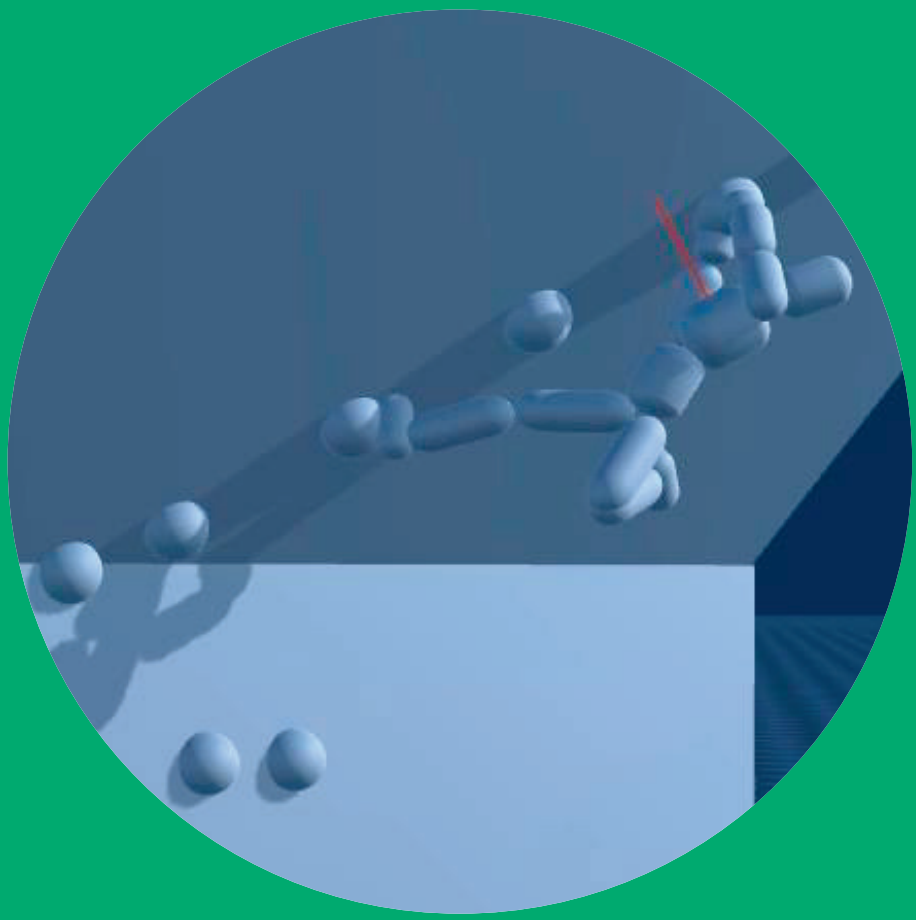
Department of Computer Science

Discovering, Synthesizing, and Learning Climbing Movements

Kourosh Naderi



Aalto University

DOCTORAL DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 87/2020

Discovering, Synthesizing, and Learning Climbing Movements

Kourosh Naderi

The public defense on 8th June 2020 at 12:00 will be available via remote technology.

Link: https://aalto.zoom.us/j/64234063678

Zoom Quick Guide: https://www.aalto.fi/en/services/zoom-quick-guide

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Science, Remote connection link (e.g. Zoom), on 8th June 2020 at 12 o'clock.

Aalto University
School of Science
Department of Computer Science
Aalto Game Research Group

Supervising professor
Professor Perttu Hämäläinen, Aalto University, Finland

Thesis advisor
Professor Ville Kyrki, Aalto University, Finland

Preliminary examiners
Professor Sheldon Andrews, École de technologie supérieure, Canada
Professor Steve Tonneau, University of Edinburgh, United Kingdom

Opponent
Professor Steven LaValle, University of Oulu, Finland

Aalto University publication series
DOCTORAL DISSERTATIONS 87/2020

ISBN 978-952-60-8886-0 (printed)
ISBN 978-952-60-8887-7 (pdf)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)
http://urn.fi/URN:ISBN:978-952-60-8887-7

Unigrafia Oy
Helsinki 2020

Finland

Printed matter
4041-0619

Aalto University

Abstract

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

Author
Kourosh Naderi

Name of the doctoral dissertation
Discovering, Synthesizing, and Learning Climbing Movements

Publisher School of Science

Unit Department of Computer Science

Series Aalto University publication series DOCTORAL DISSERTATIONS 87/2020

Field of research Computer Animation and Robotics

Manuscript submitted 17 December 2019 **Date of the defence** 8 June 2020

Permission for public defence granted (date) 6 February 2020 **Language** English

☐ **Monograph** ☒ **Article dissertation** ☐ **Essay dissertation**

Abstract

Although animation is commonly captured from real humans or edited manually through keyframes and interpolation curves, such techniques are time-consuming and expensive. Because of this, a long-standing goal of computer animation research has been to synthesize movements algorithmically, e.g., through simulating the human biomechanics and framing the animation problem as optimization of joint torques over time. During recent years, such approaches have enjoyed success with movements like bipedal locomotion, but some complex movement skills have remained challenging.

This dissertation focuses on synthesizing natural looking and human-like climbing movements in indoor bouldering, a popular and rapidly growing sport that recently was approved to the 2020 Tokyo Olympics together with speed and sport climbing. Indoor bouldering is a form of climbing that takes place relatively close to the ground on top of a soft landing surface, and does not require special equipment other than climbing shoes. Bouldering routes are usually short and they focus on complex climbing moves that require both strength and coordination. Planning and discovering the optimal or at least possible sequence of moves from the ground to the top hold is a challenging problem. The problem gets even more complicated when the planning should consider the body types of users such that the planned path and synthesized motions would be feasible for them.

This thesis proposes a high-level path planner and low-level controller for synthesizing physically plausible and human-like movements. The high-level graph-based path planner is responsible for planning a sequence of movements to the top hold while the low-level controller synthesizes the movement details through optimizing the joint actuations of a physics simulation model of a humanoid climber. Such a low-level controller might fail to follow the planned movements; the thesis proposes ways to handle this uncertainty through low-level and high-level controller interaction. In subsequent work, the approach is developed further by employing neural networks in both supervised and reinforcement learning settings.

The methods proposed in the thesis result in high-quality climbing animations without needing any reference animation or motion capture data. The work should also have applications in synthesizing other types of movements with similar characteristics, e.g., creating parkour animations based on desired footstep patterns.

Keywords Hierarchical Motion Planning, Graph Search, Sampling-based Movement Optimization, Supervised Learning, Reinforcement Learning

ISBN (printed) 978-952-60-8886-0 **ISBN (pdf)** 978-952-60-8887-7

ISSN (printed) 1799-4934 **ISSN (pdf)** 1799-4942

Location of publisher Helsinki **Location of printing** Helsinki **Year** 2020

Pages 145 **urn** http://urn.fi/URN:ISBN:978-952-60-8887-7

Preface

The work in this thesis was carried out in the Department of Computer Science of Aalto University between 2015 and 2019 in professor Perttu Hämäläinen's group. My PhD was a big chapter of my life, and it would not be possible without the endless support and encouragement from my family, friends, and colleagues. I wish to thank them for making the best and most memorable moments in these years. All I have experienced during this lifetime are cherished as learned lessons, and their reflections will enlighten my future endeavors.

My first and deepest appreciation goes to professor Perttu Hämäläinen who has given me the opportunity to pursue a doctoral degree and encouraged me during all these years to chase my dream. I wish to thank him for all his trust, patience and guidance that keep me motivated to continue my doctoral studies and research. His expertise and knowledge in the research field and his mindful interactions were enormously invaluable in conducting my research and helping me stay on track.

I also like to thank my instructor, professor Ville Kyrki, and acknowledge his positive attitude toward my research outcomes in addition to his insightful and in-depth comments on my doctoral dissertation.

I would like to thank professor Andrew Sheldon and professor Steve Tonneau for their rich and highly motivating comments and reviews. I also like to thank professor Steven Lavalle for accepting to be my opponent.

The work during my doctoral study years has been financially supported by Helsinki Doctoral Education Network in Information and Communications Technology (HICT) and Academy of Finland. I like to thank them for making this thesis possible.

I have met and exchanged ideas with many talented people during my doctoral studies. I wish to thank all of my colleagues who have made Aalto University a great place for conducting my research. Great thanks to my long-time colleague and friend Joose Rajamäki who has been always energetic, motivator and a great collaborator. I also like to thank Amin Babadi, a motivated researcher and a great collaborator, who I have enjoyed working with. Furthermore, I like to thank my co-authors Jari

Takatalo, and Jari Lipsanen. In addition, I wish to thank Juuso Toikka, Noshaba Cheema, and others in our research group who have made the workplace a better place for research and exchanging ideas.

I wish to thank my friends and family who all have been supported me during my studies. I like to thank my parents Nasser Naderi and Zohreh Ehteshaminia for their support and belief in me. Furthermore, I would like to thank my brother Kiumars Naderi. My wife's family has also been supportive during my doctoral studies. I also wish to convey my appreciation to them.

Great thanks to my wife Shaghayegh Roohi who has supported me in good and bad times. She has been the best collaborator one could ask, always supportive and ceaselessly loving during my doctoral studies.

Helsinki, May 11, 2020,

Kourosh Naderi

Contents

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I Kourosh Naderi and Joose Rajamäki and Perttu Hämäläinen. Discovering and Synthesizing Humanoid Climbing Movements. *ACM Transactions on Graphics (TOG)*, 36, 4, Article 43, 11 pages, July 2017.

II Kourosh Naderi, Jari Takatalo, Jari Lipsanen and Perttu Hämäläinen. Computer-Aided Imagery in Sport and Exercise: A Case Study of Indoor Wall Climbing. In *Proceedings of Graphics Interface (GI)*, pp. 93 - 99, Toronto, Canada. https://doi.org/10.20380/GI2018.13, May 2018.

III Kourosh Naderi, Amin Babadi and Perttu Hämäläinen. Learning Physically Based Humanoid Climbing Movements. *In Computer Graphics Forum*, vol. 37, no. 8, pp. 69-80, December 2018.

IV Kourosh Naderi, Amin Babadi, Shaghayegh Roohi, and Perttu Hämäläinen. A Reinforcement Learning Approach to Synthesizing Climbing Movements. In *IEEE Conference on Games (CoG)*, London, UK, August 2019.

Author's Contribution

Publication I: "Discovering and Synthesizing Humanoid Climbing Movements"

The author developed the algorithms, performed the testing and wrote the publication. Dr. Rajamäki and Prof. Hämäläinen contributed through commenting and discussions. Additionally, Dr. Rajamäki programmed CMA-ES optimization.

Publication II: "Computer-Aided Imagery in Sport and Exercise: A Case Study of Indoor Wall Climbing"

The author developed the technology and the user interface for the user study and wrote the technology part of the paper. Other parts were written by Dr. Takatalo and Prof. Hämäläinen. The author also conducted the user study together with Dr. Takatalo. The user study was designed by Dr. Takatalo, Dr. Lipsanen, and Prof. Hämäläinen.

Publication III: "Learning Physically Based Humanoid Climbing Movements"

The author developed the algorithms, performed the testing and wrote the publication. Mr. Babadi and Prof. Hämäläinen contributed through commenting and discussions.

Publication IV: "A Reinforcement Learning Approach to Synthesizing Climbing Movements"

The author developed the algorithms, performed the testing and wrote the publication. Ms. Roohi, Mr. Babadi, and Prof. Hämäläinen contributed through commenting and discussions.

List of Figures

Abbreviations

2D	Two Dimensional
3D	Three Dimensional
A*	A-Star
AR	Augmented Reality
APF	Artificial Potential Fields
BFS	Breadth First Search
C-PBP	Control Particle Belief Propagation
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CAI	Computer Aided Imagery
DoF	Degree of Freedom
DFS	Depth First Search
DDP	Differential Dynamic Programming
EM	Expectation Maximization
FIFO	First In First Out
GMM	Gaussian Mixture Model
GAE	Generalized Advantage Estimation
iLQR	iterative Linear Quadratic Regulator
k**NN**	k Nearest Neighbours
MPC	Model Predictive Control
MDP	Markov Decision Process

NN	Neural Networks
ODE	Open Dynamics Engine
PPO	Proximal Policy Optimization
PRM	Probabilistic Roadmap
PO-MDP	Partially Observable MDP
RL	Reinforcement Learning
RRT	Rapidly Exploring Random Trees
RRT*	Rapidly Exploring Random Tree Star
RT-RRT*	Real-Time RRT*
SAC	Soft Actor Critic
SGD	Stochastic Gradient Descent
SS-ESI	Self-Supervised Episode State Initialization
TI	Traditional Imagery

Symbols

α	Learning rate
γ	Discount factor
θ	Parameter used for policy network
μ	The mean of search distribution in CMA-ES
π_θ	Policy parametrized by θ
$\pi_\theta(\mathbf{a} \mid \mathbf{s})$	Probability density of action conditioned on state
$\rho(\mathbf{s}_0)$	Probability density of initial state $\mathbf{s}_0$
σ	A stance or an assignment of climber's hands and feet to holds
$\bar{\sigma}$	The average position of hands and feet positions in a stance
τ	An agent's simulated trajectory
τ_b	The strength of the joint attached to bone b
ω	Angular velocity
∇	Gradient
$\sum$	Summation
max	Maximize
$\mathscr{A}$	A set of actions
$A^\pi(\mathbf{s}_t, \mathbf{a}_t)$	The advantage function value at time t under policy π
$\mathbf{a}_t$	Action at time t
$\mathbf{a}_t^*$	Optimal action at time t
b_i	A bone name or id

$\mathbf{C}$	Covariance matrix of search distribution in CMA-ES
$c(\mathbf{s}_t, \mathbf{a}_t)$	A user-defined cost function that maps a state-action pair to a cost value at each time step
$c_\sigma^{nn}(\sigma_i, \sigma_{i+1})$	The effort of performing a climbing transition predicted by neural network
$c_\mathbf{s}^{nn}(\mathbf{s}, \sigma_t)$	The predicted cost function value that movement optimization will result in
$d(\mathbf{v}_i, \mathbf{v}_\mathbf{goal})$	The minimum path length from $\mathbf{v}_i$ to $\mathbf{v}_{\text{goal}}$
$D(\mathbf{x}_i, \mathbf{x}_j)$	A distance metric between $\mathbf{x}_i$ and $\mathbf{x}_j$
$\mathbf{d}_t$	The distances of climber's hands and feet to their target holds
$\mathbb{E}$	Expectation
$f(\mathbf{v}_i)$	The sum of $g(\mathbf{v}_i)$ and $h(\mathbf{v}_i)$ in A* algorithm for evaluating a graph node
$f(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$	Stochastic The state transition dynamics function or probability density
$f(\mathbf{x}_i)$	A fitness function or an objective function evaluating $\mathbf{x}_i$
f_σ	Input feature to a neural network
$f_\mathbf{s}$	Input feature to a neural network
$g(\mathbf{v}_i)$	The actual cost of the path from the start node to a graph node $\mathbf{v}_i$
$G(\tau)$	Return of a trajectory or a simulation episode
$\mathcal{G}_\sigma$	A stance graph or a graph containing all possible assignments of hands and feet to climbing holds
$h(\mathbf{v}_i)$	A heuristic function or an estimated cost from a graph node $\mathbf{v}_i$ to the goal node
$\mathcal{H}$	A set of hold positions
$\mathbf{I}$	Indicator function
$J(\pi_\theta)$	Expected return using policy π_θ
$\mathcal{N}(\mu, \mathbf{C})$	The normal distribution with mean μ and covariance matrix $\mathbf{C}$
$\mathbf{p}$	The average movement of the mean between iterations of CMA-ES

$\mathscr{P}$	A set of paths on the graph
$p(\mathbf{v}_i, \mathbf{v}_j)$	A path on the graph from $\mathbf{v}_i$ to $\mathbf{v}_j$
$P(\tau \mid \pi_\theta)$	Probability of a trajectory given policy
$p^{nn}_{\mathbf{s}}(\mathbf{s}, \sigma_t)$	predicted joint motor actuation parameters used as initial guess for movement optimization
$\mathbf{q}$	Quaternion
$r(\mathbf{s}_t, \mathbf{a}_t)$	Reward function
$\mathscr{S}$	A set of agent's states
$\mathbf{s}_t$	An agent's state at time t
$s^{nn}_{\sigma}(\sigma_i, \sigma_{i+1})$	the success rate of performing a climbing transition
$\mathbf{s}^{\mathbf{cl}}_t$	The climber's body state at time t
$\mathscr{T}$	A set of tree nodes
T	The planning horizon
t	Timestep
t'	Timestep
$\mathscr{V}$	A set of graph nodes
$\mathbf{v}_i$	i^{th} node in a graph
$V^{\pi}(\mathbf{s})$	Value function under policy of π
$\mathbf{v}$	Linear velocity
$w(\mathbf{v}_i, \mathbf{v}_{i+1})$	The weight or cost of the graph edge between nodes $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$
$\mathbf{W}$	A matrix of weight for linear mapping
$\mathscr{X}$	A set of positions
$\mathbf{x}$	Position in three-dimensional space
$\mathbf{x}_{\mathbf{ll}}$	Left foot's position
$\mathbf{x}_{\mathbf{rl}}$	Right foot's position
$\mathbf{x}_{\mathbf{lh}}$	Left hand's position
$\mathbf{x}_{\mathbf{rh}}$	Right hand's position
$\mathbf{y}_i$	Labeled output for classification or regression tasks

1. Introduction

1.1 Background

Animated characters have an important role in video games. The traditional approaches utilize motion capture data from real humans or manually edited animations through keyframes and interpolation curves to synthesize believable and plausible animations. This process is time-consuming, expensive and hard to generalize to many characters. A long-standing goal of animation research is to automate this process, and make it more efficient. The challenging task is to develop methods that synthesize realistic motions for the characters, and make them act naturally, i.e., behave closely to what humans do in the same situation [86, 9, 41, 95].

A major stream of animation research focuses on data-driven kinematic approaches that recombine motions from some database or build a machine learning model that can be conditioned on user input [53, 52, 22]. However, these methods cannot ensure physical plausibility such as contact non-penetration without post-processing. On the other hand, methods have been developed to formulate the synthesis of character animation as simulation of its biomechanics and optimization of character's muscle activations or other actuation parameters such as joint torques over time. This ensures physical plausibility, and the methods have successfully been utilized to synthesize movements such as bipedal locomotion [96], but their extensions to some complex movement skills, e.g., humanoid climbing movements, have remained challenging. This dissertation develops methods to bridge this gap, focusing on novel methods for synthesizing humanoid climbing movements.

Climbing is a type of exercise where the athlete needs to carefully select their hand and feet placements on a steep or even vertical surface. In this thesis, we focus on indoor climbing where the goal is to climb up a climbing wall and reach a so-called top hold. Indoor bouldering is a form of indoor climbing that takes place relatively close to the ground and features short

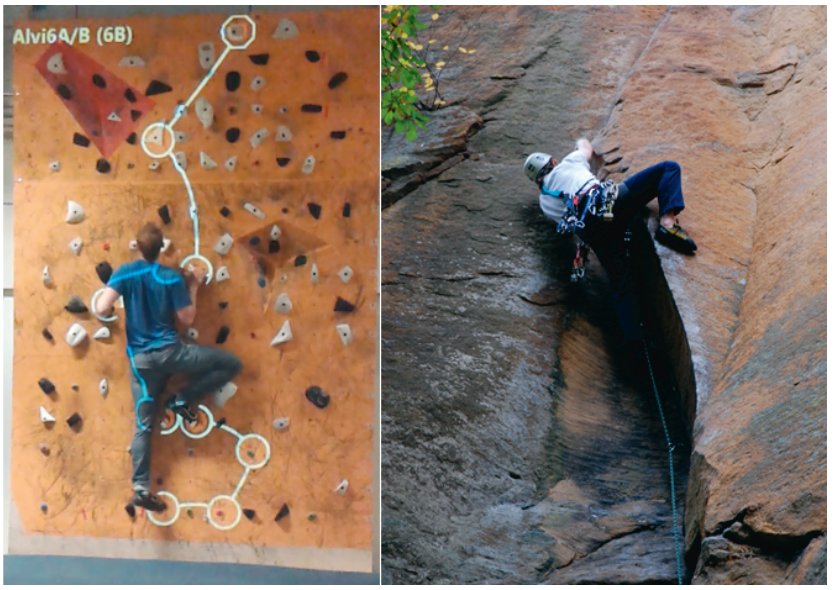


Figure 1.1. "Indoor" versus "Outdoor" climbing environment. Left: An Augmented Reality system [60] highlights the climbing route on an indoor climbing wall. Right: The rock climbing environment at Long Wall in Red River Gorge of Kentucky, USA [120]. Photo by Jarek Tuszyński, © CC-BY-3.0 & GDFL.

but challenging climbing routes (see Fig. 1.1). The sport is rapidly growing and has been approved to the 2020 Tokyo Olympics together with speed and sport climbing [90]. Also, an Augmented Reality (AR) system has been recently developed to make bouldering more exciting and motivating [60]. These developments together with the possible applications of climbing movement synthesis and control in search-and-rescue robotics [16] and computer games [77] motivate the focus of this thesis.

Controlling physically simulated characters has a long history [36]. An increasingly common approach is to tackle simulation control through sampling-based black-box optimization [124]. The methods do not directly use the gradients of the objective function, and only perform the optimization by interacting with the environment through applying actions. This makes it easier to generalize the methods to be used in many applications such as games, animations and robotics [54]. Despite of the successes of these methods in synthesizing humanoid movements, more efficient and robust methods are needed to address more complicated problems with long planning horizons. Traditionally, long planning horizons have been tackled with tree- and graph-based motion planners [110, 125]; however, these are challenging with complex characters with high-dimensional action spaces. This dissertation proposes a hierarchical motion planning approach that combines features of both approaches, progressing through steps of:

Figure 1.2. Climbing movements synthesized using the novel Computer-Aided Imagery applications developed as part of this dissertation. The user plans for hand and foot placements, and an optimization method synthesizes the movements.

- Developing a framework for synthesizing physically-based, plausible, and dynamic climbing movements. The framework includes both high-level graph-based long-horizon planning, and low-level sampling-based short-horizon movement optimization.
- Utilizing the framework to create an innovative human-computer interface for improving mental imagery practice in sports with computer aided imagery (see Fig. 1.2).
- Enhancing the hierarchical motion planning with neural networks, enabling planning more successful, agile, and less effortful movements.
- Formulating climbing movement synthesis as a Reinforcement Learning (RL) problem and training a policy useful for real-time climbing simulation.

1.2 Motivation, Aim and Focus of the Thesis

The aim of this dissertation is to develop a technology for synthesizing complex movements that need a long planning horizon. The developed methods can be utilized to both plan for and synthesize believable and plausible movements. The thesis focuses on humanoid characters which

are challenging to control because of complex movement dynamics, unactuated root, and a large number of degrees of freedom (DoF). Although the developed methods can be generalized to other applications and characters, this dissertation focuses on synthesizing humanoid climbing movements. It builds on and extends three classes of methods: sampling-based optimization, path planning, and reinforcement learning.

Sampling-based optimization: Sampling based optimization methods such as covariance matrix adaptation evolution strategy (CMA-ES) [48] have a long history in character motion synthesis [101, 46, 7]. The basic approach is to sample action sequences, simulate the corresponding state sequences, and then compute the utility of the actions to gradually evolve the sampling distribution such that good actions become more probable. The methods are well suited for short planning horizons and high-dimensional problems but the optimization becomes prohibitively slow at long horizons [47]. This dissertation contributes to optimization methods by dividing the original task into sub-tasks with shorter planning horizons, and then using sampling-based optimization to solve the sub-tasks. Furthermore, the performance of sampling based optimization is enhanced to find more successful and less effortful solutions faster by utilizing better initial guesses predicted by neural networks.

Path planning: Path planning approaches such as artificial potential field, tree search and graph search belong to traditional approaches in controlling the animated characters [23, 80]. In particular, graph search methods have a long history in games and have many practical applications in, e.g., games, robotics and traffic control systems. One of mostly utilized and efficient methods is A* [50]. One major drawback of these methods is that they get slower as the dimensionality of the problem increases. This dissertation contributes to planning methods by introducing a hierarchical motion planning method for synthesizing believable and plausible motions in high dimensional problems. A graph search method is used to plan in lower dimension and produce sub-tasks that can be solved by a low-level optimization method. Additionally, the graph search method is informed by the success rate and effort of the movements predicted by a neural network. As a result, the graph search can plan for more successful and less effortful sequence of movements that can be synthesized by the low-level controller.

Reinforcement learning (RL): RL has recently attracted the focus of many researchers, and the utilization of deep neural networks ("deep RL") makes the approach practical for many applications [6]. Similar to sampling-based optimization, RL algorithms explore random actions and learn to repeat actions with high utility. The difference is that only one action is sampled at a time, conditioned on the observed character state. Typically, a policy network takes the observation as input and outputs sampling distribution parameters such as mean and covariance. RL can require a long training process, but the resulting neural network policies

are typically much more computationally efficient than sampling-based optimization of movements on the fly. Proximal policy optimization (PPO) [104] and soft actor critic (SAC) [44] are two practical RL approaches that belong to on- and off-policy methods, respectively. Off-policy methods reuse previously collected data that can improve sample efficiency over on-policy methods [2]. On the other hand, on-policy algorithms are often more stable and easier to use [51]. Despite advances of deep RL in synthesizing humanoid movements, RL had not been utilized to synthesize physically based humanoid climbing movements before this dissertation. One reason may be that the random action exploration that RL methods utilize is less likely to produce successful climbing movements than, e.g., bipedal walking movements. Climbing movements need multilimb coordination and precision, and failures can be more dramatic than in many other tasks. This thesis proposes an RL approach for synthesizing humanoid climbing movements and develops better exploration strategies that speed up the training process.

1.3 Publications

This dissertation comprises the following four publications.

Publication I develops a hierarchical motion planner that is suitable for humanoid movement synthesis with long planning horizons. The paper was the first to solve the combined problem of planning humanoid climbing strategy and dynamic, physically based multilimb climbing movements. The method combines graph search with sampling based black-box optimization. The graph search is responsible for dividing the original problem into sub-problems that are solvable by the sampling based optimization. Publication I utilizes two sampling-based optimization methods, CMA-ES [48] and control particle belief propagation (C-PBP) [46], and compares their performance against each other.

Publication II introduces a novel interface for controlling a simulated climber and investigates how this could enable a novel form of mental practice (imagery) for sports and exercise augmented with intelligent simulation. Users play the role of graph search of Publication I. They can guide their avatar by selecting target hold positions for the AI character's hands and feet, and target rotation for its torso. Once the target posture is selected, C-PBP is utilized to synthesize the movements for the user. The users can alter their choices and explore various movements on a photogrammetry replica of a real climbing wall. After solving a climbing problem in simulation, the users try the same route on the real wall. The paper contributes an analysis of how this novel training process affects climbing performance and experience.

Publication III improves on Publication I by introducing deep neural

networks to both the high-level path planner and low-level controller in order to synthesize more dynamic and successful movements. The neural networks are trained to predict low-level movement optimization outcomes such as success rate, movement effort, and action sequence. The neural networks are used to inform the sampling distribution of the movement optimization, and for weighting the graph edges in the high-level path planner. As a result, the system produces higher quality movement in less time and is able to expand the movement repertoire of Publication I with dynamic leaps.

Publication IV proposes a reinforcement learning approach for synthesizing climbing movements in real-time. The climbing problem is formulated such that the RL approach replaces the low-level controller in the hierarchical motion planning framework of this dissertation. It is observed that basic RL training where the character always starts from a T-pose in front of the wall has problems in exploring the state space. A novel training episode initialization technique is then proposed to improve the exploration. Furthermore, the paper tests the effect of decision frequency on RL performance. Compared to other publications, Publication IV reduces the processing time that is needed to synthesize climbing movements. However, the quality of the movements synthesized by RL is not yet equal to Publication III.

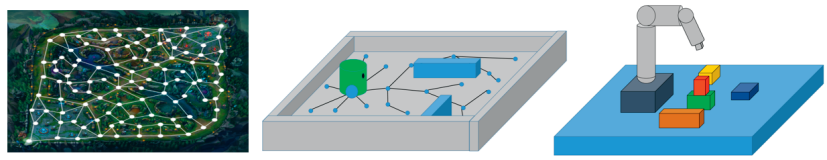
1.4 Outline of the Thesis

The next two sections provide essential background for understanding this dissertation and review some of the state of the art methods. First, Section 2 reviews path planning methods which are used in Publication I, Publication III for planning a sequence of movements. Section 2 also briefly reviews tree search, graph search and neural network guided methods for path planning. Section 3 reviews the sampling-based movement optimization methods utilized in Publication I, Publication II, and Publication III. In addition, it reviews supervised learning used in Publication III as well as reinforcement learning approaches, providing a necessary foundation for Publication IV.

Section 4 discusses the algorithms and applications that have been developed in this dissertation. It provides the details on implementations and experiments done in each publication. Finally, Section 5 discusses the contributions of the dissertation, and introduces further directions for future works.

2. High-Dimensional Path Planning

Path planning is a fundamental and demanding problem that has applications in robotics [112], computer games [23], urban traffic and so forth. Fig. 2.1 shows some of common applications in path planning. Path planning methods solve the problem of getting the agent from a starting point in the configuration space to a goal point specified by the user under certain constraints. Constraints, depending on the application at hand, may consist of safe or collision free path, low energy consumption, low risk of failing, and so on. The optimal path depends on user-defined constraints and might be the path with shortest euclidean distance between two points or the one with the lowest risk, energy or even the combined weighted sum of multiple criteria. Many variants of path planning methods have been developed to find optimal or near-optimal paths for various tasks.



a *b* *c*

Figure 2.1. Three examples of common applications in path planning. *a*) graph based search used in modern games, here illustrated on the map of League of Legends [32]. *b*) tree search for e-puck robot [112] navigation. *c*) a robot arm.

Although path planning methods are capable of solving problems such as the ones illustrated in Fig. 2.1, increasing the dimensionality of the problem (the agent's state space) makes path planning harder. In order to mitigate the curse of the dimensionality, hierarchical motion planning can be utilized [16, 19, 20]. This thesis uses a two-level hierarchy, where a high-level path planner approaches the problem in abstract form. Rather than solving the problem in the full state or configuration space, the high-level planner uses a lower dimensional representation by neglecting parts of the state space. The high-level planner produces a sequence of points in the lower-dimensional space, assuming that a low-level controller or

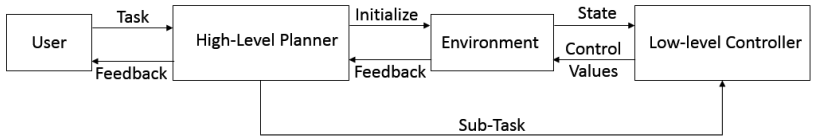


Figure 2.2. The high-level path planner and low-level controller hierarchy used in this dissertation. The flow of commands and feedback passing between user, high-level path planner, environment, and low-level controller is demonstrated.

movement optimizer can produce transitions between the points. Figure 2.2 illustrates the model of high-level planner and low-level controller interaction utilized in this thesis.

Methods for the low-level movement optimization are introduced in Section 3, and more details of the control hierarchy used in this thesis are provided in Section 4. The rest of this section reviews the relevant background for the high-level path planning. First, we review how path planning can be approached as tree or graph search since they are utilized in Publication I and Publication III. Second, as Publication III informs the path planning with neural networks, we briefly discuss the applications of neural networks in path planning

2.1 Tree Search Methods

Rapidly exploring random trees (RRT) were introduced first by LaValle in 1998 [72] to overcome the deficiency of classical path planning methods such as Artificial potential fields (APF) [63] in using knowledge from environment, being limited to low-dimensional problem, and getting stuck in local minima. APF uses attractive and repulsive fields to guide the agent toward the goal point while pushing it away from obstacles in real-time, but the method suffers from getting stuck in local-minima. On the other hand, RRT and its extensions have probabilistic completeness, i.e., as the number of samples goes to infinity the probability of finding the solution approaches one. Furthermore, the tree structure makes it easy to find a collision-free path toward a desired state by just following the path from the points on tree branches to the tree root. Fig. 2.3 illustrates the evolution of RRT methods, starting from the original paper [72], then illustrating the optimality of paths found using RRT* [61], how Informed RRT* focuses sampling of states [33], and finally, how RT-RRT* [87] combines features of previous methods for real-time path-planning in a dynamic environment.

The original RRT [72] explores the environment by expanding a tree in the collision free state space in an offline manner. As summarized in Algorithm 1, the tree starts growing from initial state $\mathbf{s}_{\text{init}}$ by 1) Sampling a point $\mathbf{x}_\mathbf{r}$ in the environment, 2) Finding the closest point ($\mathbf{s}_{\text{closest}}$) in the tree

Algorithm 1 Original Rapidly Exploring Random Trees

1: $\mathcal{T} = \{\mathbf{s}_{\text{init}}\}$
2: **for** iteration $= 1, 2, ..., \mathcal{I}_{\max}$ **do**
3: $\quad \mathbf{x}_{\text{r}} = \texttt{Unifrom}(\mathcal{X})$ //sample a random point in environment
4: $\quad \mathbf{s}_{\text{closest}} = \texttt{FindClosest}(\mathbf{x}_{\text{r}}, \mathcal{T})$ //find closest state to random sample
5: $\quad \mathbf{s}_{\text{n}} = \texttt{Move}(\mathbf{s}_{\text{closest}}, \mathbf{x}_{\text{r}})$ //Move toward sample
6: $\quad \mathcal{T} = \mathcal{T} \cup \mathbf{s}_{\text{n}}$ //Add new state to the tree
7: **end for**

to $\mathbf{x}_{\text{r}}$, 3) Attempting to move from $\mathbf{s}_{\text{closest}}$ toward $\mathbf{x}_{\text{r}}$ under agent's dynamic constraints, and 4) Reaching $\mathbf{s}_{\text{n}}$ which will be added to the tree for further expansion. The tree expansion is demonstrated in Fig 2.3a. Although in the original paper, the domain of $\mathbf{x}$ and $\mathbf{s}$ is the same, but in general, $\mathbf{x}$ can also denote some abstracted goal reached by a movement controller or optimizer, as in the control hierarchy utilized in this thesis.

The original RRT (Algorithm 1) is an offline method that utilizes an uniform sampling (Line 3), and the tree root (Line 1) is fixed during the tree expansion. There are many variants of RRT that improve performance in many aspects such as finding optimal solutions, path planning in online and real-time manner, and re-planning in dynamic environments.

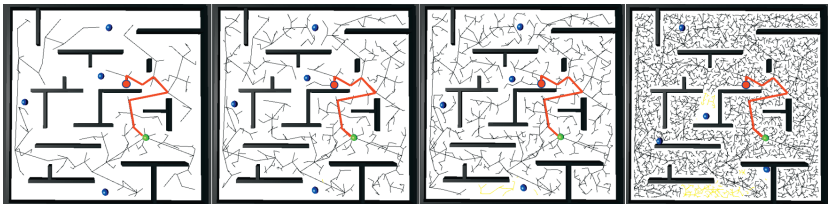
Rewiring: RRT* [61] introduces a rewiring process to tree expansion, demonstrated in Fig. 2.3b. Rewiring is the process of modifying the parent of a node $\mathbf{s}_{\text{i}}$ to a newly added node $\mathbf{s}_{\text{n}}$ when by passing from $\mathbf{s}_{\text{n}}$, $\mathbf{s}_{\text{i}}$ has lower cost compared the old parent. This happens by simple modifications to Algorithm 1 which are 1) Finding a set of neighbour nodes $\mathcal{S}_{\text{near}}$ close to $\mathbf{s}_{\text{n}}$, and 2) rewiring the nodes $\mathbf{s}_{\text{i}} \in \mathcal{S}_{\text{near}}$ when the condition is met. This alteration guarantees finding an optimal solution, but the method can be computationally expensive.

Informed Sampling: Informed RRT* focuses the sampling of $\mathbf{x}_{\text{r}}$ to improve computational efficiency [33]. The sampling at Algorithm 1 Line 3 is performed as:

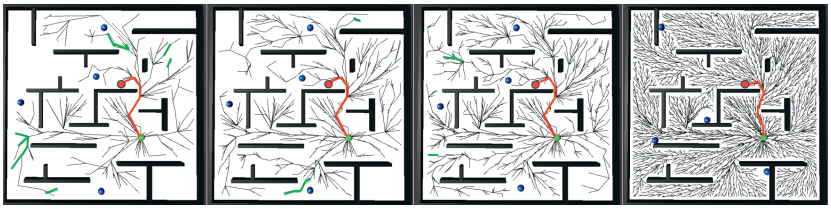
$$\mathbf{x}_{\text{r}} = \begin{cases} \texttt{Ellipse}(\mathcal{X}) & \text{path from start to goal is found} \\ \texttt{Uniform}(\mathcal{X}) & \text{otherwise} \end{cases} \tag{2.1}$$

The sampling is uniform as in basic RRT until a path from the start state to the goal point is found. After that, $\mathbf{x}_{\text{r}}$ is sampled inside an ellipse with the starting and goal points as the focal points and radius equal to the length of the current path. As the algorithm continues the length of the current path decreases, resulting in more focused sampling. This is demonstrated in Fig. 2.3c.

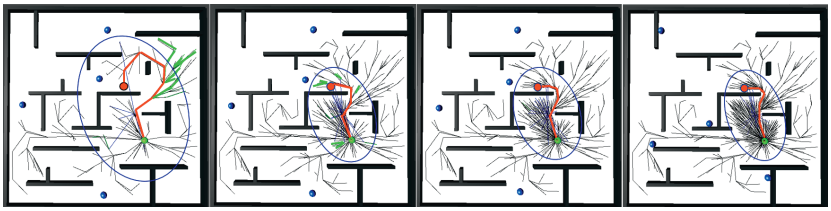
Online Planning: RT-RRT* [87], a variant of Informed RRT* and RRT*, gradually changes the structure of the tree to allow the agent to have a path to different points in the environment at all times (see Fig. 2.3d). The ability of replanning and reacting to the changes in the environment



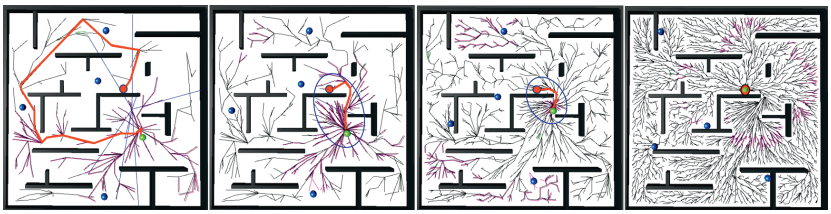
(a) Original RRT [72]. The yellow lines (area) are blocked by the dynamic obstacles.



(b) RRT* [61]. As the algorithm progresses, the tree is rewired to find more optimal paths. The green lines show the rewired nodes.



(c) Informed RRT* [33]. The method focuses the sampling of states inside an ellipse defined by the start and goal points and the path found so far. The green lines show the rewired nodes, and the ellipse is shown in blue.



(d) RT-RRT* [87] combines and extends the features of previous algorithms for real-time operation in a dynamic environment. The method adds non-uniform rewiring shown in purple. Whereas RRT* samples the tree nodes to rewire uniformly, RT-RRT* also progresses systematically away from the tree root, because the nodes around the moving agent are more important to rewire first.

Figure 2.3. Evolution of RRT algorithms. The agent and dynamic obstacles are demonstrated by green and blue spheres, respectively. The red lines demonstrate the path toward the goal point. The tree root is located at the agent.

is gained by altering Algorithm 1 in three ways: 1) Rewiring the tree at random points and by progressing systematically away from tree root (Fig.

2.3d), 2) Applying informed sampling, and 3) Changing the tree root to be located on the agent. These three components are the essence of RT-RRT*. Additionally, informed sampling helps to focus the sampling of $\mathbf{x}_r$ on a smaller part of the tree in order to rewire and enhance the current path faster. The informed sampling area gets smaller and more focused as the agent gets closer to the goal point since the tree root is located on the agent.

Sparse Planning: There exist methods that make RRT more sample efficient, e.g., [34] improves the efficiency of Algorithm 1 by altering Line 5 and using artificial potential field (APF) to move the agent multiple steps toward a random point instead of one step. Publication I and Publication III follow a sparse planning approach, utilizing a combination of graph search and movement optimization to guide tree expansion. Motivated by this, the following section provides an overview of graph search methods. It should be noted that the author also tried using the original RRT algorithm, but it was clearly not efficient enough in the case of a humanoid climber with a high-dimensional state space. A more efficient approach was needed to inform the sampling and tree expansion than simply moving one step toward a randomly explored state in Lines 3-5 of Algorithm 1.

2.2 Graph Search Methods

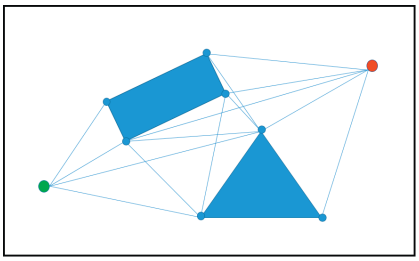
Graph search belongs to one of the early methods developed for path planning. A common way of path planning in an environment is to 1) construct a graph connecting all obstacle-free states together, and 2) find the shortest path or k-shortest paths (see Section 2.2.2) by applying a graph search method. In a graph, the environment is represented by sets of nodes $\mathcal{V}$ and edges $\mathcal{E}$. Each node can have one or more parents and children which is determined by the adjacency or connectivity matrix. The shortest path problem is formulated as finding a path $(\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_{\text{goal}})$ subject to having:

$$\min \sum_{i=0} w(\mathbf{v}_i, \mathbf{v}_{i+1}) \tag{2.2}$$

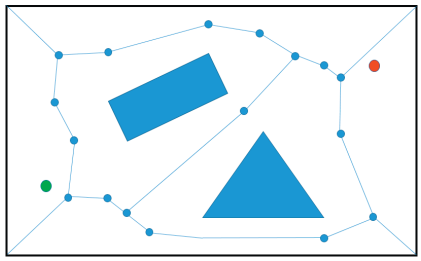
where $w(\mathbf{v}_i, \mathbf{v}_{i+1})$ is the weight on the graph edge when transiting from $\mathbf{v}_i$ to $\mathbf{v}_{i+1}$, $\mathbf{v}_i$ is adjacent to $\mathbf{v}_{i+1}$, and $\mathbf{v}_{\text{goal}}$ is the terminal or goal node. $w(\mathbf{v}_i, \mathbf{v}_{i+1})$ can be a user-defined value or it can be adjusted, e.g., by the possibilities of the actions which can lead to higher success and lower processing time in synthesizing movements [42].

2.2.1 Building the Graph

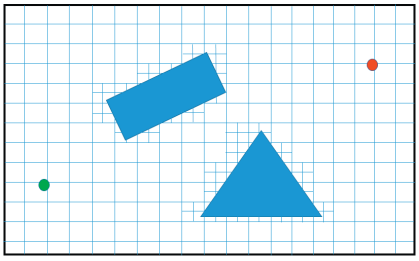
Classical methods such as grid graph [118, 111], Voronoi diagram [115, 10, 35] and visibility graph [56] exist that benefit from geometrical infor-



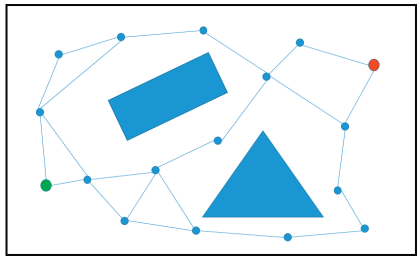
(a) Graph construction using visibility method. The start, goal, and the corners of the obstacles build the graph nodes. The line between nodes that are not passing through obstacles create graph edges.



(b) Graph constructed using Voronoi diagram method. The lines are selected to have maximum distance between the obstacles. The lines and their intersections create graph edges and nodes, respectively.



(c) Graph constructed by grid graph method. Each tile is a node and has an edge connection to its neighbour nodes. Smaller tiles are used for planning around obstacles.



(d) Graph construction by probabilistic roadmap. The nodes are sampled uniformly in the environment and obstacle free nodes and edges are chosen to build the graph.

Figure 2.4. Comparison of different methods in building the graph structure. The green and red spheres demonstrate starting and ending points, respectively.

mation of the environment and construct a roadmap in low-dimentional spaces such as 2D space with polygonal obstacles. Fig. 2.4 demonstrates differences between these methods. The grid graph discretizes the continuous environment into connected equally sized tiles. In grid graph, tiles represent the graph nodes and they are connected to their neighbours. Voronoi diagram outputs the safest regions between the obstacles in the environment [35]. The safest location between two obstacles is the line between them. The intersection of the lines builds graph nodes of Voronoi diagram and the adjacent nodes are connected to each other by the lines (see Fig. 2.4b). The visibility graph is built in the environment by connecting straight lines between the corners of the obstacles. The obstacle-free lines that are not passing through obstacles belong to graph edges. The graph nodes are the corners of the obstacles, and adjacent nodes are the connected ones (see Fig. 2.4a). There also exist methods such as probabilistic roadmap (PRM) [62] and its variants that build the

graph by sampling the points uniformly in the environment and connecting them with obstacle-free lines (see Fig. 2.4d). However, when the number of vertices and edges in the graph is limited, it is common to build the graph by enumerating the sub-solutions.

Algorithm 2 Original A* Graph Search

$\mathcal{V}_{\text{open}} = \{\mathbf{v}_{\text{init}}\}$ //list of nodes to be visited
$g(\mathbf{v}_i) = \infty \ \forall \mathbf{v}_i \in \mathcal{V}$, $g(\mathbf{v}_{\text{init}}) = 0$ //the actual cost for reaching the node
$h(\mathbf{v}_i)$ //the estimated cost for reaching the goal node from $\mathbf{v}_i$
$w(\mathbf{v}_i, \mathbf{v}_j)$ //the weight on the graph edge for transiting from $\mathbf{v}_i$ to its adjacent node $\mathbf{v}_j$
while $\mathcal{V}_{\text{open}} \neq \{\}$ **do**
    $\mathbf{v}_{\text{best}} = \underset{\mathbf{v} \in \mathcal{V}_{\text{open}}}{\operatorname{argmin}} f(\mathbf{v})$ s.t. $f(\mathbf{v}) = g(\mathbf{v}) + h(\mathbf{v})$
    `Remove` $\mathbf{v}_{\text{best}}$ from $\mathcal{V}_{\text{open}}$
    if $\mathbf{v}_{\text{best}} == \mathbf{v}_{\text{goal}}$ **then**
        `Return` the path $(\mathbf{v}_{\text{init}}, \mathbf{v}_1, ..., \mathbf{v}_{\text{goal}})$
    end if
    for each $\mathbf{v}_c$ in `Neighbour`$(\mathbf{v}_{\text{best}})$ **do**
        $g_{\text{new}}(\mathbf{v}_c) = g(\mathbf{v}_{\text{best}}) + w(\mathbf{v}_{\text{best}}, \mathbf{v}_c)$ //new cost if passing through $\mathbf{v}_{\text{best}}$
        if $g(\mathbf{v}_c) \leq g_{\text{new}}(\mathbf{v}_c)$ **then**
            `Continue`
        end if
        //updating tree structure if found a node with lower cost
        $\mathcal{V}_{\text{open}} = \mathcal{V}_{\text{open}} \cup \mathbf{v}_c$
        $g(\mathbf{v}_c) = g_{\text{new}}(\mathbf{v}_c)$
        `Parent`$(\mathbf{v}_c) = \mathbf{v}_{\text{best}}$ //path to $\mathbf{v}_c$ is found by following parents
    end for
end while
`Return` no path is found

2.2.2 Graph Search Methods

There exist many traditional methods that can search for the shortest path on a given graph based on the cost described in Eq. 2.2. Breadth-first search (BFS), depth-first search (DFS) [26] and Dijkstra [24] belong to early methods developed to search the graph. These methods do not use any heuristics in exploring the graph and finding a solution. BFS prioritizes on searching the neighbours first while DFS explores the branches first until reaches leaf nodes (nodes with no children) before backtracking. Dijkstra finds shortest paths to all graph nodes using a priority queue that enables the method to explore the nodes with lower cost first and update their neighbours' costs. Another approach to search the graph is to utilize heuristics. The heuristics guide the exploration of the nodes to

avoid traversing the whole graph and make the graph search faster. A* [50] is a popular example of these methods that utilizes heuristics in the graph search to solve a problem.

The following paragraphs reviews some of notable methods in graph searching. Furthermore, this dissertation reviews k-shortest path planning methods since the optimal strategy for solving a climbing route depends on each climbers flexibility, reach, and so on. Thus, for coaching purposes, it would be good that an AI climber can demonstrate multiple good solutions. This can be implemented by finding k-shortest paths in the high-level path planning graph.

Traditional Planning: Algorithm 2 describes A* approach that builds a tree structure on the graph in an offline manner. The method starts building the tree from $\mathbf{v}_{\text{init}}$. At each iteration, the best node $\mathbf{v}_{\text{best}}$ is selected from the open list $\mathcal{V}_{\text{open}}$ to be parent of its neighbour $\mathbf{v}_c$ if the node has lower cost by passing from $\mathbf{v}_{\text{best}}$. The neighbour node is added to the open list only if its parent has been changed to propagate the alteration through the whole graph. The process continues until it reaches the goal node or the open list gets empty. The performance of the method depends highly on the heuristics function $h(\mathbf{v})$, which should be admissible, i.e. the function values cannot overestimate the actual cost. Otherwise, the method might return a sub-optimal solution. One common heuristic is Euclidian distance to the goal.

Online Planning: There exists methods to increase the performance of A* search method. D* lite [67] is the online version of A* method that allows the method to react to changes in the environment. D* lite re-plans a path starting from current location of the agent while considering the changes in the environment when the current path to the goal node is not valid anymore, e.g. an obstacle blocks the current path.

K-Shortest Paths: k-shortest path planning is the generalization of the shortest path planning problem where the problem is to find multiple shortest paths between two graph nodes. Its applications include, e.g., robot path planning, traffic control, time scheduling, and network connection routing. In most of these applications, the user needs to see various solutions to a problem for the sake of comparison or having the next best solution ready for fast querying.

A good review of state-of-the-art methods for finding k-shortest paths on the graph can be found in [29, 28]. [127] finds the paths assuming the paths are simple, i.e. no loop exists in the path. [28] removes this assumption and returns the paths by proposing a heap structure for storing the graph paths and limiting the graph degree. In this dissertation, A* prune [79] is utilized which is a general algorithm for returning k-shortest paths between two terminal graph nodes. The method, described in Algorithm 3, applies small modifications to the original A* method to return multiple paths. It uses the heuristics function calculated by Dijkstra algorithm, and

Algorithm 3 A* Prune

Run Dijkstra to calculate $d(\mathbf{v}_i, \mathbf{v}_{\text{goal}})$ for all $\mathbf{v}_i \in \mathcal{V}$ //calculate the minimum path length from $\mathbf{v}_i$ to $\mathbf{v}_{\text{goal}}$
$\mathscr{P}_{\text{open}} = \{p(\mathbf{v}_{\text{init}}, \mathbf{v}_{\text{init}})\}$
$g(p(\mathbf{v}_{\text{init}}, \mathbf{v}_{\text{init}})) = 0$
$h(p(\mathbf{v}_{\text{init}}, \mathbf{v}_{\text{init}})) = d(\mathbf{v}_{\text{init}}, \mathbf{v}_{\text{goal}})$
$\mathscr{P}_{\text{found}} = \{\}$
while $\mathscr{P}_{\text{open}} \neq \{\}$ and $|\mathscr{P}_{\text{found}}| < k$ **do**
    $p(\mathbf{v}_{\text{init}}, \mathbf{v}_u) \leftarrow$ the first path from $\mathscr{P}_{\text{open}}$
    `Remove` $p(\mathbf{v}_{\text{init}}, \mathbf{v}_u)$ from $\mathscr{P}_{\text{open}}$
    if $\mathbf{v}_u == \mathbf{v}_{\text{goal}}$ **then**
        `Insert` $p(\mathbf{v}_{\text{init}}, \mathbf{v}_u)$ in $\mathscr{P}_{\text{found}}$
        `Continue`
    end if
    for each $\mathbf{v}_c$ in `Neighbour`$(\mathbf{v}_{\text{u}})$ **do**
        $g(p(\mathbf{v}_{\text{init}}, \mathbf{v}_c)) = g(p(\mathbf{v}_{\text{init}}, \mathbf{v}_u)) + w(\mathbf{v}_{\text{u}}, \mathbf{v}_c)$
        $h(p(\mathbf{v}_{\text{init}}, \mathbf{v}_c)) = d(\mathbf{v}_c, \mathbf{v}_{\text{goal}})$
        $f(p(\mathbf{v}_{\text{init}}, \mathbf{v}_c)) = g(p(\mathbf{v}_{\text{init}}, \mathbf{v}_u)) + h(p(\mathbf{v}_{\text{init}}, \mathbf{v}_u))$
        `Insert` $p(\mathbf{v}_{\text{init}}, \mathbf{v}_c)$ in $\mathscr{P}_{\text{open}}$ based on f in ascending order
    end for
end while
`Return` $\mathscr{P}_{\text{found}}$

keeps a list of all possible paths to explored graph nodes, sorted based on heuristics costs.

2.3 Neural Networks in Path Planning

One of earliest methods that utilizes neural networks to plan paths for intelligent agents is topologically-ordered map neurons [39, 12, 126]. The method works by forming a lattice of neurons over obstacles or the whole 2D environment, and the activation of neurons plan a path for the agents and guide them towards the target. Using a topologically-ordered map, [74] plan paths in a 2D multi-agent environment, later extended to a cooperative hunting task by [88].

There exist methods that form repulsive and attractive fields for guiding agents using neural networks. [71] introduces a neural network that has an obstacle description of a stationary 2D environment in its hidden layer and outputs a repulsive penalty function to solve the local minima problem of the artificial potential field method in navigating mobile robots. [25] uses ant colony optimization to plan a global path with least repulsive penalty function produced by a neural network that has obstacle information in its

hidden layers.

As the dimensionality of the problem increases, neural networks are typically utilized in some form of a control hierarchy because the above-mentioned approaches become less feasible. [96, 30, 31] use hierarchical reinforcement learning approaches where various low-level policy networks learn primitive motor skills such as running, walking, and etc., and a high-level policy network learns to control the low-level policies in order to satisfy the target direction and gait type in the navigation tasks.

3. Physically-Based Character Control

The automation of synthesizing animation is a long standing goal of computer animation research. The necessary element of this task is the ability to control the agent interacting with an environment. Many of traditional methods make use of recorded motions from humans performing various movements [38, 69, 124]. They utilize key-frames [68], interpolation [21], and motion re-targeting [55] to control the animated character (see Fig. 3.1). In spite of the successes of these methods in synthesizing animation, they are time consuming, expensive and cannot be generalized easily to other characters.

Figure 3.1. An example of retargeting motion captured walking animation to two different characters, the Mixamo animation system [58].

Another approach is to simulate character's behavior and its interactions with a physical environment. By applying torque and force control actions to the physically simulated character, one can ensure physical plausibility of the resulting movement, within the limits of the simulator. In order to have simulated behaviours close to a human's, many control approaches formulate the animation problem as the optimization of muscle activations or joint torques over time while simulating character's biomechanics [37]. Although there exist methods that can synthesize physically plausible and

believable animations for simple tasks such as bipedal locomotion, the problem has remained unsolved for many complex behaviours. In this dissertation, we study the synthesis of climbing movements where the following factors increase the difficulty of the optimization problem:

- The problem has high-dimensionality in both action and state spaces.
- Synthesizing a wide variety of movements with a different dynamics, ranging from controlled and precise reaching movements to high-velocity leaps.
- Planning for long sequences of movements is needed to complete the task.

A hierarchy of high-level planner and low-level movement optimizer can be utilized to mitigate the problem. The high-level planner plans for a sequence of sub-tasks in a lower dimension (see Chapter 2), and the low-level movement optimizer tries to solve each sub-task.

This dissertation utilizes sampling based optimization and reinforcement learning methods as the low-level movement optimizer to solve the planned sub-tasks, and to control the character in a physical environment. The investigated methods consider the movement optimization as a black-box problem, and the goal is to minimize the cost or maximize the reward in the optimization only by interacting with the environment. The cost or reward function is a user-defined weighted combination of various terms such as effort minimization and goal attainment.

The outline of this chapter is as follows. Section 3.1 summarizes control particle belief propagation (C-PBP) and covariance matrix adaptation evolution strategy (CMA-ES) methods as they are utilized in Publication I-Publication III for synthesizing climbing movements. Section 3.2 provides an introduction to common supervised learning methods because Publication III utilizes an iterative approach to 1) train neural networks in a supervised manner from outputs of the optimization method, and 2) enhance the sampling-based optimization method by providing better initial sampling distribution based on the neural network predictions. Finally, Section 3.3 provides a brief introduction on reinforcement learning because Publication IV formulates the synthesizing of climbing movements as a reinforcement learning problem, and utilizes proximal policy optimization (PPO) [104] with generalized advantage estimation (GAE) [103] to train a climber agent for performing various climbing movements on various climbing problems.

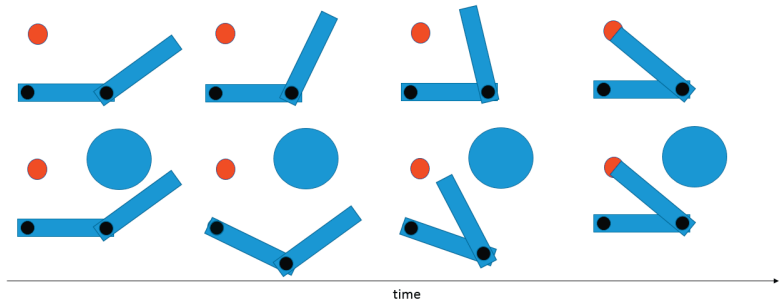


Figure 3.2. Two different trajectories of moving arm to a target (red point). The bottom trajectory is avoiding the circular obstacle while the top trajectory has a free path.

3.1 Trajectory Optimization for Character Control

In mathematical optimization, the solutions to a problem are described by one or more variables; optimization methods search for the combination of variable values that maximizes or minimizes an objective function that defines the value or cost of a solution. In trajectory optimization for character control, the agent starts at the initial state $\mathbf{s}_0$, and the goal is to find a sequence of actions ($\mathbf{a}_0$, $\mathbf{a}_1$, ..., $\mathbf{a}_T$) that minimizes a user defined cost function (see Fig. 3.2). The sequence of state-action pairs as ($\mathbf{s}_0, \mathbf{a}_0$, $\mathbf{s}_1, \mathbf{a}_1$, ..., $\mathbf{s}_T, \mathbf{a}_T$) create a trajectory for the agent, and the trajectory optimization can be formulated as:

$$
\begin{aligned}
(\mathbf{a}_0^*, \mathbf{a}_1^*, ..., \mathbf{a}_T^*) &= \underset{\mathbf{a}_0,...,\mathbf{a}_T}{\operatorname{argmin}} \sum_{t=0}^{T} c(\mathbf{s}_t, \mathbf{a}_t) \\
\text{subject to:} \quad \mathbf{s}_{t+1} &= f(\mathbf{s}_t, \mathbf{a}_t) \\
g_i(\mathbf{s}_t, \mathbf{a}_t) &\geq 0 \quad i \in \{1, ..., m\} \\
h_j(\mathbf{s}_t, \mathbf{a}_t) &= 0 \quad j \in \{1, ..., n\}
\end{aligned}
\tag{3.1}
$$

where ($\mathbf{a}_0^*$, $\mathbf{a}_1^*$, ..., $\mathbf{a}_T^*$) denote the sequence of optimal actions resulting in minimum cost, $c(\mathbf{s}_t, \mathbf{a}_t)$ is a user defined cost function mapping state-action pair to a cost value at each time step, and $f(\mathbf{s}_t, \mathbf{a}_t)$ describes the movement dynamics. $h_j(\mathbf{s}_t, \mathbf{a}_t)$ and $g_i(\mathbf{s}_t, \mathbf{a}_t)$ denote equality and inequality constraints at each time step on agent's states and actions.

The methods that exist for trajectory optimization can be categorized into two groups: 1) Model based, and 2) Model free. Model based methods utilize the dynamics function $f(\mathbf{s}_t, \mathbf{a}_t)$ to calculate optimal actions in the trajectory. Common model based approaches include iterative linear quadratic regulator (iLQR) [75] and differential dynamic programming (DDP) [81] which have been demonstrated to scale from simple dynamical systems such as an inverted pendulum to at least some humanoid

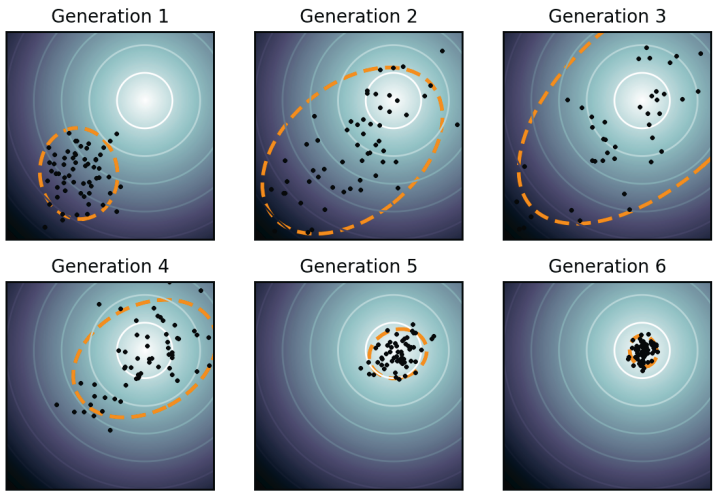


Figure 3.3. Generations (iterations) of CMA-ES algorithm in a simple 2D problem (Public Domain, [105]).

movements [116, 117]. However, contact discontinuities of complex movements can cause multimodality [45] that iLQR and DDP cannot handle as gradient-based local optimization methods. Hence, this thesis focuses on model-free sampling-based optimization methods suitable for multimodal optimization landscapes. Such model-free methods assume that $f(\mathbf{s}_t, \mathbf{a}_t)$ can only be evaluated pointwise by executing an action using simulator, and a closed-form expression of the dynamics is not available. Further, it is assumed that there are no equality or inequality constraints. This simplifies the optimization; constraint handling in sampling-based optimization and reinforcement learning is an active area of research [5, 70, 11]

3.1.1 Covariance Matrix Adaptation Evolution Strategy

One of the popular and highly utilized trajectory optimization methods is covariance matrix adaptation evolution strategy (CMA-ES). The method is an evolutionary technique that belongs to stochastic, derivative-free methods, and it is suitable for numerical optimization of non-linear or non-convex continuous optimization problems [48]. CMA-ES is based on storing the information about current solution population to the mean vector and the covariance matrix parameters. These parameters define a Gaussian distribution that is used to sample candidate solutions. Algorithm 4 highlights main points of CMA-ES method, and Fig. 3.3 demonstrates the iterations of the method on simple problem of two variables.

CMA-ES iteratively samples from a Gaussian "search distribution", and refits the distribution to the best samples. CMA-ES utilizes the following

Algorithm 4 High-level summary of CMA-ES

1: **for** iteration = $1, 2, ...$ **do**
2: Draw samples $\mathbf{x}_i \sim \mathcal{N}(\mu, \mathbf{C})$ for $i \in 1, ..., N$
3: Evaluate samples using fitness function $f(\mathbf{x}_i)$
4: Sort the samples based on $f(\mathbf{x}_i)$ and compute weights w_i based on the ranks such that best samples have highest weights.
5: Update μ and $\mathbf{C}$ using the samples and weights.
6: **end for**

heuristics in each iteration:

- **Sample pruning**: Using the default CMA-ES parameters, the worst 50% of samples are pruned and their weights are set to 0. The mean μ is updated as a weighted average of the samples.
- **Rank-μ update**: Rank-μ update stores information about the population of the current iteration, and CMA-ES first updates the covariance and only then updates the mean [48]. This has the effect of elongating the exploration distribution along the best search directions, as opposed to first updating the mean.
- **Evolution path**: The so-called evolution path heuristic maintains an estimate of the average movement of the mean between iterations, denoted $\mathbf{p}$, and adds $\mathbf{C}_1 \propto \mathbf{p}\mathbf{p}^T$ to the covariance. When CMA-ES progresses along a continuous slope of the fitness landscape, $\| \mathbf{p} \|$ is large, and the covariance is elongated and exploration is increased along the progress direction. Near convergence, when CMA-ES zigzags around the optimum in a random walk, $\| \mathbf{p} \| \approx 0$ and the evolution path heuristic has no effect.

these heuristics are used to speed up the progress and avoid premature convergence by elongating the distribution in the progress direction (see Fig. 3.3). A complete introductory on the choice of weights and coefficients in addition to mathematical formula for updating the evolution path and rank-μ covariance matrix can be found in [48].

3.1.2 Control particle belief propagation

Control particle belief propagation (C-PBP) [46] is a general-purpose model predictive control (MPC) algorithm that utilizes a multi-modal, gradient-free sampling method to effectively perform simultaneous path finding and smoothing in high-dimensional spaces. The method is suitable for online synthesis of interactive and physically plausible humanoid movements, including balancing, recovery from both small and extreme disturbances, reaching, and fully steerable locomotion in an environment with obstacles. In online control, the method is run one iteration per frame, assuming that

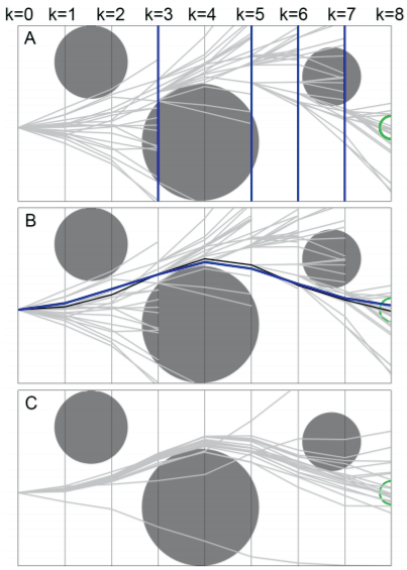


Figure 3.4. Steps of C-PBP algorithm in a simple planning problem demonstrated by [46]. Rollout timestep is denoted by k. A) simulating the rollouts forward, terminating and forking trajectories at the timesteps marked with vertical blue lines. B) determining the best sequence (black) and a smoothed and improved candidate trajectory (blue). C) simulating forward in the next iteration, informed by the results of the previous iteration.

the simulated system displays only small changes between frames. At each frame, the method simulates multiple rollouts forward in time from the current state, and the agent then takes the first action of the best rollout. The main steps of the method are demonstrated in Fig. 3.4, and can be summarized as follows:

- **Exploration**: Sample actions and explore states that result in lower cost based on the provided priors in the form of Gaussian mixture model (GMM). This forms a tree as demonstrated in Fig. 3.4A.
- **Refinement**: Smoothing the optimal path obtained in the previous phase by recursive backwards local refinement. This is shown in Fig. 3.4B.
- **Update**: Update the GMM model such that states and actions with lower cost values gets higher probability to be selected. The result is shown in Fig. 3.4C.

The method has also been simplified and augmented with machine learning to improve both optimization efficiency and movement quality [99, 100].

The main difference to CMA-ES is that whereas each sample in CMA-ES trajectory optimization typically defines a full control sequence for a short planning-horizon, C-PBP samples actions one timestep at a time,

informed by various priors encoded in a Gaussian mixture model (GMM). The method also saves simulation capacity by terminating bad trajectories early and reassigning their simulation resources to fork the best trajectories, as illustrated in Fig. 3.4, and to increase the exploration of good states and actions.

3.2 Supervised Learning for Character Control

Supervised learning is the problem of training a model such that it maps the inputs $\mathbf{x}$ to the outputs $\mathbf{y}$ where a set of N training examples in the form of $\{(\mathbf{x}_1,\mathbf{y}_1),(\mathbf{x}_2,\mathbf{y}_2),...,(\mathbf{x}_N,\mathbf{y}_N)\}$ is given. Supervised learning can be divided into two major categories, 1) classification, and 2) regression. In classification, the trained model classifies the objects using their input feature ($\mathbf{x}$) into a set of class labels ($\mathbf{y}$). In regression, the problem transform into predicting a continuous output value ($\mathbf{y}$) given a new input $\mathbf{x}$ (see Fig 3.5). As we try to predict continuous valued simulation control signals based on simulation state and goals, we deal with a regression problem, for which a common objective is to reduce the sum of squared error which is defined as follows:

$$Loss = \frac{1}{M}\sum_{i=1}^{M} \|\mathbf{y}_i - f(\mathbf{x}_i)\|_2 \tag{3.2}$$

The loss function can be defined over training, validation, and test sets. M demonstrates the number of samples in the set where the loss function is being calculated, and $f(\mathbf{x}_i)$ is the output prediction of the model.

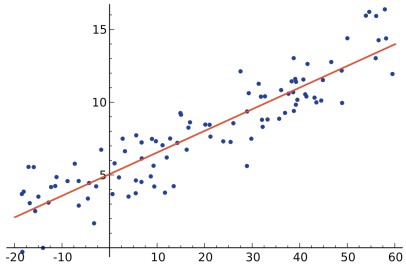


Figure 3.5. An example of linear regression. The data points are denoted by blue dots, and the red line shows a simple linear $f(\mathbf{x})$ that minimizes the loss of Eq. 3.2.

The regression task can be used to enhance sampling-based optimization of movement by interleaving optimization and model training, using previous optimization results as the training data [99, 100]. Given agent's state and goals feature vector as input to a trained model, the model can predict joints' torques or angular velocities that provide the sampling based optimization with better initial guesses [99]. There are many different

model types such as k-nearest neighbour (k-NN), linear regression, and deep neural networks [13] that can be trained for the regression task.

k-NN is the simplest and the most naive way of training a model. In k-NN, there is no training and the prediction for the new output value is done by 1) finding k nearest neighbours of new input using a distance measure, e.g. Euclidean distance, and 2) calculating the average or weighted average output values of k nearest neighbours in the regression task [57]. The weighted average output is calculated as follows:

$$\begin{aligned} \bar{\mathbf{y}} &= \sum_{i=1}^{k} w_i \mathbf{y}_i \\ w_i &= \frac{\exp[-D(\mathbf{x}, \mathbf{x}_i)]}{\sum_{i=1}^{k} \exp[-D(\mathbf{x}, \mathbf{x}_i)]}, \end{aligned} \tag{3.3}$$

where $D(\mathbf{x}, \mathbf{x}_i)$ denotes the distance between the new input $\mathbf{x}$ and the i^{th} nearest neighbour in the training set. The hyper-parameter k can be selected using cross-validation to evaluate generalization of the chosen parameter over the training set.

With k-NN, the data is the model, whereas in most other methods, one uses the data to build a compressed representation that has desirable interpolation and extrapolation capabilities. For example, linear regression trains a linear model by estimating the parameter $\mathbf{W}$ to find a linear mapping between inputs and outputs $\mathbf{y} = \mathbf{W}\mathbf{x}$. Although linear regression is simple and easy to use, the model is limited to simple data where such a linear relation can be assumed.

In recent years, the popularity of deep artificial neural networks has surged in various regression problems. Neural networks are powerful function approximators that scale well to large-scale high-dimensional data [73, 76, 40] and applying them to regression problems is easy using modern tools like Pytorch [93] and Tensorflow [1]. By increasing the size of the neural network, one can model increasingly more complex input-output relations. The parameters of the neural network can be trained numerically using stochastic gradient descent (SGD) [14, 15] or its derivations like ADAM [65]. However, the neural network architecture and activation functions have to be carefully selected in order to avoid over and/or under-fitting problems.

3.3 Reinforcement Learning for Character Control

Reinforcement learning (RL) has many practical applications in robotics, animation and games [6]. RL is a form of learning where the agent learns by trial and error. In order for RL algorithms to be able to solve the problems, they are mathematically formulated using Markov Decision

Figure 3.6. A simple illustration of a reinforcement learning agent interaction with environment for collecting experience points, adopted from [114].

Process (MDP) [113] such that the state transitions have Markov property, i.e., the next state only depends on the current state and action and not on previous states or actions. This thesis considers a finite-horizon discounted MDP defined by the tuple $\left(\mathscr{S},\mathscr{A},f,r,\rho(\mathbf{s}_0),\gamma,T\right)$ where $\mathscr{S}$ is a set of all states, $\mathscr{A}$ is a set of all actions, f is the state transition dynamics function or distribution, r is the reward function, $\rho(\mathbf{s}_0)$ is the distribution of the initial state $\mathbf{s}_0$, γ is the discount factor, and T is the maximum number of timesteps the agent acts. Infinite horizon and non-discounted MDP formulations can also be used, but the standard continuous control RL benchmarks like OpenAI Gym MuJoCo [17] use both finite horizon and discounted rewards. Some RL methods like Policy Iteration do not utilize the $\rho(\mathbf{s}_0)$ and instead iterate over all possible states [113], but such iteration is not typically possible with physically simulated agents and a typical approach is that the agent implements a reset function that samples the state from $\rho(\mathbf{s}_0)$, after which states are explored through sampling actions from $\mathscr{A}$ [17].

In the MDP formulation, the agent starts from an initial state $\mathbf{s}_0 \sim \rho(\mathbf{s}_0)$. Then at each timestep, the agent observes a state vector $\mathbf{s}_t$, takes an action $\mathbf{a}_t$, receives an instantaneous reward r_t and the next state vector $\mathbf{s}_{t+1} \sim f(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$ (see Fig. 3.6). The agent continues this until it reaches a terminal state or a total of T actions. RL methods try to maximize the expected discounted return [102, 103]:

$$J(\pi_\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t(\mathbf{s}_t,\mathbf{a}_t)\right], \tag{3.4}$$

where π_θ is a policy for sampling actions, $\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$. The policy parameters are denoted by θ. $\gamma \in [0,1)$ controls how much to take future rewards into account. With $\gamma = 0$, RL is greedy, optimizing only the instantaneous reward of each action.

Algorithm 5 provides a high-level summary of episodic on-policy reinforcement learning utilized in this thesis. On-policy RL methods approach the problem by iteratively collecting experience tuples $(\mathbf{s},\mathbf{a},\mathbf{s}',r)$ in a buffer through sampling actions from the policy, $\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, and updating the policy based on the collected experience tuples to prefer actions that maximize Eq. 3.4. The process is almost the same as the trajectory optimization

discussed in Section 3.1 – e.g., the Gaussian search distribution of CMA-ES is akin to a policy – but 1) instead of having a fixed initial state, each episode (trajectory) starts from a random state, and 2) the sampled actions are conditioned on state. Additionally, some trajectory optimization methods like C-PBP [46] assume that the state transition dynamics are deterministic, whereas RL dynamics are usually assumed stochastic.

Algorithm 5 High-level summary of episodic on-policy RL

1: **for** iteration = 1,2,... **do**
2: **while** simulation budget is not exceeded **do**
3: Initialize the agent to a state $\mathbf{s}_0 \sim \rho(\mathbf{s}_0)$
4: Run agent on policy π_θ for T timesteps or until a terminal state
5: **end while**
6: Update policy parameters θ based on experience tuples $\left[\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\right]$ collected in this iteration
7: **end for**

The RL algorithms can be divided into model-based and model-free. In model based algorithms, a model of both reward and forward dynamics is either learned or given, and the agent tries to find the best policy based on the approximated model as well as observations collected from the real environment [94]. Model-free algorithms, on the other hand, try to find the best policy only by interacting with the environment. Model-based algorithms can achieve better results with less collected experience, but so far, they have had problems with scaling to complex high-dimensional problems such as synthesizing humanoid movements [43, 6]. Recently, model-free RL algorithms have undergone rapid development [106, 104, 107, 102, 83] with major successes such as playing Atari games from pixel inputs [84, 85] and controlling humanoid locomotion with rapid velocity changes [104]. This dissertation focuses on model-free RL algorithms for training the agent in the bouldering task.

In formulating a problem into RL format one should consider the following factors:

- **Markov Property:** The selected agent's observation/state vector should have Markov property meaning the future state distribution should be independent of the history of states and actions beyond the current state and action.
- **Fully vs Partially Observable Environment:** An environment is called fully observable if $\mathbf{s}_t$ if what the agent can observe at time t is sufficient for acting optimally. RL is usually easier with fully observed state, although modern RL methods may also work for partially observed MDP:s (PO-MDP:s), e.g., using a recurrent policy network [122].

- **Exploration vs Exploitation:** How to explore the environment is a central issue in RL. In order to gather higher future reward, the agent needs to explore and try new actions, and at the same time it needs to utilize or exploit already explored good actions. Thus, there is a trade-off between utilizing the current information and trying new actions in the hope of getting higher expected return value. This is known as the exploration-exploitation dilemma and it is a central and well researched problem in RL [114]. Many RL algorithms initially explore more randomly and the action distribution's variance is gradually reduced as the policy converges to the highest-value actions. In humanoid movement control, the policy is often Gaussian and conditioned on the state [104, 102, 44], which is reminiscent of running CMA-ES optimization in parallel for multiple states. The policy network receives state as input and outputs the mean and variance.

- **On-Policy vs Off-Policy:** On-policy methods can only update the policy based on the experience tuples generated by following the policy being optimized, i.e., the collected experience must be discarded after each update. Off-policy methods can utilize the data from previous training iterations or offline data available in external sources, e.g., from expert demonstrations, which can allow better sample-efficiency [44].

- **Reward Function:** Designing an informative reward function r_t that can guide the policy learning is perhaps the most crucial task in making RL work efficiently. For example designing a sparse reward, e.g. giving reward for success $\{+1\}$ or failure $\{-1\}$ in a task, is easy, but it slows the training process as most of the agent's experience does not provide any information to guide the learning.

3.3.1 Policy Gradient

The most popular RL algorithms today are based on policy gradient methods where the gradient of the expected return $J(\pi_\theta)$, i.e. the gradient of Eq. 3.4, is used to update the policy's parameters. The following reviews the derivation of the so-called REINFORCE policy gradient update, using the notation of [4], originally developed by [123]. REINFORCE and its variants are also introduced in detail in [113].

The policy parameters θ can be updated using the gradient as:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta) \tag{3.5}$$

where α is the learning rate. Let us denote a trajectory as $\tau = [\mathbf{s}_0, \mathbf{a}_0, ..., \mathbf{s}_T]$ and episode return as $G(\tau) = \sum_{t=0}^{T} \gamma^t r_t(\mathbf{s}_t, \mathbf{a}_t)$. Now, $J(\pi_\theta)$ can be expressed

as $J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[G(\tau)]$, and $\nabla_\theta J(\pi_\theta)$ is calculated as:

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[G(\tau)] \\ \nabla_\theta J(\pi_\theta) &= \nabla_\theta \int_\tau P(\tau \mid \pi_\theta) G(\tau) \\ \nabla_\theta J(\pi_\theta) &= \int_\tau \nabla_\theta P(\tau \mid \pi_\theta) G(\tau)\end{aligned} \tag{3.6}$$

where $P(\tau \mid \pi_\theta)$ is the probability of the trajectory τ. To calculate $\nabla_\theta P(\tau \mid \pi_\theta)$, the gradient log trick is used as follows:

$$\begin{aligned}\nabla_\theta \log P(\tau \mid \pi_\theta) &= \frac{\nabla_\theta P(\tau \mid \pi_\theta)}{P(\tau \mid \pi_\theta)} \\ \nabla_\theta P(\tau \mid \pi_\theta) &= P(\tau \mid \pi_\theta) \nabla_\theta \log P(\tau \mid \pi_\theta).\end{aligned} \tag{3.7}$$

Thus, $\nabla_\theta J(\pi_\theta)$ gets the form of:

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \int_\tau P(\tau \mid \pi_\theta) \nabla_\theta \log P(\tau \mid \pi_\theta) G(\tau) \\ \nabla_\theta J(\pi_\theta) &= \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log P(\tau \mid \pi_\theta) G(\tau)]\end{aligned} \tag{3.8}$$

To calculate $\nabla_\theta \log P(\tau \mid \pi_\theta)$, we need to expand $P(\tau \mid \pi_\theta)$ to its components as:

$$\begin{aligned}P(\tau \mid \pi_\theta) &= \rho(\mathbf{s}_0) \prod_{t=0}^{T} f(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \\ \log P(\tau \mid \pi_\theta) &= \log \rho(\mathbf{s}_0) + \sum_{t=0}^{T} \{\log f(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) + \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)\}\end{aligned} \tag{3.9}$$

where $\rho(\mathbf{s}_0)$ is the probability of being at $\mathbf{s}_0$, and $f(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ is the transition probability. By taking the gradient of Eq. 3.9 with respect to θ and zeroing out independent components we have:

$$\nabla_\theta \log P(\tau \mid \pi_\theta) = \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t). \tag{3.10}$$

By substituting Eq. 3.10 in Eq. 3.8, we get:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) \sum_{t'=0}^{T} \gamma^{t'} r_{t'}(\mathbf{s}_{t'}, \mathbf{a}_{t'})\right] \tag{3.11}$$

which is independent of the transition distribution $f(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$. Intuitively, moving along $\nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)$ makes action $\mathbf{a}_t$ more probable in state $\mathbf{s}_t$. The sum of trajectory rewards $\sum_{t'=0}^{T} \gamma^{t'} r_{t'}(\mathbf{s}_{t'}, \mathbf{a}_{t'})$ acts as a multiplier for the gradient, making high-value actions more probable than others. In a

causal system, however, the value of an action should not be influenced by past rewards, and as proven in [3], Eq. 3.11 can be simplified as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\gamma^t \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)\sum_{t'=t}^{T}\gamma^{t'-t} r_{t'}(\mathbf{s}_{t'}, \mathbf{a}_{t'})\right] \tag{3.12}$$

In practice, policy gradient methods replace the expectation in Eq. 3.12 with an average over simulated trajectories or episodes as follows [113]:

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}\gamma^t \left[\nabla_\theta \log \pi_\theta(\mathbf{a}_t^{(i)} \mid \mathbf{s}_t^{(i)})\sum_{t'=t}^{T}\gamma^{t'-t} r_{t'}(\mathbf{s}_{t'}^{(i)}, \mathbf{a}_{t'}^{(i)})\right] \tag{3.13}$$

where the (i) superscripts denote simulated trajectory or episode indices. The methods alternate between collecting experience and performing gradient updates on θ based on variations of Eq. 3.13. However, such Monte Carlo gradient estimates can have high variance, which can slow down the training process or make it unstable. Next section discusses how the issue is handled in Proximal Policy Optimization [104], a modern policy gradient method utilized in this thesis.

3.3.2 Proximal Policy Optimization

As collecting new data for each gradient step is very computationally expensive, RL methods employ various improvements. A popular first choice for continuous control problems in on-policy RL algorithms is Proximal Policy Optimization (PPO) [104] that has stable policy updates, is scalable to high-dimensional problems, and requires only moderate parameter tuning. Publication IV utilizes PPO which is based on an advantage-based gradient estimate as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) A^\pi(\mathbf{s}_t, \mathbf{a}_t)\right] \tag{3.14}$$

where the advantage function $A^\pi(\mathbf{s}_t, \mathbf{a}_t)$ is an estimate of how much of an improvement an action has over the current policy. Comparing Eq. 3.14 and Eq. 3.12, one sees that the advantage function replaces the sum of rewards in Eq. 3.12. The motivation is to reduce gradient variance while still being simple to estimate from the collected experience [103]. The advantage is treated as not depending directly on θ, and it is expressed as $A^\pi(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V^\pi(\mathbf{s}_{t+1}) - V^\pi(\mathbf{s}_t)$, where $V^\pi(\mathbf{s})$ is the value or expected on-policy return from state $\mathbf{s}$, and $\mathbf{s}_{t+1} \sim f(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. In principle, one can train a neural network with simulated episode returns to approximate $V^\pi(\mathbf{s})$, but the associated bias and variance easily result in unstable RL. This is why PPO utilizes Generalized Advantage Estimation (GAE), a simple but powerful way to estimate the advantages [103]. On the top of reducing gradient variance, PPO benefits from taking multiple gradient

steps on the data collected in each iteration, and limits the divergence between the old and updated policies to mitigate the instability caused by the update.

Although PPO is widely used, e.g., as the default optimizer of the Unity ML Agents framework [59], later RL algorithms have provided further improvements, e.g., by combining expectation-maximization (EM) -style fitting of the policy to high-value actions with efficient usage of off-policy data [2, 98].

4. Developed Algorithms and Applications

This chapter reviews the novel algorithms and applications developed in this dissertation. First, Publication I focuses on solving the combined problem of planning a climbing strategy and synthesizing the required movements. The solution is able to synthesize more realistic and dynamic climbing, in particular allowing the physically simulated agent to move two limbs at a time, whereas previous work had simplified the problem by only allowing one limb moves or not simulating the full movement dynamics. Next, Publication II proposes a novel application for the technology of Publication I in computer-aided mental practice of climbing, and evaluates the approach through prototyping and a user study.

While the solution of Publication I was able to produce high-quality results, it was fairly computationally expensive, requiring 1 or 2 minutes of CPU time to solve climbing problems of a modest size. This is why Publication III proposes improvements to Publication I using machine learning, and Publication IV investigates an alternative problem formulation and solution using deep reinforcement learning.

If this thesis had been started one or two years later, there could have been more focus on RL methods. Work on Publication I begun in 2015, when the best simulated humanoid control results were still obtained using trajectory optimization approaches [117, 45, 46]. It was only in 2017 when proximal policy optimization (PPO) demonstrated that deep reinforcement learning is able to produce policies that can robustly handle, e.g., humanoid locomotion combined with rapid direction changes and getting back up after falling [104], which was previously demonstrated with the C-PBP method utilized in this thesis for the low-level controller part [46].

4.1 Publication I: A Control Hierarchy for Discovering Climbing Movements

In Publication I, a hierarchical motion planner is proposed to synthesize climbing movements. In the hierarchy, graph based search is used as the

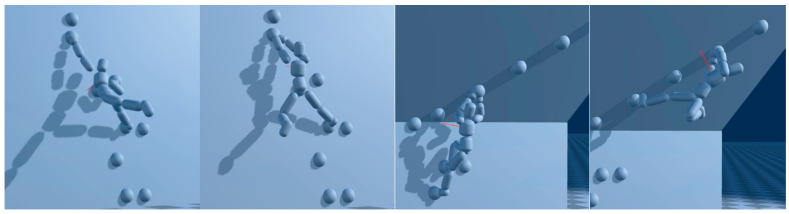


Figure 4.1. Four different climbing moves on flat (left) and 45 degrees overhanging (right) walls, as synthesized in Publication I

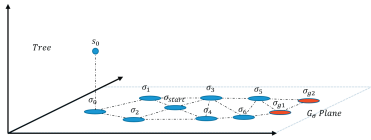
high-level path planner to plan a path or a sequence of climbing moves by outputting goals for each individual climbing move – which holds to reach with which limbs. Then, a sampling based optimization method is utilized as a low-level controller to synthesize the climbing movements on the planned path by optimizing the individual moves such that the goals are reached by the limbs. The main idea behind using the hierarchical motion planning is to split a long-horizon climbing problem into short-horizon sub-problems. Finding multiple good short movement trajectories is much easier than optimizing a single long one because optimization complexity can grow exponentially with dimensionality, and movement trajectory optimization also tends to get increasingly ill-conditioned with longer horizons [47].

Fig. 4.1 illustrates a number of climbing movements that are synthesized by the proposed hierarchical motion planner. The hold positions are in $\mathbb{R}^3$ and denoted by $\mathbf{x}_h$. The high-level path planner can also plan for free hands or feet for the climber which is denoted by $\mathbf{x}_{-1}$. The assignment of the holds to the climber's limbs or *a climbing stance* is denoted by $\sigma = [\mathbf{x}_{\text{ll}}, \mathbf{x}_{\text{rl}}, \mathbf{x}_{\text{lh}}, \mathbf{x}_{\text{rh}}]$ where $\mathbf{x}_{\text{i}} \in \{\mathbf{x}_{-1}, \mathbf{x}_h\}$ for $i = [\text{ll,rl,lh,rh}]$. The agent starts in a T-Pose at state $\mathbf{s}_0$ and $\sigma_0 = [\mathbf{x}_{-1}, \mathbf{x}_{-1}, \mathbf{x}_{-1}, \mathbf{x}_{-1}]$ in front of the wall, and the task is to get either of its hands to the top or goal hold $\mathbf{x}_g$ while passing through a user-specified stance σ_{start}. All the goal stances containing $\mathbf{x}_g$ as a hand hold are denoted by $\sigma_g = \{\sigma | \mathbf{x}_g \in \{\mathbf{x}_{\text{lh}}, \mathbf{x}_{\text{lh}}\}\}$. The paths that are returned by the high-level planner have the form of $p(\sigma_0, \sigma_g) = (\sigma_0, ..., \sigma_{\text{start}}, ..., \sigma_g)$.

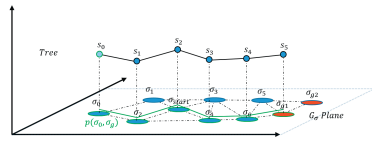
First, rapidly-exploring random trees (RRT) [72] was tried to grow a tree of paths in the climber's state space, however, it did not work as planning in high-dimensional space using randomized actions can take a long time and sometimes makes it impossible to find a feasible path [34]. The final solution still builds a tree in the state space, but it is guided by graph search in the stance space (see Fig. 4.2).

4.1.1 Building the Stance Graph

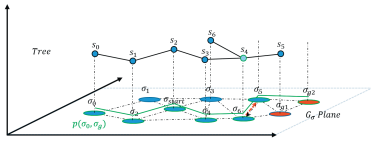
The high-level path planner's *stance graph* contains all possible climbing stances that a climber can reach. The graph can grow prohibitively large even for short climbing routes. For a practical graph search implementa-



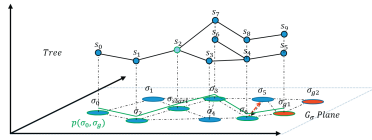
(a) Initialization of tree root of $\mathbf{s}_0$ and the stance graph.



(b) The low-level controller successfully grows the tree on the sampled path of $p(\sigma_0, \sigma_g)$ from the stance graph.



(c) The low-level controller failed to grow the tree on the second sampled path from $\mathbf{s}_4$ toward σ_5 and instead ended up in σ_3 at $\mathbf{s}_6$. The transition cost for σ_6-to-σ_5 is increased, and is now higher than σ_5-to-σ_6.



(d) The low-level controller successfully grows the tree on the third sampled path starting from $\mathbf{s}_2$. Although there already are tree nodes representing σ_6 and σ_{g1}, new nodes are added for the new states, illustrating the one-to-many mapping of stances and states.

Figure 4.2. An illustration of the hierarchical motion planner of Publication I. The xz-plane denotes the stance dimensions, and vertical axis denotes the additional dimensions of climber body state $\mathbf{s}$. The approach grows a tree in the state space (in this abstract figure the 3D space) guided by the lower-dimensional stance graph (here the xz-plane). As illustrated in subfigure (d), a single stance may map to multiple states, but tree edges uniquely map to stance graph edges. All paths start from σ_0 (corresponding to a T-pose in front of the wall) and pass through stance σ_{start} where the limbs are on desired starting holds of the climbing route. There are two goal stances in the graph denoted by σ_{g1} and σ_{g2}. The starting tree node for growing the tree for each planned path is denoted by green color. The planned path of $p(\sigma_0, \sigma_g)$ is denoted by green solid lines on the stance graph. Red dashed edges in the graph denote increased edge cost after failure.

tion, there are two main questions to consider: 1) Can graph connectivity be pruned? 2) given a pruned graph, can one prioritize which edges to search using some heuristics like the Euclidian distance to goal used in A*?

Publication I utilizes different sets of rules for exploring new stances and pruning their connectivity. The rules can be divided into two sets as follows:

- **Climber is on the ground:** Exploring the short transition sequences from σ_0 (T-pose) to σ_{start}. In these sequences, each edge transition is possible when 1) the first stance is σ_0 and next stance has least one connected hand, 2) the first stance has one connected hand and the next stance has at least one more connected hand or

foot, and 3) the transition satisfies the shared pruning rules explained below. In other words, the climber should get from T-pose to the wall without using any other hold than those included in σ_{start}.

- **Climber is on the wall:** Exploring transition sequences from σ_{start} to σ_g by gradually expanding stances in feasible regions around the already added stances. In these sequences, an edge transition is possible when 1) one hand and at least two limbs are connected to holds in both stances, and 2) the shared pruning rules below are satisfied.

The shared pruning rules that determine the feasibility of new stances and moves are defined as follows:

1. At most two limb-to-hold assignments are different in a transition between two stances.
2. Foot holds cannot be placed higher than 0.1cm of the highest hand hold position in a stance. In other words, the climber should avoid being upside down.
3. If all hands and feet are connected in a stance, the number of unique holds should be at least three. This ensure stable balance.
4. Hand-to-hand, foot-to-foot, and hand-to-foot distances should not exceed the climber's body measurements.

using all above rules, Publication I explores new stances and prunes their connectivity by building the graph starting from an initial stance and recursively adding stances.

Publication I uses the following heuristics for evaluating the climber's movements (graph edges) and stances (graph nodes). The graph search prioritizes paths with lower costs. The path cost is the sum of its edge and node costs, which encode an approximation of how a real climber tends to favor stable, comfortable and low-energy movements when possible.

At each stance, the node cost is defined as the sum of the following cost components:

- Penalizing for having free hands or feet. This makes the climber favor stable movements.
- Cost for *matching*, i.e., having two feet attached to the same hold. Matching is usually cumbersome in real climbing, especially if the holds are small.
- Cost of having hands or feet cross each other, or either of hands being lower than any of the feet. Such stances usually correspond to cumbersome poses that limit further progress.

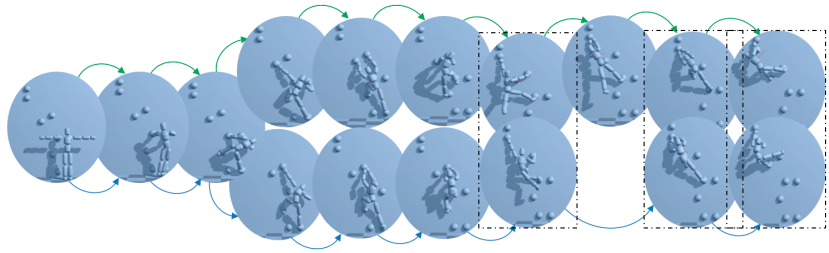


Figure 4.3. Two paths (green and blue arrows) to the top hold in a bouldering problem of Publication I. The dashed rectangles highlight the one-to-many mapping from stances (assignments of holds to limbs) to climber states.

- Penalty for having a hand and foot too close to each other. This avoids unstable balance.

For all connecting stances, the edge cost is the sum of the following:

- A movement cost based on the distances between initial and target holds.
- A cost for moving more than one limb. This favors easy and low-energy movements.
- A cost for having less than three unique holds connected to hands and feet during the transition, if the distance between the holds is less than a threshold, which makes movement unstable.
- A dynamic edge cost that is increased in case of failure in synthesizing planned movement by the low-level controller.

Using the above heuristics in the graph search leads to movements that are more likely to be synthesized successfully by the low-level controller. However, the last item in the list above adds the capability of replanning in case of failures, and avoiding risky moves.

4.1.2 Synthesizing Motions on Stance Graph

The following briefly describes the process of synthesizing climbing movements on the stance graph.

Finding multiple paths: A part of the attraction of climbing is that it is both a physical and a mental game. Well-designed climbing problems often have multiple solutions and different body types and physical abilities can also require multiple solutions. This is why A* prune was chosen as the graph search algorithm, as it can return a number of shortest paths ($p(\sigma_0, \sigma_g)$) on the graph that represent different climbing strategies (Fig. 4.3). This also compensates for the inaccuracy of the heuristics; finding multiple sequence of climbing movements increases the probability that at least some of them are practical ones.

Handling failed moves: The cost components and pruning rules in Section 4.1.1 only encode a priori assumptions of the problem. In practice, some moves may fail either because the assumptions were incorrect or the low-level control optimization failed. This uncertainty is handled by adding a dynamic edge cost that is increased whenever a move fails. After this, the graph search is run again using A* prune while keeping the list of already successfully simulated paths untouched.

Synthesizing motion on the planned path: Once a path $p(\sigma_0, \sigma_g)$ is planned, the low-level controller tries to simulate each edge σ_i-σ_{i+1} in the planned path that has not been simulated yet. Simulating the movement corresponding to an edge results in a movement trajectory, and the trajectories form a tree starting from the initial state of the climber standing in front of the climbing wall. Thus, finding edges that have not been simulated amounts to traversing down the tree following the planned stance path. To simulate the edge of σ_i-σ_{i+1}, the system performs the following:

1. Detach the climber's hands and feet that switch holds between the initial and goal stance.
2. Use the low-level controller to simulate from starting climbing state at σ_i to reach the target hold positions specified in σ_{i+1}.
3. Attach the climber's hands and feet to target holds if they are in contact with the holds.

In the dynamics simulation, gripping a hold or placing a foot on it is approximated as connecting the hold and limb with a ball-and-socket joint. Thus, the attaching and detaching above amounts to creating and destroying the joints. In case of failure, the cost of failed edge is updated, and A* prune is run again as explained above.

4.1.3 Sampling-based Optimization

Two sampling-based black box optimization methods, CMA-ES and C-PBP discussed in Section 3.1, are used as the low-level controller in Publication I to synthesize climbing movements on the planned path $p(\sigma_0, \sigma_g)$. Both methods try to optimize a weighted sum of the following cost components, each corresponding to a movement goal or desired quality:

- Move the limbs towards the target holds: Squared distance from targets.
- Keep the center of mass close to the wall: Squared distance from wall.
- Keep the posture close to a default natural climbing pose (see Fig. 4.4): Sum of squered joint angle differences between current and

default pose. Note that when the climber is on the ground, the default pose is the T-pose (Fig. 4.4, left).

- Maintain a desired facing direction: Angular distance between current and target chest direction. In Publication I, the target is to always face the wall. The motivation is that this usually makes the subsequent moves easier. For example, if the climber swings around and ends up with its back to the wall, continuing from such a state will be difficult.
- Minimize movement velocity: The sum of squared speeds of the climber's simulated body parts.
- Minimize the torque and force exerted from the joints: The sum of squared applied joint motor torques and the forces exerted by hands on the hand holds.

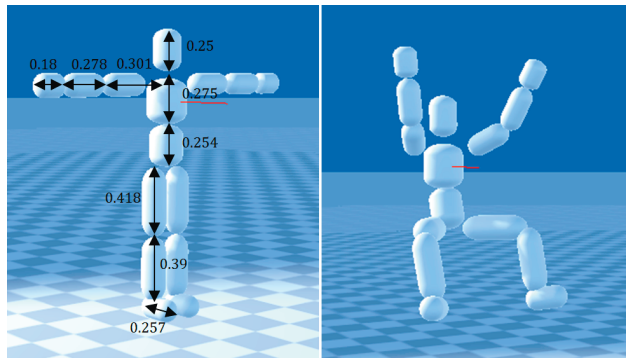


Figure 4.4. Two default postures used during optimization. Left: climber at T-Pose. Right: climber at default climbing pose.

The optimization process comprises of 1) sampling control parameters, 2) simulating the corresponding climber state trajectories, 3) evaluating the cost function based on the controls and state, and 4) refining the control sampling distribution based on the costs. Both optimization methods optimize joint motor target velocities over a planning horizon. However, they differ from each other in:

1. C-PBP simulates 1.5 seconds in the future, sampling new actions every 4 timesteps, while CMA-ES samples full actions sequences defined by two keyframes. The keyframes specify target angular velocities for joint motors, and at each timestep, the angular velocities are interpolated linearly between the keyframes. Keyframe durations are sampled in the range $[0.25, 0.75]$.
2. C-PBP and CMA-ES use 64 and 256 samples per iteration, respectively.

3. C-PBP utilizes Gaussian mixture models while CMA-ES considers only one Gaussian model for the sampling distribution.

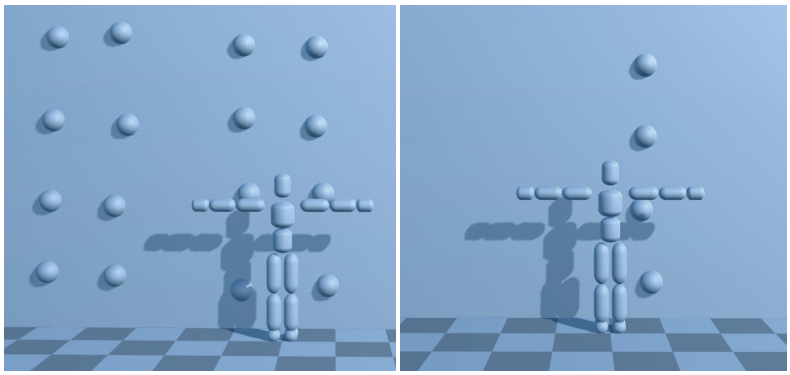


Figure 4.5. Two examples of scalability test environments of Publication I. The environments are built using grids of holds with some random deviation added to the hold positions.

4.1.4 Summary of Results

Publication I evaluates the performance of the graph search method and the optimization methods used in high-level path planner and low-level controller using various measurements. The results can be summarized as follows:

- Experienced climbers comment that CMA-ES produces more determined and skilled movement, while C-PBP looks more like a hobbyist. This may be largely due to the sampling noise of C-PBP that makes the movements look uncertain and shaky, whereas the interpolation of the CMA-ES action parameterization results in smoother movements.
- C-PBP noise can be reduced with increased sampling budget.
- Increasing the number of CMA-ES control points results in a higher number of optimized parameters and more failures of the optimization.
- CMA-ES needs more iterations with a similar sampling budget due to oscillation around the final cost minimum found.
- CMA-ES with the system's settings, i.e. linearly interpolating between two control points, is more reliable in terms of avoiding greedy behaviors that lead to failures. To evaluate this, 20 simulations were run by following the first route planned by A* prune in a fixed bouldering route for both optimization methods with 4 failure cases for

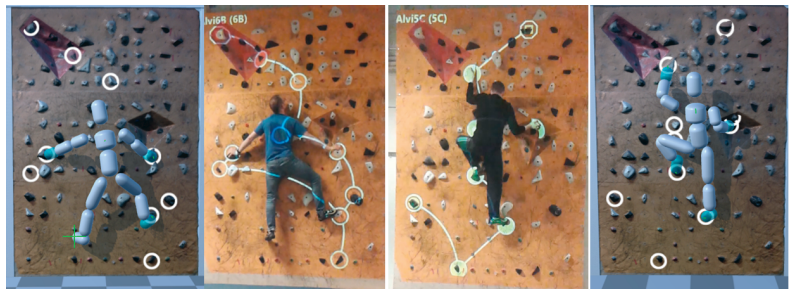


Figure 4.6. Middle: Real climbers climbing two different routes shown with projected graphics. Sides: Screenshot of Publication II's CAI application, with a photogrammetry-based 3D model of the climbing wall and a simulated climber that the user can control.

CMA-ES and 11 failure cases for C-PBP. The greediness of C-PBP can be adjusted with the resampling threshold parameter, but it appears difficult to tune.

- The maximum torque allowed for the joint motors can be easily adjusted to simulate stronger or weaker climbers. Weakening the agent increases the average time needed to find a route, as many of the tried paths fail; qualitatively, the successful climbs exhibit more calm and conservative movements.

- To evaluate the scalability of the method, the number of holds was increased in a grid of $\{\text{rows} \times \text{cols}\}$ with $\{\text{rows} = [2,8], \text{cols} = [1,4]\}$, and the method was run to find 10 paths over 6 different simulations. Examples of the test grids are shown in Fig. 4.5. The data indicates that with such hold grids, both graph size and the planning and simulation time grow roughly linearly as a function of wall height.

4.2 Publication II: Computer Aided Imagery in Climbing

Publication II introduces and evaluates a novel human-computer interface that enables a new form of mental sport practice called computer aided imagery (CAI). The interface utilizes a photogrammetry model of a real climbing wall and a virtual climber with body proportions scaled to match the user based on computer vision measurements (see Fig 4.6). The system uses the algorithms of Publication I in synthesizing plausible and believable physically-based simulations to allow the users better imagine themselves on the climbing wall. The users can interact with the interface and control their avatar or the simulated climber through mouse and keyboard to plan for a sequence of climbing movements leading the character to the top hold. The climbing movements are defined by:

1. Setting target holds for the climber's hands and feet.
2. Specifying target direction for the climber's chest, allowing the user to specify moves where one needs to have one side against the wall.

Using these controls allows the user to discover various climbing movements and solve the climbing route before trying to climb on the actual climbing wall. The main purpose of the interface is to replace or augment the traditional mental imagery phase of preparing for a climb and thinking of possible movement strategies.

Mental imagery is a powerful cognitive, emotional, and motivational tool that has a strong impact on human behavior [49, 64, 108]. However, it is a difficult skill and people have different abilities to visualize movements [18]. Motivated by this, Publication II provides the users with a novel interface for investigating the climbing movements. This section briefly describes the developed technology.

4.2.1 Environment Modeling

The model is built using Autodesk ReCap 360 [78], a photogrammetry software that creates a 3D model of an object from its pictures taken from different angles (see Fig. 4.7). To model a large object such as climbing wall with the 2.44m width and 3.47m height, the author photographed 50 pictures dividing the wall into 3 segments: bottom, middle, and top sections. For each section, the pictures are taken from multiple angles, rotating around the vertical axis in a half circle. The 3D model has approximately less than 1cm inaccuracy in the hold shapes and location. The users can freely rotate and zoom the virtual camera for better inspection of the holds.

The interface utilizes open dynamics engine (ODE) [109] to simulate climber's movements and interaction with holds. The sampling based optimization methods of Publication I are utilized to control the character and synthesize climbing movements without motion captured data or inverse kinematics that could limit the realism and diversity of the movements. The holds are modeled as simple ball-and-socket joints, similar to Publication I. However, the hold modeling is extended by manually annotating the holds with ideal pulling/pushing directions corresponding to the directions from which one can place one's fingers in a cavity or behind a ledge. The ideal direction is used in the movement optimization to synthesize climbing movements that end in more realistic and practically feasible postures.

4.2.2 Adapting the Climber Model to the User

The simulated climber needs to be adjusted to participants' body measurements in order to enable users to copy the simulated movements (e.g., reach a specific hold from a specific pose). The interface utilizes Microsoft

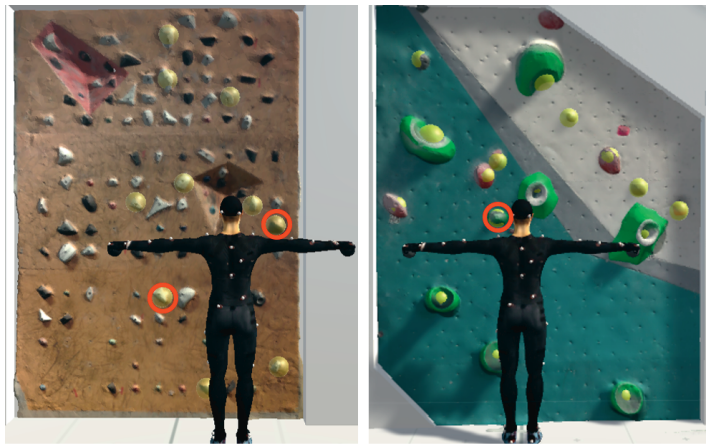


Figure 4.7. Two examples of climbing routes implemented in the simulation, using photogrammetry models of real climbing walls. The red circles denote the starting hand holds.

Kinect V2 sensor and SDK [82] to locate each participant's body joints while standing in T-pose. The located joint positions are used to adjust simulated climber's joint positions and bone lengths.

As a limitation, it should be noted that scaling the body model is not enough for fully realistic climbing simulation. The system does not model finger details, which means that in trying out some climbing stategy, the users have to estimate whether they are able to grasp the holds or whether their hands might slip. Measuring and modeling the user's flexibility and strength is also deferred to future work.

4.2.3 Character Control

In order to allow the users to interact with the interface in real-time and with minimal latency, Publication II uses the online version of C-PBP as opposed to the offline version which is used in Publication I, except when both hands are detached from holds (e.g. when the climber is on the ground). When the climber is on the ground, the movements to get limbs to the holds on the climbing wall are more challenging to optimize, and the increased robustness of offline optimization justifies the added computation time.

Compared to Publication I, the user playes the role of the high-level graph search, and specifies target holds by clicking and dragging limbs to the holds in the climbing environment. Limbs can also be defined as "free" in order to be used for balancing or frictional contacts with the wall. Furthermore, the user can define a target angle for the simulated climber's torso. The optimization process can be initiated by the user by pressing enter.

The C-PBP movement optimization attempts to minimize a cost function expressed as a weighted sum of the following costs:

- Obeying user input: Cost for hands and feet distances to the target holds, and angular distance between the current and target chest direction.
- Conserving energy: Penalty for center of mass distance to the wall.
- Natural movement: Cost for joint angular distance from a default climbing posture (hanging with arms straight to minimize fatigue, see Fig. 4.4).
- Preferring calm and controlled movement: Cost for speed of body parts.
- Feasibility of postures: Penalty for angular distance between the actual and ideal pulling or pushing direction.

Compared to Publication I, the cost function used here is the same except for the last term that allows the simulated climber to adapt to different hold rotations.

4.2.4 Summary of Results

The interface prototype was evaluated in a user study that compares two different preparation methods for climbing, namely CAI and traditional imaginary (TI). The user study was conducted on 20 participants (6 female, 14 male) with a wide range of climbing experience. All the participants climbed four bouldering routes with TI and other four with CAI, using a randomized block design. The overall objective of the study was to better understand the added value of utilizing movement simulation and AI technology in physical activity and sport. The results can be summarized as follows:

- The experiment indicates that preparing with CAI forces a climber to think in 3rd person, and to plan and explore the whole route and body movements in more detail, e.g. CAI made climbers to consider their legs and torso in addition to their hands. However, using TI, climbers prepare for the climb by just checking few first holds and imagining how their hands interact with the holds.
- Using CAI for preparing to climb made participants consider a climbing route as more complex and more difficult to comprehend and climb. This can reduce participants' pre-climb confidence and make them more careful. However, after the climb, these feelings turned into positive evaluations of elevated level of own skill, confidence and decreased sense of route difficulty. In other words, CAI increased

participants' post-climb feeling of competence. The feeling that the simulation was more complex than reality can be considered a limitation of the system. However, it is perhaps better to be inaccurate in this way than make the participants think that a route is easy and then be disappointed or take unnecessary risks.

- Although the majority of the climbers prefer to solve the climbing puzzles by themselves, CAI technology may have benefits as a "route guide" or a something that helps climbers communicate, demonstrate and teach new and difficult routes to each other.
- In the final interviews, climbers considered CAI to save some energy, because one doesn't have to try everything physically on the wall.
- Although participants were allowed to visually inspect the real wall while exploring it with CAI, almost all the climbers mentioned missing more precise hold and finger modeling. This is clearly a potential topic for future work.
- Although the CAI prototype provided the participants with their body measurements, they asked for more realism such as exact holds, hands/fingers, movement, body capabilities, and looks.

4.3 Publication III: Enhancing Climbing Movements using Neural Network Predictions

Publication III utilizes hierarchical motion planning similar to Publication I (see Section 4.1). The high-level path planner uses A* prune to propose sequences of hand and feet placements on the climbing wall, and the low-level controller utilizes a CMA-ES method to synthesize climbing movements on the planned movement sequences. Publication III extends the approach by utilizing neural networks in both the graph search and CMA-ES optimization, with multiple benefits:

1. Reducing the number of heuristics used in building the graph and pruning the edges.
2. Reducing the number of hand-tuned heuristics used in the graph search. Edge costs are based on neural network predictions of stance-to-stance movement success rate and effort. Using these allows the graph search to produce more successful and less effortful sequences of climbing movements, illustrated in 4.8).
3. Better initial guesses for CMA-ES, which allow finding higher quality movement trajectories in less time.

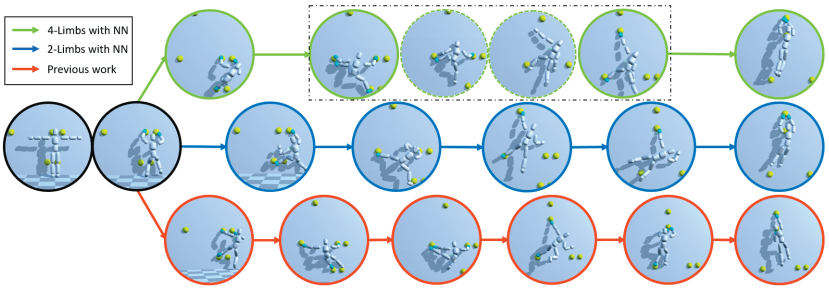


Figure 4.8. Three paths to the top hold in a bouldering problem. Top: A solution involving a dynamic leap (dashed circles) emerges when moving all 4 limbs at the same time is allowed. Middle: A different strategy where movement is restricted to only 2 limbs moving at a time, similar to Publication I. Bottom: The system of Publication I produces more cumbersome movements, and in this problem does not find a solution until after 10 minutes of failed attempts. The top and middle solutions were found on the first try (less than 1 minute of CPU time) using the improved system of Publication III.

The use of neural networks in both graph search and the sampling based optimization allows the motion planner to plan for and synthesize highly dynamic climbing movements such as jumping, as illustrated in Fig. 4.8. Compared to Publication I, Publication III removes the constraint of only switching at most two limbs to new holds in a single move.

4.3.1 System and Training Process

Building on Publication I, Publication III uses the same notation introduced in Section 4.1. The hold positions are denoted by $\mathbf{x}_h$, and the assignment of holds to climber's hands and feet (i.e., stances) are denoted by σ. The climber's state is denoted by $\mathbf{s}$. The climber starts on the ground from T-pose at σ_0, and the goal is to reach the top hold at σ_g. The stance graph, a graph containing all assignments of holds to the climber's hands and feet, is denoted by $\mathcal{G}_\sigma$, and the planned path is denoted by $p(\sigma_0, \sigma_g)$.

Neural Networks: The neural networks used in the stance graph and the low-level controller are denoted by NN_σ and $NN_\mathbf{s}$, respectively. NN_σ outputs:

- $s_\sigma^{nn}(\sigma_i, \sigma_t)$: the predicted success rate of performing a transition from an initial stance to a target stance by the low-level controller.
- $c_\sigma^{nn}(\sigma_i, \sigma_t)$: the predicted effort used by the low-level controller in performing the transition.

The input feature vector of NN_σ is composed as:

$$f_\sigma(\sigma_i, \sigma_t) = \begin{bmatrix} \sigma_i - \bar{\sigma}_i, \\ \sigma_t - \bar{\sigma}_i, \\ \bar{\sigma}_t - \bar{\sigma}_i, \\ \texttt{connected}(\sigma_i), \\ \texttt{connected}(\sigma_t) \end{bmatrix} \tag{4.1}$$

where σ_i and σ_t denote the initial and target stances, $\bar{\sigma}_i$ denotes the average of hand and feet positions in σ_i, and $\texttt{connected}(\sigma_i)$ specifies whether climber's hands and feet are connected to a climbing hold or not. $NN_\mathbf{s}$ outputs:

- $p^{nn}_\mathbf{s}(\mathbf{s}, \sigma_t)$: predicted joint motor actuation parameters that are used as an initial guess for the optimization of the movement from $\mathbf{s}$ to σ_t.
- $c^{nn}_\mathbf{s}(\mathbf{s}, \sigma_t)$: predicted of cost function value that the movement optimization will yield.

Given initial climber's state, and the target stance of σ_t, $NN_\mathbf{s}$'s input feature vector is composed as:

$$f_\mathbf{s}(\mathbf{s}, \sigma_t) = \begin{bmatrix} \mathbf{q}^{-1}_{\text{spine}} \mathbf{v}_{\text{spine}}, \\ \mathbf{q}^{-1}_{\text{spine}}(\mathbf{x}_{\text{b}_1} - \mathbf{x}_{\text{spine}}), \ldots, \mathbf{q}^{-1}_{\text{spine}}(\mathbf{x}_{\text{b}_n} - \mathbf{x}_{\text{spine}}), \\ \mathbf{q}^{-1}_{\text{spine}}(\mathbf{v}_{\text{b}_1} - \mathbf{v}_{\text{spine}}), \ldots, \mathbf{q}^{-1}_{\text{spine}}(\mathbf{v}_{\text{b}_n} - \mathbf{v}_{\text{spine}}), \\ \mathbf{q}^{-1}_{\text{spine}}(\sigma_t - \mathbf{x}_{\text{spine}}), \\ \mathbf{q}^{-1}_{\text{spine}}(\bar{\sigma}_t - \mathbf{x}_{\text{spine}}), \\ \texttt{connected}(\sigma_\mathbf{s}), \\ \texttt{connected}(\sigma_t) \end{bmatrix} \tag{4.2}$$

where $\sigma_\mathbf{s}$ is the climber's stance at the initial state of the optimization, $\mathbf{q}_{\text{spine}}$ is the quaternion of the climber's root at $\mathbf{s}$, and $\mathbf{x}_{b_i}$ and $\mathbf{v}_{b_i}$ denote the positions of the climber's bones and velocities at $\mathbf{s}$, respectively. $b_1, \ldots, b_n$ denote all 15 bones in the feature vector including the climber's arms, legs, and torso.

Training Process: Fig. 4.9 demonstrates a randomized training environment for the climber, and a dynamic jump performed by the climber. The scenes are generated randomly by perturbing holds around a 4-by-4 grid followed by selecting 2 holds per row that are closest to a randomly drawn line on the climbing wall. The training process repeats the following steps until convergence:

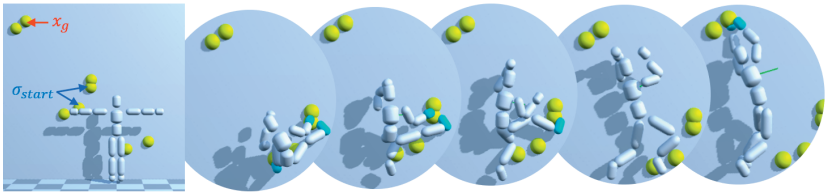


Figure 4.9. Square: A random learning scene. Circles: A jumping sequence to the target hold where hands are separated from holds for a small amount of time.

1. Randomize the training holds.
2. Collect training data to a FIFO buffer by performing random climbing movements.
3. Train the neural networks on all data in the FIFO.

Inspired by the informed online trajectory optimization of [99], $NN_{\mathbf{s}}$ and NN_{σ} are trained in a supervised manner. At each training iteration, neural networks are trained for 5 epochs with minibatch size of 500 using Adam optimizer [65]. Both neural networks have 3 equally-sized hidden layers. The number of neurons per layer is 100 for $NN_{\mathbf{s}}$ and 30 for NN_{σ}. Publication III uses bipolar [27] scaled exponential linear units [66] activation functions for hidden layers and linear activation function for the output layer. To avoid overfitting, the networks are re-trained from scratch if test error starts increasing while training error keeps decreasing.

Data Collection: To collect the data points on a randomized scene, first a stance graph (see Section 4.3.2) is built on the randomized holds. Next, a path $p(\sigma_0, \sigma_g)$ is planned on the graph using A* Prune, and the data points are collected by optimizing the movements corresponding to edges on and around the planned path. Half of the time, a neighbor stance around $\sigma_i \in p(\sigma_0, \sigma_g)$ is selected for simulation of the around planned path. The initial population has 50% random trajectories in addition to the trajectories sampled around the prediction of $NN_{\mathbf{s}}$. For each randomized scene at most 100 climbing movements are explored. A* Prune plans for the next path $p(\sigma_0, \sigma_g)$, if the low-level controller either completes the planned path or fails to follow it. In case of failure the edge cost is increased to make A* prune avoid it when planning for the next path.

The optimization produces two types of training data: 1) whether the optimization is successful or not (s_{σ}^{nn}) which is evaluated by the climber being at σ_t, and 2) for successful optimization, the training data comprises the value of objective function ($c_{\mathbf{s}}^{nn}$), sum of squared joint actuation torques (c_{σ}^{nn}) and the best found control parameters ($\mathbf{p}_{\mathbf{s}}^{nn}$). As the optimization results have a high variance in the initial stages of training, the initial results are forgotten by keeping training data in a FIFO memory.

Curriculum Learning: Publication III adopts a curriculum learning [8] approach of progressing from simple to complex moves to minimize

the overall optimization failure rate. Utilizing initial guess prediction of $NN_{\mathbf{s}}$ in optimizing multiple limb movements when $NN_{\mathbf{s}}$ is trained on simpler movements helps in increasing the success rate of performing the movements. The system starts collecting the data and training by simulating the simplest movements where one limb is switching holds, then proceed to two and three multiple limb movements, and finally to four-limb dynamic leaps. The training is finished with a mixture of all types of moves with the random exploration rate decreased to 25%. The system uses separate FIFO buffers for each training stage to ensure that the training data contains similar amounts of data for each type of movement. Each buffer holds 10K samples. The training minibatches are sampled from all FIFO buffers.

4.3.2 Neural Networks in Graph Building

The building of the stance graph is more simple than in Publication I. As the low-level controller is able to synthesize jumping movements, the stance graph is almost fully connected. However, the system utilizes the success rate prediction from NN_{σ} to prune the edges.

Exploration and Connectivity: To enumerate all possible stances in a given scene with N holds, one needs to account for assignments of all limbs to all holds, which translates to a computational complexity of $O(N^4)$. In practice, one can reduce the computational complexity to $O(NM^3)$ by assigning a hold to a hand, and enumerating over feasible regions for the other limbs. M is the number of holds that can possibly belong to a stance once one limb has been assigned to a hold. Usually, $M < N$. The system utilizes the following rules to prune infeasible stances:

- Hand-to-hand, foot-to-foot, and hand-to-foot distances do not exceed body limits.
- At least one hand is on a hold, except for the initial stance σ_0 where the climber stands beside the wall.
- Foot holds are not higher than 0.5m above the highest hand hold. This prevents weird upside down climbing while still allowing simulation of climbing techniques such as heel hooks.

Compared to Publication I, the approach of Publication III produces more valid stances, but the amount is manageable for small walls or sparsely placed holds.

When a new stance is created, one needs to check the connectivity to all existing stances in the graph which has a worst-case complexity of $O(N^2M^6)$. However, the system uses the following rules to prune the graph edge connectivity. The stance graph has an edge between two stances σ_i and σ_j if and only if:

- The distance between the average hold locations of σ_i and σ_j is less than $d_{\text{mean}} = 2r_{\text{body}}$. Publication I uses $r_{\text{body}} = 2\text{m}$.
- The distance between the holds assigned to a limb in σ_i and σ_j is less than $d_{\text{limb}} = 2r_{\text{body}}$.
- If σ_i is the initial stance, σ_j will have both hands on the predefined starting holds of the route.
- The predicted success probability of movement is greater than 50%, i.e., $s_\sigma^{nn}(\sigma_i, \sigma_j) > 0.5$ (naturally, the system omits this during training).

Evaluating Graph Edges: The outputs of NN_σ, i.e. effort prediction $c_\sigma^{nn}(\sigma_i, \sigma_j)$ and success rate prediction $s_\sigma^{nn}(\sigma_i, \sigma_j)$, are incorporated in the graph edge costs to guide the path planning. Given a pair of connected stance nodes of σ_i and σ_j, the edge cost is calculated as:

$$\begin{aligned} c(\sigma_i, \sigma_j) = {} & c_\sigma^{nn}(\sigma_i, \sigma_j) \\ & + c_{\text{risk}} \left(e^{s_\sigma^{nn}(\sigma_i, \sigma_j)} - 1 \right) \\ & + c_{\text{dyn}}(\sigma_i, \sigma_j) \\ & + c_{\text{free}} N_{\text{free}}(\sigma_j) \end{aligned} \tag{4.3}$$

where c_{risk} and c_{free} are tuning parameters that penalize the path planner in selecting paths with low success rate and more free limbs, respectively. $N_{\text{free}}(\sigma_j)$ denotes the number of free limbs at σ_j, and $c_{\text{dyn}}(\sigma_i, \sigma_j)$ is the dynamic cost that is increased when the low-level controller fails to perform transition from σ_i to σ_j. The total path cost of $p(\sigma_0, \sigma_g)$ is the sum its edge costs, and a path with low cost is likely to yield low effort and high success rate when optimized by the low-level controller.

4.3.3 Neural Networks in the Low-level Controller

Covariance matrix adaptation evolution strategy (CMA-ES), a black-box sampling-based derivative-free optimization method (see Section 3.1.1), is used as the low-level controller to synthesize climbing movements. Given an initial climbing state and target stance, CMA-ES tries to find the best trajectory or a sequence of climbing states and actions. Compared to Publication I, the optimization can synthesize more expressive and dynamic climbing movements because:

- The predictions of $NN_{\mathbf{s}}$ help in initializing the optimization.
- The time of letting go of the initial holds is optimized, as opposed to the climber letting go of them at the beginning of each move. This is crucial to enable dynamic leaps where the climber has to first swing and gain momentum without letting go of the initial holds.

- Instead of optimizing just two segments of linearly interpolated joint angular velocities, Publication III optimizes a Catmull-Rom spline of body poses defined by joint angles.

Objective Function: The objective function is composed as the sum of a control cost and a state cost. The control cost – an estimate of movement effort – is computed as weighted sum of the following terms over the trajectory:

- Squared torques applied on the joints.
- Squared forces applied on the hands by the simulated hand-hold joints, i.e., an estimate of finger strain.

NN_σ learns to predict the control cost as c_σ^{nn}. The state cost is composed of weighted sum of the following costs at the final state of trajectory simulation:

- Squared distance from hands and feet to their target holds.
- Squared distance of the center of mass from wall.
- A cost for not facing the wall.
- A penalty for deviating from the default posture shown in Fig. 4.4.
- A penalty for fast body movements.

$NN_\mathbf{s}$ learns to predict the sum of the control and state costs as $c_\mathbf{s}^{nn}$.

Control Parameterization: Each trajectory is defined as a nonuniform Catmull-Rom spline with 4 control points. Each control point defines a pose as a vector of 22 joint angles for the simulation model's actuated joints and a time offset to the previous control point within range $[0.1, 0.5]$. The system also has 4 additional control variables defining the times at which the hands and feet let go of their initial holds, which was observed to be important for controlled leaping. Thus, the total dimensionality of the optimization problem is $4 \times (22 + 1) + 4 = 96$. The total trajectory simulation time is the sum of first 3 control points' time offsets, and the time to let go of the hands and feet are sampled between 0% and 80% of the total trajectory time.

Optimization Process: At each iteration, CMA-ES samples a set of simulation control parameter vectors, and the objective function value for each vector is computed by simulating the corresponding trajectory. The CMA-ES initial sampling distribution is constructed differently from Publication I as follows:

- Half of the population is from an diagonal Gaussian distribution with the control point pose angle means corresponding to the default pose (see Fig. 4.4) and the standard deviation of 60 degrees. The times are sampled uniformly within the range $[0.1, 0.5]$.

- One sample is dedicated to the control parameters predicted by $NN_{\mathbf{s}}$.
- The rest of the samples are generated by mutating the predicted pose angles with Gaussian noise with the standard deviation of 10 degrees. The times are sampled around the outputs of the neural network with the standard deviation of 0.1 seconds.

The simulations use Open Dynamics Engine (ODE) [109] with a time step of 1/30 seconds. The system terminates the optimization process if one of the following conditions are met: 1) if a maximum number of 120 iterations is reached, 2) if the best found objective function value is more than $c_{\mathbf{s}}^{nn}$, i.e. the cost predicted by $NN_{\mathbf{s}}$, and it stays fixed for 30 iterations, and 3) if the best found objective function value is less than $c_{\mathbf{s}}^{nn}$ and it stays fixed for 15 iterations.

4.3.4 Summary of Results

Various tests were designed and implemented to evaluate the performance of the method. The results can be summarized as:

- Compared to Publication I, the movements are qualitatively more effective and diverse while the motion planner is more simple to finetune as there is no need for more than just a few hand-crafted graph pruning rules and edge cost heuristics.
- The method can plan for dynamic leaps when needed, and prefers less effortful and more successful movements, while Publication I needs complex heuristics in the graph search, and is limited to at most two limbs switching holds during a move.
- The neural network -enhanced graph search plans for sequences of movements that the low-level controller can handle more successfully and with less effort.
- The sampling based optimization is more capable of synthesizing complex climbing movements such as dynamic leaps when it is augmented with $NN_{\mathbf{s}}$ predictions.
- A comparison to human climber demonstrates qualitatively similar climbing movements. However, the simulated climber is substantially faster than an amateur human climber. The system also lacks details of the types of holds on the wall, assuming that all holds are easy to grasp from any direction. This makes the simulated climber take shortcuts not available for the real climber.

Similar to the comments of the participants of Publication II, the last finding above highlights the need to model both the climber body and finger details. However, this makes the planning problem considerably

harder, as the simulation needs a much smaller timestep and possibly double precision arithmetic to precisely process the large mass differences of fingers and the rest of the body as well as the collisions and friction between the climbing holds and small but fast-moving fingers.

4.4 Publication IV: Reinforcement Learning Framework for Climbing Movements

Publication IV proposes a reinforcement learning (RL) approach for synthesizing climbing movements in physically based simulation. The policy trained by RL replaces the sampling based optimization used in the earlier publications. Although modern policy gradient methods such as proximal policy optimization (PPO) [104] can handle high-dimensional continuous states and actions, climbing turns out to be a particularly difficult problem where a naive application of RL fails easily. Publication IV proposes and compares exploration strategies that improve RL efficiency.

4.4.1 Reinforcement Learning Formulation

In reinforcement learning, the agent learns through interacting with the environment. At each timestep t, the agent observes the current state vector $\mathbf{s}_t$, samples an action $\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)$. π_θ is the policy parameterized by θ, mapping the observed state to a distribution over actions. After executing the action $\mathbf{a}_t$, the agent receives a scalar reward r_t, and observes the new state vector $\mathbf{s}_{t+1}$. The agent tries to maximize the expected cumulative future-discounted reward, i.e., $\mathbb{E}\left[\sum_{t=0} \gamma^t r_t\right]$, where $\gamma \in [0,1)$ is the discount factor, and t denotes episode timestep.

Agent's Observation: In Publication IV, the observation vector $\mathbf{s}_t$ is defined as follows:

$$\mathbf{s}_t = \left\langle \mathbf{s}_t^{\text{cl}}, \sigma_c, \sigma_d, \text{I}(\sigma_c), \text{I}(\sigma_d), \text{I}(\sigma_c \neq \sigma_d) \right\rangle \tag{4.4}$$

where σ_c and σ_d are the agent's current and desired stances or the assignment of agent's hands and feet to hold positions. σ_d is set by the user or a high-level path planner while σ_c is determined by the current state of the climber. $\text{I}(\sigma_c)$ and $\text{I}(\sigma_d)$ indicate whether the hands and feet are attached to the holds in the current and desired stance, respectively. $\text{I}(\sigma_c \neq \sigma_d)$ expresses which elements in the current and desired stances differ from each other. $\mathbf{s}_t^{\text{cl}}$ is the climber's body state at time t, defined as:

$$\mathbf{s}_t^{\text{cl}} = \left\{ \left\langle \mathbf{x}_b, \mathbf{v}_b, \mathbf{q}_b, \omega_b, \tau_b, \text{I}_{\text{wall}}(b), \text{I}_{\text{ground}}(b) \right\rangle \forall b \in \mathscr{B} \right\} \tag{4.5}$$

where $\mathbf{x}_b$, $\mathbf{q}_b$, $\mathbf{v}_b$, and ω_b are position, quaternion, linear and angular velocities of the bones. τ_b determines the strength of the joint attached to

bone b. $\mathrm{I}_{\mathrm{wall}}(b)$ and $\mathrm{I}_{\mathrm{ground}}(b)$ indicate whether the bone is in contact with the climbing wall and/or the ground, respectively.

Agent's Action: The sampled action vector $\mathbf{a}_t$ at timestep t has 44 values in the range $[-1,1]$. The first 30 elements of $\mathbf{a}_t$ denote the target angles for the climber's joints, and the last 14 elements are mapped to the range $[0,\tau_{\max}]$ to be used as the joints' strength, i.e. the maximum torque that the joint motors of the physics simulator are allowed to use to reach the target angles. Joint limits and $\tau_{\max} = 250\mathrm{Nm}$ are set such that the climber has human-like limitations.

Agent's Reward: Designing an informative reward function is a challenging task. The reward function of Publication IV is composed as the sum of two components:

- r_t^{σ}: similar to the highly successful Deepmimic system [95], this is a distance-based reward to guide the policy towards getting the hands and feet closer to the target stance at each timestep.
- r_t^{τ}: to encourage smooth movements, the system incorporates a reward term based on the strengths of the joints.

More specifically, r_t^{σ} and r_t^{τ} are computed as follows:

$$\begin{aligned} r_t^{\sigma} &= \mathrm{I}(\mathbf{d}_t) \sum_{i=\{\mathrm{ll,rl,lh,rh}\}} \begin{bmatrix} k_{\sigma} \exp(-c_{\sigma} \parallel \sigma_{c,i} - \sigma_{d,i} \parallel) \\ + \mathrm{I}(\sigma_{c,i} = \sigma_{d,i}) \end{bmatrix} - \mathrm{I}_{\mathrm{ground}}(\mathbf{s}) \\ r_t^{\tau} &= k_{\tau} \mathrm{I}(\mathbf{d}_t) \exp\left[-c_{\tau} \sum_{i \in \{31,\ldots,44\}} (\mathbf{a}_t[i] + 1)^2 \right], \end{aligned} \tag{4.6}$$

where k_{σ}, k_{τ}, c_{σ}, and c_{τ} are tuning parameters. k_{σ} is larger than k_{τ} to prioritize getting hands and feet to target holds over relaxing the joints. $\mathrm{I}(\mathbf{d}_t) = 1$ if climber's hands and feet are getting closer to the targets, otherwise $\mathrm{I}(\mathbf{d}_t) = 0$. As opposed to sparse reward, e.g. $r_t = \{-1,1\}$ that specifies success or failure, the designed reward function guide the policy at every step toward getting closer to the target positions while reducing the strength used by the climber.

Training Episodes and Termination Condition: Publication IV uses PPO [104], which assumes that experience is collected during training as episodes that start from some initial climbing state and run up to a time limit T_{episode} or a terminal climbing state. In Publication IV, a climbing state is considered terminal if the climber reaches the target stance σ_d or any part of the climber's body except its feet touches the ground.

4.4.2 Exploration Strategies

A big part of the difficulty of humanoid climbing as an RL problem is exploration. Recall from Section 3.3.1 that policy gradient updates increase

the probability of actions that happened to result in high rewards. To make this efficient, the explored actions need to include some good actions. Furthermore, to be able to improve the action distribution for all relevant states (e.g., various poses on the climbing wall, various target hold configurations for different limbs), the agent needs to experience those states. However, if one initializes the agent to stand in front of a climbing wall and explores with random movements, the agent mostly just falls on the ground. Even with millions of simulations, it is very unlikely that the agent manages to get on the wall let alone explore feasible climbing moves.

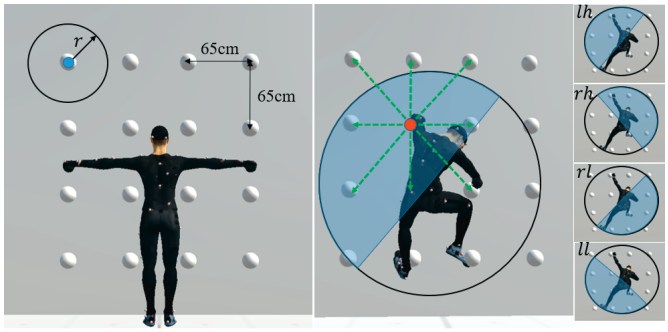
DeepMimic [95] solves the problem through Reference State Initialization (RSI), i.e., initializing episodes from random motion capture data frames, which ensures that the agent explores relevant states. However, collecting collecting motion capture data is expensive and may not be possible for all apolications, which calls for better exploration methods that do not depend on data. Motivated by this, Publication IV developed an exploring strategy for target stances (movement goals), and investigated and compared multiple approaches to exploring agent states and actions.

Exploration of Target Stance: To sample a random target stance, the system first randomly perturbs all holds on the wall around the initial grid locations within $r = 33$cm demonstrated in Fig. 4.10-(a). Then, it randomly selects 1 or 2 limbs to move toward new targets and sample target holds for them among the neighboring holds, as illustrated in Fig. 4.10-(b). The neighboring holds include the following:

- $\mathbf{x}_{-1}$, i.e., the target can be just freeing the limb.
- The holds inside of the shaded area for the limb as shown in Fig. 4.10-(c). The shaded area is within $r_{\text{body}} = 130$cm around the climber's hip position.
- The holds connected by dashed lines to the hold closest to the moving limb.

The target holds for the moving limbs are sampled uniformly from the neighboring holds, using rejection sampling to prune invalid targets that do not satisfy the following:

- Connectivity Limit: At least one hand should be connected in both current and target stances, except for when the climber is at T-pose σ_0 and only the target stance needs to have at least one connected hand.
- Distance Limit: The distances between hand-to-hand, foot-to-foot, and hand-to-foot should not exceed the climber's body reach in any stance.
- Limb Movement Limit: the current and target stances should have at most two different elements.



(a) Training Environment (b) Neighboring Holds for Left Hand (c)

Figure 4.10. (a) Initial hold positions with their randomization radius r. The climber is at T-pose in front of wall. (b) The neighboring hold set of the left hand is the union of the holds that are inside the shaded area and the ones that are connected by dashed lines to the hold closest to the left hand, shown by red dot. (c) The four shaded areas show the valid regions of the holds to be reached by the climber's hands and feet with respect to climber's hip position.

Exploration of Climber's State: Publication IV compares and studies the effect of two strategies for episode state initialization. This is inspired by Peng et al. [95] who recently demonstrated that initializing episodes from randomly selected motion capture frames can dramatically improve learning. Without this, the episodes fail to initially explore much of the relevant states. The investigated strategies are as follows:

- BaseLine: Initializes the climber to the T-pose in front of a randomized wall, as illustrated in Fig 4.10, and climbing continues one target stance after another. Reset happens after failure movement or a time limit. This is similar to common continuous control benchmarks like OpenAI Gym MuJoCo [17], where the initial state distribution has zero or only moderate variance.
- Self-Supervised Episode State Initialization (SS-ESI): Gradually expands the set of possible initial climbing states in $\mathscr{S}$ by adding the end result of each climbing move to $\mathscr{S}$ if at least one hand is connected to a hold. The episode initial states are sampled uniformly from $\mathscr{S}$. The $\mathscr{S}$ is cleared after each PPO iteration.

With SS-ESI, the episode trajectories fork previous trajectories, forming an exploration tree. In order to ensure diverse exploration for the approach:

- The initial episode climbing state is randomly relocated on the climbing wall, as illustrated in Fig. 4.11. This relocation is done only if all hands and feet are connected to avoid breaking the dynamics simulation.
- The sample distribution of target stances is balanced such that a similar amounts of feet and hand moves are sampled.

(a) (b) (c)

Figure 4.11. Randomly relocating the climber on the climbing wall to allow more diverse movement. The positions of the holds not used by the climber are also randomly perturbed.

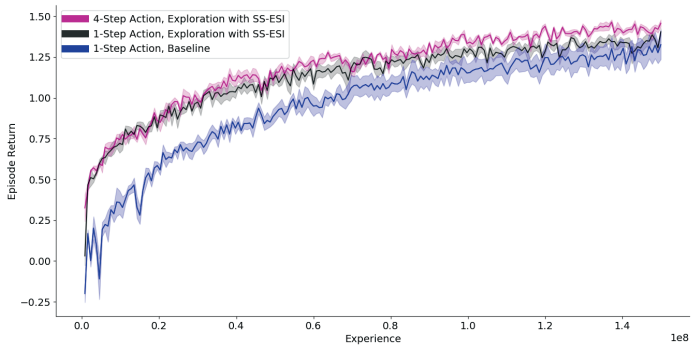
In both the baseline and SS-ESI approaches, each episode is terminated after a single climbing move. This makes value estimation much less noisy compared to when the training episodes have multiple movements, as value does not depend on target stances sampled after the current move. In the baseline version, the first episode starts from T-pose, and if the movement reaches the target stance, the next episode continues from the same state. Otherwise, or if episode time limit was reached, the next episode again starts from T-pose. A more naive baseline was already tested, where all episodes started from T-pose and each episode attempted multiple moves until failure or time limit, but this resulted in basically no progress.

Exploration of Action Setting: Publication IV compares both single-step and multi-step actions. Multi-step actions simulate every action that comes from the policy for L steps instead of only one step. Although increasing L produces smoother movements, using multi-step actions yields less experience tuples for the same amount of timesteps simulated, and in the extreme cases it reduces the expressive range of the movements and makes the learning task harder. However, in moderate cases using multi-step actions yields better training results (see Fig. 4.12). Publication IV uses $L = 4$.

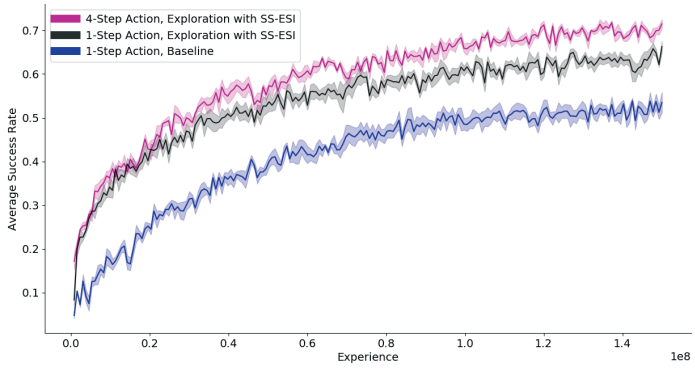
4.4.3 Summary of Results

The RL climber's performance was measured by the success rate and the cumulative reward. A brief summary of the results are as follows:

- SS-ESI explores both hand and foot moves much more evenly during all training iterations. In contrast, the baseline approach is biased towards learning hand moves, as shown in Fig 4.13.
- As illustrated in Fig. 4.12a, the baseline learns much slower initially, although the difference in becomes smaller as training progresses.
- The baseline approach has lower success rate than SS-ESI in per-



(a) Mean and standard deviation of episode return.



(b) Mean and standard deviation of episode success rate.

Figure 4.12. The learning curves of the tested exploration strategies. The means and standard deviations are from 5 independent training runs.

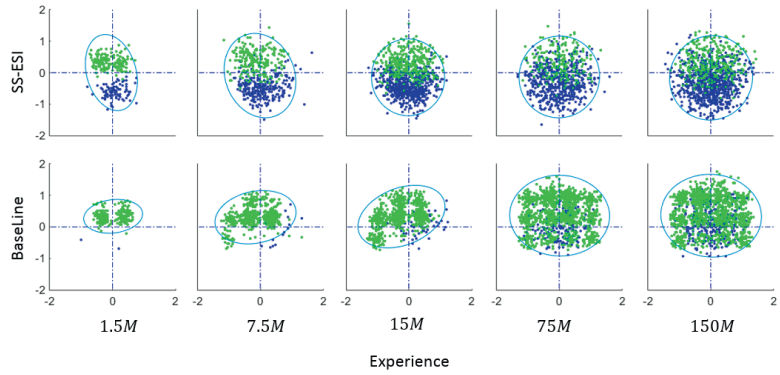


Figure 4.13. The effect of self-supervised episode state initialization (SS-ESI) approach on the diversity of successful transitions during training. The green and blue dots show successful transitions of hands and feet, respectively, in the climber's local coordinates.

forming climbing movements. This difference does not vanish as training progresses, as shown in Fig 4.12b.

- Multi-step actions lead to more rapid learning, and higher episode returns than 1-step action. Also, the multi-step actions produce smoother, visually more pleasing movements. A visual inspection of the movements reveals that multi-step actions lead to wider movement arcs and thus wider exploration of climber states, which may explain the better performance.
- Increasing the number of simulation steps per action makes collecting experience more costly. If the step count is increased too much, it also limits the complexity and temporal resolution of movements. However, the evaluation data indicates that a moderate increase of steps per action can be beneficial, compared to the baseline of taking a new action at every simulation step.

5. Discussion and Conclusions

This dissertation focuses on developing algorithms and applications that are suitable for planning and synthesizing movements in complex environments such as simulated wall climbing. This section highlights central findings, and discusses future directions for the research.

5.1 Summary of the Algorithms and Applications

The hierarchical motion planning developed in Publication I introduces an algorithm capable of synthesizing dynamic climbing movements (see Section 4.1). Although the high-level path planner and the low-level controller was already used by [16] to synthesize static climbing movements for a low degree of freedom (DoF) crawler, the combination of graph search method with sampling based optimization methods makes it possible to synthesize dynamic climbing movements for high dimensional characters.

Publication II developed a novel interface that allows the user to discover climbing movements and helps users imagine themselves solving real climbing problems. Although the users highlighted many positive aspects of the technology, they also reported the simulated movements were not realistic and not close enough to their movements. This realism gap manifested in two main ways:

1. The low-level controller was able to perform climbing movements that are hard or impossible for the users, e.g., hanging from tiny holds with no support for the feet.

2. The low-level controller was unable to perform some movements that are easy for the users. An example of this was dynamic leaps.

The first issue comes from simplification of the hold models as ball-and-socket joints, and not modeling climbers' finger detail. The second issue was largely due to the inefficiency of the sampling based movement optimization. The finger modeling remains as future work, but Publication III and Publication IV already developed the movement optimization further.

Publication III dived into the problem of synthesizing more successful and agile movements, utilizing supervised learning and curriculum learning to enhance the movement planning and optimization. Using the predictions of neural networks, with the iterative training process introduced in Section 4.3.1, allows the hierarchical motion planner to synthesize less effortful and more successful movements. The neural networks provide the planner with:

1. Better initial guesses for the sampling-based movement optimization.
2. Better evaluation and pruning of the graph edges.

Although the method synthesizes more efficient and diverse climbing movement including dynamic leaps, the method remains offline, i.e. the neural networks only inform the movement optimization instead of directly predicting useful control sequences. This motivated the RL approach of Publication IV, the final publication included in this disseratation.

Publication IV addressed the problem of synthesizing climbing movements in real-time, utilizing a reinforcement learning approach that results in a neural network policy that is able to control the simulated climber, in contrast to having to optimize each move. The publication highlights the importance of improving the exploration of relevant states and actions, and proposes and evaluates different strategies for this purpose (see Section 4.4.2). The proposed strategies improve the results over a naive baseline similar to standard continuous control benchmarks like OpenAI Gym MuJoCo [17].

5.2 Suggestions for Future Work

One of the major issues in synthesizing realistic and believable climbing movements is hold modeling. In this thesis, a simple ball-and-socket joint model is used such that a joint is created between an end effector and a target joint if it is closer than some threshold distance. However, this hold modeling overpowers the simulated agent compared to a human climber who may slip, get tired from hanging onto the holds or simply cannot grasp some holds. Thus, one research direction is toward better contact and hold modeling for climbing movements. One might be interested in utilizing a more advanced physics simulator such as Mojuco [119] for better contact and force modeling, and simulate hands with fingers [89] to synthesize grasping different holds with different shapes.

In spite of the successes of the low-level controller in the sampling based optimization methods of Publication I to Publication III, there is a need in having a method capable of synthesizing high-quality movements in real-time performance. Publication III provides an approach for synthesizing

high-quality movements, however it cannot perform the process in real-time. Publication IV successfuly formulates a reinforcement learning approach for the problem of synthesizing climbing movements, but the quality of the results is not yet on par with Publication III. One research direction is to enhance the quality of the movements synthesized by RL towards the ones synthesized in Publication III. One key question is how to combine the efficiency of RL with the smooth movements resulting from spline-based movement optimization. It is possible to use RL with each action defining a spline, but at least based on the author's initial experiments, training becomes very computationally expensive because evaluating each action requires simulating long movement trajectories.

An alternative approach could be to utilize imitation learning with motion capture data, but high quality and diverse enough datasets are not yet available. However, this could be mitigated with proper data augmentation, e.g., following the recent impressive results of [92]. Such data could possibly be collected on a large scale using low-cost computer vision similar to [97], either in climbing centers or based on Internet video.

Another exciting research avenue is utilizing reinforcement learning and imitation learning on the high-level path planner to model the climbing strategies planned by human climbers. Furthermore, the model should take into account differences in climber skill levels and route difficulty. Same as above, large scale vision-based data collection might be helpful.

5.3 Conclusions

The capability of computers in learning and solving complicated tasks is growing rapidly. Recent major milestones include mastering two-player fully observable games such as Go [106] but also multiplayer and only partially observable games such as Dota2 [91] and StarCraft [121]. Although performance of algorithms is not yet human-level in synthesizing physical and believable motions in computer animation, games, and robotics, technology is advancing fast. This dissertation contributes to this research stream, focusing on indoor climbing as the test problem, and develops an approach for planning and synthesizing of such complex movements.

This dissertation introduces a combination of high-level path planner and low-level controller to synthesize plausible and believable motions for a complex 3D humanoid character with a high number of degrees of freedom and a long planning horizon. A particular strength is that the algorithms developed in this dissertation do not need any reference animations or motion capture data; instead, all movements emerge from optimization of relatively simple objective functions. The technology was also demonstrated and evaluated in a user study where climbers used it for mental preparation of real climbing.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*, 2018.

[3] Josh Achiam. Proof for Do not Let the Past Distract You. `https://spinningup.openai.com/en/latest/spinningup/extra_pg_proof1.html`, 2019.

[4] Josh Achiam. Spinning Up in Deep RL: Intro to Policy Optimization. `https://readthedocs.com/projects/openai-education-spinningup/downloads/pdf/latest/`, April 2019. pp 43-51.

[5] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 22–31. JMLR. org, 2017.

[6] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[7] Amin Babadi, Kourosh Naderi, and Perttu Hämäläinen. Intelligent middle-level game control. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2018.

[8] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[9] Elisabetta Bevacqua, Romain Richard, and Pierre De Loor. Believability and co-presence in human-virtual character interaction. *IEEE computer graphics and applications*, 37(4):17–29, 2017.

[10] Priyadarshi Bhattacharya and Marina L Gavrilova. Roadmap-based path planning-using the voronoi diagram for a clearance-based shortest path. *IEEE Robotics & Automation Magazine*, 15(2):58–66, 2008.

[11] Rafał Biedrzycki. Handling bound constraints in cma-es: An experimental study. *Swarm and Evolutionary Computation*, page 100627, 2019.

[12] Ni Bin, Chen Xiong, Zhang Liming, and Xiao Wendong. Recurrent neural network for robot path planning. In *International Conference on Parallel and Distributed Computing: Applications and Technologies*, pages 188–191. Springer, 2004.

[13] Hanen Borchani, Gherardo Varando, Concha Bielza, and Pedro Larrañaga. A survey on multi-output regression. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 5(5):216–233, 2015.

[14] Léon Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, Nimes, France, 1991. EC2.

[15] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[16] Timothy Bretl. Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem. *The International Journal of Robotics Research*, 25(4):317–342, 2006.

[17] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[18] Nichola Callow and Ross Roberts. Imagery research: An investigation of three issues. *Psychology of Sport and Exercise*, 11(4):325–329, 2010.

[19] Salvatore Candido, Yong-Tae Kim, and Seth Hutchinson. An improved hierarchical motion planner for humanoid robots. In *Humanoids 2008-8th IEEE-RAS International Conference on Humanoid Robots*, pages 654–661. IEEE, 2008.

[20] Min Gyu Choi, Jehee Lee, and Sung Yong Shin. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics (TOG)*, 22(2):182–203, 2003.

[21] Loïc Ciccone, Cengiz Öztireli, and Robert W Sumner. Tangent-space optimization for interactive animation control. *ACM Transactions on Graphics (TOG)*, 38(4):101, 2019.

[22] Simon Clavet. Motion Matching and The Road to Next-Gen Animation. `http://twvideo01.ubm-us.net/o1/vault/gdc2016/Presentations/Clavet_Simon_MotionMatching.pdf`, March 2016. As part of presentations at GDC 2016 talks.

[23] Xiao Cui and Hao Shi. A*-based pathfinding in modern computer games. *International Journal of Computer Science and Network Security*, 11(1):125–130, 2011.

[24] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[25] Haibin Duan and Linzhi Huang. Imperialist competitive algorithm optimized artificial neural networks for ucav global path planning. *Neurocomputing*, 125:166–171, 2014.

[26] E. F. Moore. The shortest path through a maze. pages 285 – 292. In Proceedings of the International Symposium on the Theory of Switching, Harvard University Press, 1959.

[27] Lars Eidnes and Arild Nøkland. Shifting mean activation towards zero with bipolar activation functions. *arXiv preprint arXiv:1709.04054*, 2017.

[28] David Eppstein. Finding the k shortest paths. *SIAM Journal on computing*, 28(2):652–673, 1998.

[29] David Eppstein. *k-Best Enumeration*, pages 1–4. Springer US, Boston, MA, 2008.

[30] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.

[31] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. *arXiv preprint arXiv:1710.09767*, 2017.

[32] Riot Games. League of Legends Game. `https://na.leagueoflegends.com/en/`, 2009.

[33] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004, Sep. 2014.

[34] Ian Garcia and Jonathan P How. Improving the efficiency of rapidly-exploring random trees using a potential function planner. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 7965–7970. IEEE, 2005.

[35] Santiago Garrido, Mohamed Abderrahim, and Luis Moreno. Path planning and navigation using voronoi diagram and fast marching. *IFAC Proceedings Volumes*, 39(15):346–351, 2006.

[36] Thomas Geijtenbeek and Nicolas Pronost. Interactive character animation using simulated physics: A state-of-the-art review. In *Computer Graphics Forum*, volume 31, pages 2492–2515. Wiley Online Library, 2012.

[37] Thomas Geijtenbeek, Michiel Van De Panne, and A Frank Van Der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics (TOG)*, 32(6):206, 2013.

[38] Weidong Geng and Gino Yu. Reuse of motion capture data in animation: a review. In *International Conference on Computational Science and Its Applications*, pages 620–629. Springer, 2003.

[39] Roy Glasius, Andrzej Komoda, and Stan CAM Gielen. Neural network dynamics for path planning and obstacle avoidance. *Neural Networks*, 8(1):125–133, 1995.

[40] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[41] Antti Granqvist, Tapio Takala, Jari Takatalo, and Perttu Hämäläinen. Exaggeration of avatar flexibility in virtual reality. In *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play*, pages 201–209. ACM, 2018.

[42] Michael X Grey, Aaron D Ames, and C Karen Liu. Traversing environments using possibility graphs for humanoid robots. *arXiv preprint arXiv:1608.03845*, 2016.

[43] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

[44] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[45] Perttu Hämäläinen, Sebastian Eriksson, Esa Tanskanen, Ville Kyrki, and Jaakko Lehtinen. Online motion synthesis using sequential monte carlo. *ACM Transactions on Graphics (TOG)*, 33(4):51, 2014.

[46] Perttu Hämäläinen, Joose Rajamäki, and C Karen Liu. Online control of simulated humanoids using particle belief propagation. *ACM Transactions on Graphics (TOG)*, 34(4):81, 2015.

[47] Perttu Hämäläinen, Juuso Toikka, and Karen Liu. Visualizing movement control optimization landscapes. *arXiv preprint arXiv:1909.07869*, 2019.

[48] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[49] Lew Hardy and Nichola Callow. Efficacy of external and internal visual imagery perspectives for the enhancement of performance on tasks in which form is important. *Journal of Sport and Exercise Psychology*, 21(2):95–112, 1999.

[50] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[51] Matthew Hausknecht and Peter Stone. On-policy vs. off-policy updates for deep reinforcement learning. In *Deep Reinforcement Learning: Frontiers and Challenges, IJCAI 2016 Workshop*, 2016.

[52] Rachel Heck and Michael Gleicher. Parametric motion graphs. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 129–136. ACM, 2007.

[53] Daniel Holden, Taku Komura, and Jun Saito. Phase-functioned neural networks for character control. *ACM Trans. Graph.*, 36(4):42:1–42:13, July 2017.

[54] Tito Homem-de Mello and Güzin Bayraksan. Monte carlo sampling-based methods for stochastic optimization. *Surveys in Operations Research and Management Science*, 19(1):56–85, 2014.

[55] Ming-Kai Hsieh, Bing-Yu Chen, and Ming Ouhyoung. Motion retargeting and transition in different articulated figures. In *Ninth International Conference on Computer Aided Design and Computer Graphics (CAD-CG'05)*, pages 6–pp. IEEE, 2005.

[56] Han-Pang Huang and Shu-Yun Chung. Dynamic visibility graph for path planning. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2813–2818. IEEE, 2004.

[57] Sadegh Bafandeh Imandoust and Mohammad Bolandraftar. Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background. *International Journal of Engineering Research and Applications*, 3(5):605–610, 2013.

[58] Adobe Systems Incorporated. Mixamo: Get Animated. `https://www.mixamo.com/#/`. Copyright 2019 Adobe Systems Incorporated.

[59] Arthur Juliani, Vincent-Pierre Berges, Esh Vckay, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018.

[60] Raine Kajastila and Perttu Hämäläinen. Augmented climbing: Interacting with projected graphics on a climbing wall. In *Proceedings of the Extended Abstracts of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI EA '14, pages 1279–1284, New York, NY, USA, 2014. ACM.

[61] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *CoRR*, abs/1005.0416, 2010.

[62] Lydia Kavraki, Petr Svestka, and Mark H Overmars. *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, volume 1994. Unknown Publisher, 1994.

[63] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[64] Jackie Kiewa. Control over self and space in rockclimbing. *Journal of Leisure Research*, 33(4):363–382, 2001.

[65] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[66] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980, 2017.

[67] Sven Koenig and Maxim Likhachev. D* lite. *Aaai/iaai*, 15, 2002.

[68] Lucas Kovar and Michael Gleicher. Automated extraction and parameterization of motions in large data sets. In *ACM Transactions on Graphics (ToG)*, volume 23, pages 559–568. ACM, 2004.

[69] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *ACM SIGGRAPH 2008 classes*, page 51. ACM, 2008.

[70] Oliver Kramer. A review of constraint-handling techniques for evolution strategies. *Applied Computational Intelligence and Soft Computing*, 2010, 2010.

[71] Valeri Kroumov and Jianli Yu. Neural networks based path planning and navigation of mobile robots. *Recent advances in mobile robotics*, pages 174–190, 2011.

[72] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[73] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[74] Howard Li, Simon X Yang, and Mae L Seto. Neural-network-based path planning for a multirobot system with moving obstacles. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 39(4):410–419, 2009.

[75] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pages 222–229, 2004.

[76] Shiyu Liang and Rayadurgam Srikant. Why deep neural networks for function approximation? *arXiv preprint arXiv:1610.04161*, 2016.

[77] Benoît Libeau, Alain Micaelli, and Olivier Sigaud. Transfer of knowledge for a climbing virtual human: A reinforcement learning approach. In *2009 IEEE International Conference on Robotics and Automation*, pages 2119–2124. IEEE, 2009.

[78] N. Lievendag. How to use atuodesk recap 360. http://3dscanexpert:com/123d-catch-alternative-recap-360-sketchfab-mobile-android-workflow/, 2017.

[79] Gang Liu and KG Ramakrishnan. A* prune: an algorithm for finding k shortest paths subject to multiple constraints. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, volume 2, pages 743–749. IEEE, 2001.

[80] Abraham Sánchez López, René Zapata, and Maria A Osorio Lama. Sampling-based motion planning: A survey. *Computación y Sistemas*, 12(1):5–24, 2008.

[81] David Mayne. A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *International Journal of Control*, 3(1):85–95, 1966.

[82] Microsoft. How to use kinect v2.0. https://developer:microsoft:com/en-us/windows/kinect/develop, 2013.

[83] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[84] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[85] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[86] Maxim Mozgovoy, Marina Purgina, and Iskander Umarov. Believable self-learning ai for world of tennis. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–7. IEEE, 2016.

[87] Kourosh Naderi, Joose Rajamäki, and Perttu Hämäläinen. Rt-rrt*: A real-time path planning algorithm based on rrt*. In *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, MIG '15, pages 113–118, New York, NY, USA, 2015. ACM.

[88] Jianjun Ni and Simon X Yang. Bioinspired neural network for real-time cooperative hunting by multirobots in unknown environments. *IEEE Transactions on Neural Networks*, 22(12):2062–2077, 2011.

[89] Tim Olsen, Sheldon Andrews, and Paul Kry. Computational climbing for physics-based characters. In *Poster presented at the ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA posters)*, 2014.

[90] Olympic.org. IOC APPROVES FIVE NEW SPORTS FOR OLYMPIC GAMES TOKYO 2020. https://www.olympic.org/news/ioc-approves-five-new-sports-for-olympic-games-tokyo-2020, August 2016. Retrieved 2016-08-08.

[91] OpenAI. OpenAI Five. https://openai.com/five/, 2019.

[92] SOOHWAN PARK, HOSEOK RYU, SEYOUNG LEE, SUNMIN LEE, and JEHEE LEE. Learning predict-and-simulate policies from unorganized human motion data. 2019.

[93] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[94] Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, Kam-Fai Wong, and Shang-Yu Su. Deep dyna-q: Integrating planning for task-completion dialogue policy learning. *arXiv preprint arXiv:1801.06176*, 2018.

[95] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.*, 37(4):143:1–143:14, July 2018.

[96] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):41, 2017.

[97] Xue Bin Peng, Angjoo Kanazawa, Jitendra Malik, Pieter Abbeel, and Sergey Levine. Sfv: Reinforcement learning of physical skills from videos. *ACM Transactions on Graphics (TOG)*, 37(6):178, 2019.

[98] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[99] Joose Rajamäki and Perttu Hämäläinen. Augmenting sampling based controllers with machine learning. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, page 11. ACM, 2017.

[100] Joose Julius Rajamäki and Perttu Hämäläinen. Continuous control monte carlo tree search informed by multiple experts. *IEEE transactions on visualization and computer graphics*, 2018.

[101] Spyridon Samothrakis, Samuel A Roberts, Diego Perez, and Simon M Lucas. Rolling horizon methods for games with continuous states and actions. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8. IEEE, 2014.

[102] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[103] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[104] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[105] Sentewolf. Custom image displaying the directional optimization of the cma-es algorithm. `https://en.wikipedia.org/wiki/CMA-ES#/media/File:Concept_of_directional_optimization_in_CMA-ES_algorithm.png`, 2008.

[106] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[107] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.

[108] Paul J Silvia. What is interesting? exploring the appraisal structure of interest. *Emotion*, 5(1):89, 2005.

[109] R. Smith. Open dynamics engine v0.5 user guide. `http://www:ode:org/ode-latest-userguide:pdf`, 2006.

[110] Omar Souissi, Rabie Benatitallah, David Duvivier, AbedlHakim Artiba, Nicolas Belanger, and Pierre Feyzeau. Path planning: A 2013 survey. In *Proceedings of 2013 International Conference on Industrial Engineering and Systems Management (IESM)*, pages 1–8. IEEE, 2013.

[111] Nathan R Sturtevant. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, 2012.

[112] Muhammad Saleem Ullah Khan Sumbal and Marc Carreras. Environment detection and path planning using the e-puck robot. *Magister Program, University of Giron, Catalonia*, 2010.

[113] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, USA, 2018.

[114] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

[115] Osamu Takahashi and Robert J Schilling. Motion planning in a plane using generalized voronoi diagrams. *IEEE Transactions on robotics and automation*, 5(2):143–150, 1989.

[116] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913. IEEE, 2012.

[117] Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1168–1175. IEEE, 2014.

[118] Sebastian Thrun and Arno Bücken. Integrating grid-based and topological maps for mobile robot navigation. In *Proceedings of the National Conference on Artificial Intelligence*, pages 944–951, 1996.

[119] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[120] Jarek Tuszyński. Climber on Autum (5.9-) at Long Wall in Red River Gorge of Kentucky. `https://commons.wikimedia.org/wiki/File:Red_River_Gorge_-_Long_Wall_-_Autum_2.jpg`, October 2009. Attribution: Jarek Tuszyński / CC-BY-3.0 & GDFL.

[121] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, pages 1–5, 2019.

[122] Daan Wierstra, Alexander Foerster, Jan Peters, and Juergen Schmidhuber. Solving deep memory pomdps with recurrent policy gradients. In *International Conference on Artificial Neural Networks*, pages 697–706. Springer, 2007.

[123] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[124] Shihong Xia, Lin Gao, Yu-Kun Lai, Ming-Ze Yuan, and Jinxiang Chai. A survey on human performance capture and animation. *Journal of Computer Science and Technology*, 32(3):536–554, May 2017.

[125] Liang Yang, Juntong Qi, Dalei Song, Jizhong Xiao, Jianda Han, and Yong Xia. Survey of robot 3d path planning algorithms. *Journal of Control Science and Engineering*, 2016:5, 2016.

[126] Simon X Yang and Chaomin Luo. A neural network approach to complete coverage path planning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(1):718–724, 2004.

[127] Jin Y Yen. An algorithm for finding shortest routes from all source nodes to a given destination in general networks. *Quarterly of Applied Mathematics*, 27(4):526–530, 1970.

Errata

Publication I

In Section 5.3, "For each move, we optimize joint motor target velocities over a planning horizon of 1.5 seconds (C-PBP offline) or 0.5-2.0 seconds (CMA-ES)." should be "For each move, we optimize joint motor target velocities over a planning horizon of 1.5 seconds (C-PBP offline) or 0.5-1.5 seconds (CMA-ES)."

Publication IV

The caption of Fig. 2-(b), should be "The neighboring hold set of the left hand is the union of the holds that are inside the shaded area and the ones that are connected by dashed lines to the hold closest to the left hand, shown by red dot.", and in Section IV-B, "The holds connected by dashed-lines to the hold attached to the moving limb." should be "The holds con-

Aalto-DD 87/2020

ISBN 978-952-60-8886-0 (printed)
ISBN 978-952-60-8887-7 (pdf)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

Aalto University
School of Science
Department of Computer Science
www.aalto.fi

BUSINESS + ECONOMY

ART + DESIGN + ARCHITECTURE

SCIENCE + TECHNOLOGY

CROSSOVER

DOCTORAL DISSERTATIONS