manually devising a complete architecture of a working solver. Given the diversity of the tasks to be faced, one cannot rely on predefined representations and concepts, and often not even on preset solving strategies, and it becomes essential to allow the learner to construct them on the fly and, in particular, to *learn* what to construct *given the context of the given task*.

In the past, abstract reasoning tasks have been most often approached with algorithms relying exclusively on symbolic representations, typically involving some form of principled logic-based inference. While this can be successful for problems posed “natively” in symbolic terms (e.g. Hofstadter, 1995), challenges start to mount up when a symbolic representation needs to be inferred from a low-level, e.g. visual, representation (Bongard, 1967; Chollet, 2019). The recent advances in deep learning and increasing possibilities of their hybridization with symbolic reasoning (see Section **Related work**) opened the door to architectures that combine “subsymbolic” processing required to perceive the task with sound symbolic inference.

Addressing the above challenges requires a flexible, expressive, structural, and compositional formalism that permits the learner to experiment with alternative representations and features of the task, combine and validate them, and so learn successful Problem Solving strategies. While devising completely new mechanisms for this purpose is arguably possible, it is natural in our opinion to reach the realm of programming languages, and to pose the problem of solving abstract reasoning problems as a *program synthesis task*. In this formulation, the solver is an intelligent, reactive agent that perceives the task and attempts to synthesize a program in a bespoke *symbolic interface* (DSL) which, once executed, produces the solution to the task.

This study is based on these premises and introduces TransCoder, a neuro-symbolic

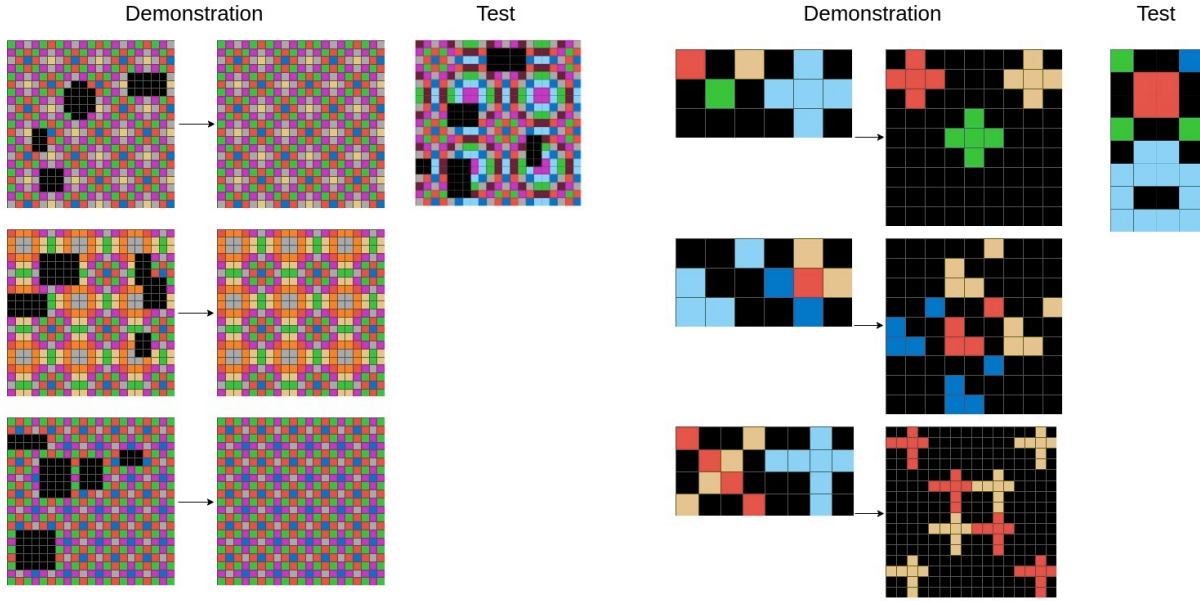


Figure 12.1: Examples from the Abstract Reasoning Corpus Dataset.

architecture that relies on both neural and symbolic perception of the problem and involves programmatic representations to detect and capture relevant patterns in low-level representation of the task, infer higher-order structures from them, and encode the transformations required to solve the task. TransCoder is designed to handle the tasks from the ARC, a popular benchmark that epitomizes the above-mentioned challenges. Our main contributions include (i) the original neuro-symbolic architecture that synthesizes programs that are syntactically correct by construction, (ii) the “learning from mistakes” paradigm for generating synthetic tasks of adequate difficulty to provide TransCoder with a learning gradient, (iii) an advanced perception mechanism to reason about small-size rasters of variable size, and (iv) an empirical assessment on the ARC suite.

12.1.1 Abstract Reasoning Corpus

Abstract Reasoning Corpus (ARC) Chollet, 2019 is a collection of 800 tasks of a visual nature, partitioned into 400 training tasks and 400 testing tasks.¹ Each task comprises a set of input-output pairs that exemplify a mapping/rule to be discovered. That set is divided into *demonstrations* and *tests*. The demonstration set consists of three to six such pairs, while the test set typically contains a single pair. The solver has full access only to demonstrations; for tests, it can observe only the input parts (Figure 12.1). The test set acts as an independent verification mechanism for evaluating the solutions generated by the solver. Essentially, the solver is challenged to extract the inherent problem definition from demonstrations and subsequently apply this learned knowledge to generate solutions for the inputs in tests. This distinction between the roles of the demonstrations and tests emphasizes the importance of generalization and the ability of the solver to extrapolate beyond the provided examples.

The inputs and outputs in a pair are raster images. Images are usually small (at most 30 by 30 pixels) and each pixel can assume one of 10 color values, represented as a categorical variable (there is no implicit ordering of colors).

For each ARC task, there exists at least one processing rule (unknown to the solver) that maps the input raster of each demonstration to the corresponding output raster. The solver is expected to infer² such a rule from the demonstrations and apply it to the test input rasters to produce the corresponding outputs. The output is then submitted to the oracle which, based on the knowledge of the true/desired outputs, returns a binary response informing

about the correctness/incorrectness of the solution.

The ARC collection is extremely heterogeneous in difficulty and nature, featuring tasks that range from simple pixel-wise image processing (like re-coloring of objects), to mirroring of the parts of the image, to combinatorial aspects (e.g. counting objects), to intuitive physics (e.g. an input raster to be interpreted as a snapshot of moving objects and the corresponding output presenting the next state). In quite many tasks, the black color should be interpreted as the background on which objects are presented; however, there are also tasks with rasters filled with “mosaics” of pixels, with no clear foreground-background separation (see, e.g., the left example in Figure 12.1). Raster sizes can vary between demonstrations, and between the inputs and outputs; in some tasks, it is the *size* of the output raster that conveys the response to the input. Because of these and other characteristics, ARC is widely considered extremely hard: in the Kaggle contest accompanying the publication of this benchmark³, which closed on the 28th of May 2020, the best contestant entry algorithm achieved an error rate of 0.794, i.e. solved approximately 20% of the tasks from the (unpublished) evaluation set, and most entries relied on a computationally intensive search of possible input-output mappings. Section **Related work** reviews selected ARC solvers proposed to date.

12.1.2 Related work

TransCoder engages programs, i.e. a class of symbolic entities, to process and interpret the input data, and thus bears similarity to several past works on neuro-symbolic systems, of which we review only the most prominent ones.

In synthesizing programs in response to input (here: task), TransCoder resembles the Neuro-Symbolic Concept Learner (NSCL, Mao et al., 2019). NSCL was designed to solve Visual Query Answering tasks (Johnson et al., 2016) using object-centric representation and learn to parameterize a semantic parser that translated a NL query about scene content to a DSL program which, once executed, produced the answer to the question. The system was trained with reinforcement learning (RL), with a binary reward function that reflected the correctness of the answer provided to the query. Interestingly, NSCL’s DSL was implemented in a differentiable fashion, which allowed it to inform its perception subnetwork in training.

With respect to abstract reasoning, the key characteristic aspect of the ARC suite, a similar “object-centric” approach is taken by Assouel et al., 2022, which operates on a category of ARC tasks. The work relies on strong assumptions about the tasks from the considered category and provides a DSL for manipulating and transforming objects typical for rasters used in those tasks. This formalism allows the authors to generate new unseen tasks, which are subsequently used to train their model, in a fashion similar to TransCoder. However, the proposed approach does not involve program synthesis nor a DSL *per se*.

The most successful approaches in the ARC 2020 Kaggle challenge often relied on an assembly of various processing techniques including brute-force search, handcrafting features, manual data labeling, and DSL-guided search. The results of the competition corroborated the awareness of the usefulness of the program synthesis approach with data-tailored DSLs, and resulted in approaches which engaged, among others, descriptive grids and the minimum-description length principle (Ferré, 2021), graph-based representations (Xu et al., 2022), or combining DreamCoder (Ellis et al., 2021) with a search graph (Alford et al., 2021). To a greater or lesser extent, many of those methods engaged also exhaustive search, which is notably absent in TransCoder.

In using a program synthesizer to produce new tasks, rather than only to solve the presented tasks, TransCoder bears some similarity to the DreamCoder (Ellis et al., 2021). DreamCoder’s training proceeds in cycles comprising *wake*, *dreaming*, and *abstraction* phases which realize respectively solving the original problems, training the model on “replays” of

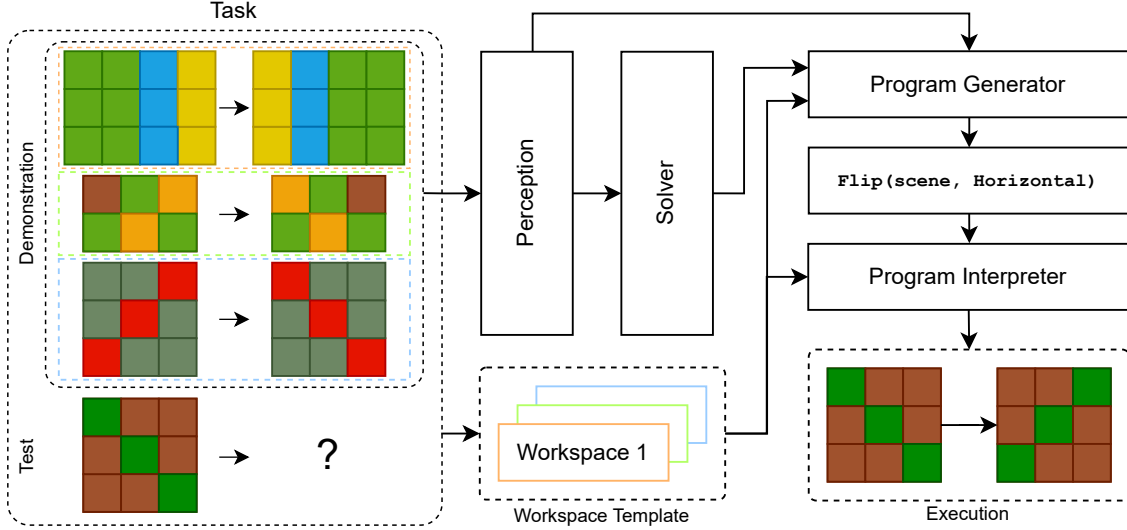


Figure 12.2: The overall architecture of TransCoder comprises Perception, Solver, and Program Generator. The former two are purely neural, while the Generator combines symbolic processing and information from the neural model. The Generator relies also on symbolic working memory (Workspace), which maintains both entities specific to individual demonstrations (local) as well as those common for all demonstrations and predefined concepts (global).

the original problems and on “fantasies” (synthetic problems), and refactorization of the body of synthesized programs. The last phase involves identification of the often-occurring and useful snippets of programs, followed by encapsulating them as new functions in the DSL, which is meant to facilitate “climbing the abstraction ladder”. The DreamCoder was shown to achieve impressive capabilities on a wide corpus of problems, ranging from typical program synthesis to symbolic regression to interpretation of visual scenes. For a thorough review of other systems of this type, the reader is referred to (Chaudhuri et al., 2021).

Recent advancements in large language models (LLMs) have led researchers to explore their capabilities in abstract reasoning tasks, notably the ARC and RAVEN (Zhang et al., 2019) benchmarks. An extensive analysis of ARC was conducted in Moskvichev et al., 2023, resulting in a comparison of Kaggle 2020 competition winner methods, GPT 4, and human performances on ConceptARC, a categorized version of the ARC dataset proposed in that study. One of the main findings of this study was human’s superior capability, compared to all considered algorithmic methods. Extensive research on LLMs applied to ARC tasks and other problems (Mirchandani et al., 2023) has demonstrated significant improvements through prompt engineering techniques in ARC solving, and pointed to the overall capacity of those models as sequence modelers and quite successful pattern extrapolators. These advances have enabled LLMs to surpass the accuracy of many DSL-based methods, achieving accuracy rates of approximately 10% without fine-tuning, and often via zero-shot learning via prompting.

The perception module draws inspiration from the vision transformer architecture (Dosovitskiy et al., 2020), interleaving self-attention and cross-attention blocks (Vaswani et al., 2017) with reduction blocks (Lin et al., 2014).

12.2 Details

The broad scope of visual features, object properties, alternative interpretations of images, and inference mechanisms required to solve ARC tasks suggest that devising a successful solver requires at least some degree of symbolic processing. Among others, objects in ARC rasters tend to be small and “crisp”, so that even a single-pixel difference between them matters (not mentioning the fact that some tasks feature single-pixel objects). The conventional convolution-based perception typical for contemporary deep learning models is not sufficient to capture this level of detail, also because they are prepared to process categorical, unordered colors. Moreover, such precision is required not only in perception but also when producing output rasters to match those in demonstrations or generate the response to the test image.

It is also clear that reasoning conducted by the solver needs to be *compositional*; for instance, in some tasks, objects must be first delineated from the background and then counted, while in others objects need to be first counted, and only then the foreground-background distinction becomes possible. It is thus essential to equip the solver with the capacity to rearrange the inference steps in an (almost) arbitrary fashion.

Following the motivations outlined in Section 12.1, the above observation is a strong argument for representing the candidate solutions as *programs* and forms our main motivation for founding TransCoder on *program synthesis*, where the solver can express a candidate solution to the task as a program in a *Domain Specific Language* (DSL), a bespoke programming language designed to handle the relevant entities. Because (i) the candidate programs are to be generated in response to the content of the (highly visual) task, and (ii) it is desirable to make our architecture efficiently trainable with a gradient to the greatest degree possible, it becomes natural to control the synthesis using a neural model. TransCoder is thus a *neuro-symbolic system* that comprises the following components (Figure 12.2):

- **Perception:** encodes demonstration input-output pairs into latent representations z_i of fixed dimensionality, capturing inter-example relationships, where i points to the representation of the i -th demonstration,
- **Solver:** aggregates latent representations z_i from the perception module to a problem latent space $\mathcal{Z}$ and samples a latent program representation $z \sim \mathcal{Z}$,
- **Program Generator (Generator for short):** based on the latent program representation z and additional information detailed in Section **Workspace**, constructs an Abstract Syntax Tree (AST) of the synthesized program p ,
- **Program Interpreter (Interpreter for short):** executes p on input rasters to produce the corresponding output rasters, which can be then compared with the expected output rasters.

In training, the Interpreter applies p independently to each of the input rasters of demonstrations and returns the predicted output rasters, which are then confronted with the true output rasters using a loss function. In testing, p is applied to the test raster and the resulting raster is submitted to the oracle that determines its correctness.

We detail the components of TransCoder in the following sections. For technical details, see Appendix A.

12.2.1 The perception module

The perception module comprises the *raster encoder* and the *demonstration encoder*.

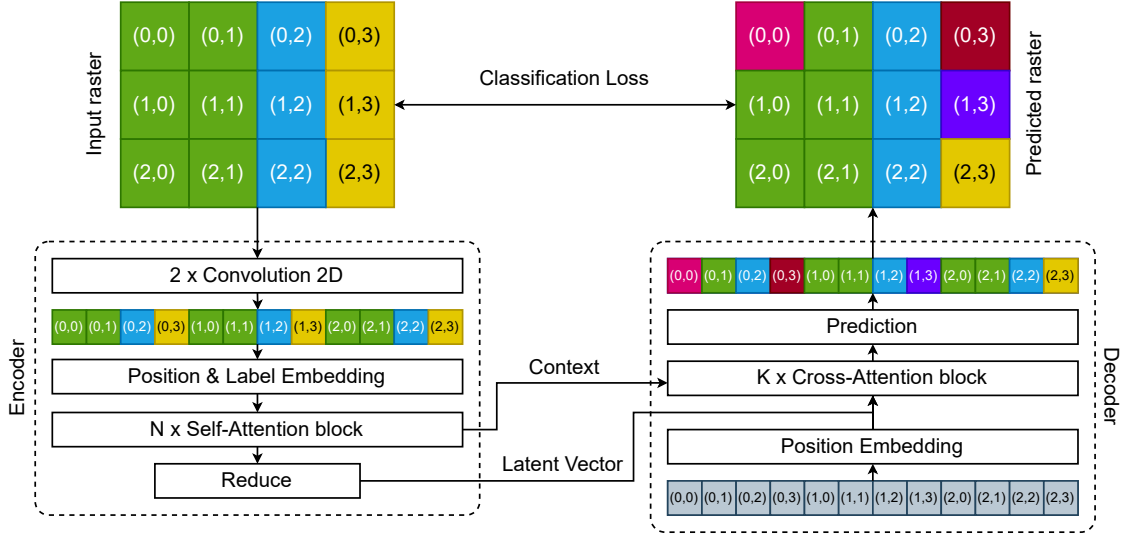


Figure 12.3: The autoencoder architecture used to pre-train the raster encoder in Perception. Left: the encoder (used in TransCoder after pre-training). Right: the decoder (used only in pre-training and then discarded).

As argued at the beginning of this section (and verified in our preliminary studies), typical convolutional layers are insufficient for capturing the information that is essential in ARC tasks. Therefore, we abandon purely convolutional processing in favor of encoding rasters as one-dimensional sequences of tokens that convey information about pixels. Prior to the sequential processing block, the data undergoes a preprocessing with two 2D convolutional layers. The layers preserve the spatial dimensions of the input tensor, ensuring that both height and width remain unchanged, which is essential due to extremely small rasters in the dataset. The information about the raster size itself, often crucial for solving the task, is also conveyed by the length of the sequence.

Our raster encoder is based on an attention module that allows processing rasters of different sizes, which is required when solving ARC tasks (see, e.g., the task shown in Figure 12.2). Each pixel resulting from the last convolutional layer gives rise to a separate token, tagged with the color *and* (x, y) coordinates, the latter acting as input to a *positional embedding layer*, architecturally similar to those employed in transformer models (Vaswani et al., 2017), but adapted for a 2D spatial context. This embedding strategy allows the model to explicitly incorporate spatial information into its representation of the input data. The resulting tokens are flattened into a one-dimensional sequence (see the left part of Figure 12.3), which is then processed by a series of eight self-attention blocks. Subsequently, a global average pooling operation reduces the final sequence to a single, fixed-size vector, yielding a compact representation of the raster. These raster representations are then passed to the demonstration encoder.

The raster encoder can be trained alongside the other TransCoder components; however, this turned out to be computationally inefficient. More importantly, when training the entire architecture end-to-end, it is hard to determine the component that should be blamed for the failure to deliver the correct answer. For these reasons, we pre-train the raster encoder within an autoencoder architecture, where the encoder is combined with a compatible decoder that can reproduce varying-length sequences of pixels (and thus the input raster) from a fixed-dimensionality latent. The decoder is shown in the right part of Figure 12.3 and comprises a sequence of steps that attempt to recover the spatial and color information from the fixed-dimensional latent.

The objective of pre-training is to develop an encoder capable of effectively encapsulating

the essential information from the input data in the latent representation. To achieve this, we train the autoencoder in the self-supervision mode, by confronting it with a “fill-in” task. The decoder is tasked with predicting the correct pixel value (pixel color) at a given location, conditioned on the latent representation of the input raster from the encoder. By providing the decoder with the complete set of target pixel positions, the requirement for the decoder to determine the output raster is relaxed (compared to an alternative scenario in which one could require the decoder to restore the *entire* sequence, i.e. both colors and coordinates). The decoder then concatenates the embeddings of target positions with the reduced raster representation to prevent the encoder from overspecializing on the “fill-in” tasks. As the “query” position representation contains partial raster representation, the encoder is forced to reason in terms of dependencies between pixels and their colors and coordinates, rather than overfitting to specific locations and colors of pixels. The decoder then processes the tokens with a stack of cross-attention blocks. Eventually, a fully connected layer produces per-pixel predictions, which are confronted with the actual pixels from the input sequence using pixel-wise categorical cross-entropy loss on the color variable.

The autoencoder is pre-trained until convergence, using all rasters (inputs, outputs and tests) collected from the training part of the ARC collection. In this process, we intend to make feature extraction invariant to permutations of colors, while accounting for the black color serving the special role of the background in a large fraction of tasks. To this end, the rasters derived from the ARC are treated as dynamic templates, which are being randomly re-colored on the fly, while preserving the distinctions of colors. This enforces permutation-invariance in the space of colors and improves generalization. As a result of pre-training, the raster encoder used in the experimental part of this study achieves the per-pixel reconstruction accuracy of 99.99% and the per-raster accuracy of 96% on the testing part of the original ARC. The resulting encoder is thus guaranteed to pass on almost the entirety of information available in the input rasters.

Upon completion of pre-training, the raster encoder is prepended to the demonstration encoder in TransCoder. For each demonstration pair, the demonstration encoder invokes the raster encoder twice, separately for the input and output raster, concatenates the resulting latent vectors, and passes them through a two-layer MLP. This is repeated for all demonstrations, and the output vectors produced by the MLP are chained into a sequence, which is then subject to processing with four consecutive self-attention blocks. The final sequence z_i is the representation of the demonstrations.

12.2.2 Solver

The sequential latent z_i produced by the Perception task forms a representation of demonstrations of the given ARC task. As the raster encoder has been almost perfectly pre-trained via auto-association (see previous section), we expect this sequence to convey the entirety of information about the *content* of the task⁴. However, this does not imply the capacity of *solving* the task. Therefore, the role of the Solver is to map z_i to a latent representation z of the program to be synthesized in the DSL, which is meant to act on the task and solve it.

Technically, the Solver is implemented as a two-layer MLP preceded by averaging over the sequence of z_i s. In principle, the resulting latent vector z could be directly passed to the program generator (Section **Program generator**). However, as in most programming languages, in our DSL (detailed in Section **The symbolic interface**) the relationship between the programs (their syntax) and their input-output behaviors (semantics) is many-to-one, i.e. the same semantics can be implemented with more than one program. As a result, a given ARC task can be solved with more than one program, and this very program can be a

solution to more than one ARC task.

To account for this many-to-many correspondence, we make the Solver stochastic by following the blueprint of the Variational Autoencoder (Kingma & Welling, 2013): the last layer of Solver’s MLP does not directly produce z , but parameterizes the normal distribution $\mathcal{Z}(z_\mu, z_\sigma)$ with two vectors z_μ and z_σ and then calculates $z = z_\mu + z_\sigma N(0, 1)$ where $N(0, 1)$ comes from a random number generator. The resulting latent z is subsequently passed to the Program Generator, accompanied by the additional information about the task harvested from the Workspace.

The variational layer allows thus the Solver to propose alternative solutions to the same ARC task. Ideally, one would hope z_μ to represent the centroid of the set of programs that solve a given task (and thus share in this sense a common semantics), while z_σ to model the extent of that set in the latent space (i.e., the syntactic variations between those programs). Of course, such ideal convergence is not guaranteed, and the subsequent stages of processing do not rely on its emergence. However, as we found in preliminary experiments, the variational layer improves also exploration in training.

The presence of sampling makes TransCoder non-deterministic and brings tangible benefits in training: as we found in preliminary experiments, it significantly improves exploration, helping the method to produce non-trivial programs, and is particularly important for generating synthetic tasks (Section **Learning from mistakes**). If desired, sampling can be disabled at test time, with z_μ passed as z , allowing for deterministic replication of TransCoder’s querying. Importantly, even though sampling may lead to different programs in response to the same ARC task, each of those programs has well-defined, *deterministic* semantics, and will always produce the same output when applied to a given input.

12.2.3 Workspace

The Workspace is the core symbolic module of TransCoder, which is meant to provide the Program Generator (Section **Program generator**) with additional information, alongside the latent z resulting from the Solver. It supplies the Generator with two types of symbolic entities, i.e. (i) the **percepts** representing parts of the rasters in demonstrations, and (ii) the **elements of the DSL**, i.e. its instructions and constants. The access to both these types of elements is provided in a unified manner, by embedding them in the same space, which allows them to be retrieved on-demand by the Generator during program synthesis, composed in an arbitrary way (limited only by the syntax of the DSL), and ultimately “tried out” via program execution.

Concerning the **access to percepts in the Workspace**, we supplement the information provided by the Perception module with selected symbolic percepts inferred independently from the task via direct “symbolic parsing” of images. This supplementation is essential, as many ARC tasks involve qualitative and combinatorial concepts (e.g. counting of objects) that cannot be naturally conveyed by the neural nature of processing and representation provided by the Perception module. We provide two data structures for this purpose:

- *Workspace Template* that holds the abstract placeholders for the entities that appear in *all* task’s demonstrations,
- *Workspaces* that “instantiate” that template with concrete values derived from particular demonstrations.

The relation between these two storages can be likened to a dictionary, with the entries in the Workspace Template acting as *keys* that index specific *values* (realizations) in the Workspace of a specific demonstration. For instance, the *Scene* key in the Workspace

Template may have a different value in each Workspace. For the task shown in Figure 12.2, the Workspace Template contains the *Scene* key, and its values in the first two Workspaces are different, because they feature different sets of colors:

```

1 scene_0 = Region( # first demonstration
2     positions=[[0,0], [0,1], [0,2], ...],
3     colors=[Green, Green, Blue, ...]
4 )
5 scene_1 = Region( # second demonstration
6     positions=[[0,0], [0,1], [0,2], ...],
7     colors=[Brown, Green, Orange, ...]
8 )

```

The list of workspace keys is predefined and includes constants (universal keys shared between all ARC tasks, e.g. *zero*, *one*), local invariants (values that repeat within a given task, across all demonstrations, e.g. the set of used colors), and local keys (information specific to a single demonstration pair, e.g. an input *Region*). For a complete list of available keys, see **Appendix**.

The workspaces require appropriate “neural presentation” for the Generator of DSL programs. For a given task, all keys available in the Workspace Template are first embedded in a Cartesian space, using a learnable embedding similar to those used in conventional deep learning. This representation is *context-free*, i.e. each key is processed independently, and the embedding has in total as many entries as there are unique keys defined in DSL.

Concerning the **access to the elements of the DSL**, they are also embedded in the same Cartesian space so that the Generator can choose from them, i.e. from the DSL instructions and constants from Tables 12.8 and 12.7, respectively, alongside the symbolic percepts. In this way, the elements of the DSL are presented to the Generator (on the neural level) in the context of the given task, and, among others, constants (such as “Red”) may have a different embedding depending on the perception result. This is expected to facilitate alternative interpretations of the roles of particular percepts; for instance, while the black pixels should often be interpreted as the background, some tasks are exceptions to this rule.

12.2.4 The symbolic interface

The DSL we devised for TransCoder is a typed, functional programming language, which allows conveniently representing programs as ASTs. The leaves of AST trees are responsible for fetching input data and constants, inner nodes are occupied by DSL instructions/functions, and the root of the tree produces the return value of the entire program. Each tree node has a well-defined type. The DSL features concrete data types (e.g. *Int*, *Bool*, *Region*) and generic data types (e.g. *List/T*). Tables 12.1 and 12.2 presents the complete list of, respectively, types and instructions.

Within this DSL, a complete program defines a mapping from a set of input values to a corresponding output value of DSL types. Program execution is deterministic, i.e. application of a given program to a given input will always produce the same output; however, the variational layer included in the solver (Section **Solver**) enables TransCoder to respond with a different program in training and so provide exploration of the problem space. Program execution leverages the Workspace for selective retrieval of necessary items using keys. When TransCoder is queried on an ARC problem, the same synthesized DSL program is executed independently for each test Workspace.

The DSL features 40 operations presented in Table 12.2 (and in Table 12.8 in more detail). These can be divided into:

- data-composing operations (form a more complex data structure from constituents, e.g. *Pair*, *Rect*),

Name	Description
Arithmetic	An abstract type that implements basic arithmetic operations such as addition and subtraction
Bool	Logical type
Color	Refers to the categorical value of pixels. It can take one of ten values
Comparable	An abstract type that implements basic operations that allow objects to be compared with each other
Int	Signed integer (32 bit)
Loc	A location consisting of two integers
Connectivity	The type of neighborhood used by the FloodFill operation; possible values are n4 and n8
Direction	The type of direction used by the Rotate operation; possible values are cw (clockwise) and cww (counterclockwise)
Orientation	The type of direction used by the Flip operation; possible values are vertical and horizontal
Region	An object representing any list of pixels and their colors
List[T]	A generic type representing a list of objects of a compatible type
Pair[T, L]	A generic type representing a pair of objects of a compatible type

Table 12.1: The list of types available in the TransCoder’s DSL.

- property-retrieving operations (fetch elements or extract simple characteristics from data structures, e.g. *Width*, *Area* or *Length*),
- data structure manipulations (e.g. *Head* of the list, *First* of a *Pair*, etc.), arithmetics (*Add*, *Sub*, etc.), and
- region-specific operations (high-level transformations of drawable objects, e.g. *Shift*, *Paint*, *FloodFill*).

Our DSL features also higher-order functions, among them *Map* and *Filter*, which apply a subprogram to the elements of a compound data structure like a *List*. The complete definition of the DSL can be found in **Appendix**.

The type system in Table 12.1 together with the operations in Table 12.8 implicitly define the grammar of the DSL. We do not present it formally, as the type-parametricity (the generics) make it context-dependent and quite complex. Crucially, the program generator detailed in the subsequent Section **Program generator** traverses the combined tree of types and function signatures, and so guarantees the resulting programs to be syntactically correct and thus executable. For similar reasons, we do not attempt to formalize the semantics of the DSL, other than via our implementation of the DSL interpreter, which translates each instruction of the DSL from Table 12.2 into a snippet of Python code. This implementation and the NL description in Table 12.8 provide informal, yet quite precise, denotational semantics of our DSL.

There is a number of arguments in favor of using a bespoke DSL rather than relying on generic languages like Prolog or/and related approaches like Inductive Logic Programming (Muggleton & Watanabe, 2014). First, our DSL is already equipped with adequate types for coordinates, regions of connected pixels, and generics (like *Pair*). This is accompanied with a number of bespoke functions that embody “natural” operations on these data types. These elements of the DSL greatly facilitate learning, especially in the difficult initial phase, which is critical for successful formation of a model. Secondly, a fair share of ARC puzzles are ‘operational’ in nature, i.e. require the input panel to be somehow *transformed* into the answer panel. It is thus natural to express such transformations as executable sequences (or

Name	Signature
Add	(A: Arithmetic, A: Arithmetic) ->A: Arithmetic
Area	(Region) ->Int
Crop	(Region, Loc, Loc) ->Region
Deduplicate	(List[A]) ->List[A]
Diff	(List[A], List[A]) ->List[A]
Draw	(Region, Union[Region, List[Region]]) ->Region
Equals	(A, A) ->Bool
Filter	(List[A], (A->Bool)) ->List[A]
First	(Pair[A, Type]) ->A
Flip	(Region, Orientation) ->Region
FloodFill	(Region, Color, Connectivity) ->List[Region]
GroupBy	(List[A], (A->B)) ->List[Pair[B, List[A]]]
Head	(List[A]) ->A
Height	(Region) ->Int
Intersection	(List[A], List[A]) ->List[A]
LBC	(Region) ->Loc
LTC	(Region) ->Loc
Len	(List[Type]) ->Int
Line	(Loc, Loc, Color) ->Region
Loc	(Int, Int) ->Loc
Map	(List[A], (A->B)) ->List[B]
MostCommon	(List[A], (A->B)) ->B
Neg	(Union[A, B: Bool]) ->Union[A, B: Bool]
Paint	(Region, Color) ->Region
Pair	(A, B) ->Pair[A, B]
Pixels	(Region) ->List[Region]
RBC	(Region) ->Loc
RTC	(Region) ->Loc
Rect	(Loc, Loc, Color) ->Region
Reverse	(List[A]) ->List[A]
Rotate	(Region, Direction) ->Region
Scale	(Region, Union[Int, Pair[Int, Int]]) ->Region
Second	(Pair[Type, A]) ->A
Shift	(Region, Union[Loc, Pair[Int, Int]]) ->Region
Sort	(List[A], (A->Comparable)) ->List[A]
Sub	(A: Arithmetic, A: Arithmetic) ->A: Arithmetic
Tail	(List[A]) ->A
Union	(List[A], List[A]) ->List[A]
Width	(Region) ->Int
Zip	(List[A], List[B]) ->List[Pair[A, B]]

Table 12.2: Definitions of the operations available in the TransCoder's DSL.

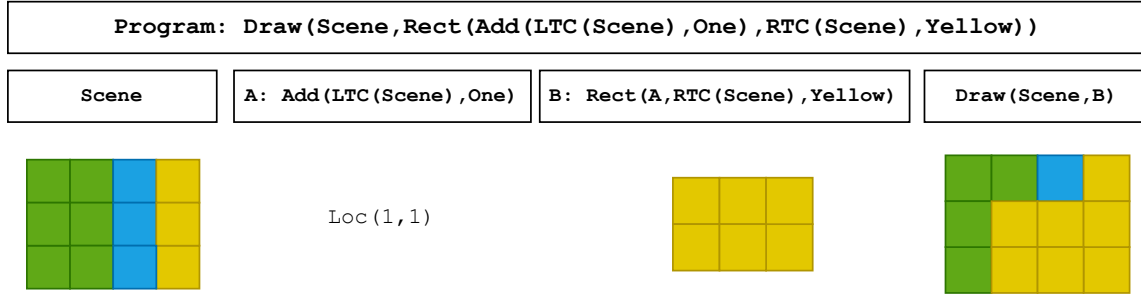


Figure 12.4: Execution process of an example program applied to an input raster (scene). For clarity, we mark the key subprograms with A and B . A adds a constant value of 1 to both coordinates of the upper left corner, i.e. $\text{Loc}(0, 0)$, producing $\text{Loc}(1, 1)$; this operation exemplifies implicit value casting built-in in our DSL (alike the “broadcasting” of arrays in Python). B uses the coordinates calculated by A to draw a yellow rectangle. Finally, the program renders the rectangle produced by B on the input raster and returns the so modified image.

other control structures) of steps that transform the initial state (as observed in the input panel) into a resulting state, represented by the output panel. Examples include moving objects, connecting pixels with segments, mirroring fragments of the input image, counting objects in the input image and “expressing” the obtained number in the output raster, and more. Expressing such staged operations in, e.g., Prolog, which is declarative rather than imperative, while hypothetically possible, would be quite cumbersome.

Figure 12.4 shows an example program and the process of its application to an input raster, including the effects of intermediate execution steps.

12.2.5 Program generator

The Program Generator is a bespoke architecture based on the blueprint of the doubly recurrent neural network; see Alvarez-Melis and Jaakkola, 2017 for a simple variant of DRNN. The latent z obtained from the Solver (Section **Solver**) determines the initial state of this network, which then recursively iterates over the AST nodes of the program being generated.

Before the generation loop, an embedding of each key presented by the Workspace Template is enriched with the information in the latent z_j produced by the Perception (Section **The perception module**) by applying two subsequent cross-attention blocks. This gives the Generator access to precise, symbolic information on demonstration representation, while enabling its “cross-linking” with the presumably less precise, but raster-wide (and thus in this sense global) representation provided by the neural Perception. The resulting vectors form the *contextual embeddings* z_{s_j} of the symbols (values in the Workspace) which is accessed by the Generator in the program generation process described below.

In each iteration, the Generator receives the data on the context, including the current tree size (the number of already generated AST nodes), the depth of the node in the AST, the parent of the current node, and the return type of the node. For the root node, the return type is *Region*; for other nodes, it is determined recursively by the types of arguments required by DSL functions picked in previous iterations. The Generator is also supplied with the set of symbols available in the workspaces (including the elements of the DSL), via their embeddings. From this set, the Generator selects only the symbols that meet the requirements regarding the type and the maximum depth of the tree. Then, it applies an attention mechanism to the embedded representations of the selected symbols. The outcome of attention is the symbol to be “plugged” into the AST at the current location. Figure 12.5

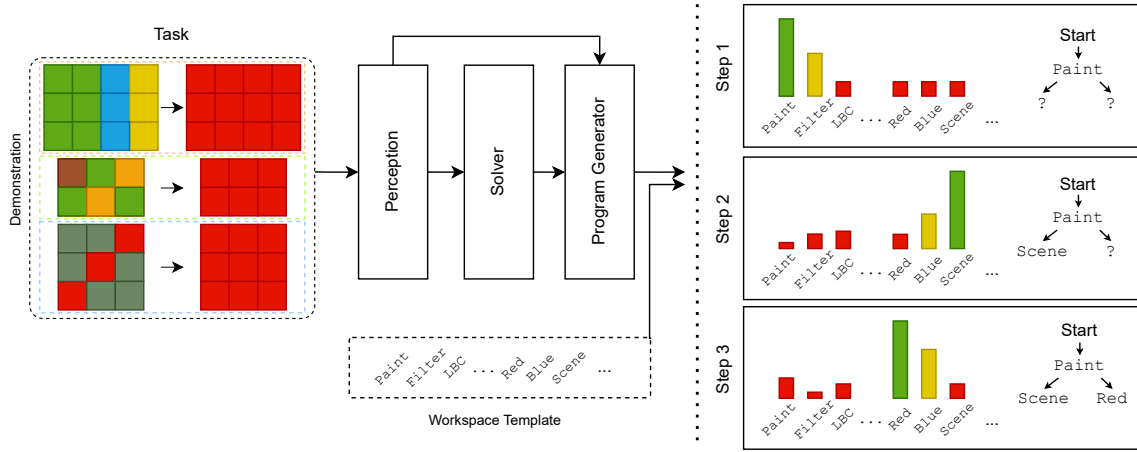


Figure 12.5: When generating a program, TransCoder uses symbol embeddings available in the Workspace Template. At each step of the breadth-first order generation of a program tree (AST), it computes the probability distribution over all symbols and then selects the one with the highest score. As this process is compliant with DSL’s grammar, some operations may be not available at a given step (the model still calculates the probabilities for such operations, but is not allowed to select from them). The generator works as long as there are any arguments missing (e.g., in step 1, from the definition of the *Paint* operation, it follows that two descendants should be generated).

illustrates a few iterations of this process.

By chaining the calls of DSL instructions via type consistency, the generated programs are *guaranteed to be syntactically correct*. This is an essential advantage of TransCoder, compared to alternative mechanisms based on, e.g., language models. In those approaches, programs are stochastically generated sequences of tokens, which need to be checked for syntactic correctness before execution. By providing syntactic correctness by design, the generation mechanism proposed here is more elegant and computationally efficient.

The Generator also determines the hidden state of the DRNN to be passed to each of the to-be generated children subtrees. This is achieved by merging the current state with a learnable embedding indexed with children’s indices, so that generation in deeper layers of the AST tree is informed about node’s position in the sequence of parent’s children.

The generation process iterates recursively until the current node requests no children, which terminates the current branch of the AST (but not the others). It is also possible to enforce termination by narrowing down the set of available symbols. To efficiently utilize computational resources, we adopted the dynamic batching mechanism presented in (J. Bednarek, Piaskowski, & Krawiec, 2018).

12.2.6 Training

TransCoder can be trained with reinforcement learning (RL) or supervised learning (SL). In RL, the program p synthesized by the Generator is applied to the query raster and returns an output raster, which is then sent to the Oracle. The Oracle deems it correct or not, and that response determines the value of the reward, which is then used to update the Generator. The reward value is 1 when the Oracle’s answer is true for all pairs of demonstration and test, otherwise, the value is 0. Unfortunately, the a priori odds for a generated program to solve the given task are minuscule. As a result, training TransCoder only with RL is usually ineffective and inefficient, especially in the early stages, when the generated programs are almost entirely random: most episodes lead to zero reward and, consequently, no parameter

updates. Therefore, based on preliminary experimenting, we abandoned the idea of training TransCoder end-to-end with RL.

The ineffectiveness of RL motivates the SL mode, in which we use the program that solves the given task (target) to guide the Generator. We directly confront the actions of the Generator (i.e. the AST nodes it produces) with the target program node-by-node, and apply a loss function (categorical cross-entropy) that rewards choosing the right symbols at individual nodes and penalizes the incorrect choices. In SL, every prediction produces non-zero gradients, and thus non-zero updates for the model’s parameters (unless the target program has been perfectly generated). We employ the AdamW (Loshchilov & Hutter, 2017) optimizer with default parameter values provided in the TensorFlow (Abadi et al., 2016) library. Additionally, within each outer loop (consecutive Training phases culminating in the transition to the Exploration phase), the learning rate is adjusted according to the method introduced in (Smith, 2018) (with 5 warm-up and 25 decay inner cycles).

The prerequisite for SL is the availability of target programs; as those are not given in the ARC benchmark (in any form, not mentioning our DSL), we devise a method for producing them online during training, presented in Section **Learning from mistakes**. In general, the specific program used as the target will be *one of many programs* that map the input panels of the task’s demonstrations to the corresponding output panels (see Section **Abstract Reasoning Corpus**). Deterministic models cannot realize one-to-many mappings, and the variational layer introduced in Section **Solver** is meant to address this limitation. Upon the ultimate perfect convergence of TransCoder’s training, we would expect the *deterministic* output of the Solver (corresponding to z_μ) to abstractly represent the common semantic of all programs that solve the presented task, and the *variational* layer to sample the latents that cause the Generator to produce concrete programs *with that very semantics*.

12.2.7 Learning from mistakes

The programs produced by the Generator are *syntactically correct by construction*. Therefore, barring the occasional run-time errors (e.g., attempt to retrieve the head of an empty list), a generated program⁵, applied to an input raster, will always produce some *response* output raster. By applying such a program p to each of input rasters I of some task T , we obtain the corresponding list of responses O . We observe that the resulting raster pairs $(x, y) \in I \times O$ form another ARC task T' , to which p is the solution (usually *one of* possible solutions, to be precise). The resulting pair (T', p) forms thus a complete example that can be used to train TransCoder in SL mode, as explained in the previous section, where T' is the task to be solved and p is the corresponding target program.

This observation allows us to *learn from mistakes*: whenever the Generator produces a program p that fails the presented training task T , we pair the inputs with the actual outputs generated by p , add the *synthetic task* (T', p) formed in this way to the working collection S of *solved tasks*, and subsequently use them for . Crucially, we expect T' to be on average easier than T , and thus provide the training process with a valuable “learning gradient”. By doing so, we intend to help the model make progress in the early stages of training, when its capabilities fall behind the difficulty of the original ARC tasks.

To model the many-to-many relation between tasks and programs, we implement S as a relational database to facilitate the retrieval of all programs (known to date) that solve a given task, and vice versa — the retrieval of all tasks solved by a given program. We disallow duplicates in S .

The top-level learning process starts with $S = \emptyset$ and the *working training set* L filled with the original ARC tasks, and stages learning into cycles of *Exploration*, *Training*, and *Reduction* phases (Figure 12.6). As TransCoder spends most of its time iterating between

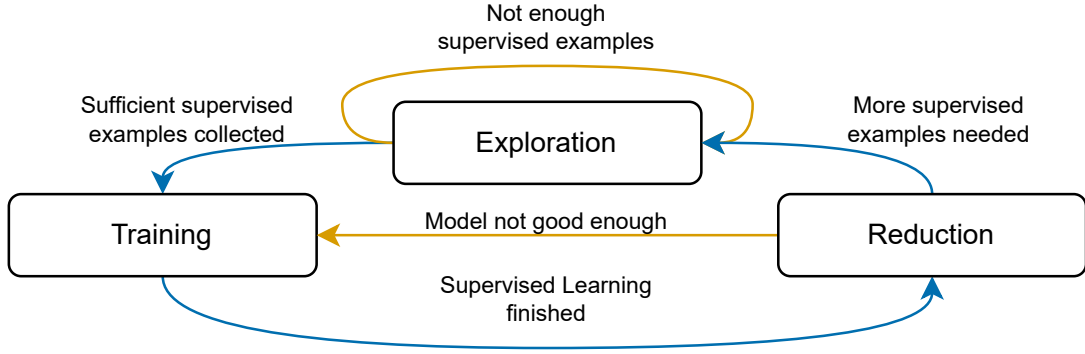


Figure 12.6: The state diagram of TransCoder’s training, including learning from mistakes.

Training and Reduction, this will be referred as the *inner loop* of the method; Exploration happens more sparingly, and thus we consider it a part of the *outer loop*.

Exploration The purpose of this phase is to provide synthetic training examples needed in subsequent phases. A random task T is drawn from L and the Generator is queried on it. If the generated program p solves T , the pair (T, p) is added to S . Otherwise, it is checked whether the response rasters produced by p for T ’s demonstrations meet basic criteria of nontriviality, i.e. are non-empty and vary by demonstration. If T passes this test, (T, p) is added to the S and L . This continues until enough new tasks have been added to S .

Training Training consists in applying SL to a random subset of tasks drawn from S . The execution of this phase ends after iterating m times through all training examples in the drawn subset.

Reduction Reduction starts with selecting from S a subset of tasks with known target programs. Then, those tasks are grouped by programs; a group of tasks with the same program forms a *category*. Next, n categories are drawn at random. Finally, k tasks are drawn for each category. The tasks selected in this way form a working subset $L' \subset L$.

In the next step, TransCoder is evaluated on L' . For each task $T \in L'$, if the program produced by the Generator solves T , it is marked as *learned* and removed from S (if present in S). Otherwise, T is marked as *not learned* and added to S (if not present in S)⁶.

Finally, the results for L' are grouped according to the category from which the tasks come and the average percentage β of solved tasks within each of them is calculated. If β reaches the minimum threshold $\beta_{\min}$ in n_s consecutive iterations of the Training stage, we declare stagnation and switch to the Exploration phase, invoking a new iteration of the outer loop; otherwise, we move to the Training phase (i.e., continue iterating in the inner Training-Reduction loop in Figure 12.6).

12.3 Results

In the following experiment, we examine TransCoder’s capacity to provide itself with a “reflexive learning gradient”, meant as continuous supply of synthetic tasks at the level of difficulty that facilitates further improvement. Therefore, we focus on the dynamics of the learning process.

Setup To ensure a sufficiently large pool of training examples, each Exploration phase lasts until the set of solved tasks S contains at least 8192 tasks and 32 unique solutions. During each Training phase, a random subset of 8192 tasks is sampled without replacement from S . The Training phase comprises $m = 4$ iterations over this sampled subset, with the order of tasks randomized in each iteration. For the Reduction phase, we set the number of categories to be drawn for L' to $n = 512$, the number k of tasks to be drawn from each category to 16, the solving threshold $\beta_{\min}$ to 30%, and the number of stagnation iterations $n_s = 3$. Additionally, if the model achieves $\beta \geq 80\%$, the reduction phase stops and another Training phase begins. The programs synthesized by the Generator are limited to 64 nodes, a maximum depth of 8, and at most 2 nestings of higher-order functions. Each invocation of a nested high-order function is considered as a separate program and is also required to meet the above constraints.

To enhance the efficiency and effectiveness of program synthesis, two mechanisms are introduced to constrain the search space and mitigate redundancy in generated programs. The first mechanism, termed **blacklisting**, employs a set of predefined rules that prohibits the Generator from composing specific operations. The complete list of these rules is shown in Table 12.3. For instance, the first rule in that table prevents the model from generating a program where a *Shift* parent operation (which translates a Region by a given offset vector) has another *Shift* operation as its child; this would imply the former being executed immediately after the latter, which is inefficient, as the same overall effect can be achieved by a single invocation of *Shift* with the combined translation vector as an argument. Technically, each rule applies to a specific position (index) in the parent’s list of children. Blacklisting eliminates semantically redundant program structures and works in pair with neural-guided program generation.

The second mechanism, termed **normalization**, aims to shorten the generated programs by applying a set of predefined, semantics-preserving simplification rules. Given a synthesized program, the normalization process identifies applicable rules and iteratively applies them until no further simplification is possible, resulting in an irreducible normal form. This process eliminates redundant or inefficient code segments, leading to more concise and interpretable programs. The complete list of normalization rules is presented in Table 12.4. In contrast to blacklisting, normalization is applied *after* program generation, i.e. to complete synthesized programs, and as such does not directly interfere with the training process.

Normalization plays an important role in the preparation of a data set for a training cycle by reducing multiple semantically equivalent programs to the same normal form⁷. Thanks to this procedure, during SL, the generator does not have to produce the inessential “dead code”. Moreover, normalization prevents the generator from infinite nesting of redundant subprograms (which is particularly likely in the initial training stage).

Metrics The primary metric is the percentage of tasks solved from the testing subset of the original ARC (**ARCRate**). However, because of the difficulty of this corpus, this metric is very coarse. To assess the progress in a more fine-grained way, we prepared a collection of 205,745 synthetic tasks by collecting them from several past runs of the method. This collection is fixed and contains tasks that are on average easier to solve than those from the ARC suite; the percentage of tasks solved from that collection will be referred to as **SynthRate**.

Results We report two representative runs of TransCoder (Run 1 and Run 2) that used the same setting of hyperparameters but varied in the random initialization of the model’s weight. This difference, together with the “cumulative” nature of the search process, resulted in those runs taking significantly different courses, as shown in the following. Both runs

Parent Operation	Position	Disallowed Child Operation(s)
Shift	0	Shift
Paint	0	Paint, Rect, Line
Scale	0	Scale
Crop	0	Crop
Neg	0	Neg
Area	0	Flip, Rotate, Shift, Paint, Rect, Line
Width	0	Flip, Rotate, Paint, Shift, Rect, Line
Height	0	Flip, Rotate, Paint, Shift, Rect, Line
LTC	0	Flip, Rotate, Paint, Shift, Rect, Line
LBC	0	Flip, Rotate, Paint, Shift, Rect, Line
RTC	0	Flip, Rotate, Paint, Shift, Rect, Line
RBC	0	Flip, Rotate, Paint, Shift, Rect, Line
First	0	Pair
Second	0	Pair
Reverse	0	Reverse, Union, Deduplicate, Intersection, Diff
Deduplicate	0	Deduplicate, Reverse
Head	0	Reverse, Union, Deduplicate, Intersection, Diff
Tail	0	Reverse, Union, Deduplicate, Intersection, Diff
Sort	0	Sort, Reverse
MostCommon	0	Deduplicate, Reverse
FloodFill	0	Paint, Rect, Line, Rotate, Flip, Shift
Draw	1	FloodFill, Pixels

Table 12.3: The blacklisting rules that are integrated into the Generator’s symbol selection process and prevent it from composing certain pairs of parent-child operations.

Add(a, zero) Add(zero , a) → a	Intersection(a, a) → a
Add(a, Neg(b)) → Sub(a, b)	Union(a, a) → a
Add(Neg(a), b) → Sub(b, a)	Shift(a, zero) → a
Sub(a, zero) → a	Shift(a, LTC(scene)) → a
Sub(zero , a) → Neg(a)	Paint(Paint(a, c2), c1) → Paint(a, c1)
Sub(a, Neg(b)) → Add(a, b)	Scale(a, one) → a
Neg(zero) → zero	Flip(Flip(a, dir1), dir1) → a
Neg(Neg(a)) → a	

Table 12.4: Normalization rules applied to the synthesized programs. Bold arguments state for predefined symbols (e.g. constant “zero”) from Workspace. For some rules, we also compare entire subprograms since the same subprograms always return the same values.

lasted for 5 outer cycles of the method (Figure 12.6), i.e. entered the Exploration phase 4 times (the last invocation of the Exploration phase only marks the moment at which the model is evaluated).

Figures 12.7a, 12.7b, and 12.7c present the dynamics of the training process. We time the runs with the number of Training-Reduction cycles, as this is the part of the state graph from Figure 12.6 that the method iterates over the most. The number of those cycles is highly correlated with the incurred computational expense. The invocations of the Exploration phase, marked with vertical lines, happen much more sparingly, triggered by the conditions discussed at the end of Section **Details**.

The continuous lines in Figure 12.7a show SolveRate, the percentage of tasks solved, estimated from a random sample drawn from the current set S . Because S varies dynamically along training, this quantity is not objective, yet illustrates the dynamics of training. The sudden drops in performance occur right after the completion of the Exploration phase, which augments S with new tasks that the method cannot yet solve. Notice however that, at least for Run 1, the drops tend to be less prominent in consecutive cycles, which may signal that TransCoder gradually improves its skills of solving the newly generated synthetic tasks.

To provide a more objective assessment, the dashed lines in the Figure 12.7b present SynthRate, the percentage of tasks solved from a precomputed suite of over 200k synthetic tasks. The curves of this metric exhibit fast and steady growth, demonstrating TransCoder’s capacity for continuous progress. Importantly, for both runs, these curves do not tend to saturate, suggesting the potential for additional increase upon further continuation of training.

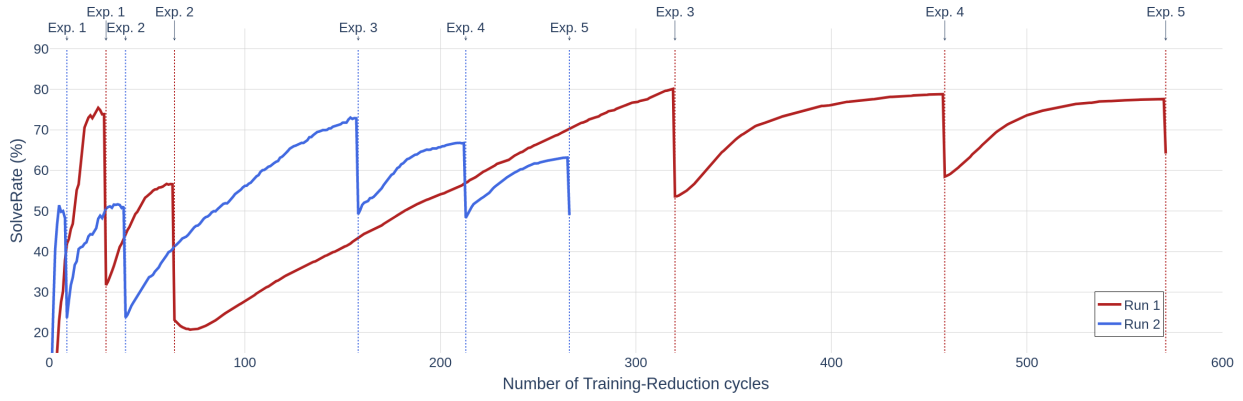
Figure 12.7c presents the dynamics of the working sets of tasks manipulated in TransCoder, synced in time with Figure 12.7a and 12.7b. Each invocation of the Exploration phase expands the working set of tasks S (dotted line) and the L set (solid line). In the Training-Reduction iterations that follow, TransCoder gradually learns to solve those tasks, which causes those solved ones to be removed from S and added to the ‘learned’ set (dashed lines).

The primary feature distinguishing Run 1 from Run 2 is that the latter tended to “consume” (i.e. successively learn to solve) the tasks from S at a higher average rate than the former. This is particularly evident in the third cycle of both runs (after Exp. 2), which lasts over 350 Training-Reduction cycles for Run 1, while only slightly more than 100 cycles for Run 2. This tendency causes Run 2 to invoke the Exploration phase more often, and, as a result, expose TransCoder to new tasks earlier in the process. This, in turn, leads also to faster growth of the L set for Run 2.

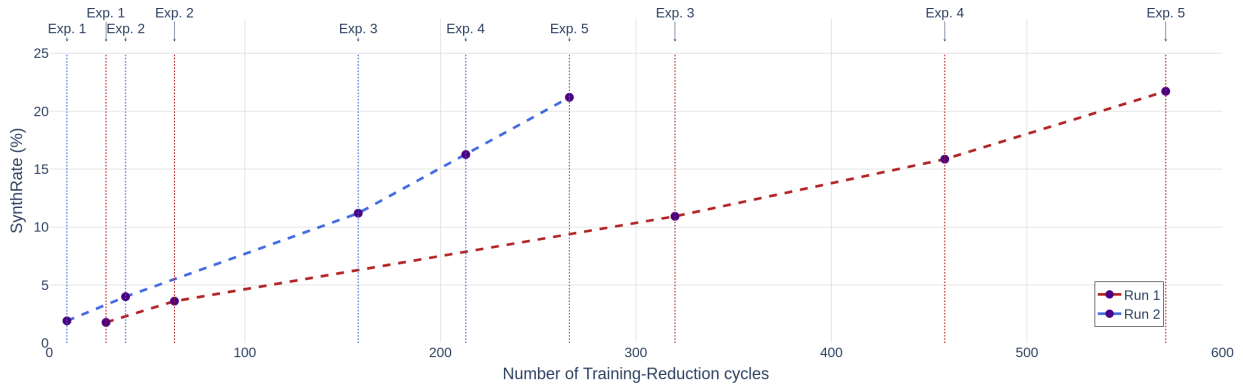
Another difference between Run 1 and Run 2 is that the former one tends to “exhaust” its working set S almost entirely towards the end of each cycle, while for Run 2 each cycle ends with S still holding a substantial number of tasks. It is tempting to hypothesize that this helped Run 2 to make faster progress and ultimately achieve almost the same value of SynthRate as Run 1 (almost 21.2% vs $\sim$ 21.7%), yet at a much lower computational expense ($\sim$ 260 cycles vs. almost 700 cycles). This claim needs, however, to be verified in a separate study.

Table 12.5 presents the metrics at the completion of consecutive training cycles of Run 1 (cf. Figure 12.6). The monotonous increase of SynthRate positively impacts also the ARCRate, which ultimately achieves the all-high value of 3% at the end of the run, suggesting that the skills learned from the synthetic, easier tasks translate into more effective solving of the harder original ARC tasks.

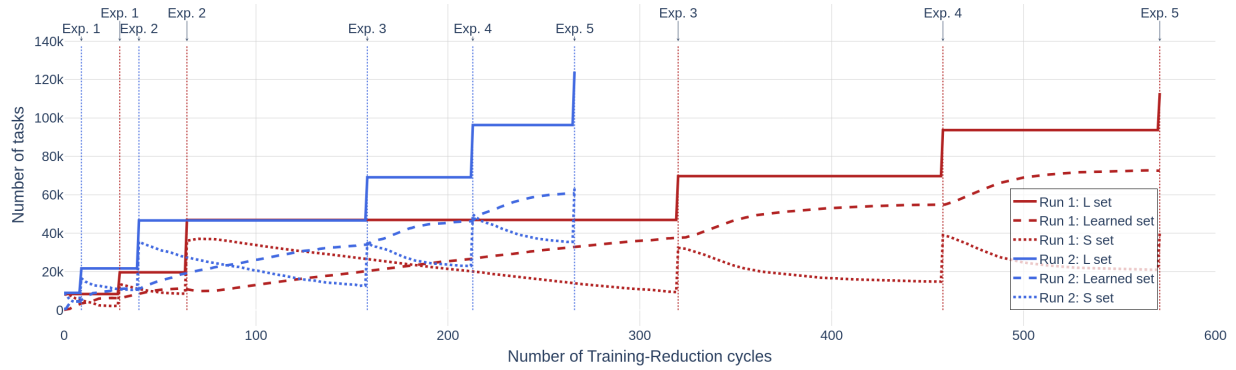
Table 12.6 presents the results in terms of *historical progress*, i.e. how TransCoder fares in consecutive cycles on the synthetic tasks it generated in the previous cycles. This concept has been introduced in Miconi, 2009 to assess the progress on coevolutionary algorithms in absence of well-defined objective yardsticks. The table shows the rate of solved programs achieved by the snapshots of TransCoder trained for a given number of cycles (in rows) on the



(a) SolveRate – the success rate on a set of tasks that dynamically changes during training.



(b) SynthRate – the success rate on a precomputed, fixed set of 205,745 synthetic tasks.



(c) The number of tasks in the working training set L , in the set of solved tasks S , and the number of learned tasks. Once a task has been solved, it is moved from S to the learned set, hence the total size of these both sets remains constant within a given outer cycle of the method.

Figure 12.7: The dynamics of two runs of TransCoder that used different random initializations. Vertical lines indicate the activations of the Exploration phase. Run 1 and Run 2 lasted 4 and 2 days respectively on on NVIDIA A100 GPUs.

Cycle	1	2	3	4	5
SynthRate	1.77%	3.61%	10.93%	15.86%	21.72%
ARCRate	1.75%	2.00%	2.25%	2.75%	3.00%

Table 12.5: SynthRate and ARCRate immediately before the commencement of particular training cycles (i.e. before Exploration phases), for Run 1.

TransCoder’s snapshot from cycle	Synthetic task set S from cycle				
	1	2	3	4	5
1	41.12%	1.18%	0.16%	0.03%	0.02%
2	34.09%	40.42%	0.06%	0.04%	0.00%
3	40.62%	32.92%	55.33%	0.78%	0.38%
4	42.02%	23.57%	47.69%	56.31%	0.86%
5	36.66%	24.10%	44.00%	52.84%	58.47%

Table 12.6: Performance of the snapshots of the model from a given cycle (row) on the synthetic examples collected in a given cycle (column), for Run 1. For instance, TransCoder from the first cycle (the first row in the table) solves 41.14% of the synthetic tasks created in the same cycle, 1.18% of the tasks created in the second cycle, 0.16% from the third, 0.03% from the fourth, and 0.02% from the fifth cycle.

tasks from S collected in particular cycles (in columns).¹ Similarly to previous results, the table demonstrates overall consistent improvement of the model’s performance. The success rate on S collected in the first cycle (first column) is noticeably high, which we attribute to the fact that in this initial cycle the method produces the simplest, and thus the easiest, synthetic tasks. Compared to them, the rates in the second column tend to be lower, but then experience monotonous increase until the diagonal of the table. This indicates that the knowledge acquired by TransCoder in previous cycles is not forgotten and helps it solve the ‘historical’ tasks even if they are not part of its working training set anymore. The success rates right off the diagonal are much lower, which was expected, as they mark the performance on the synthetic tasks that TransCoder has not been yet trained on.

Figure 12.8 shows examples of generated tasks with solutions, i.e. the (T, p) pairs added to S during training. The left-hand program first fills the scene (which has initially the same dimensions as the input raster) with the red color. Then, it “measures” the height of the input raster by locating its left bottom corner (the LBC function), which will in general have coordinates $(0, y_{\max})$ (the coordinates are 0-based). Finally, it crops the scene to the rectangle that spans from this point/corner to $(1, 1)$. The resulting output raster will thus be a rectangle that is always two pixels wide, while its height is smaller by one than the input raster. The program synthesized for the right-hand task shown in Figure 12.8 also involves cropping of the input raster, but the extent of the cropping rectangle is determined in a more complex fashion, by flood-filling the input raster and locating the first pixel of the resulting region. Examples of programs that create new synthetic problems from subsequent Exploration phases are presented in Table 12.9.

12.3.1 Conclusions and future work

This study corroborated our preliminary findings from J. Bednarek and Krawiec, 2024 and confirmed TransCoder’s capacity to provide itself with a learning gradient by generating synthetic tasks of moderate difficulty, which close the “skill gap” existing between the capabilities of an initial untrained and inexperienced solver and the difficulty of ARC tasks. Crucially, the generative aspect of this architecture, combined with expressing candidate solutions in a DSL, allows the method to obtain concrete DSL programs, use them as concrete targets for the synthesis process (Generator), and so gradually transform an unsupervised learning problem into a supervised one. As evidenced by the experiment, facilitated in this way provides informative learning guidance, unparalleled when compared to we engaged in preliminary experiments.

¹Calculated off-line, after the completion of the run.

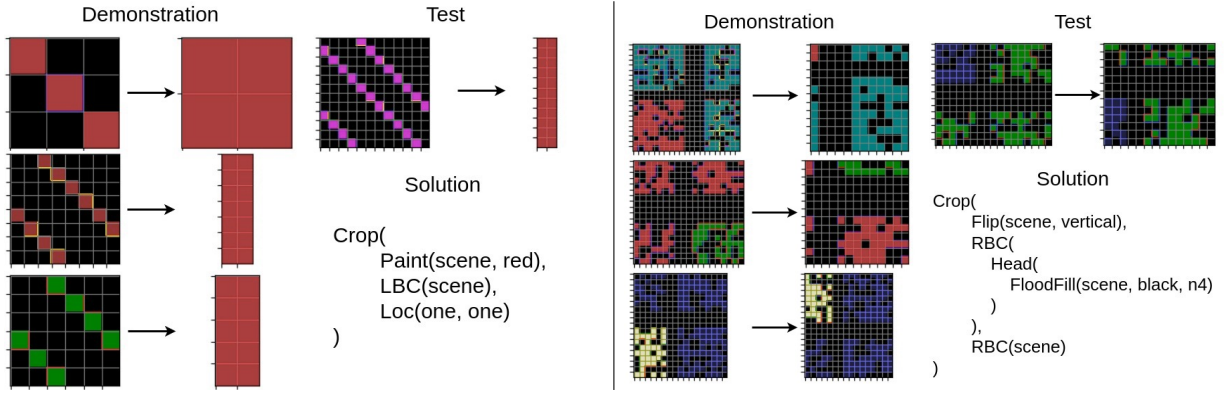


Figure 12.8: Examples of (task, program) pairs synthesized by TransCoder.

In a sense, this research is venturing into the domain of *relational learning* and, by employing deep learning components to this aim, explores the area of *relational deep learning* (see, e.g., Fey et al., 2024). In the current architecture of TransCoder, this is achieved with the quite rudimentary means of variational layer, which allows us to relate, in a many-to-many, bi-directional fashion, the ARC tasks to the DSL programs that solve them. Prospectively, it would be interesting to engage other mechanisms for this purpose, perhaps such that allow discarding the, not necessarily desirable, stochasticity of the variational mapping.

The modularity of the proposed architecture allows TransCoder to be adapted to other types of data than those considered in our DSL and, more generally, than those specific to the realm of ARC tasks. In particular, the Solver and Generator modules are independent of the input data type, while the only type-specific module is Perception. This ensures relatively loose coupling between the method and the DSL, and facilitates introducing changes in the latter, or even replacing it entirely. Future work will include applying the approach to other benchmarks in different domains, developing alternative DSLs, transferring abstraction and reasoning knowledge between datasets, and prioritizing the search in the solution space to solve the original ARC tasks.

12.4 Appendix

12.4.1 Specification of the DSL

The specification of the DSL comprises:

- The list of predefined symbols (constants) in Table 12.7,
- The list of operations (instructions) in Table 12.8.

Table 12.7: The list of predefined symbol keys available in the DSL. The keys *Zero*, *One*, *Horizontal*, *Vertical*, *N4*, *N8*, *Cw*, and *Ccw* represent constant values and are always present in a Workspace. *Colors* are only available if they appear within a given task. *Scene* is a key relative to a specific pair of demonstrations. *FunctionalInput* is a special key used by higher-order functions.

Name	Type	Description
Zero	Int	A constant '0'
One	Int	A constant '1'
Horizontal	Orientation	A categorical value used for the Flip operation
Vertical	Orientation	A categorical value used for the Flip operation
N4	Connectivity	A categorical value used for the FloodFill operation
N8	Connectivity	A categorical value used for the FloodFill operation
Cw	Direction	A categorical value used for the Rotate operation
Ccw	Direction	A categorical value used for the Rotate operation
Black	Color	A categorical value of color from ARC
Blue	Color	A categorical value of color from ARC
Red	Color	A categorical value of color from ARC
Green	Color	A categorical value of color from ARC
Yellow	Color	A categorical value of color from ARC
Grey	Color	A categorical value of color from ARC
Fuchsia	Color	A categorical value of color from ARC
Orange	Color	A categorical value of color from ARC
Teal	Color	A categorical value of color from ARC
Brown	Color	A categorical value of color from ARC
Scene	Region	A Region representing input raster from an input-output demonstration pair
Functional Input	-	A special key available only during execution/generation of functional operation subprogram

Table 12.8: Definitions of the operations available in the TransCoder's DSL.

Name	Signature	Description
Add	(A: Arithmetic, A: Arithmetic) ->A: Arithmetic	Adds two objects of the same type inheriting from Arithmetic
Area	(Region) ->Int	Calculates the surface area of a region
Crop	(Region, Loc, Loc) ->Region	Cuts a subregion based on the top left and bottom right vertices
Deduplicate	(List[A]) ->List[A]	Removes duplicates from the list
Diff	(List[A], List[A]) ->List[A]	Performs a difference operation on two lists
Draw	(Region, Union[Region, List[Region]]) ->Region	Overlays a Region or list of Regions on the given input region
Equals	(A, A) ->Bool	Compares two objects of the same type
Filter	(List[A], (A->Bool)) ->List[A]	Filters the list of input objects based on the result of the subroutine run on each input element
First	(Pair[A, Type]) ->A	Gets the first element of the input pair
Flip	(Region, Orientation) ->Region	Performs a vertical or horizontal flip of the input region
FloodFill	(Region, Color, Connectivity) ->List[Region]	Performs segmentation of the input Region using the Flood Fill algorithm and the background color and neighborhood type (n4 or n8)
GroupBy	(List[A], (A->B)) ->List[Pair[B, List[A]]]	Groups objects from the input list based on the results of the subroutine
Head	(List[A]) ->A	Gets the first element of the input list
Height	(Region) ->Int	Returns the height of the region
Intersection	(List[A], List[A]) ->List[A]	Returns the intersection of two lists
LBC	(Region) ->Loc	Returns the location of the left bottom corner
LTC	(Region) ->Loc	Returns the location of the left top corner
Len	(List[Type]) ->Int	Returns the length of the input list
Line	(Loc, Loc, Color) ->Region	Creates a single-color Region that represents a line between two points
Loc	(Int, Int) ->Loc	Location object constructor
Map	(List[A], (A->B)) ->List[B]	Performs the transformation operation of each element of the input list using a subroutine
MostCommon	(List[A], (A->B)) ->B	Returns the most frequently occurring object from the input list based on the value returned by the subroutine applied to the list elements
Neg	(Union[A, B: Bool]) ->Union[A, B: Bool]	Performs a negation operation on the input object
Paint	(Region, Color) ->Region	Colors the input region a solid color
Pair	(A, B) ->Pair[A, B]	The constructor of an object of type Pair
Pixels	(Region) ->List[Region]	Returns a list of pixels of the input region

Table 12.8: Definitions of the operations available in the TransCoder’s DSL.

Name	Signature	Description
RBC	(Region) -> Loc	Returns the location of the right bottom corner
RTC	(Region) -> Loc	Returns the location of the right top corner
Rect	(Loc, Loc, Color) -> Region	Creates a single-color Region representing a rectangle bounded by the input locations
Reverse	(List[A]) -> List[A]	Reverses the input list
Rotate	(Region, Direction) -> Region	Rotates the Input Region clockwise or counterclockwise
Scale	(Region, Union[Int, Pair[Int, Int]]) -> Region	Scales the input Region according to the integer argument
Second	(Pair[Type, A]) -> A	Returns the second element of the input pair
Shift	(Region, Union[Loc, Pair[Int, Int]]) -> Region	Shifts the input Region by an integer argument
Sort	(List[A], (A -> Comparable) -> List[A]	Sorts the input list based on the result of the subroutine applied to the list
Sub	(A: Arithmetic, A: Arithmetic) -> A: Arithmetic	Subtracts two objects of the same type inheriting from Arithmetic
Tail	(List[A]) -> A	Gets the last element of the input list
Union	(List[A], List[A]) -> List[A]	Returns the sum of two input lists
Width	(Region) -> Int	Returns the width of the region
Zip	(List[A], List[B]) -> List[Pair[A, B]]	Creates a list of pairs of corresponding elements in the input lists

12.4.2 Examples of synthetic data

Examples of programs creating new synthetic problems generated in the subsequent phases of Exploration are presented in Table 12.9.

12.5 Implications

The primary implication of this chapter lies in the implementation of the neuro-symbolic framework for abstract reasoning, demonstrating how the theoretical components of Sensing and Acting can be instantiated in a purely cognitive domain. In this architecture, a symbolic sensing is reflected by the *Perception Module*, which processes a dual input—visual grid-based demonstrations and symbolic specifications—to yield a unified latent representation for the target program, alongside specific neural embeddings for the problem’s symbols. The symbolic acting is constructed from a composite of a *Solver* and a *Program Generator*. These neural modules utilize the learned latent representations to synthesize algorithms within Domain Specific Language (DSL), strictly adhering to syntactic rules and type constraints. Finally, the physical environment is replaced by the *Program Interpreter*, which executes the synthesized code to realize the interaction with the problem state.

Beyond the architecture, the introduction of the *task generation* mechanism directly addresses the objective of automatic problem space construction defined in Section 10.1.3. By leveraging the explicitness of the DSL and the executability of the synthesized logic, the system implements a mechanism for analyzing failures: it identifies synthesis attempts that failed the original objective but produced coherent outputs, re-contextualizing them as valid solutions to new, synthetic problems. This creates a “reflexive learning gradient”, shifting the paradigm from passive fitting to a static distribution (as seen in the previous chapter) to an active, generative exploration of the domain.

In the broader context of the thesis, this chapter serves as the empirical verification of the *structural isomorphism* hypothesis. It proves that the “limits of abstraction” are defined not by the lack of physical reality, but by the agent’s ability to simulate a feedback loop. Just as the robotic agents in Part II learned to walk by interacting with a physics simulator, the TransCoder agent learns to reason by interacting with a “logical simulator” of its own creation. This draws the final parallel of the dissertation: “reasoning” is effectively a generalized form of Acting. This realization confirms that the proposed composite neuro-symbolic architecture is capable of bridging the gap between learning from real data and reasoning about abstract concepts.

Exploration	Synthesized program
1	Crop(scene, RTC(scene), Neg(LBC(scene))) Crop(Rotate(scene, cw), Loc(zero, one), Loc(zero, one)) Crop(scene, Loc(Add(one, one), one), Loc(Width(scene), Height(scene))) Rotate(Crop(scene, Loc(one, one), Loc(one, Sub(one, one))), ccw) Draw(Flip(scene, vertical), Rotate(scene, ccw))
2	Crop(Flip(scene, vertical), Loc(zero, Height(scene)), LTC(scene)) Crop(Flip(scene, vertical), Loc(zero, Height(scene)), LTC(scene)) Crop(Flip(scene, vertical), Loc(one, one), Head(Map(Sort(Pixels(scene), Width(scene)), RBC(scene)))) Crop(Rotate(Flip(scene, horizontal), cw), Loc(one, one), Loc(one, zero)) Crop(Rotate(scene, ccw), Loc(one, one), LBC(scene)) Crop(Rotate(Flip(scene, horizontal), ccw), Loc(Height(scene), zero), RTC(scene))
3	Crop(scene, Loc(Width(scene), Neg(one)), Sub(Loc(Width(scene), zero), LBC(scene))) Rotate(Crop(scene, RBC(scene), Loc(one, one)), CCW) Crop(Paint(scene, black), Loc(one, one), Loc(Add(Width(scene), Width(scene)), Area(scene))) Crop(Rotate(scene, ccw), Loc(Len(Pixels(scene)), one), RBC(scene)) Crop(Flip(scene, vertical), Add(LTC(Crop(scene, RTC(scene), RBC(scene))), RTC(scene)), Loc(zero, one))
4	Crop(Rotate(scene, ccw), Loc(one, one), LBC(scene)) Rotate(Crop(scene, Loc(zero, Height(scene)), LTC(scene)), ccw) Rotate(Crop(scene, Loc(Height(scene), one), RBC(scene)), ccw) Flip(Draw(Scale(scene, Neg(one)), Crop(Scale(Rotate(scene, ccw), Neg(one)), Neg(RBC(scene)), RTC(scene))), vertical) Crop(scene, Loc(one, Add(one, one)), Loc(one, one))
5	Crop(Rotate(Flip(scene, horizontal), ccw), Loc(zero, one), Loc(Add(Width(scene), Width(scene)), Width(scene))) Crop(Rotate(Scale(scene, Neg(one)), ccw), Loc(Height(scene), one), RBC(scene)) Crop(Rotate(Flip(scene, horizontal), ccw), LTC(scene), RTC(scene)) Draw(scene, Rotate(Crop(scene, RTC(Draw(scene, Rotate(scene, cw))), RBC(Head(Pixels(scene)))), ccw)) Crop(scene, Loc(Neg(one), Area(scene)), Loc(Area(scene), Add(one, one)))

Table 12.9: The examples of synthesized programs across successive exploration phases for Run 1.

Chapter 13

Conclusions

13.1 Summary of contributions

This part of the thesis explored the domain of abstract Problem Solving, positing that the architectural principles of embodied agents—specifically the feedback loop between Sensing and Acting—can be successfully elevated to the cognitive domain. The chapters presented here formalized this approach through a progressive evolution of the framework, validating the central hypothesis of structural isomorphism between robotic control and abstract reasoning, introduced in Section 10.1.3 (**H3**).

The first contribution, detailed in Chapter 11, established the architectural skeleton for this translation. We introduced SAPS, an end-to-end neural architecture capable of mapping high-level problem descriptions (NL) directly into executable actions (ASTs). Although this work operated in a supervised regime, it successfully demonstrated the structural precedent: that a purely neural system could be trained to produce syntactically correct symbolic logic without external post-processing. This formalized the concept of a “simple” Sensing-Acting loop where the act of solving is equivalent to the act of synthesis, laying the necessary groundwork for more complex iterative reasoning.

The second contribution, detailed in Chapter 12, provided empirical verification of the hypothesis in the rigorous domain of ARC. We proposed TransCoder, a neuro-symbolic architecture that upgraded the perceptual module to encode visual grid-based demonstrations into latent features, effectively constructing the problem space. Simultaneously, the Acting component was evolved into a generative mechanism capable of navigating this space via Domain Specific Language. Crucially, this work implemented the objective of “automatic construction of the problem space” through the task generation mechanism. By treating failed program executions as valid solutions to synthetic tasks, the system generated its own curriculum, creating a dynamic feedback loop that jointly optimized the neural representation and symbolic strategy.

13.2 Implications and broader context

The research presented in this part has significant implications for the development of general-purpose AI systems. By shifting the output of the decision loop from static classification labels to executable programs, we move towards systems that are inherently interpretable and verifiable. The ability to inspect the synthesized code allows for a level of transparency—revealing the “reasoning steps” of the agent—that is often absent in purely connectionist approaches.

Furthermore, the task generation methodology introduced in Chapter 12 addresses a fundamental bottleneck in modern AI: the reliance on massive, human-annotated datasets. In Section 1.3.4 we established that embodied learning is inherently bounded by the “friction of reality”—the physical constraints of hardware degradation, safety risks, and the unyielding passage of real time. Here, in the cognitive domain, task generation entirely circumvents this friction. We demonstrated that a neuro-symbolic agent operating within a generative abstract space can create its own “reflexive learning gradient” by safely and rapidly exploring

the solution space, formalizing its own execution failures as new training examples. The **frictionless self-simulation** suggests a path forward for autonomous agents that must operate in novel environments where physical trial-and-error is too costly or predefined training data are unavailable. In the broader context of the thesis, this implies that “reasoning” is not a static capability, but a *dynamic process of construction*, where the agent continuously explores its internal symbolic language to better describe and manipulate the perceived reality.

This part addresses the scope outlined by the thesis title: “*Composite neuro-symbolic architectures for learning and reasoning from real and abstract data.*” Although the preceding parts, Sensing and Acting, successfully validated the application of neuro-symbolic ideas to real data—grounding the decision-making loop in the physical constraints of proprioception and locomotion—this part extends the investigation to the domain of abstract data.

Crucially, this part serves as the unification of the thesis narrative, connecting the core ideas from both Sensing and Acting within the framework of neuro-symbolic AI. By introducing the concepts of symbolic sensing and symbolic acting, we shift the operational domain from physical interaction to pure cognition. In doing so, we essentially join the disparate capabilities of perception and intervention for a greater purpose: the resolution of abstract problems. This transition demonstrates that the proposed composite architectures are not artifacts of embodiment, but domain-agnostic mechanisms capable of reasoning about logical structures as effectively as they reason about physical properties.

Furthermore, this part expands the thesis focus from merely operating within a pre-defined environment to the active mechanism of Problem Solving itself. We establish the framework in the context of the problem space theory, but significantly extend this classical concept with the capability of automatic problem space construction. Through the introduced mechanism of task generation—which creates a “reflexive learning gradient” by learning from failures—we show that a neuro-symbolic system can actively build and populate the latent manifold it navigates. This capability confirms the universality of the approach and completes the research arc proposed in this dissertation, bridging the final gap between physical interaction and cognitive abstraction.

Part IV

Thesis summary

Chapter 14

Discussion

14.1 Summary of key findings

This thesis integrates connectionist representation learning and symbolic reasoning through composite neuro-symbolic architectures. Based on the hypothesis of structural isomorphism between embodied control and cognitive decision-making, this research evaluates the mechanisms required to ground abstract symbols across physical and logical domains. Rather than isolated outcomes from robotic control and program synthesis, these findings support a unified framework for machine cognition. This section synthesizes the evidence collected across the three capabilities—Sensing, Acting, and Problem Solving—to verify the hypotheses posed at the outset of each investigation.

14.1.1 Validation of symbolic sensing (Part I)

Part I evaluated how artificial agents ground representations in vision-denied environments, where standard exteroceptive sensors become unreliable. Section 2.1 hypothesized that semantic meaning could be extracted directly from low-level physical interactions:

H1. *We hypothesize that a specialized perceptive module can successfully abstract raw, low-level sensory data acquired during robot-environment interaction into high-level, quantifiable features (e.g. symbolic properties) that are sufficiently meaningful and accurate to be directly utilized by the upstream decision-making and control systems to improve overall task performance.*

This hypothesis was empirically validated through the development and evaluation of the haptic localization (Chapter 4). The experimental results demonstrated two findings regarding the nature of neuro-symbolic sensing:

1. **Abstraction of noisy signals:** We confirmed that deep learning architectures can robustly extract discrete terrain classes (e.g., “gravel”, “sand”, “concrete”) from high-frequency, noisy force-torque signals. The model achieved high classification accuracy even under varying gait patterns, proving that semantic properties are invariant to the specific mechanics of interaction.
2. **Sufficiency of symbolic constraints:** Crucially, the localization experiments verified that these abstract symbols are not merely descriptive labels but functional control states. By integrating these classes into a Monte Carlo Localization framework, the robot was able to maintain a bounded pose estimate in a flat environment. This validates the premise that reducing high-dimensional sensory data to low-dimensional symbolic states does not result in a loss of critical information; rather, it provides the noise invariance necessary for robust operation.

These findings collectively confirm that the symbolic sensing module serves as a bridge between the physical world and the robot’s internal belief state. This interface addresses the symbol grounding problem for non-visual navigation.

14.1.2 Validation of abstract learning for physical policies (Part II)

Part II evaluated the execution domain of Acting, addressing the high computational overhead of controlling agents with complex dynamics—such as soft-robotic end-effectors—using traditional analytical methods. In Section 6.1, we hypothesized that the robustness of analytical control could be preserved while significantly reducing this computational cost:

H2. *We hypothesize that a neural network acting as an “Apprentice” can distill the complex, non-linear decision boundaries of an analytical “Oracle” solely from synthetic data. Furthermore, we posit that a policy learned from this high-level abstract representation captures the essential dynamics of the interaction sufficiently well to allow for direct, zero-shot transfer to the physical world, effectively bridging the gap between analytical rigor and real-time execution.*

Experimental results from the foothold selection and soft-locomotion frameworks (Chapters 7 and 8) validate this hypothesis. This evaluation yielded the following core findings:

1. **Computational compression:** Results show that the decision manifolds of computationally expensive planners, such as Quadratic Programming based Model Predictive Control, can be approximated by a convolutional neural network. The Apprentice policy reduced inference time from the order of seconds to milliseconds. This reduction enabled the real-time control of mechanically adaptive soft feet on-board without requiring explicit online dynamic modeling.
2. **Efficacy of synthetic abstraction:** Policies trained entirely on synthetic elevation maps and simplified physics models successfully controlled the physical ANYmal robot. This validates the premise of “abstract learning”—the neural network generalizes by interacting with the structural constraints of physics provided by the simulator rather than real-world environments. This approach achieved zero-shot transfer, bypassing the sample inefficiency of classical RL on hardware.

These results indicate that generative simulation functions as a viable proxy for physical environments. This baseline allows agents to learn control policies through interactions restricted to simulated physical laws.

14.1.3 Validation of structural isomorphism in reasoning (Part III)

Part III applied this neuro-symbolic framework to abstract Problem Solving. In Section 10.1, we posited the structural isomorphism hypothesis, evaluating whether the mechanisms of abstract reasoning share structural commonalities with embodied control:

H3. *We hypothesize that the challenge of automatic fitting to abstract problems can be resolved by establishing a structural isomorphism between the embodied decision-making loop and the cognitive reasoning process. By elevating mechanical Sensing to symbolic perception and physical Acting to program synthesis, we postulate that a unified neuro-symbolic architecture can utilize program execution traces as a training signal analogous to physical feedback, enabling the joint optimization of problem perception and solution strategy.*

Evaluation of the TransCoder architecture and the task generation mechanism on the Abstract Reasoning Corpus (ARC) (Chapter 12) tests this hypothesis. The empirical results support this structural isomorphism across two areas:

1. **Reasoning as navigation in a latent space:** This framework models the reasoning process as a decision-making task, where the perception module (symbolic sensing) extracts grid features and the solver (symbolic acting) synthesizes a program. The capacity of the TransCoder to solve unseen synthetic ARC tasks indicates that logical reasoning can be framed as a trajectory optimization problem, structurally parallel to the robot path planning evaluated in Part II.
2. **Reflexive learning via execution feedback:** The task generation experiments provided the strongest evidence for the hypothesis. By allowing the agent to generate its own synthetic tasks and learn from execution traces, the system actively navigates the solution space, iteratively approaching the target task by solving a sequence of intermediate problems. This validates that logical simulation (executing code) provides a feedback signal as rich and constraining as physical simulation (physics engine), enabling the agent to self-correct and construct its own curriculum.

These findings synthesize the core themes of this work, indicating that intelligent behavior—whether navigating a sewer or solving a puzzle—emerges from the same neuro-symbolic feedback loop. This loop integrates state perception, (symbolic) action execution, and learning from (simulated) consequences.

14.1.4 Cross-cutting finding: the meta-structural role of constraints

Beyond the domain-specific validations detailed above, this thesis establishes a fundamental cross-cutting finding regarding the role of structural constraints in deep learning systems.

Across all three parts of this research, results demonstrate that *imposing structural inductive biases via symbolic interfaces does not limit the expressivity of neural models. Instead, these constraints stabilize learning within high-dimensional, noisy, or sparse-reward environments.*

This structural role of constraints applies to each contribution:

- In **Sensing** (Part I), we constrained the semantics of the environment (i.e. a map was limited to a defined set of symbols—terrain types) and the perceptive network to predict discrete semantic classes (terrain types), forcing it to translate continuous sensor noise into a stable, low-dimensional state space.
- In **Acting** (Part II), we constrained the policy network to approximate the manifold of an analytical physics solver, forcing it to adhere to the strict kinematic and dynamic limits of the robot.
- In **Problem Solving** (Part III), we constrained the reasoning network to output tokens within a strict Domain Specific Language (DSL), forcing the search process to remain within the bounds of executable logic.

This collective evidence challenges the purely connectionist view that unconstrained “end-to-end” learning from raw data is universally superior. Instead, our results demonstrate that the definition of a formal structure—whether a set of terrain labels, a physics equation, or a grammar—provides the necessary scaffold that allows neural networks to generalize from limited experience. Effectively, we conclude that “meaning” (semantics) is acquired most efficiently when the learning process is bounded by “structure” (syntax or physics), using constraints to bridge the data-efficiency gap.

14.2 Interpretation of results

The findings presented in the preceding section provide empirical validation for the specific neuro-symbolic architectures developed in this work. However, the broader significance of this research emerges when these results are interpreted collectively rather than in isolation. While the domains of haptic localization (Part I), soft-robotic locomotion (Part II), and abstract program synthesis (Part III) appear distinct, our analysis reveals that they are solved by the same underlying mechanisms. In this section, we interpret the experimental evidence through the lens of the leitmotif: that *abstract reasoning is structurally isomorphic to embodied control*. We demonstrate that the success of the proposed methods stems from two fundamental principles: the use of symbolic interfaces to ground neural intuition, and the use of generative simulation to substitute for physical experience.

14.2.1 The isomorphism of embodied and cognitive navigation

The strongest theoretical interpretation of our findings is the confirmation that cognitive reasoning can be effectively modeled as a form of embodied navigation. By juxtaposing the results of Part II and Part III, we observe a strict structural parallelism between the two domains.

In the robotic domain, the agent perceives a physical state (e.g., terrain geometry), selects a motor action (e.g., a foothold), and receives feedback from the environment (e.g., stability or slip). In the cognitive domain, as evidenced by the TransCoder architecture, the agent perceives an abstract state (e.g., a grid configuration; a set of predefined symbols), selects a symbolic action (e.g., a DSL-guided program), and receives feedback from the interpreter (e.g., a valid output or an error). In both cases, the complete execution of the decision loop serves as the starting point for the next iteration. Thus, both approaches implement the search of the problem space using “steps”—either physical or abstract. Both scenarios are aimed by the ultimate goals, respectively, reaching a position on the map and solving an abstract reasoning task.

Our results imply that these processes are not merely analogous but isomorphic. The success of the task generation mechanism can be interpreted as logical proprioception. Just as a robot relies on ground reaction forces to correct its physical trajectory, the reasoning agent relies on execution traces to correct its logical trajectory. This re-frames program synthesis not as a language translation task, but as a *trajectory optimization* task in the problem’s latent space. Consequently, we interpret the intelligence exhibited by the system not as a high-level mystical process, but as a standard control loop applied to a non-physical manifold.

14.2.2 The efficacy of symbolic interface as inductive biases

A critical question in current AI research is why neuro-symbolic architectures should be preferred over purely connectionist “end-to-end” models. Our results provide a clear interpretation: symbolic interfaces serve as necessary inductive biases that reduce the dimensionality of the learning problem.

In the domain of Sensing, the raw haptic data stream is continuous and noisy. By forcing the network to bottleneck this information into a discrete symbolic state (e.g., “sand”), we effectively collapse the state space, leading to the opportunity to use these symbols to describe the world surrounding the agent (e.g. a map). We interpret the success of the haptic particle filter not merely as a classification success, but as a validation that *semantic abstraction* is

the preferred method for handling sensor noise. The symbol acts as a stable anchor in a chaotic signal.

Similarly, in the domain of Problem Solving, the space of all possible computer programs is combinatorially infinite. By restricting the agent to a Domain Specific Language (DSL), we impose a structural constraint that makes the search for a solution tractable.

Consequently, we interpret the function of the neural network in our architecture not as the creator of logic, but as the *intuition engine* that navigates a pre-defined logical structure. The DL component provides the pattern-matching capability to guess the next move, while the symbolic interface provides the “rails” that ensure the move is valid.

14.2.3 Generative simulation as a substitute for experience

Across Acting and Problem Solving capabilities, this research relies on synthetic data to address sparse real-world sampling. These outcomes validate **generative simulation** as a functional substitute for physical or human experience.

In Part II, Oracle (a physics simulator) generated the training distribution for the robot. In Part III, Task Generator (a logic engine) generated the training distribution for the reasoning agent. In both cases, the neural network never observed the “real” test distribution (the physical ANYmal robot or the hidden ARC evaluation set) during training.

We interpret the observed zero-shot transfer capabilities not as a miraculous generalization, but as the result of *intensive internal rehearsal*. Because the agent possesses a robust internal model—whether it is the equations of motion or the semantics of a DSL—it can simulate millions of interactions rapidly. This allows the agent to explore failure modes and refine its policy in a safe, synthetic environment. Consequently, we conclude that intelligent agents do not strictly require massive external datasets to learn; if provided with a high-fidelity simulator/verifier, they can generate their own experience, effectively “dreaming” solutions to novel problems before encountering them in reality.

14.3 Discussion of implications

The results presented in this thesis have significant implications that extend beyond the performance metrics on haptic localization or abstract reasoning benchmarks. By demonstrating that robust, data-efficient learning emerges from the interplay between neural intuition and symbolic constraints, this work challenges the prevailing trend of purely data-driven, end-to-end optimization. Our findings suggest a paradigm shift in the design of intelligent systems: moving away from the “tabula rasa” assumption toward architectures that leverage structural priors and generative simulation as foundational elements. The following sections discuss how this shift impacts the fields of neuro-symbolic AI, Robotics, and the broader pursuit of General Artificial Intelligence.

14.3.1 Implications for neuro-symbolic AI: the role of structure

A significant implication of this research concerns the architectural philosophy of artificial intelligence. In an era dominated by large-scale, end-to-end models that learn representations from massive, unstructured datasets, our findings argue for the critical utility of formal structure. We posit that symbolic interfaces are not merely useful constraints, but necessary prerequisites for overcoming the “data wall”—the finite limit of high-quality human-generated data.

Structure as a catalyst, not a bottleneck. Historically, manual feature engineering or hand-crafted logic was viewed as a bottleneck to scalability. However, our results across both physical and abstract domains imply that structure (e.g. symbols, rules, constraints) acts as a powerful catalyst when assessed in the context of an entire system. It imposes a strong inductive bias that drastically reduces the search space, allowing neural networks to learn high-fidelity policies from sparse data. This implies that future neuro-symbolic research should focus less on how to remove symbols to make systems fully differentiable, and more on how to design expressive symbolic interfaces that neural networks can easily manipulate.

The emergence of reflexive learning. Perhaps the most profound implication stems from the *task generation* results in Part III. We demonstrated that when an agent operates within a structured language (code; including construction rules), it gains the ability to verify its own outputs through execution. These characteristics support the paradigm of **reflexive learning**: systems that improve not by observing more data, but by reflecting on the validity of their own actions. This suggests that the path to robust AI lies in architectures that can generate, execute, and debug their own behavior, effectively closing the loop without human intervention.

This thesis advocates a transition from “Big Data” toward “Better Definitions.” The outcomes indicate that defining the problem space—via the DSL or a physics model—can be as critical to system performance as the architecture of the neural learner itself.

14.3.2 Implications for robotics: autonomy in extreme environments

Within robotics, the results primarily concern the design of autonomous agents operating in extreme or resource-constrained environments.

The primacy of proprioception in vision-denied settings. Current robotic systems rely primarily on exteroceptive sensing, such as LiDAR and cameras. While effective in structured settings, this dependency introduces vulnerability in optically compromised environments—such as smoke-filled disaster zones, subterranean sewer networks, or deep mines. The performance of the haptic particle filter (Part I) indicates that *proprioception need not be a secondary fallback modality*. Integrating haptic feedback into a symbolic framework (symbolic sensing) allows robots to localize and reason about their surroundings using touch alone, provided domain knowledge such as an environment map is available. These outcomes support the viability of non-visual autonomy, allowing hardware to operate robustly when visual sensors are compromised.

The democratization of agile control. Furthermore, the *Oracle-Apprentice framework* (Part II) addresses the hardware-compute trade-off limiting soft and morphological robotics. Controlling complex non-linear bodies conventionally requires computationally intensive analytical solvers, necessitating limited operation or substantial onboard computing. This reliance increases hardware complexity and potential points of failure. These outcomes indicate that the computational burden can be compressed into lightweight neural networks while maintaining physical robustness. This compression supports an *embedded intelligence*—the deployment of high-frequency, adaptive control policies directly on low-power microcontrollers. This architecture expands the utility of high-fidelity control to resource-constrained, agile hardware platforms.

14.3.3 Theoretical implications: grounding symbols in action

As established by Searle’s “Chinese Room” argument, a computer that processes symbols based solely on their shape (syntax) possesses no understanding of their content (semantics); it is effectively following a rulebook for a language it does not speak (Searle, 1990). Harnad later formalized this as the “dictionary loop” problem: defining a symbol using only other symbols is circular, like trying to learn a language from a dictionary written entirely in that language (Harnad, 1999). To break this loop, symbols must be grounded in the agent’s experience.

Our findings extend the empirical work of Glenberg and Kaschak, who demonstrated that humans understand language by unconsciously simulating the associated physical actions (Glenberg & Kaschak, 2002). We argue that this principle applies to artificial agents. In our framework, a symbol is grounded not by a static label, but by the *predictive consequences of action*. Just as a human understands the phrase “open the drawer” by simulating the arm’s motion, our agent understands a terrain class or a code token by simulating the feedback that will result from executing it. Meaning is therefore defined as the capacity to predict the outcome of an action.

In our unified framework, the meaning of a symbol is defined strictly by its operational role in the agent’s feedback loop:

- **Physical domain:** In Part I, a terrain class is not grounded in a visual label, but in the specific dynamic relationship between motor commands and sensory feedback. A symbol such as “sand” is characterized by the slippage and resistance the actuators experience when interacting with it. Therefore, the symbol represents an *affordance*: it is a compact encoding of the permissible control policies required to traverse that region.
- **Abstract domain:** In Part III, the meaning of DSL symbol is not fixed but is fluidly determined by two layers of context:
 1. *Global task:* A static attribute like `Black` has no intrinsic semantic value; it acquires meaning only relative to the task goal (e.g., functioning as “background” in one task versus “noise” in another).
 2. *Local action:* An operator’s meaning shifts based on its position in the program. For instance, `Width` may represent a static property when applied to an input grid, but transforms into a dynamic parameter (e.g., a shift magnitude of an object on a canvas) when applied to the result of an intermediate calculation. Thus, abstract meaning is emergent from the *sequential interaction* of operators.

14.4 Strengths and limitations of the study

The empirical results presented in the preceding chapters validates composite neuro-symbolic architectures in solving complex tasks across both physical and abstract domains. While the proposed framework successfully addresses the challenges of data scarcity and computational efficiency, its applicability is bounded by specific design choices—most notably the reliance on structural priors such as semantic maps, physics simulators, and domain-specific languages. This section outlines the strengths of the research design while transparently acknowledging the limitations that define the current boundaries of the approach.

14.4.1 Strengths of the research design

The research design in this thesis features three distinct properties that contrast with conventional approaches in robotics and machine learning.

Cross-domain unification. A primary characteristic of this study is the empirical evaluation of a single conceptual framework—combining symbolic interfaces with neural perception—across divergent domains. Validating the same core architecture on physical hardware (ANYmal robot) and abstract logical benchmarks (ARC) supports the proposed structural isomorphism. These outcomes indicate the framework generalizes effectively across modalities rather than serving as a narrow application.

Systemic integration of semantics. A distinct strength of the proposed architecture is that it treats symbols as integrated functional components of the entire system. In Part I, the terrain classes act as a shared prior on the semantic map—understood by human operators yet interpreted by the robot through direct physical interaction. Similarly, in Part III, symbols are not static tokens but are dynamically contextualized by the predicted consequences of their execution. This design forces the neural networks to look beyond local pattern recognition and instead “adjust” their representations to align with the global semantics of the system, effectively grounding low-level inference in high-level system constraints, in a manner consistent with human learning and understanding.

Independence from human annotation. The research design also addresses the “Data Scarcity” bottleneck common in deep learning architectures. Leveraging *generative simulation*—via a physics engine (Part II) or a task generation mechanism (Part III)—enables self-supervised learning. These properties stabilize performance when human-annotated data is constrained or unavailable. Consequently, this shift moves the burden of supervision from manual labeling to automated verifiers.

14.4.2 Limitations and boundaries

The proposed framework, however, exhibits specific limitations that bound its current applicability.

Dependence on accurate priors and interfaces. The primary limitation of this research is its dependency on the quality of the expert-designed symbolic interface. Unlike end-to-end approaches that learn representations directly from data, this framework assumes a valid physics model (Part II) or an expressive DSL (Part III). Misspecified or incomplete structural priors bound the agent’s performance. The system cannot “invent” new physical or logical operators outside of its provided grammar; it only optimizes within the parameter space defined by the designer.

The persistence of the sim-to-real gap. Although Oracle-Apprentice framework significantly improves robustness, it does not completely eliminate the “sim-to-real” gap. The success of the transfer relies on the assumption that the simulation captures the *dominant* dynamics of the real world. In scenarios where physical phenomena are extremely difficult to model analytically—such as the complex non-linear friction of soft materials or chaotic fluid dynamics in a sewer—the policies learned from the simulator may still experience performance degradation. **Apprentice is only as wise as Oracle allows it to be.**

Engineering complexity of generative environments. Finally, shifting the paradigm from “data collection” to “simulation design” introduces a new form of engineering overhead. Building a high-fidelity haptic simulator or a task generation engine capable of creating a curriculum requires significant domain expertise and development time. In domains where data is abundant and cheap (e.g., standard computer vision tasks), the engineering effort required to build these neuro-symbolic scaffolds may outweigh the benefits compared to standard supervised learning. Thus, this approach is most justified in “frontier” domains where data is the limiting factor, such as the ARC challenge.

14.5 Directions for future research

The research conducted in this thesis has shown a foundation for the design of composite neuro-symbolic agents. However, it has also highlighted the boundaries of what is achievable with fixed structural priors. Although we have successfully demonstrated that given a valid physics model or DSL an agent can explore a defined environment, the origins of these structures remain rooted in human engineering. Consequently, the most promising directions for future research lie in relaxing these constraints—moving from systems that operate within a fixed symbolic scaffold to systems capable of discovering, refining, and adapting that scaffold autonomously. The following avenues represent the next logical steps in the evolution of this framework.

14.5.1 Automated discovery of symbolic primitives (DSL learning)

Although this thesis successfully demonstrated the power of reasoning within a structured DSL, it remained a fixed artifact of human engineering. A critical direction for future research is the development of mechanisms for the **automated discovery of symbolic primitives**. Currently, the “vocabulary” of the agent (e.g., specific grid operations) must be defined a priori. Future systems should possess the capability to identify repeating sub-routines in their own successful solutions and compress them into new, reusable primitives—a process often referred to as “library learning.”

Cognitive domain: library abstraction. In the context of program synthesis, this aligns with the principles established by the DreamCoder framework (Ellis et al., 2021). Future iterations of the TransCoder architecture could implement a “sleep phase”, where the agent analyzes the programs it generated during the “wake phase” to abstract common patterns into higher-order functions. This would allow the agent to evolve its own DSL, adapting its language to the specific distribution of tasks it encounters, effectively growing its own conceptual understanding over time (Grand et al., 2024).

Physical Domain: automated oracle discovery. In the embodied domain, a significant limitation of the current framework is its reliance on pre-existing analytical “Oracles” (e.g., the foothold score). Although we successfully leveraged a known physical model for foothold selection, many robotic tasks involve dynamics that are either unknown or too complex to model analytically. Future research should therefore leverage techniques from *Symbolic Regression* and *Equation Discovery* (Brunton et al., 2016; Schmidt & Lipson, 2009) to automate the creation of these “Oracles”. Rather than depending on a human engineer to derive a cost function, an agent could observe empirical interactions to synthesize its own governing equations. This would empower the system to “discover” new “Oracles” for complex

phenomena—such as non-linear soft-body deformation—where no closed-form analytical solution currently exists.

14.5.2 Online adaptation and continual learning

A critical limitation of the current Oracle-Apprentice framework is its offline nature; the policy is distilled from a fixed simulator and deployed as a static inference model. A vital direction for future research is the transition to **online adaptation**, enabling the agent to refine its internal models and policies in real-time as it interacts with the physical world.

Closing the reality gap in real-time. Although our approach used a robust “one-size-fits-all” policy, true autonomy requires the ability to adapt to changes in the robot’s own morphology (e.g., damage, wear) or drastic environmental shifts. Future work could integrate *meta-learning* or *Rapid Motor Adaptation* (RMA) techniques (Kumar et al., 2021), where the agent infers a latent “environment vector” from recent history and conditions its policy on this context. This would allow the symbolic interface to be dynamic variables estimated on-the-fly rather than static constants.

Self-supervised continual learning. Furthermore, the reliance on a pre-defined simulator can be relaxed by incorporating *Self-Supervised Learning* directly on the hardware. By treating the discrepancy between the “predicted state” (from the internal model) and the “measured state” (from sensors) as a training signal, the agent can continuously fine-tune its inverse dynamics model. This aligns with recent advances in *continual learning* for robotics (Lesort et al., 2019), ensuring that the agent does not suffer from catastrophic forgetting but accumulates physical wisdom over its operational lifespan. Ultimately, this leads to systems where the “training phase” never ends, but is co-extensive with the robot’s existence.

14.5.3 Closing the loop: from abstract to physical and back

Finally, the ultimate goal of this research trajectory is to close the loop between high-level abstract reasoning and low-level physical interaction. Although this thesis demonstrated that these domains share *structural isomorphism*, they were investigated in isolation. Future research must focus on integrated architectures where the flow of information is bidirectional: abstract plans guiding physical actions, and physical feedback refining abstract models.

Semantic planning via human-readable abstractions. The recent emergence of Vision-Language-Action (VLA) models, such as *PaLM-E* (Driess et al., 2023) and *RT-2* (Brohan et al., 2023), validates the potential of using high-level semantic abstractions to drive embodied control. A critical opportunity for future work lies in utilizing the **human-readable symbolic representations** developed in this thesis—specifically the semantic terrain maps—as a direct interface for these foundation models. Unlike opaque latent vectors, our symbolic primitives effectively share a vocabulary with natural language. This alignment allows VLA models to extend their reasoning capabilities directly into the physical world: a command such as “avoid the slippery patch” becomes executable not through vague approximation, but by querying the robot’s internal map for specific, linguistically grounded symbols, effectively allowing the model to reason about physical properties as if they were text.

Bottom-up: physical failure as reflexive learning. Conversely, the loop must be closed from the bottom up to enable true **reflexive learning**. In this context, a physical failure is defined as a measurable divergence between the expected state (e.g., the target goal) and the

actual state achieved by the robot. Rather than simply discarding these “execution errors,” the agent should treat them analogously to the synthetic tasks generated in Part III. If a robot intends to walk forward but moves in a different direction (e.g. walking on an inclined plane), this discrepancy becomes a new data point: a valid demonstration of how to generate motion in that specific scenario. By retrospectively treating these unintended outcomes as new “synthetic tasks,” the agent can learn to operate in new environments by probing the world, without explicit human supervision, effectively transforming every failure into a successful lesson.

Chapter 15

Conclusions

This thesis aimed to explore the boundaries between embodied interaction and abstract cognition, challenging the traditional dichotomy that separates the physical intelligence of a robot from the logical intelligence of a reasoning engine. Throughout the preceding chapters, we have traversed the landscape of intelligent agency—from the granular, noisy reality of haptic sensing (Part I), through the dynamic complexity of legged locomotion (Part II), to the structured domain of program synthesis (Part III). Although these investigations spanned distinct modalities, they were united by a singular scientific objective: to demonstrate that the mechanisms of robust learning are structurally isomorphic across physical and logical domains. This final chapter synthesizes the outcomes of this journey, restating the core research problem, summarizing the unified solution, and defining the specific contributions this work leaves to the scientific community.

15.1 Restatement of the research problem and aims

The research presented in this thesis was motivated by a fundamental dichotomy that has long characterized the field of artificial intelligence (AI): the divide between *connectionist* approaches, which excel at pattern recognition in noisy, unstructured environments, and *symbolic* approaches, which provide the rigor of logical reasoning and physical consistency.

Although Deep Learning has achieved unprecedented success in perceptual tasks, it has historically struggled with data efficiency, interpretability, and adherence to strict constraints. Conversely, classical symbolic methods—such as analytical physics solvers or formal logic—offer precision but lack the robustness required to operate in the chaotic noise of the real world. This thesis addressed the challenge of bridging this gap through composite neuro-symbolic architectures, specifically targeting three critical limitations in the state of the art:

1. **The semantic gap in Sensing:** The inability of standard robotic sensors to extract discrete, task-relevant states (such as terrain types) from continuous haptic feedback in vision-denied environments.
2. **The computational gap in Acting:** The high computational cost of analytical physics solvers for complex morphologies (such as soft robotics), limiting the deployment of advanced control policies on real-time hardware.
3. **The expressiveness gap in reasoning:** The lack of structured symbolic tools necessary to traverse the combinatorial problem space. This deficit requires large human-annotated datasets to approximate reasoning through pattern matching, which restricts self-correction and zero-shot generalization.

This research investigates the hypothesis of **structural isomorphism** between these domains. The results indicate that grounding physical signals into robot actions requires a mechanism mathematically distinct from, yet structurally identical to, the process mapping visual abstractions into logical programs. Ultimately, this thesis evaluates whether a unified framework—leveraging symbolic interfaces and generative simulation—can mitigate data scarcity and robustness limitations across physical and cognitive environments.

15.2 Synthesis of the thesis

This thesis progressively increases in abstraction, moving from physical robotic sensing to logical problem-solving. Although organized into three distinct investigations, these components jointly outline the development of an integrated autonomous agent.

From physical noise to semantic symbols (Part I). The synthesis begins within Sensing, addressing agent localization in vision-denied environments. Part I shows that high-dimensional, noisy signals from physical interaction (force/torque data) contain latent symbolic structures. The haptic particle filter demonstrates how a neural network can extract discrete semantic classes—such as “gravel” or “sand”—from continuous signals and use them in specific task of localization. This process translates unstructured physical data into a “symbolic map,” grounding subsequent decision-making.

From abstract laws to physical agility (Part II). Following the processing of physical data, Part II addresses Acting. Here, symbolic integration occurs via abstract policy learning. The Oracle-Apprentice framework synthesizes locomotion policies for complex morphologies, including soft-robotic feet. This architecture trains a neural network to approximate a computationally expensive analytical solver (Oracle). These outcomes indicate that a reactive policy can “distill” physical dynamics into real-time control. By utilizing the “synthetic experience” of a physics engine, the agent circumvents the need for extensive physical trial-and-error.

From embodied control to cognitive reasoning (Part III). Part III extends the neuro-symbolic paradigm to Problem Solving, evaluating whether cognitive reasoning is structurally isomorphic to embodied control. The TransCoder architecture treats program synthesis as a navigation problem within a latent space. These outcomes indicate that the execution “feedback loop” of a generated program is mathematically analogous to the proprioceptive feedback of a physical limb. Through task generation, the system circumvents reliance on static datasets by generating and solving synthetic tasks. This mechanism mirrors the simulation-based learning established in Part II.

Unified perspective. These three components jointly outline an integrated model of intelligent agency. Equipping the agent with symbolic sensing and symbolic acting enables dual processes of discovery and grounding. The agent uses these mechanisms to map the problem space, applying structured interfaces to navigate potential solutions. The system then evaluates these symbols through their *predictive consequences* rather than relying on static syntax. While these functionalities operate concurrently within the Problem Solving domain (Part III), they derive directly from the distinct embodied architectures in Parts I and II. This alignment suggests that abstract reasoning structurally parallels physical interaction.

15.3 Statement of contribution and significance

This research makes several contributions to the intersection of robotics, machine learning, and neuro-symbolic artificial intelligence. These contributions are significant not only for their immediate technical utility but for the broader methodological shift they represent—moving from purely data-driven to structure-driven learning. The primary contributions are:

15.3.1 A framework for symbolic haptic perception (Part I)

This thesis contributes the haptic particle filter, a novel probabilistic framework for state estimation in vision-denied environments. Crucially, it marks the successful introduction of a neural-based approach for terrain classification directly from proprioceptive sensors (force/torque).

Significance. Unlike traditional proprioceptive methods that rely on raw signal thresholds, this framework introduces a semantic layer that classifies terrain types in real-time. This enables robots to localize and navigate based solely on touch, significantly enhancing autonomy in optically compromised settings such as disaster zones. As pioneer work in this domain, it validates the efficacy of deep learning for haptic interpretation, opening wide possibilities for extending and inventing new neural architectures specialized for proprioceptive sensing.

15.3.2 The Oracle-Apprentice framework distillation method for soft robotics (Part II)

We presented the approach for controlling complex, non-linear morphologies by distilling analytical Oracles into neural Apprentices.

Significance. This method demonstrates a solution to the computational bottleneck in soft robotics. It shows the deployment of sophisticated, physics-grounded control policies on resource-constrained embedded hardware, bridging the gap between the accuracy of model predictive control (MPC) and the speed of neural inference.

15.3.3 The TransCoder and logical proprioception architecture (Part III)

Within abstract reasoning, this work introduces the TransCoder architecture and the concept of logical proprioception. Rather than a standard sequence model, this **Transformer-based architecture** integrates directly with a DSL to embed strict syntactic constraints, rule adherence, and dynamic type checking into the generation process.

Significance. This architecture shows that reasoning tasks (ARC) can be addressed without large datasets by using the execution trace of generated code as a feedback signal. Theoretically, this approach addresses the symbol grounding problem in AI by grounding abstract symbols in context—either through the *predictive consequence* of their execution (action context) or through their relation to a specific *task definition* (e.g., the semantic meaning of a color). These findings suggest that code execution functions as a structure-preserving grounding mechanism for abstract logic.

15.3.4 Validation of generative simulation as a data substitute

Across these domains, this thesis empirically validates generative simulation—via task generation and a physics engine—as a practical substitute for large-scale human annotation.

Significance. By demonstrating zero-shot transfer in both robotic locomotion and program synthesis, this work provides an alternative to the “Big Data” paradigm. These results indicate that agents equipped with a valid internal simulator can self-generate the data required to operate within a domain. This mechanism reduces reliance on external human supervision during the primary learning phase.

15.4 Final concluding remarks

The future of artificial intelligence does not lie in discarding the data-rich foundation models of today, but in grounding them within the well-structured methodologies of human cognition. We view the trajectory of AI as an evolution of a tool that is beginning to function within realities familiar to us.

The progression of this research—from physical haptic feedback to the abstract syntax of computer programs—indicates that subsequent architectural developments depend on converging structural methodologies. This interface combines the high-dimensional pattern matching of neural networks with formal frameworks of **problem spaces**, **symbol grounding**, and **abstract reasoning**.

Our findings indicate that the successful integration of AI into daily human activity depends fundamentally on cognitive alignment; to establish the trust necessary for sustained collaboration, artificial systems must adopt reasoning patterns compatible with our own. Furthermore, this evolution addresses a critical technical bottleneck: current purely data-driven methodologies are inherently constrained by the boundaries of their training distributions. To transcend these empirical limits, AI must move beyond statistical pattern recognition. By embracing symbolic structures, interpreting the environment through causal relationships, and constructing internal models to traverse solution spaces, future systems can move past data-distribution dependencies and learn dynamically from the friction of their own mistakes.

We conclude that the ultimate promise of neuro-symbolic AI is the creation of agents that combine the intuitive perception of System 1 with the deliberate, logical rigor of System 2. By showing the structural isomorphism between physical embodiment and abstract thought, this thesis provides the blueprint for machines that do not just process information, but truly inhabit the physical and logical worlds defined by human experience.

Bibliography

- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., ... Zheng, X. (2016). Tensorflow: A system for large-scale machine learning. *CoRR*, *abs/1605.08695*, 265–283. <https://doi.org/10.48550/arxiv.1605.08695>
- Alford, S., Gandhi, A., Rangamani, A., Banburski, A., Wang, T., Dandekar, S., Chin, J., Poggio, T. A., & Chin, P. (2021). Neural-guided, bidirectional program search for abstraction and reasoning. *CoRR*, *abs/2110.11536*. <https://doi.org/10.48550/arxiv.2110.11536>
- Aljalbout, E., Golkov, V., Siddiqui, Y., & Cremers, D. (2018). Clustering with deep learning: Taxonomy and new methods. *CoRR*, *abs/1801.07648*. <https://doi.org/10.48550/arxiv.1801.07648>
- Alvarez-Melis, D., & Jaakkola, T. S. (2017). Tree-structured decoding with doubly-recurrent neural networks. *International Conference on Learning Representations*. <https://openreview.net/forum?id=HkYhZDqyg>
- Assouel, R., Rodriguez, P., Taslakian, P., Vazquez, D., & Bengio, Y. (2022). Object-centric compositional imagination for visual abstract reasoning. *ICLR2022 Workshop on the Elements of Reasoning: Objects, Structure and Causality*. <https://openreview.net/forum?id=rCzfIruU5x5>
- ATI Industrial Automation. (2026). Mini45 Force/Torque Sensor Specifications.
- Ba, J., Kiros, R., & Hinton, G. E. (2016). Layer normalization. *CoRR*, *abs/1607.06450*.
- Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, *abs/1409.0473*.
- Bajcsy, R. (1988). Active perception. *Proceedings of the IEEE*, 76, 966–1005. <https://doi.org/10.1109/5.5968>
- Balog, M., Gaunt, A. L., Brockschmidt, M., Nowozin, S., & Tarlow, D. (2016). Deepcoder: Learning to write programs. *CoRR*, *abs/1611.01989*.
- Barasuol, V., Camurri, M., Bazeille, S., Caldwell, D. G., & Semini, C. (2015). Reactive trotting with foot placement corrections through visual pattern classification. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5734–5741. <https://doi.org/10.1109/IROS.2015.7354191>
- Barsalou, L. W. (1999). Perceptual symbol systems. *Behavioral and Brain Sciences*, 22(4), 577–660. <https://doi.org/10.1017/S0140525X99002149>
- Bednarek, J., Bednarek, M., Kicki, P., & Walas, K. (2019). Robotic touch: Classification of materials for manipulation and walking. *IEEE International Conference on Soft Robotics (RoboSoft)*, 527–533. <https://doi.org/10.1109/ROBOSOFT.2019.8722819>
- Bednarek, J., Bednarek, M., Wellhausen, L., Hutter, M., & Walas, K. (2019). What am i touching? learning to classify terrain via haptic sensing. *IEEE International Conference on Robotics and Automation (ICRA)*, 7187–7193. <https://doi.org/10.1109/ICRA.2019.8794478>
- Bednarek, J., & Krawiec, K. (2024). Learning to solve abstract reasoning problems with neurosymbolic program synthesis and task generation. In T. R. Besold, A. d’Avila Garcez, E. Jimenez-Ruiz, R. Confalonieri, P. Madhyastha, & B. Wagner (Eds.), *Neural-symbolic learning and reasoning* (pp. 386–402). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-71167-1_21

- Bednarek, J., & Krawiec, K. (2026). Solving Abstract Reasoning Problems with Neurosymbolic Program Synthesis and Task Generation. *Neurosymbolic Artificial Intelligence*. <https://doi.org/10.1177/29498732261419309>
- Bednarek, J., Maalouf, N., Pollayil, M. J., Garabini, M., Catalano, M. G., Grioli, G., & Belter, D. (2020). Cnn-based foothold selection for mechanically adaptive soft foot. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 10225–10232. <https://doi.org/10.1109/IROS45743.2020.9340910>
- Bednarek, J., Piaskowski, K., & Bednarek, M. (2018). Methods of enriching the flow of information in the real-time semantic segmentation using deep neural networks. *2018 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, 163–167. <https://doi.org/10.23919/SPA.2018.8563422>
- Bednarek, J., Piaskowski, K., & Krawiec, K. (2018). Ain't nobody got time for coding: Structure-aware program synthesis from natural language. *CoRR*, *abs/1810.09717*. <https://doi.org/10.48550/arxiv.1810.09717>
- Bednarek, M., Kicki, P., Bednarek, J., & Walas, K. (2021). Gaining a sense of touch object stiffness estimation using a soft gripper and neural networks. *10(1)*. <https://doi.org/10.3390/electronics10010096>
- Bednarek, M., Łysakowski, M., Bednarek, J., Nowicki, M. R., & Walas, K. (2021). Fast haptic terrain classification for legged robots using transformer. *2021 European Conference on Mobile Robots (ECMR)*, 1–7. <https://doi.org/10.1109/ECMR50962.2021.9568808>
- Belter, D., Bednarek, J., Lin, H.-C., Xin, G., & Mistry, M. (2019). Single-shot foothold selection and constraint evaluation for quadruped locomotion. *2019 International Conference on Robotics and Automation (ICRA)*, 7441–7447. <https://doi.org/10.1109/ICRA.2019.8793801>
- Belter, D., Łabecki, P., Fankhauser, P., & Siegwart, R. (2016). Rgb-d terrain perception and dense mapping for legged robots. *Int. J. Appl. Math. Comput. Sci.*, *26(1)*, 81–97. <https://doi.org/10.1515/amcs-2016-0006>
- Belter, D., Łabźcki, P., & Skrzypczyński, P. (2016). Adaptive motion planning for autonomous rough terrain traversal with a walking robot. *Journal of Field Robotics*, *33(3)*, 337–370. <https://doi.org/10.1002/rob.21610>
- Belter, D., & Skrzypczyński, P. (2011). Rough terrain mapping and classification for foothold selection in a walking robot. *Journal of Field Rob.*, *28(4)*, 497–528. <https://doi.org/10.1002/rob.20397>
- Belter, D., Wietrzykowski, J., & Skrzypczyński, P. (2018). Employing natural terrain semantics in motion planning for a multi-legged robot. *Journal of Intelligent & Robotic Systems*. <https://doi.org/10.1007/s10846-018-0865-x>
- Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 610–623. <https://doi.org/10.1145/3442188.3445922>
- Bloesch, M., Burri, M., Sommer, H., Siegwart, R., & Hutter, M. (2018). The Two-State Implicit Filter Recursive Estimation for Mobile Robots. *IEEE Robotics and Automation Letters*, *3(1)*, 573–580. <https://doi.org/10.1109/LRA.2017.2776340>
- Bongard, M. M. (1967). The problem of recognition. *M.: Nauka*.
- Bosworth, W., Whitney, J., Kim, S., & Hogan, N. (2016). Robot locomotion on hard and soft ground: Measuring stability and ground properties in-situ. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 3582–3589. <https://doi.org/10.1109/ICRA.2016.7487541>
- Brandão, M., Shiguematsu, Y. M., Hashimoto, K., & Takanishi, A. (2016). Material recognition cnns and hierarchical planning for biped robot locomotion on slippery terrain. *2016*

- IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, 81–88. <https://doi.org/10.1109/HUMANOIDS.2016.7803258>
- Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Chen, X., Choromanski, K., Ding, T., Driess, D., Dubey, A., Finn, C., Florence, P., Fu, C., Arenas, M. G., Gopalakrishnan, K., Han, K., Hausman, K., Herzog, A., Hsu, J., Ichter, B., ... Zitkovich, B. (2023). Rt-2: Vision-language-action models transfer web knowledge to robotic control. <https://doi.org/10.48550/arxiv.2307.15818>
- Brunton, S. L., Proctor, J. L., & Kutz, J. N. (2016). Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15), 3932–3937. <https://doi.org/10.1073/pnas.1517384113>
- Buchanan, R., Bednarek, J., Camurri, M., Nowicki, M. R., Walas, K., & Fallon, M. (2021). Navigating by touch: Haptic monte carlo localization via geometric sensing and terrain classification. *Autonomous Robots*, 45(6), 843–857. <https://doi.org/10.1007/s10514-021-10013-w>
- Buchanan, R., Camurri, M., & Fallon, M. (2020). Haptic Sequential Monte Carlo Localization for Quadrupedal Locomotion in Vision-Denied Scenarios. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. <https://doi.org/10.1109/iros45743.2020.9341128>
- Catalano, M. G., Grioli, G., Farnioli, E., Serio, A., Piazza, C., & Bicchi, A. (2014). Adaptive synergies for the design and control of the pisa/iit soft hand. *The International Journal of Robotics Research*, 33(5), 768–782. <https://doi.org/10.1177/0278364913518998>
- Chalon, M., Reinecke, J., & Pfanne, M. (2013). Online in-hand object localization. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2977–2984. <https://doi.org/10.1109/IROS.2013.6696778>
- Chaudhuri, S., Ellis, K., Polozov, O., Singh, R., Solar-Lezama, A., & Yue, Y. (2021). Neurosymbolic programming. *Foundations and Trends® in Programming Languages*, 7(3), 158–243. <https://doi.org/10.1561/25000000049>
- Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., ... Zaremba, W. (2021). Evaluating large language models trained on code. *CoRR*, abs/2107.03374. <https://doi.org/10.48550/arxiv.2107.03374>
- Chen, X., Liu, C., & Song, D. X. (2018). Tree-to-tree neural networks for program translation. *CoRR*, abs/1802.03691.
- Chitta, S., Vernaza, P., Geykhman, R., & Lee, D. (2007). Proprioceptive localization for a quadrupedal robot on known terrain. *IEEE International Conference on Robotics and Automation (ICRA)*, 4582–4587. <https://doi.org/10.1109/ROBOT.2007.364185>
- Cho, K., van Merriënboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. <https://doi.org/10.3115/v1/D14-1179>
- Chollet, F. (2019). On the measure of intelligence. *CoRR*, abs/1911.01547. <https://doi.org/10.48550/arxiv.1911.01547>
- Christe, J., & Kottege, N. (2016). Acoustics based terrain classification for legged robots. *IEEE International Conference on Robotics and Automation (ICRA)*. <https://doi.org/10.1109/icra.2016.7487543>
- Chun-Hung Chen & Kumar, V. (1996). Motion planning of walking robots in environments with uncertainty. *Journal of Robotic Systems*, 4(10), 3277–3282 vol.4. <https://doi.org/10.1109/ROBOT.1996.509212>

- Dahiya, R. S., Metta, G., Valle, M., & Sandini, G. (2010). Tactile sensing—from humans to humanoids. *IEEE Transactions on Robotics*, 26(1), 1–20. <https://doi.org/10.1109/TRO.2009.2033627>
- Dallaire, P., Walas, K., Giguere, P., & Chaib-draa, B. (2015). Learning Terrain Types with the Pitman-Yor Process Mixtures of Gaussians for a Legged Robot. *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 3457–3463. <https://doi.org/10.1109/IROS.2015.7353859>
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. <https://doi.org/10.1109/CVPR.2009.5206848>
- Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, *abs/1810.04805*. <https://doi.org/10.48550/arxiv.1810.04805>
- Devlin, J., Uesato, J., Bhupatiraju, S., Singh, R., Mohamed, A., & Kohli, P. (2017). Robustfill: Neural program learning under noisy I/O. *ICML*, 70, 990–998.
- Di Carlo, J., Wensing, P. M., Katz, B., Bledt, G., & Kim, S. (2018). Dynamic Locomotion in the MIT Cheetah 3 Through Convex Model-Predictive Control. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1–9. <https://doi.org/10.1109/IROS.2018.8594448>
- Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, *abs/2010.11929*. <https://doi.org/10.48550/arxiv.2010.11929>
- Driess, D., Xia, F., Sajjadi, M. S. M., Lynch, C., Chowdhery, A., Ichter, B., Wahid, A., Tompson, J., Vuong, Q., Yu, T., Huang, W., Chebotar, Y., Sermanet, P., Duckworth, D., Levine, S., Vanhoucke, V., Hausman, K., Toussaint, M., Greff, K., ... Florence, P. (2023). Palm-e: An embodied multimodal language model. <https://doi.org/10.48550/arxiv.2303.03378>
- Dulac-Arnold, G., Mankowitz, D. J., & Hester, T. (2019). Challenges of real-world reinforcement learning. *CoRR*, *abs/1904.12901*. <https://doi.org/10.48550/arxiv.1904.12901>
- Ellis, K., Wong, C., Nye, M., Sablé-Meyer, M., Morales, L., Hewitt, L., Cary, L., Solar-Lezama, A., & Tenenbaum, J. B. (2021). Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 835–850. <https://doi.org/10.1145/3453483.3454080>
- Fankhauser, P., Bjelonic, M., Dario Bellicoso, C., Miki, T., & Hutter, M. (2018). Robust rough-terrain locomotion with a quadrupedal robot. *IEEE International Conference on Robotics and Automation (ICRA)*, 5761–5768. <https://doi.org/10.1109/ICRA.2018.8460731>
- Fankhauser, P., Bloesch, M., Gehring, C., Hutter, M., & Siegwart, R. (2014). Robot-centric elevation mapping with uncertainty estimates. *International Conference on Climbing and Walking Robots (CLAWAR)*. https://doi.org/10.1142/9789814623353_0051
- Ferré, S. (2021). First steps of an approach to the ARC challenge based on descriptive grid models and the minimum description length principle. *CoRR*, *abs/2112.00848*. <https://doi.org/10.48550/arxiv.2112.00848>
- Fey, M., Hu, W., Huang, K., Lenssen, J. E., Ranjan, R., Robinson, J., Ying, R., You, J., & Leskovec, J. (2024). Position: Relational deep learning - graph representation learning on relational databases. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, & F. Berkenkamp (Eds.), *Proceedings of the 41st international conference*

- on machine learning (pp. 13592–13607, Vol. 235). PMLR. <https://proceedings.mlr.press/v235/fey24a.html>
- Filitchkin, P., & Byl, K. (2012). Feature-based terrain classification for littledog. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, 1387–1392. <https://doi.org/10.1109/IROS.2012.6386042>
- Fink, G., & Semini, C. (2020). Proprioceptive Sensor Fusion for Quadruped Robot State Estimation. *IEEE International Conference on Intelligent Robots and Systems (IROS)*. <https://doi.org/10.1109/iros45743.2020.9341521>
- Focchi, M., Orsolino, R., Camurri, M., Barasuol, V., Mastalli, C., Caldwell, D., & Semini, C. (2018). Heuristic planning for rough terrain locomotion in presence of external disturbances and variable perception quality (A. Grau, Y. Morel, A. Puig-Pey, & F. Cecchi, Eds.). https://doi.org/10.1007/978-3-030-22327-4_9
- Forster, C., Carlone, L., Dellaert, F., & Scaramuzza, D. (2017). On-Manifold Preintegration for Real-Time Visual-Inertial Odometry. *IEEE Transactions on Robotics*, 33(1), 1–21. <https://doi.org/10.1109/TRO.2016.2597321>
- Fox, D., Thrun, S., Burgard, W., & Dellaert, F. (2001). Particle filters for mobile robot localization. In *Sequential monte carlo methods in practice* (pp. 401–428). Springer New York. https://doi.org/10.1007/978-1-4757-3437-9_19
- Frappier, M. (2018). The book of why: The new science of cause and effect. *Science*, 361, 855–855. <https://api.semanticscholar.org/CorpusID:52133264>
- Garcez, A. d., & Lamb, L. C. (2023). Neurosymbolic ai: The 3rd wave. *Artif. Intell. Rev.*, 56(11), 12387–12406. <https://doi.org/10.1007/s10462-023-10448-w>
- Garcia, J., & Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1), 1437–1480.
- Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2), 155–170. [https://doi.org/10.1016/S0364-0213\(83\)80009-3](https://doi.org/10.1016/S0364-0213(83)80009-3)
- Giguère, P., Dudek, G., Saunderson, S., & Prahacs, C. (2006). Environment Identification for a Running Robot Using Inertial and Actuator Cues. *Robotics: Science and Systems*. <https://doi.org/10.15607/rss.2006.ii.035>
- Giusti, A., Guzzi, J., Cireşan, D. C., He, F. L., Rodríguez, J. P., Fontana, F., Faessler, M., Forster, C., Schmidhuber, J., Caro, G. D., Scaramuzza, D., & Gambardella, L. M. (2016). A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots. *IEEE Robotics and Automation Letters*, 1(2), 661–667. <https://doi.org/10.1109/lra.2015.2509024>
- Glenberg, A. M., & Kaschak, M. P. (2002). Grounding language in action. *Psychonomic Bulletin & Review*, 9(3), 558–565. <https://doi.org/10.3758/BF03196313>
- Grand, G., Wong, L., Bowers, M., Olausson, T. X., Liu, M., Tenenbaum, J. B., & Andreas, J. (2024). Lilo: Learning interpretable libraries by compressing and documenting code. <https://doi.org/10.48550/arxiv.2310.19791>
- Graves, A., Fernández, S., & Schmidhuber, J. (2005). Bidirectional lstm networks for improved phoneme classification and recognition. *Proceedings of the 15th International Conference on Artificial Neural Networks: Formal Models and Their Applications - Volume Part II*, 799–804. https://doi.org/10.1007/11550907_126
- Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines. <https://doi.org/10.48550/arxiv.1410.5401>
- Gualtieri, M., Ten Pas, A., Saenko, K., & Platt, R. (2016). High precision grasp pose detection in dense clutter. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 598–605. <https://doi.org/10.1109/IROS.2016.7759114>

- Harnad, S. (1999). The symbol grounding problem. *CoRR*, *cs.AI/9906002*. <https://doi.org/10.48550/arxiv.cs/9906002>
- Hartley, R., Ghaffari, M., Eustice, R. M., & Grizzle, J. W. (2020). Contact-aided invariant extended Kalman filtering for robot state estimation. *The International Journal of Robotics Research*, *39*(4), 402–430. <https://doi.org/10.1177/0278364919894385>
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *2015 IEEE International Conference on Computer Vision (ICCV)*, 1026–1034. <https://doi.org/10.1109/ICCV.2015.123>
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. <https://doi.org/10.1109/CVPR.2016.90>
- Hinton, G., & Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507. <https://doi.org/10.1126/science.1127647>
- Hinton, G. E., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. *CoRR*, *abs/1503.02531*. <https://doi.org/10.48550/arxiv.1503.02531>
- Hoepflinger, M., Remy, C., Hutter, M., Haag, S., & Siegwart, R. (2010). Haptic terrain classification on natural terrains for legged robots. *International Conference on Climbing and Walking Robots (CLAWAR)*, 785–792. https://doi.org/10.1142/9789814329927_0097
- Hoepflinger, M. A., Hutter, M., Gehring, C., Bloesch, M., & Siegwart, R. (2013). Unsupervised identification and prediction of foothold robustness. *International Conference on Robotics and Automation (ICRA)*, 3293–3298. <https://doi.org/10.1109/ICRA.2013.6631036>
- Hoepflinger, M. A., Hutter, M., Gehring, C., Fankhauser, P., & Siegwart, R. (2014). Haptic Foothold Suitability Identification and Prediction for Legged Robots. *International Conference on Climbing and Walking Robots (CLAWAR)*, 425–432. https://doi.org/10.1142/9789814623353_0050
- Hoffmann, M., Stepanova, K., & Reinstein, M. (2014). The Effect of Motor Action and Different Sensory Modalities on Terrain Classification in a Quadruped Robot Running with Multiple Gaits. *Robotics and Autonomous Systems*, *62*(12), 1790–1798. <https://doi.org/10.1016/j.robot.2014.07.006>
- Hofstadter, D. R. (1995). *Fluid concepts & creative analogies : Computer models of the fundamental mechanisms of thought*. Basic Books.
- Höpfinger, M., et al. (2010). Haptic Terrain Classification for Legged Robots. *IEEE International Conference on Robotics and Automation (ICRA)*, 2828–2833. <https://doi.org/10.1109/robot.2010.5509309>
- Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., & Burgard, W. (2013). Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Auton. Robots*, *34*(3), 189–206. <https://doi.org/10.1007/s10514-012-9321-0>
- Hüllermeier, E., & Waegeman, W. (2019). Aleatoric and epistemic uncertainty in machine learning: A tutorial introduction. *CoRR*, *abs/1910.09457*. <https://doi.org/10.48550/arxiv.1910.09457>
- Hutter, M., Gehring, C., Jud, D., Lauber, A., Bellicoso, C. D., Tsounis, V., Hwangbo, J., Bodie, K., Fankhauser, P., Bloesch, M., Diethelm, R., Bachmann, S., Melzer, A., & Hoepflinger, M. (2016). Anymal-a highly mobile and dynamic quadrupedal robot. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 38–44. <https://doi.org/10.1109/IROS.2016.7758092>
- Hyun, D. J., Seok, S., Lee, J., & Kim, S. (2014). High speed trot-running: Implementation of a hierarchical controller using proprioceptive impedance control on the mit cheetah. *Int. J. Rob. Res.*, *33*(11), 1417–1445. <https://doi.org/10.1177/0278364914532150>

- Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., & Girshick, R. B. (2016). CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. *CoRR*, *abs/1612.06890*. <https://doi.org/10.48550/arxiv.1612.06890>
- Kahneman, D. (2011). *Thinking, fast and slow*. Farrar, Straus; Giroux. https://www.amazon.de/Thinking-Fast-Slow-Daniel-Kahneman/dp/0374275637/ref=wl_it_dp_o_pdT1_nS_nC?ie=UTF8&colid=151193SNGKJT9&coliid=I3OCESLZCVDL7
- Kalakrishnan, M., Buchli, J., Pastor, P., & Schaal, S. (2009). Learning locomotion over rough terrain using terrain templates. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 167–172. <https://doi.org/10.1109/IROS.2009.5354701>
- Kertesz, C. (2016). Rigidity-Based Surface Recognition for a Domestic Legged Robot. *IEEE Robotics and Automation Letters*, *1*(1), 309–315. <https://doi.org/10.1109/LRA.2016.2519949>
- Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization (Y. Bengio & Y. LeCun, Eds.). *CoRR*, *abs/1412.6980*. <https://doi.org/10.48550/arxiv.1412.6980>
- Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *CoRR*, *abs/1312.6114*. <https://doi.org/10.48550/arxiv.1312.6114>
- Kober, J., Bagnell, J. A., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, *32*(11), 1238–1274. <https://doi.org/10.1177/0278364913495721>
- Kolter, J. Z., Rodgers, M. P., & Ng, A. Y. (2008). A control architecture for quadruped locomotion over rough terrain. *2008 IEEE International Conference on Robotics and Automation*, 811–818. <https://doi.org/10.1109/ROBOT.2008.4543305>
- Kolvenbach, H., Bärtschi, C., Wellhausen, L., Grandia, R., & Hutter, M. (2019). Haptic Inspection of Planetary Soils With Legged Robots. *IEEE Robotics and Automation Letters*, *4*(2), 1626–1632. <https://doi.org/10.1109/LRA.2019.2896732>
- Kolvenbach, H., Wisth, D., Buchanan, R., Valsecchi, G., Grandia, R., Fallon, M., & Hutter, M. (2020). Towards autonomous inspection of concrete deterioration in sewers with legged robots. *Journal of Field Robotics*. <https://doi.org/10.1002/rob.21964>
- Kopicki, M., Detry, R., Adjigble, M., Stolkin, R., Leonardis, A., & Wyatt, J. L. (2015). One-shot learning and generation of dexterous grasps for novel objects. *The International Journal of Robotics Research*, *35*(8), 959–976. <https://doi.org/10.1177/0278364915594244>
- Koval, M. C., Pollard, N. S., & Srinivasa, S. S. (2016). Manifold representations for state estimation in contact manipulation. In M. Inaba & P. Corke (Eds.), *Robotics research: The 16th international symposium isrr* (pp. 375–391). Springer International Publishing. https://doi.org/10.1007/978-3-319-28872-7_22
- Krawiec, K., Bładek, I., & Swan, J. (2017). Counterexample-driven genetic programming. *Proceedings of the Genetic and Evolutionary Computation Conference*, 953–960. <https://doi.org/10.1145/3071178.3071224>
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Proceedings of the 26th international conference on neural information processing systems - volume 1* (pp. 1097–1105). Curran Associates Inc. <https://doi.org/10.1145/3065386>
- Krotkov, E., & Simmons, R. (1996). Perception, planning, and control for autonomous walking with the ambler planetary rover. *The International Journal of Robotics Research*, *15*(2), 155–180. <https://doi.org/10.1177/027836499601500204>
- Kumar, A., Fu, Z., Pathak, D., & Malik, J. (2021). RMA: rapid motor adaptation for legged robots. *CoRR*, *abs/2107.04034*. <https://doi.org/10.48550/arxiv.2107.04034>

- Lake, B. M., Ullman, T. D., Tenenbaum, J. B., & Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, e253. <https://doi.org/10.1017/S0140525X16001837>
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, 521(7553), 436–444.
- Lederman, S. J., & Klatzky, R. L. (1987). Hand movements: A window into haptic object recognition. *Cognitive Psychology*, 19(3), 342–368. [https://doi.org/10.1016/0010-0285\(87\)90008-9](https://doi.org/10.1016/0010-0285(87)90008-9)
- Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., & Hutter, M. (2020). Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47). <https://doi.org/10.1126/scirobotics.abc5986>
- Lenskiy, A. A., & Lee, J. (2010). Terrain images segmentation in infra-red spectrum for autonomous robot navigation. *International Forum on Strategic Technology 2010*, 33–38. <https://doi.org/10.1109/IFOST.2010.5667916>
- Lesort, T., Lomonaco, V., Stoian, A., Maltoni, D., Filliat, D., & Rodríguez, N. D. (2019). Continual learning for robotics. *CoRR*, abs/1907.00182. <https://doi.org/10.48550/arxiv.1907.00182>
- Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2015). End-to-end training of deep visuomotor policies. *CoRR*, abs/1504.00702. <https://doi.org/10.48550/arxiv.1504.00702>
- Li, C., & Wand, M. (2016). Combining markov random fields and convolutional neural networks for image synthesis. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2479–2486. <https://doi.org/10.1109/cvpr.2016.272>
- Lin, M., Chen, Q., & Yan, S. (2014). Network in network. <https://doi.org/10.48550/arxiv.1312.4400>
- Ling, W., Blunsom, P., Grefenstette, E., Hermann, K. M., Kociský, T., Wang, F., & Senior, A. (2016). Latent predictor networks for code generation. *CoRR*, abs/1603.06744, 599–609. <https://doi.org/10.18653/v1/P16-1057>
- Loshchilov, I., & Hutter, F. (2017). Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101. <https://doi.org/10.48550/arxiv.1711.05101>
- Luo, S., Bimbo, J., Dahiya, R., & Liu, H. (2017). Robotic tactile perception of object properties: A review. *CoRR*, abs/1711.03810. <https://doi.org/10.48550/arxiv.1711.03810>
- Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., Ojea, J. A., & Goldberg, K. (2017). Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. In N. M. Amato, S. S. Srinivasa, N. Ayanian, & S. Kuindersma (Eds.), *Robotics: Science and systems*. <https://doi.org/10.15607/RSS.2017.XIII.058>
- Manuelli, L., & Tedrake, R. (2016). Localizing external contact using proprioceptive sensors: The contact particle filter. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5062–5069. <https://doi.org/10.1109/iros.2016.7759743>
- Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B., & Wu, J. (2019). The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. <https://doi.org/10.48550/arxiv.1904.12584>
- Marcus, G. (2018). Deep learning: A critical appraisal. *CoRR*, abs/1801.00631. <https://doi.org/10.48550/arxiv.1801.00631>
- Miconi, T. (2009). Why coevolution doesn't "work": Superiority and progress in coevolution. In L. Vanneschi, S. Gustafson, A. Moraglio, I. De Falco, & M. Ebner (Eds.), *Proceedings of the 12th european conference on genetic programming, eurogp 2009* (pp. 49–60, Vol. 5481). Springer. https://doi.org/10.1007/978-3-642-01181-8_5
- Mikolov, T., Chen, K., Corrado, G. S., & Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

- Mirchandani, S., Xia, F., Florence, P., Ichter, B., Driess, D., Arenas, M. G., Rao, K., Sadigh, D., & Zeng, A. (2023). Large language models as general pattern machines. <https://doi.org/10.48550/arxiv.2307.04721>
- Moravec, H. (1988). *Mind children: The future of robot and human intelligence*. Harvard University Press. <https://books.google.pl/books?id=56mb7XuSx3QC>
- Moskvichev, A., Odouard, V. V., & Mitchell, M. (2023). The conceptarc benchmark: Evaluating understanding and generalization in the arc domain. <https://doi.org/10.48550/arxiv.2305.07141>
- Mrva, J., & Faigl, J. (2015). Feature Extraction for Terrain Classification with Crawling Robots. *Proceedings ITAT 2015: Information Technologies - Applications and Theory, Slovensky Raj, Slovakia, September 17-21, 2015.*, 179–185.
- Muggleton, S. H., & Watanabe, H. (2014). *Latest advances in inductive logic programming*. IMPERIAL COLLEGE PRESS. <https://doi.org/10.1142/p954>
- Nair, V., & Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on International Conference on Machine Learning*, 807–814. <http://dl.acm.org/citation.cfm?id=3104322.3104425>
- Namin, S. T., Najafi, M., Salzmann, M., & Petersson, L. (2015). Cutting Edge: Soft Correspondences in Multimodal Scene Parsing. *2015 IEEE International Conference on Computer Vision (ICCV)*, 1188–1196. <https://doi.org/10.1109/iccv.2015.141>
- Narváez, F. Y., Gregorio, E., Escolà, A., Rosell-Polo, J. R., Torres-Torriti, M., & Cheein, F. A. (2018). Terrain classification using tof sensors for the enhancement of agricultural machinery traversability. *Journal of Terramechanics*, 76, 1–13. <https://doi.org/10.1016/j.jterra.2017.10.005>
- Newell, A., Simon, H. A., et al. (1972). *Human problem solving* (Vol. 104). Prentice-hall Englewood Cliffs, NJ.
- Oleynikova, H., Taylor, Z., Fehr, M., Siegwart, R., & Nieto, J. (2017). Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. <https://doi.org/10.1109/iros.2017.8202315>
- Otsu, K., Ono, M., Fuchs, T. J., Baldwin, I., & Kubota, T. (2016). Autonomous terrain classification with co- and self-training approach. *IEEE Robotics and Automation Letters*, 1(2), 814–819. <https://doi.org/10.1109/LRA.2016.2525040>
- Pan, J., Chitta, S., & Manocha, D. (2012). Fcl: A general purpose library for collision and proximity queries. *2012 IEEE International Conference on Robotics and Automation*, 3859–3866. <https://doi.org/10.1109/ICRA.2012.6225337>
- Paszke, A., Chaurasia, A., Kim, S., & Culurciello, E. (2016). Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*. <https://doi.org/10.48550/arxiv.1606.02147>
- Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543. <https://doi.org/10.3115/v1/D14-1162>
- Pfeifer, R., & Bongard, J. (2006, October). *How the body shapes the way we think: A new view of intelligence*. The MIT Press. <https://doi.org/10.7551/mitpress/3585.001.0001>
- Piazza, C., Della Santina, C., Gasparri, G. M., Catalano, M. G., Grioli, G., Garabini, M., & Bicchi, A. (2016). Toward an adaptive foot for natural walking. *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, 1204–1210. <https://doi.org/10.1109/humanoids.2016.7803423>
- Pitman, J., & Yor, M. (1997). The two-parameter poisson-dirichlet distribution derived from a stable subordinator. *The Annals of Probability*, 855–900. <https://doi.org/10.1214/aop/1024404422>

- Polosukhin, I., & Skidanov, A. (2018). Neural program search: Solving programming tasks from description and examples, 11.
- Rabinovich, M., Stern, M., & Klein, D. (2017). Abstract syntax networks for code generation and semantic parsing. *CoRR*, *abs/1704.07535*, 1139–1149. <https://doi.org/10.18653/v1/P17-1105>
- Rebula, J. R., Neuhaus, P. D., Bonnlander, B. V., Johnson, M. J., & Pratt, J. E. (2007). A controller for the littledog quadruped walking on rough terrain. *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 1467–1473. <https://doi.org/10.1109/ROBOT.2007.363191>
- Roennau, A., Kerscher, T., Ziegenmeyer, M., Zöllner, J., & Dillmann, R. (2009). Six-legged walking in rough terrain based on foot point planning. *Mobile robotics: Solutions and challenges*, 591–598. https://doi.org/10.1142/9789814291279_0072
- Romera, E., Alvarez, J. M., Bergasa, L., & Arroyo, R. (2017). Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *IEEE Transactions on Intelligent Transportation Systems*, *PP*(19), 1–10. <https://doi.org/10.1109/TITS.2017.2750080>
- Russell, S., Russell, S., & Norvig, P. (2020). *Artificial intelligence: A modern approach*. Pearson. <https://books.google.pl/books?id=koFptAEACAAJ>
- Saارينen, J. P., Andreasson, H., Stoyanov, T., & Lilienthal, A. J. (2013). 3d normal distributions transform occupancy maps: An efficient representation for mapping in dynamic environments. *Int. J. Rob. Res.*, *32*(14), 1627–1644. <https://doi.org/10.1177/0278364913499415>
- Schlangen, D. (2020). Targeting the benchmark: On methodology in current natural language processing research. *CoRR*, *abs/2007.04792*. <https://doi.org/10.48550/arxiv.2007.04792>
- Schmidt, M., & Lipson, H. (2009). Distilling free-form natural laws from experimental data. *Science*, *324*(5923), 81–85. <https://doi.org/10.1126/science.1165893>
- Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*, 2673–2681. <https://doi.org/10.1109/78.650093>
- Schwendner, J., Joyeux, S., & Kirchner, F. (2014). Using embodied data for localization and mapping. *Journal of Field Robotics*, *31*(2), 263–295. <https://doi.org/10.1002/rob.21489>
- Searle, J. R. (1990). Minds, brains, and programs. In M. A. Boden (Ed.), *The philosophy of artificial intelligence* (pp. 67–88). Oxford University Press.
- Siciliano, B., & Khatib, O. (2008). *Springer handbook of robotics*. Springer Berlin Heidelberg. <https://books.google.pl/books?id=Xpgi5gSuBxsC>
- Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2010). *Robotics: Modelling, planning and control*. Springer London. <https://books.google.pl/books?id=jPCAFmE-logC>
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, *550*(7676), 354–359. <https://doi.org/10.1038/nature24270>
- Smith, L. N. (2018). A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. *CoRR*, *abs/1803.09820*. <https://doi.org/10.48550/arxiv.1803.09820>
- Spelke, E. S., & Kinzler, K. D. (2007). Core knowledge. *Developmental Science*, *10*(1), 89–96. <https://doi.org/10.1111/j.1467-7687.2007.00569.x>
- Stallkamp, J., Schlupsing, M., Salmen, J., & Igel, C. (2012). Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, *32*, 323–332. <https://doi.org/10.1016/j.neunet.2012.02.016>
- Sturm, J., Engelhard, N., Endres, F., Burgard, W., & Cremers, D. (2012). A Benchmark for the Evaluation of RGB-D SLAM Systems. *IEEE/RSJ International Conference on*

- Intelligent Robots and Systems (IROS)*, 573–580. <https://doi.org/10.1109/IROS.2012.6385773>
- Sukhbaatar, S., Szlam, A., Weston, J., & Fergus, R. (2015). End-to-end memory networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems 28* (pp. 2440–2448). Curran Associates, Inc. <http://papers.nips.cc/paper/5846-end-to-end-memory-networks.pdf>
- Sünderhauf, N., Brock, O., Scheirer, W. J., Hadsell, R., Fox, D., Leitner, J., Upcroft, B., Abbeel, P., Burgard, W., Milford, M., & Corke, P. (2018). The limits and potentials of deep learning for robotics. *CoRR*, *abs/1804.06557*. <https://doi.org/10.48550/arxiv.1804.06557>
- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning - an introduction, 2nd edition*. MIT Press. <http://www.incompleteideas.net/book/the-book-2nd.html>
- Tai, K. S., Socher, R., & Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. *ACL*. <https://doi.org/10.3115/v1/p15-1150>
- Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic robotics*. MIT Press. <https://books.google.pl/books?id=2Zn6AQAAQBAJ>
- Tolomei, S., Belter, D., Bednarek, J., Angelini, F., & Garabini, M. (2025). Learning-based foot-shape-aware foothold selection for quadrupedal robots. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *55*(8), 5234–5247. <https://doi.org/10.1109/TSMC.2025.3563974>
- Valada, A., Oliveira, G., Brox, T., & Burgard, W. (2016). Deep multispectral semantic scene understanding of forested environments using multimodal fusion. *The 2016 International Symposium on Experimental Robotics (ISER 2016)*. https://doi.org/10.1007/978-3-319-50115-4_41
- Valada, A., Spinello, L., & Burgard, W. (2018). Deep feature learning for acoustics-based terrain classification. In A. Bicchi & W. Burgard (Eds.), *Robotics research: Volume 2* (pp. 21–37). Springer International Publishing. https://doi.org/10.1007/978-3-319-60916-4_2
- Valsecchi, G., Grandia, R., & Hutter, M. (2020). Quadrupedal locomotion on uneven terrain with sensorized feet. *IEEE Robotics and Automation Letters*, *5*(2), 1548–1555. <https://doi.org/10.1109/LRA.2020.2969160>
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *CoRR*, *abs/1706.03762*. <https://doi.org/10.48550/arxiv.1706.03762>
- Vespa, E., Nikolov, N., Grimm, M., Nardi, L., Kelly, P. H. J., & Leutenegger, S. (2018). Efficient octree-based volumetric slam supporting signed-distance and occupancy mapping. *IEEE Robotics and Automation Letters*, *3*(2), 1144–1151. <https://doi.org/10.1109/LRA.2018.2792537>
- Vezzani, G., Pattacini, U., Battistelli, G., Chisci, L., & Natale, L. (2017). Memory Unscented Particle Filter for 6-DOF Tactile Localization. *IEEE Transactions on Robotics*, *33*(5), 1139–1155. <https://doi.org/10.1109/TRO.2017.2707092>
- Villarreal, O. A., Barasuol, V., Camurri, M., Franceschi, L., Focchi, M., Pontil, M., Caldwell, D. G., & Semini, C. (2019). Fast and continuous foothold adaptation for dynamic locomotion through CNNs. *IEEE Robotics and Automation Letters*, *4*(2), 2140–2147. <https://doi.org/10.1109/lra.2019.2899434>
- Walas, K., & Nowicki, M. (2014). Terrain Classification using Laser Range Finder. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5003–5009. <https://doi.org/10.1109/IROS.2014.6943273>

- Walas, K. (2015). Terrain Classification and Negotiation with a Walking Robot. *Journal of Intelligent & Robotic Systems*, 78(3-4), 401–423. <https://doi.org/10.1007/s10846-014-0067-0>
- Walas, K., & Belter, D. (2011). Supporting locomotive functions of a six-legged walking robot. *Int. J. Appl. Math. Comput. Sci.*, 21(2), 363–377. <https://doi.org/10.2478/v10006-011-0027-9>
- Wellhausen, L., Dosovitskiy, A., Ranftl, R., Walas, K., Cadena, C., & Hutter, M. (2019). Where Should I Walk? Predicting Terrain Properties from Images Via Self-Supervised Learning. *IEEE Robotics and Automation Letters*, 4(2), 1509–1516. <https://doi.org/10.1109/LRA.2019.2895390>
- Winkler, A., Havoutis, I., Bazeille, S., Ortiz, J., Focchi, M., Dillmann, R., Caldwell, D., & Semini, C. (2014). Path planning with force-based foothold adaptation and virtual model control for torque controlled quadruped robots. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 6476–6482. <https://doi.org/10.1109/icra.2014.6907815>
- Wooden, D., Malchano, M., Blankespoor, K., Howardy, A., Rizzi, A. A., & Raibert, M. (2010). Autonomous navigation for bigdog. *2010 IEEE International Conference on Robotics and Automation*, 4736–4741. <https://doi.org/10.1109/ROBOT.2010.5509226>
- Xu, Y., Khalil, E. B., & Sanner, S. (2022). Graphs, constraints, and search for the abstraction and reasoning corpus. <https://doi.org/10.48550/arxiv.2210.09880>
- Yin, P., & Neubig, G. (2017). A syntactic neural model for general-purpose code generation. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 440–450. <https://doi.org/10.18653/v1/P17-1041>
- Zhang, C., Gao, F., Jia, B., Zhu, Y., & Zhu, S. (2019). RAVEN: A dataset for relational and analogical visual reasoning. *CoRR*, *abs/1903.02741*. <https://doi.org/10.48550/arxiv.1903.02741>