

PLANNING HYBRID DRIVING-STEPPING LOCOMOTION FOR
GROUND ROBOTS IN CHALLENGING ENVIRONMENTS

DISSERTATION

zur Erlangung des Doktorgrades (Dr. rer. nat.)
der Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von
TOBIAS KLAMT
aus Hannover

Bonn, Juni 2019



Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen
Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

Erster Gutachter: Prof. Dr. Sven Behnke
Zweiter Gutachter: Prof. Dr. Maren Bennewitz
Tag der Promotion: 23. Januar 2020
Erscheinungsjahr: 2020

ABSTRACT

Ground robots capable of navigating a wide range of terrains are needed in several domains such as disaster response or planetary exploration. Hybrid driving-stepping locomotion is promising since it combines the complementary strengths of the two locomotion modes. However, suitable platforms require complex kinematic capabilities which need to be considered in corresponding locomotion planning methods. High terrain complexities induce further challenges for the planning problem.

We present a search-based hybrid driving-stepping locomotion planning approach for robots which possess a quadrupedal base with legs ending in steerable wheels allowing for omnidirectional driving and stepping. Driving is preferred on sufficiently flat terrain while stepping is considered in the vicinity of obstacles. Steps are handled in a hierarchical manner: while only the connection between suitable footholds is considered during planning, those steps in the resulting path are expanded to detailed motion sequences considering the robot stability. To enable precise locomotion in challenging terrain, the planner takes the individual robot footprint into account. The method is evaluated in simulation and in real-world applications with the robots Momaro and Centauro. The results indicate that the planner provides bounded sub-optimal paths in feasible time. However, the required fine resolution and high-dimensional robot representation result in too large state spaces for more complex scenarios exceeding computation time and memory constraints.

To enable the planner to be applicable in those scenarios, the method is extended to incorporate three levels of representation. In the vicinity of the robot, the detailed representation is used to obtain reliable paths for the near future. With increasing distance from the robot, the resolution gets coarser and the degrees of freedom of the robot representation decrease. To compensate this loss of information, those representations are enriched with additional semantics increasing the scene understanding. We further present how the most abstract representation can be used to generate an informed heuristic. Evaluation shows that planning is accelerated by multiple orders of magnitude with comparable result quality. However, manually designing the additional representations and tuning the corresponding cost functions requires a high effort.

Therefore, we present a method to support the generation of an abstract representation through a convolutional neural network (CNN). While a low-dimensional, coarse robot representation and corresponding action set can be easily defined, a CNN is trained on artificially

generated data to represent the abstract cost function. Subsequently, the abstract representation can be used to generate a similar informed heuristic, as described above. The CNN evaluation on multiple data sets indicates that the learned cost function generalizes well to real-world scenes and that the abstraction quality outperforms the manually tuned approach. Applied to hybrid driving-stepping locomotion planning, the heuristic achieves similar performance while design and tuning efforts are minimized.

Since a learning-based method turned out to be beneficial to support the search-based planner, we finally investigate if the whole planning problem can be solved by a learning-based approach. Value Iteration Networks (VINs) are known to show good generalizability and goal-directed behavior, while being limited to small state spaces. Inspired by the above-described results, we extend VINs to incorporate multiple levels of abstraction to represent larger planning problems with suitable state space sizes. Experiments in 2D grid worlds show that this extension enables VINs to solve significantly larger planning tasks. We further apply the method to omnidirectional driving of the Centauro robot in cluttered environments which indicates limitations but also emphasizes the future potential of learning-based planning methods.

ZUSAMMENFASSUNG

Bodenroboter, welche eine Vielzahl von Untergründen überwinden können, werden in vielen Anwendungsgebieten benötigt. Beispielszenarien sind die Katastrophenhilfe oder Erkundungsmissionen auf fremden Planeten. In diesem Kontext ist hybride Fahr-/Lauf-Fortbewegung vielversprechend, da sie die sich ergänzenden Stärken der beiden Fortbewegungsarten miteinander vereint. Um dies zu realisieren benötigen entsprechende Roboter allerdings komplexe kinematische Fähigkeiten, welche auch in adäquaten Ansätzen für die Planung dieser Fortbewegung berücksichtigt werden müssen. Anspruchsvolle Umgebungen mit komplexen Untergründen erhöhen dabei zusätzlich die Anforderungen an die Bewegungsplanung.

In dieser Arbeit wird ein suchbasierter Ansatz für kombinierte Fahr-/Lauf-Fortbewegungsplanung vorgestellt. Die adressierten Zielplattformen sind vierbeinige Roboter, deren Beine in lenkbaren Rädern enden, so dass sie omnidirektional fahren und laufen können. Auf ausreichend ebenem Untergrund wird generell Fahren bevorzugt, während der Planer Laufmanöver in der Nähe von Hindernissen in Erwägung zieht. Schritte werden dabei in einer hierarchischen Art und Weise realisiert: Während des Planens werden nur Verbindungen zwischen geeigneten Auftrittsflächen gesucht. Nur solche Schritte, die im Ergebnispfad enthalten sind, werden anschließend zu detaillierten Bewegungsabläufen verfeinert, welche die Balance des Roboters sicherstellen. Um präzise Fortbewegung in anspruchsvollen Umgebungen zu ermöglichen, betrachtet der Planer die spezifischen Aufstandsflächen der vier Füße. Der Ansatz wurde sowohl in simulierten als auch in realen Tests mit den Robotern Momaro und Centauro evaluiert, wobei der Planer in der Lage war, Lösungspfade von ausreichender Qualität in zulässiger Zeit zu generieren. Allerdings ergeben die benötigte feine Planungsauflösung und die hochdimensionale Roboterrepräsentation große Zustandsräumen. Diese würden für komplexere oder größere Planungsprobleme die zulässige Rechenzeit und den verfügbaren Speicher überschreiten.

Damit der Planer auch eben diese komplexeren oder größeren Planungsprobleme handhaben kann, wird eine Erweiterung des Ansatzes beschrieben, welche mehrere Repräsentationslevel mit einbezieht. In unmittelbarer Umgebung des Roboters wird die zuvor beschriebene detaillierte Repräsentation genutzt, um hochwertige Pfade für die nahe Zukunft zu erzeugen. Mit zunehmendem Abstand vom Roboter wird die Auflösung gröber und die Anzahl der Freiheitsgrade in der Roboterrepräsentation sinkt. Um den mit dieser Vergrößerung einhergehenden Informationsverlust zu kompensieren, werden diese Repräsentationen mit zusätzlicher Semantik ausgestattet, welche das

Szenenverständnis erhöht. Darüber hinaus wird beschrieben, wie die Repräsentation mit dem höchsten Abstraktionsgrad zur Berechnung einer effektiven Heuristik genutzt werden kann. Die Evaluation in Simulationsumgebungen zeigt, dass der Planungsprozess um mehrere Größenordnungen beschleunigt werden kann, während die Ergebnisqualität vergleichbar bleibt. Allerdings sind das manuelle Gestalten der zusätzlichen Repräsentationen und das dazugehörige Parametrisieren der Kostenfunktionen sehr arbeitsintensiv.

Um diesen Aufwand zu reduzieren, wird daher eine Methode beschrieben, welche die Gestaltung einer abstrakten Repräsentation durch ein *Convolutional Neural Network* (CNN) unterstützt. Während eine grobe, niedrigdimensionale Roboterrepräsentation und ein dazugehöriges Aktionsset einfach definiert werden können, wird ein CNN auf künstlich erzeugten Daten trainiert, um die abstrakte Kostenfunktion zu lernen. Anschließend kann die so erzeugte abstrakte Repräsentation genutzt werden, um die bereits zuvor erwähnte effektive Heuristik zu berechnen. In der Evaluation des CNNs auf verschiedenen Datensätzen zeigt sich, dass die gelernte Kostenfunktion auch mit Daten aus realen Umgebungen funktioniert und dass die generelle Ergebnisqualität oberhalb der Ergebnisse mit manuell erzeugten Repräsentationen liegt. Die Anwendung der Methode zur Planung hybrider Fahr-/Lauf-Fortbewegung zeigt, dass die so erzeugte Heuristik gleichwertige Ergebnisse wie die Heuristik auf Basis manuell erzeugter Repräsentation liefert, während der Aufwand zur Gestaltung und Parametrisierung deutlich verringert wurde.

Da sich gezeigt hat, dass eine lernbasierte Methode den klassischen suchbasierten Ansatz effektiv unterstützen kann, wird in dieser Arbeit abschließend untersucht, ob das gesamte Planungsproblem durch eine lernbasierte Methode gelöst werden kann. *Value Iteration Networks* (VINs) sind in diesem Zusammenhang ein vielversprechender Ansatz, da sie bekanntlich ein gutes zielorientiertes Planungsverhalten lernen und das Gelernte auf unbekannte Situationen verallgemeinern können. Allerdings ist ihre bisherige Anwendung auf kleine Zustandsräume begrenzt. Durch die zuvor beschriebenen Ergebnisse motiviert, wird eine Erweiterung von VINs beschrieben, so dass diese auf verschiedenen Abstraktionsleveln planen, um größere Planungsprobleme in Zustandsräumen entsprechender Größe darzustellen. Experimente in 2D-Rasterumgebungen zeigen, dass die beschriebene Methode VINs in die Lage versetzt, deutlich größere Planungsprobleme zu lösen. Darüber hinaus wird die beschriebene Methode benutzt, um omnidirektionale Fahrmanöver für den Centauro-Roboter in anspruchsvollen Umgebungen zu planen. Gleichzeitig werden hier aber auch die momentanen, hardware-bedingten Grenzen rein lernbasierter Ansätze sowie ihr zukünftiges Potential aufgezeigt.

ACKNOWLEDGMENTS

First of all, I would like to give special thanks to Prof. Dr. Sven Behnke for his advice, inspiring research ideas, fruitful discussions, and for providing me the space that I needed to develop my research and write this thesis.

I would like to thank all members of the Autonomous Intelligent Systems group for the great time I had in Bonn. I enjoyed the very friendly and loyal, but also productive atmosphere and I will gladly think back to many eventful conference and project trips.

My special gratitude goes to Max Schwarz and Diego Rodriguez who showed extensive helpfulness and patience to support me with my—especially in the beginning—limited programming skills. Furthermore, I would like to thank Michael Schreiber for his assistance in innumerable technical and logistical problems. I would also like to thank the students that I supervised for their contributions to my research.

Deepest thanks belong to my wife Ann-Ki and my family who always supported me along my way to finalize this thesis.

This work was partially funded by the European Union's Horizon 2020 Programme under Grant Agreement 644839 (CENTAURO).

Für Ann-Ki

CONTENTS

1	INTRODUCTION	1
1.1	List of Contributions	3
1.2	Publications	5
1.3	Outline	6
2	RELATED WORK	9
2.1	Hybrid Driving-Stepping Locomotion Robots	9
2.2	Robot Motion Planning Method Overview	12
2.2.1	Search-based Planning	13
2.2.2	Sampling-based Planning	14
2.2.3	Optimization- and Potential Field-based Planning	15
2.2.4	Learning-based Planning	16
2.3	Planning on Multiple Representation	18
2.4	Robot Locomotion Planning Approaches	21
3	A SEARCH-BASED APPROACH TO HYBRID DRIVING-STEPPING LOCOMOTION PLANNING	25
3.1	System Overview	26
3.2	Map Generator	28
3.3	Path Planner	33
3.3.1	Robot Representation	33
3.3.2	Planning Method	37
3.3.3	Action Set and Cost Function	42
3.3.4	Heuristic	48
3.3.5	Implementation	49
3.4	Path Expander	51
3.4.1	Step Expansion	51
3.4.2	Vertical Foot Positions	53
3.5	Controller	54
3.6	Evaluation	55
3.6.1	Robot Orientation Cost Factor	55
3.6.2	Heuristic Weight Comparison	58
3.6.3	Cluttered Staircase Scenario	59
3.6.4	Real Robot Application	61
3.7	Conclusion	64
4	PLANNING HYBRID DRIVING-STEPPING LOCOMOTION ON MULTIPLE LEVELS OF ABSTRACTION	67
4.1	System Overview	69
4.2	Environment and Robot Representation	72
4.2.1	Level 1 Representation	72
4.2.2	Level 2 Representation	74

4.2.3	Level 3 Representation	75
4.3	Action Set, and Cost Function	78
4.3.1	Level 1	78
4.3.2	Level 2	79
4.3.3	Level 3	81
4.3.4	Level Transition	81
4.4	Abstract Representation-based Heuristic	82
4.5	Continuous Path Refinement	83
4.6	Evaluation	85
4.6.1	Representation Level Performance	85
4.6.2	Heuristic Comparison	86
4.6.3	Start State Influence	89
4.7	Conclusion	90
5	TOWARDS LEARNING ABSTRACT REPRESENTATIONS FOR LOCOMOTION PLANNING IN HIGH-DIMENSIONAL STATE SPACES	93
5.1	System Overview	94
5.2	Problem Statement	96
5.3	Planning Representations	96
5.3.1	Detailed Representation	97
5.3.2	Abstract Representation	98
5.4	Abstract Cost Network	99
5.4.1	Network Architecture	99
5.4.2	Training	101
5.4.3	Abstract Representation-based Heuristic	102
5.5	Evaluation	103
5.5.1	Abstraction Quality	104
5.5.2	Application to Planning	105
5.6	Conclusion	108
6	VALUE ITERATION NETWORKS ON MULTIPLE LEVELS OF ABSTRACTION	109
6.1	System Overview	110
6.2	Method	111
6.2.1	Network Architecture	112
6.2.2	Application	116
6.2.3	Training	120
6.3	Evaluation	122
6.3.1	Path Planning in 2D Random Obstacle Grid Worlds	123
6.3.2	Path Planning in 2D Maze Grid Worlds	125
6.3.3	Planning Omnidirectional Robot Driving Loco- motion with Footprint Consideration	127
6.4	Conclusion	130
7	DISCUSSION	133

LISTS OF FIGURES, TABLES, AND ACRONYMS	137
BIBLIOGRAPHY	144

INTRODUCTION

Robots find their way into more and more areas of our lives. For well-defined tasks in known environments, robots are often employed since they are faster, stronger, more precise, or less costly than humans. Today, one of the main challenges in robotic development is to extend their applicability to complex, unstructured, and unknown real-world environments. Solving this challenge will enable robots to solve tasks in numerous new domains.

Many of these domains require ground robots to move over a variety of terrains. Examples are delivery services, disaster response, planetary exploration, as well as domestic application. In general, ground robots either possess a wheeled/tracked or legged locomotion architecture. Driving locomotion is advantageous to overcome large, sufficiently flat terrains since it is fast, safe, and energy efficient. On the other hand, stepping locomotion is more suitable for considerably more challenging terrains, since only isolated footholds are needed. To acquire the ability of overcoming a large variety of terrains, hybrid driving-stepping locomotion is promising since it combines the advantages of both locomotion types.

However, robotic platforms with hybrid driving-stepping locomotion capabilities exhibit complex kinematics with many degrees of freedom (DoF) posing challenges to the control. Teleoperation of such systems is usually slow and puts a high cognitive load on the operator which increases the risk of mistakes. Hence, autonomous locomotion planning and execution is needed. However, the combination of well-studied, isolated planning approaches for driving and stepping locomotion is not sufficient. Hybrid locomotion platforms, as considered in this thesis, possess unique motion capabilities, such as driving individual feet on the ground relative to the robot base while being under load, which need to be considered in the planning to exploit all kinematic capabilities.

In this thesis, a hybrid driving-stepping locomotion planning approach is presented, which describes all kinematic capabilities in a holistic motion planning problem. The method addresses the navigation of quadrupedal ground robots in challenging environments including rough surfaces, debris, and staircases. The planner prefers omnidirectional driving and considers stepping maneuvers in environments which potentially cannot be overcome through driving. It furthermore considers individual robot foot configurations to enable precise navigation planning. Different motion planning approaches

are discussed, and a graph search algorithm is chosen to solve the planning problem.

One of the main challenges of motion planning for systems with multiple and complex kinematic capabilities is computational efficiency. The high number of DoF usually results in rapidly growing state spaces, which become infeasibly large for longer planning queries such that path generation cannot be performed in reasonable time - an effect that is also observed for the presented planning problem. To overcome this limitation, several works propose the utilization of additional coarser planning representations. By choosing coarser resolutions or robot representations with fewer DoF, the same problem can be described with smaller state spaces while the original detailed representation is only employed in certain situations. However, such coarsening bears the risk of discarding valuable information resulting in bad or false planning decisions.

In this thesis, it is described how multiple representations with different levels of abstraction enable efficient planning for large and complex problems while minimizing the above-described risks caused by information loss. Motivated by the manner how humans plan their locomotion, a high degree of detail is only given in the vicinity of the current position. With increasing distance from that position, the plan becomes coarser but with additional semantics increasing the "understanding" of the situation. It is further shown, how an abstract representation can be used to generate an informed heuristic which has knowledge about the robot capabilities and the environment, and efficiently guides the planner towards the goal. Those applications require the different representations to provide similar situation assessments which can be obtained through similar cost functions. While a method to manually tune these cost functions is presented first, it is subsequently described how machine learning can be employed to minimize manual tuning efforts: a CNN is trained on generated artificial data and learns to estimate costs for given planning tasks.

It is finally investigated if machine learning methods can be used to solve the whole planning problem. In the last years, hardware developments have enabled the implementation of neural network architectures with increasing complexity. This provided the foundation for a variety of learning-based motion planning approaches. However, those planners are only applicable to small planning problems of certain complexity to obtain manageable state space sizes. Inspired by the results of planning with multiple representations with different levels of abstraction, this idea is transferred to learning-based planning. A learning-based planner is presented which employs such multiple representations and is capable of solving significantly larger and more complex planning problems compared to its original implementation without these multiple representations. This increases its applicability to real-world planning problems

1.1 LIST OF CONTRIBUTIONS

Planning omnidirectional driving in challenging environments: Challenging environments, which are, e.g., cluttered with obstacles, induce several challenges to locomotion planning: precise planning and collision checking are required despite the high computational effort. Moreover, a desirable robot behavior should consider available safety margins to obstacles, the knowledge about different collision properties of different robot parts, and the preference of certain driving orientations to enable safe and effective locomotion strategies.

In this thesis, cost representations for individual robot feet and the robot base are developed. Foot costs are designed to cause the robot to pass obstacles considering available safety margins. Base costs consider the underlying terrain. Cost computation is based on 2D height maps to provide high computational efficiency. A search-based locomotion planner is presented that considers individual foot and base costs and enables precise omnidirectional driving in challenging environments.

Stepping locomotion near obstacles: Most mobile robots either possess a wheeled or legged locomotion approach. Only few platforms provide hybrid driving-stepping capabilities coming along with a high number of DoF. This induces challenges to corresponding locomotion planning since planning representations feature rapidly growing state spaces. So far, only few works have addressed the problem of hybrid driving-stepping locomotion planning, and, to the best of our knowledge, no previous works proposed such locomotion planning for challenging environments, which includes, e.g., stair climbing.

In this thesis, the developed driving planner is extended to consider stepping in the vicinity of obstacles. This includes steps themselves, as well as stepping-related motions of individual feet and the robot base which provide a reliable robot stability. Steps are represented in a hierarchical manner: while only simplified steps are considered during planning, steps in the resulting path are expanded to detailed motion sequences. The planner is extended with anytime characteristics to provide paths in feasible time.

A complete hybrid locomotion planning and execution pipeline: The developed locomotion planner is embedded in a planning pipeline which contains an environment perception module processing laser scanner and optionally other sensor data, the planner itself, and a controller capable of executing the generated paths. The pipeline is demonstrated in simulation environments and with the real Momaro and Centauro robots.

Planning on multiple levels of abstraction: While the developed hybrid locomotion planner is applicable to environments of limited size, larger queries result in infeasible long computation times and infeasibly large memory requirements. This observation can be generalized to most high-dimensional motion planning problems in challenging environments. The required state space sizes are too large to be efficiently handled by traditional planning approaches. In the literature, this is usually addressed through additional planning representations with coarser resolutions or lower-dimensional robot representations which reduce the state space size but might discard valuable information.

In this thesis, the hybrid driving-stepping locomotion planner is extended to incorporate multiple representations with different levels of abstraction. A detailed representation is chosen in the vicinity of the robot. With increasing distance from the robot position, the representation gets coarser and lower-dimensional but semantically enriched. Hence, while spatial precision decreases, scene “understanding” increases. All representations are integrated in a single planning problem in order to avoid undesired effects of coarse-to-fine planning approaches.

Informed heuristics from abstract representations: Several planning approaches incorporate a heuristic function to guide the search towards the goal. An effective heuristic has considerable influence on the planner performance. However, most popular heuristic functions are purely geometry-based and do not include knowledge about the environment or the robot capabilities. This leads to poor planning performances in cases where the optimal path significantly differs from a free space solution, such as in challenging environments.

In this thesis, methods which generate heuristics from abstract representations are presented. Those heuristics have knowledge about the environment and the robot capabilities and provide an informed guidance for the planner accelerating path generation by multiple orders of magnitude compared to popular heuristics. A first presented method relies on manual parametrization of the abstract representation which achieves good results but requires high effort. A second method represents all tuning intensive components as a Convolutional Neural Network (CNN) which is trained on generated artificial data. This achieves an even better abstraction performance while tuning efforts are minimized.

Increased real-world applicability of learning-based planners: Learning-based planners are promising to efficiently solve challenging planning problems since they can learn a certain scene “under-

standing” while exploiting the effects of massive parallelization through implementations on a graphics processing unit (GPU). Value Iteration Networks (VINs) have shown interesting results with goal-directed behavior and generalization capabilities to unseen domains. However, similar to other learning-based planners, their application is restricted by currently available hardware to small, low-dimensional planning domains with limited state space sizes.

In this thesis, an extension for VINs that incorporates multiple levels of abstraction is presented. Again, while regions close to the current robot position are presented in high detail, the representation gets coarser but semantically enriched with increasing distance from the robot. The extension is fully differentiable and is integrated in the network architecture such that it can be learned without considerable further parametrization. It enables the method to plan paths on significantly larger maps and solve planning queries for more complex planning tasks, such as omnidirectional driving in challenging environments, which considerably increases the method’s real-world applicability.

1.2 PUBLICATIONS

Parts of this thesis have been published in journals and conference proceedings. The publications are listed in chronological order:

T. Klamt and S. Behnke:

Anytime Hybrid Driving-Stepping Locomotion Planning

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, Canada, September 2017

Finalist for Best Paper Award on Safety Security and Rescue Robotics

M. Schwarz, M. Beul, D. Droeschel, T. Klamt, C. Lenz, D. Pavlichenko, T. Rodehutsors, M. Schreiber, N. Araslanov, I. Ivanov, J. Razlaw, S. Schüller, D. Schwarz, A. Topalidou-Kyniazopoulou, and S. Behnke:

DRC Team NimbRo Rescue: Perception and Control for Centaur-like Mobile Manipulation Robot Momaro

The Darpa Robotics Challenge Finals: Humanoid Robots To The Rescue, Springer Tracts in Advanced Robotics (STAR), vol. 121, pp. 145-190, April 2018

T. Klamt and S. Behnke:

Planning Hybrid Driving-Stepping Locomotion on Multiple Levels of Abstraction

IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, May 2018

T. Klamt and S. Behnke:

Towards Learning Abstract Representations for Locomotion Planning in High-dimensional State Spaces

IEEE International Conference on Robotics and Automation (ICRA),
Montreal, Canada, May 2019

D. Schleich, T. Klamt, and S. Behnke:

Value Iteration Networks on Multiple Levels of Abstraction

Robotics: Science and Systems (RSS),
Freiburg, Germany, June 2019

T. Klamt, M. Schwarz, C. Lenz, L. Baccelliere, D. Buongiorno, T. Cichon, A. Di Guardo, D. Droeschel, M. Gabardi, M. Kamedula, N. Kashiri, A. Laurenzi, D. Leonardis, L. Muratore, D. Pavlichenko, A. Selvam Periyasamy, D. Rodriguez, M. Solazzi, A. Frisoli, M. Gustmann, J. Roßmann, U. Süss, N. G. Tsagarakis, and S. Behnke:

Remote Mobile Manipulation with the Centauro Robot: Full-body Telepresence and Autonomous Operator Assistance

Accepted for Journal of Field Robotics (JFR), Wiley, 2019

T. Klamt, D. Rodriguez, L. Baccelliere, X. Chen, D. Chiaradia, T. Cichon, M. Gabardi, P. Guria, K. Holmquist, M. Kamedula, H. Karaoguz, N. Kashiri, A. Laurenzi, C. Lenz, D. Leonardis, E. Mingo Hoffman, L. Muratore, D. Pavlichenko, F. Porcini, Z. Ren, F. Schilling, M. Schwarz, M. Solazzi, M. Felsberg, A. Frisoli, M. Gustmann, P. Jensfelt, K. Nordberg, J. Roßmann, U. Süss, N. G. Tsagarakis, and S. Behnke:

**Flexible Disaster Response of Tomorrow
Final Presentation and Evaluation of the CENTAURO System**

IEEE Robotics and Automation Magazine, Special Issue on Humanoid Robot Applications in Real World Scenarios, vol. 26(4), pp. 59-72, December 2019

The following publication is closely related to the presented content in this thesis and has been written during the time in which the presented research has been conducted:

T. Klamt, D. Rodriguez, M. Schwarz, C. Lenz, D. Pavlichenko, D. Droeschel, and S. Behnke:

Supervised Autonomous Locomotion and Manipulation for Disaster Response with a Centaur-like Robot

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),
Madrid, Spain, October 2018

1.3 OUTLINE

This thesis consists of seven chapters of which Chapter 3 to Chapter 6 describe the scientific contributions. Each of these chapters contains

an individual evaluation and discussion of the presented method. Presented approaches usually build upon the content of previous chapters. Nevertheless, chapters are written self-contained such that they can be read individually. Figure 1.1 gives an overview of the thesis and the relation between chapters.

Chapter 2 gives a comprehensive overview of related work in the fields of hybrid driving-stepping robotic platforms, robot motion planning methods, and their extension to incorporate multiple representations. It further describes relevant applications of those methods to plan robot locomotion in a variety of domains. This related work builds the basis for the scientific contributions described in this document.

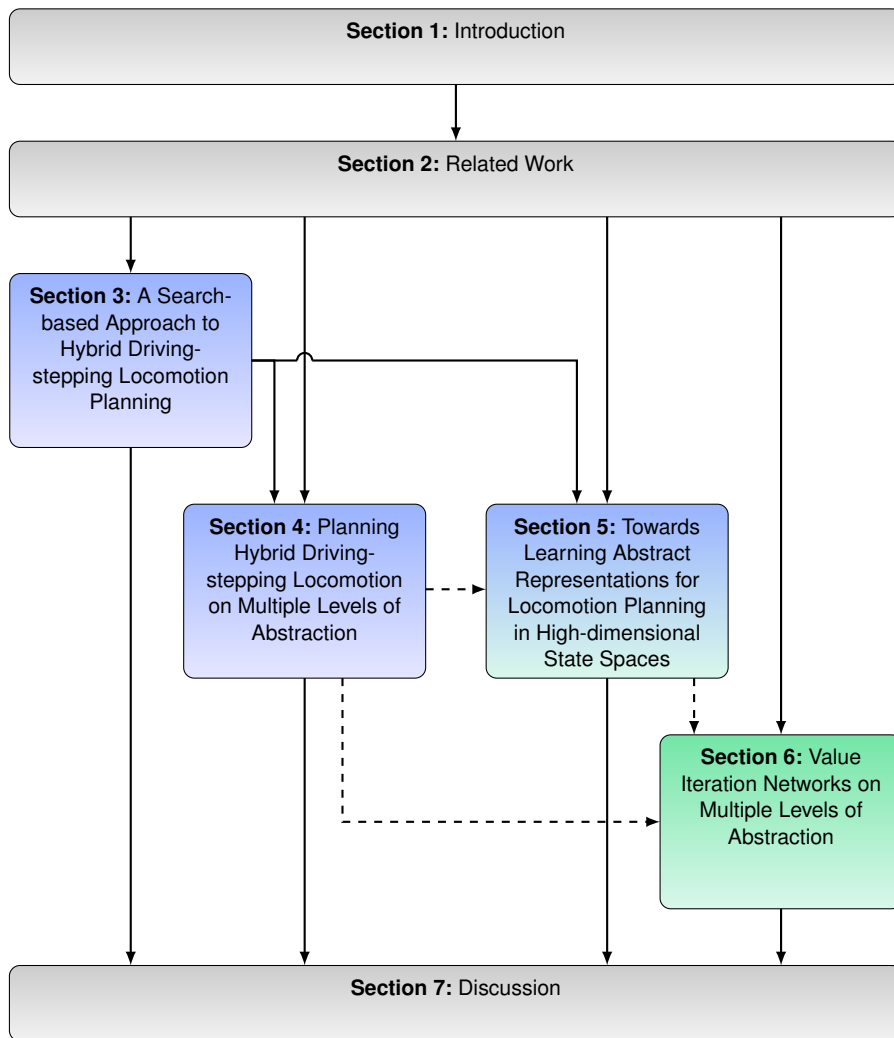


Figure 1.1: Overview of the document structure. Blue background color indicates search-based planning approaches, green indicates learning-based planning approaches. Continuous lines indicate that chapters build on each other. Dashed lines visualize if a chapter is motivated by results from another chapter. All chapters are self-contained and can be read isolated.

In **Chapter 3**, a search-based approach to hybrid driving-stepping locomotion planning in challenging environments is presented. It is described how the employed planning method and the corresponding planning representation are derived from the characteristics of the robot platforms and the target environments. The planner is capable of generating sub-optimal paths for environments of limited size and complexity in feasible time.

Chapter 4 extends the planner described in Chapter 3 to incorporate multiple planning representations with different degrees of abstractions. While the environment and robot are described in high detail in the vicinity of the robot, the representation gets coarser and semantically enriched with increasing distance from the robot. It is described how the different representations are jointly incorporated by the planner or used to generate an informed heuristic. The parametrization of the different representations is manually tuned. This enables the planner to provide paths for significantly larger and more complex queries, compared to the planner in Chapter 3, in feasible time.

Chapter 5 describes a further extension to the planner presented in Chapter 3. Inspired by the results of Chapter 4, an abstract representation is developed to generate an informed heuristic for the planner. However, the tuning intensive environment representation and cost function are replaced by a CNN which is trained on generated artificial data. This shows similar planning performance to the planner described in Chapter 4, while the abstraction quality is even improved, and manual tuning efforts are minimized.

Chapter 6 goes a step further. While Chapter 5 demonstrates that a learning-based approach can support a traditional planning approach, this chapter investigates if the whole planning problem can be solved by a learning-based planner. Value Iteration Networks (VINs) are a promising approach in this context but are limited to small state spaces. Inspired by the results of Chapter 4 and Chapter 5, VINs are extended to incorporate multiple levels of abstraction. This increases their applicability to more complex and larger planning problems.

Chapter 7 finally concludes the document. Achieved results are discussed, suitable applications are described, and potential future works are proposed.

RELATED WORK

In this chapter, we give an overview of the state of the art which forms the basis for the work described in this thesis. Hybrid driving-stepping locomotion requires robots which can perform both locomotion strategies, but which may vary in their implementation details. Section 2.1 gives an overview of respective platforms. For planning corresponding locomotion, we give an overview of motion planning algorithms in Section 2.2. Planning performance can be improved by employing multiple representations. Section 2.3 describes existing works which address this topic. Finally, Section 2.4 summarizes relevant examples of motion planning algorithms applied to robot locomotion planning.

2.1 HYBRID DRIVING-STEPPING LOCOMOTION ROBOTS

Driving locomotion and stepping locomotion possess complementary advantages. Driving is well suited to traverse sufficiently flat terrains in a fast, energy efficient, and stable manner. However, it is limited to suitable terrains with limited slopes and height differences, and obstacle-free paths. Stepping locomotion is usually slower, less stable, and has higher energy consumption, but enables robots to overcome much more challenging terrains, since it only requires adequate footholds. Hence, combining these two locomotion modes is a promising idea for application domains in which a wide variety of terrains has to be traversed. Example domains are disaster response, search and rescue scenarios, delivery service, or planetary exploration. Corresponding platforms vary in their complexity and kinematic capabilities.

A simple method to enable robust locomotion in rough terrain is to attach active wheels to passive limbs which function as a suspension system. Siegwart et al. (2002) proposed the Shrimp platform with six motorized wheels in a rhombus configuration attached to passive spring suspension arms (see Figure 2.1 a). Takahashi et al. (2006) proposed a quadruped robot where each axis can passively rotate (yaw) and can be actively shifted in the longitudinal direction (see Figure 2.1 b). Although these passive leg designs do not allow for lifting individual feet, they enable the proposed platforms to overcome rough terrain such as climbing stairs.

In 2015, the DARPA Robotics Challenge (DRC) pushed research teams to develop robots which are capable of mobile manipulation including navigation in terrain with variable complexity. Four of the

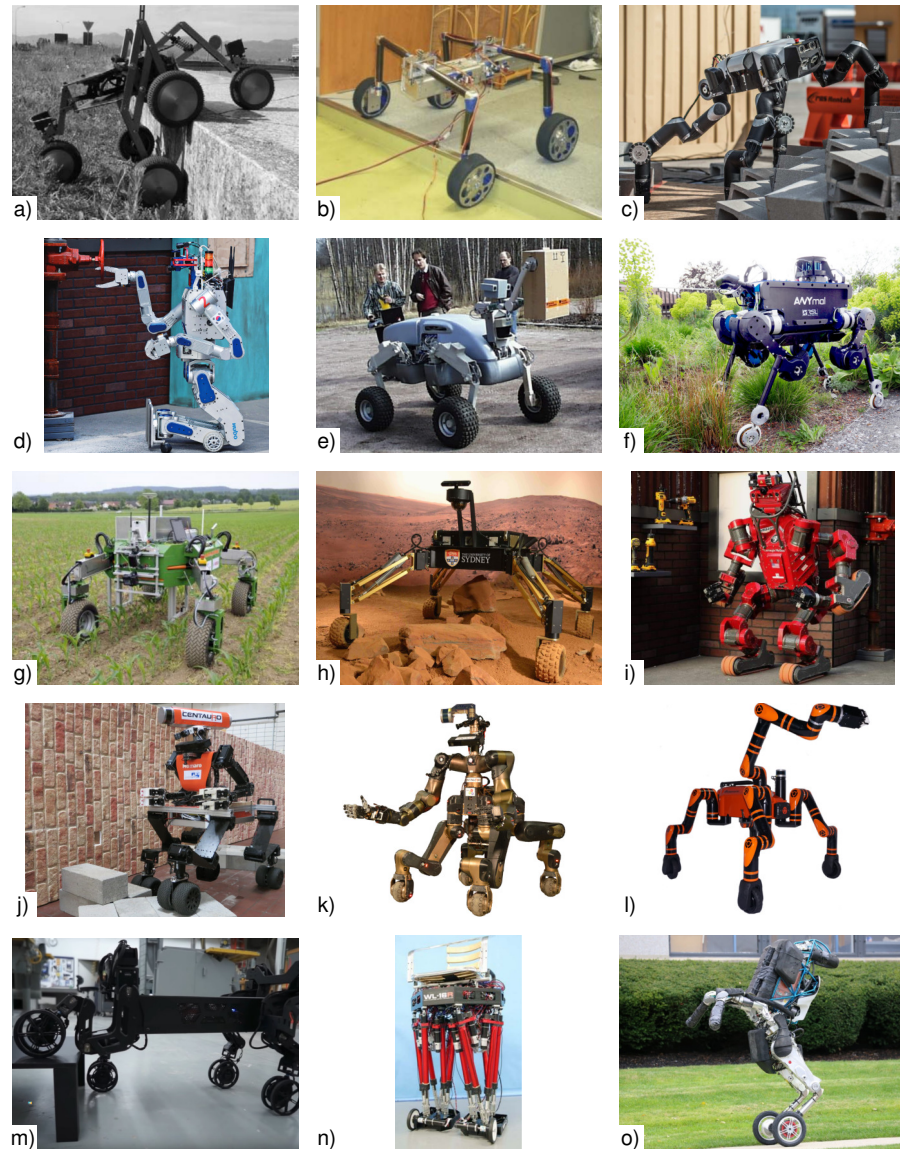


Figure 2.1: Examples of hybrid driving-stepping robot platforms. References refer to image sources. *a)* Shrimp (Siegwart et al., 2002), *b)* Platform by Takahashi et al. (Takahashi et al., 2006), *c)* RoboSimian (Hebert et al., 2015), *d)* DRC-HUBO (Zucker et al., 2015), *e)* WorkPartner (Halme et al., 2001), *f)* ANYmal on Wheels (Bjelonic et al., 2019), *g)* BoniRob (Biber et al., 2012), *h)* MAMMOTH (Reid et al., 2016a), *i)* CHIMP (Fontaine, 2016), *j)* Momaro (Schwarz et al., 2017), *k)* Centauro (Klamt et al., 2019b), *l)* RoboMantis (Motiv Robotics, 2019), *m)* Hyundai Cradle Walking Car Concept (The Wheel Network, 2019), *n)* WS2/WL16 (Hashimoto et al., 2005), *o)* Handle (Boston Dynamics, 2019).

five best teams chose a hybrid driving-stepping platform with active limbs and active wheels which is an indication for the comprehensive flexibility of this locomotion strategy. Two of these platforms possessed a kinematic structure which requires posture changes to switch between driving and stepping locomotion: RoboSimian is a quadrupedal robot with four 7-DoF generalized limbs which can be used for walking and for manipulation (Hebert et al., 2015). It further has two active wheels at its trunk and two caster wheels at two of the limbs allowing for driving locomotion (see Figure 2.1 c). To switch between stepping and driving locomotion, the robot has to lower itself to a predefined sitting configuration. The DRC winner platform DRC-HUBO (Zucker et al., 2015) is a humanoid robot, which, in addition to walking, is able to lower itself to a kneeling configuration and perform driving locomotion over flat terrain. This is possible due to four small wheels at its knees and ankles (see Figure 2.1 d).

Although RoboSimian and DRC-HUBO provide isolated driving and stepping locomotion capabilities, the necessity for posture changes to switch between them prohibits additional valuable locomotion features. Platforms which have wheels attached to the end of their limbs, such that those wheels also function as feet during walking can switch between the two locomotion modes in any configuration. Furthermore, this design allows to change the configuration of ground contact points (which we refer to as the robot footprint) while under load.

In other words, those robots can drive individual feet relative to the robot base while under load. This is helpful for, e.g., obtaining stable configurations for stepping. Hence, most hybrid-driving stepping robots follow this design approach.

An early example is the WorkPartner platform (Halme et al., 2001, 2003), a quadrupedal robot designed to work interactively with humans in outdoor environments (see Figure 2.1 e). It possesses four active 3-DoF legs ending in active wheels. Steering is realized through a yaw joint between the front and rear joint of the base. Halme et al. proposed *rolking*, a mix between driving and walking to navigate challenging terrain. A similar kinematic structure is featured by the ANYmal robot with attached wheels (Bjelonic et al., 2019) (see Figure 2.1 f). In contrast to WorkPartner, it is missing the yaw joint in its base. Instead, steering is realized through leg motions or walking.

Examples from other domains are the agricultural robot BoniRob (Ruckelshausen et al., 2009) (see Figure 2.1 g) and the planetary rovers MAMMOTH (Reid et al., 2014) (see Figure 2.1 h) and Sherpa (Cordes et al., 2014). All three systems feature four 3-DoF legs with parallel kinematics ending in actively driven wheels enabling omnidirectional driving. A significantly more complex planetary rover is the NASA ATHLETE possessing six legs with six DoF each, ending in wheels (Wilcox et al., 2007).

Robot Footprint

The footprint of a robot configuration is the individual set of ground contacts relative to the robot base center.

Drive a Foot

The maneuver of driving a robot foot describes the motion of an individual foot relative to the robot base while keeping ground contact and while being under load. This maneuver is only feasible for robots possessing legs ending in wheels.

A to some extent unique robot is CHIMP (Stentz et al., 2015), another DRC participant. It has a roughly anthropomorphic body plan and powered tracks at each of its feet and wrists (see Figure 2.1 i). Hence, it is able to perform statically stable bipedal and quadrupedal driving and walking locomotion.

The fourth hybrid locomotion DRC participant was Momaro with its centaur-like body plan (Schwarz et al., 2017). It possesses a complex kinematic structure to address a wide range of mobile manipulation tasks (see Figure 2.1 j). Regarding locomotion, its quadrupedal lower body consists of four 4-DoF legs ending in an actively driven, 360° steerable pair of wheels each. Hence, posture changes are not needed to switch between locomotion modes. This enables Momaro to effectively navigate a wide range of terrains. The centaur-like platform Centauro (Klamt et al., 2019b) can be seen as a successor of Momaro. It has a similar body plan which is extended by an additional hip yaw joint for each leg and improved, compliant actuation (see Figure 2.1 k).

A similar lower body design with four 5-DoF legs and omnidirectional driving was chosen for RoboMantis (Motiv Robotics, 2019) (see Figure 2.1 l). Interestingly, RoboMantis is the predecessor of RoboSimian and one of the main kinematic differences is that posture changes are no longer required to switch between driving and stepping locomotion. The same lower body design was chosen for the recently presented Hyundai Cradle Walking Car Concept (The Wheel Network, 2019) (see Figure 2.1 m).

Finally, few platforms address the idea of bipedal hybrid driving-stepping locomotion. Hashimoto et al. (2005) extended the bipedal robot WL-16 which is capable of carrying a human is extended by WS-2, a shoe with attached wheels (see Figure 2.1 n). In 2016, Boston Dynamics introduced its bipedal robot Handle (Boston Dynamics, 2019) which shows impressive dynamic locomotion, including jumping, with its two 2-DoF legs and attached wheels (see Figure 2.1 o).

State vs. Configuration

In this document, a robot state contains the set of parameters completely describing how a robot is positioned in the environment. In contrast, the robot configuration only describes the position of the limbs relative to the base (e.g., the leg configuration) while neglecting the absolute robot position and orientation.

2.2 ROBOT MOTION PLANNING METHOD OVERVIEW

Motion planning is one of the major challenges in robot development. Given a robot start state and a desired goal state, a motion planning algorithm shall output a path or trajectory (if temporal information is also included) to reach the goal state while staying in the space of feasible states, and frustrating self-collisions and collisions with the environment. The path shall be optimal or sub-optimal with respect to a given cost function. Furthermore, computation time and hardware requirements are limited, since planning algorithms usually run on the robot hardware during missions. Over the years, many methods have been proposed to solve robot motion planning problems. They can be categorized according to their basic principles which

are either search-based planning (Section 2.2.1), sampling-based planning (Section 2.2.2), optimization-based planning (Section 2.2.3), or learning-based planning (Section 2.2.4). This section gives a theoretical overview of those methods.

2.2.1 Search-based Planning

Search-based motion planning algorithms discretize the state space to a mathematical graph. Nodes represent states while edges represent actions and their specific weights represent the corresponding costs. The discretization resolution has to be chosen carefully and is a trade-off between the result quality and computation time. A fine, uniform resolution is required to represent robot states with sufficient precision while resulting in large state spaces which are costly to be searched. A coarse resolution leads to smaller state spaces and accelerates planning while coming along with the risk that important states are discarded during discretization leading to wrong results or results which are far away from an optimal solution.

Subsequently, graph search is performed on this generated graph. A basic algorithm is the Dijkstra algorithm (Dijkstra, 1959) which performs a deterministic, greedy search starting from the start state. For each node n , its shortest distance from the start node $g(n)$ and its predecessor $p(n)$ on this path are saved. At initialization, the distance for the start node is assigned zero $g(s) = 0$ and $g(n_i) = \infty$ for all other nodes. The algorithm then iteratively visits the unvisited node with the shortest known distance value. It computes the distance for each of its neighboring nodes m_j as the sum of the distance of the current node and the costs of the respective action to the neighbor node $g(n) + c(n, m_j)$. If this distance is lower than the saved neighbor node distance $g(m_j)$, its distance value is updated, and the current node is saved as the neighbor's predecessor $p(m_j) = n$. The algorithm stops if either no unvisited nodes are left or if the goal node is visited. In the latter case, the reversed resulting path can be generated by starting at the goal node and following the predecessor entries until reaching the start node. By reversing this path, the optimal solution with respect to the chosen discretization is obtained.

The Dijkstra algorithm is extended by the A* algorithm (Hart et al., 1968). It uses a heuristic function $h(n, g)$ estimating the distance from a given node n to the goal node g . The sum of the distance from the start node to the current node and the heuristic value from the current node to the goal node $f(n) = g(n) + h(n, g)$ can be used as an estimate for the path length from the start node, through the current node, and to the goal node. Instead of visiting the node with the shortest distance from the start node, the A* algorithm visits those nodes with the lowest f -value—the shortest estimated resulting path length. Obviously, the performance of this algorithm is very dependent on

the used heuristic function. To guarantee optimality, it always has to underestimate the real shortest path length. An often-used heuristic for motion planning is the Euclidean distance since it is easy to implement and can be computed efficiently.

To speed up planning, the heuristic function can be weighted with a factor $\epsilon > 1$ increasing the attraction of the search front towards the goal. This results in bounded sub-optimal results which are at maximum ϵ times more expensive than the optimal solution.

Anytime Repairing A* (ARA*) (Likhachev et al., 2004) uses this effect to quickly generate a first sub-optimal solution with a large ϵ and iteratively approach the optimal solution by decreasing ϵ and performing additional searches while processing results from previous searches.

Further extensions of A* consider dynamic environments (Stentz, 1995), bi-directional planning (Nannicini et al., 2008), incorporation of multiple heuristics (Aine et al., 2016), or parallelization on GPUs (Zhou and Zeng, 2015), to only name a few.

2.2.2 *Sampling-based Planning*

Sampling-based planning algorithms randomly draw samples from the state space and try to find connections between these samples to obtain paths from the start to the goal state.

An established method is the Rapidly-exploring Random Tree (RRT) algorithm (Lavalle, 1998). It builds a space-filling tree, rooted at the start state, to efficiently search high-dimensional, non-convex state spaces. For each drawn sample, the nearest state in the existing tree is searched. If the connection from the sample to this nearest state is collision-free and obeys all constraints, the sample is added to the tree. However, the length of this connection is limited by a growth factor. If the connection is longer, an intermediate state with the maximum allowed distance to the nearest state is interpolated between the two and is used instead of the originally sampled state. Due to the random sampling character, tree expansion is biased towards large, unsearched areas. Furthermore, the algorithm can be biased towards the goal or other specific areas by introducing a certain probability to draw samples from these desired states. After finding an initial solution, further samples improve the result quality. RRT converges towards the optimal solution with infinite samples and can be categorized as a single-query algorithm.

The Probabilistic Roadmap (PRM) algorithm (Kavraki et al., 1996) is a second widely used sampling-based method. In contrast to RRT, it can be categorized as a multi-query algorithm. The planner consists of two phases, a learning phase and a query phase. In the learning phase, a roadmap, represented as a graph, is built. This is done by iteratively sampling states from the state space. If a sampled state is fea-

sible, it is added to the graph. Subsequently, a local planner is used to find connections to neighbor states. Typically, this search is restricted to all neighbors within a given distance or the k nearest neighbors. Feasible connections are added to the graph. This is repeated until a desired roadmap density is obtained. In the query phase, for a given start and goal state, both are connected to the roadmap and a graph search algorithm, such as the Dijkstra algorithm, is used to find the shortest path in this graph. Similar to RRT, PRMs are probabilistically complete.

Several works proposed extension to RRT and PRM. The well-established RRT* (Karaman and Frazzoli, 2011) extends the standard RRT algorithm by two features improving the search for nearest neighbors and introducing local tree rewiring. Similarly, PRM* (Karaman and Frazzoli, 2011) improves PRM. Further extensions for RRT address the weakness of planning through narrow passages (Zhang and Manocha, 2008), improve sampling after an initial solution has been found (Gammell et al., 2014), or combine this with bi-directional searches (Burget et al., 2016), to only name a few examples. Similarly, PRM's capabilities to handle narrow passages are improved in (Hsu et al., 2003). An overview over PRM extensions is given in (Geraerts and Overmars, 2004).

A sampling-based method which combines the ideas of RRT and PRM is the Fast Marching Tree (FMT*) algorithm (Janson et al., 2015). A set of samples is drawn from the state space at first. In a second step, a tree which is routed at the start state is grown through this set.

In many cases, output paths of sampling-based planners need a successive optimization to, e.g., remove redundant, unfeasible, or jerky motions and to obtain a path which can be executed by the robot (Ratliff et al., 2009).

2.2.3 Optimization- and Potential Field-based Planning

The initial idea of optimization-based methods is to improve the path quality of an input path which is already feasible. While for RRT and PRM, after an initial solution has been found, further sampling also can be seen as an optimization. The employment of an explicit optimization method allows to introduce additional constraints which were not present during the initial path generation to limit its complexity. Examples are the Elastic Bands (Quinlan and Khatib, 1993) and Elastic Strips (Brock and Khatib, 2002) algorithms.

While the input trajectory for the above-mentioned algorithms has to be collision-free, this requirement was dropped for more recent methods. Hence, a naïve input trajectory, such as the linear connection between the start and goal state, is sufficient, which extends those optimizers to stand-alone planning algorithms. Covariant Hamiltonian

Optimization for Motion Planning (CHOMP) (Ratliff et al., 2009) uses covariant gradient techniques for the cost function and is able to plan paths including higher-order dynamics. Since it relies on a gradient of the cost function, it is vulnerable to getting stuck in local minima. The Stochastic Trajectory Optimization for Motion Planning (STOMP) algorithm (Kalakrishnan et al., 2011b) overcomes this issue by using stochastic optimization. Iteratively, noisy trajectories are sampled around the best currently known trajectory, their quality is assessed with respect to the given cost function, and the best known trajectory is shifted accordingly. In (Pavlichenko and Behnke, 2017) this has been extended to assign costs not to the keyframes of a trajectory but to the transitions between them. Nevertheless, for complex planning problems and infeasible initial trajectories, those methods might have problems to generate feasible solutions due to the lack of environment understanding. Their strength is the optimization within feasible trajectory spaces and not the generation of an initial feasible solution.

Similar to the Elastic Bands and Elastic Strips method, which internally use potential field-like computation, Potential Fields (Khatib, 1986) themselves are another approach to motion planning. Inspired from the concept of electrical charges, the environment is represented with attractive and repulsive potentials (e.g., obstacles have a repelling force while the goal has an attractive force). The robot behaves in this environment like an electrical charge would do, finding a way to the bottom of the potential sum. However, this method is vulnerable to local minima and oscillations which limits its application to environments with limited complexity.

2.2.4 *Learning-based Planning*

In recent years, intensive research has been performed on learning-based motion planning which uses machine learning approaches to solve navigation tasks without performing extensive searches. In general, one can distinguish between supervised learning methods and reinforcement learning (RL) methods. In supervised learning, each training example comes with a desired solution. This is not the case for RL, where the agent explores the state space by itself, receives rewards for its actions, and learns from its experience. However, the boundaries between the two are not exactly defined, some of the proposed methods can be used for both, and since we are mainly interested in the requirements, capabilities, and limitations of learning-based planners in general, they are not further distinguished.

Levine et al. (2016), trained a CNN to directly map camera images to motor commands to solve manipulation tasks. The same was done in (Bojarski et al., 2016) for steering commands of a self-driving car.

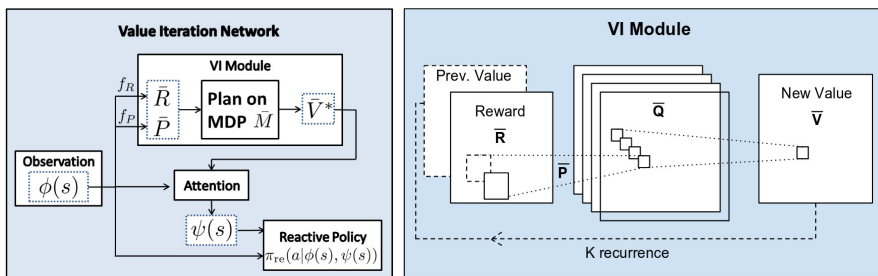


Figure 2.2: Value Iteration Networks (VINs). *Left*) General structure of the method. *Right*) The Value Iteration module. From Tamar et al. (2016).

Those systems learn how to act in a given situation, but they are lacking a long-term goal-directed behavior.

This has been addressed by VINs (Tamar et al., 2016): most planning problems can be described as a Markov Decision Process (MDP), a discrete mathematical framework describing decision problems as a state and action space, transition probabilities, and a reward for each state-action pair (Bellman, 1957). The goal is to find a policy which results in high long-term rewards. A common method to find an optimal policy is Value Iteration (VI) which calculates for each state the expected long-term reward by iteratively applying the Bellman equation (Bellman, 2013 [1957]). The optimal policy is obtained by always moving the agent to the state with the highest expected long-term reward. VINs approximate the VI algorithm by rewriting it as a CNN, as depicted in Figure 2.2. Since this planning module is fully differentiable, standard backpropagation can be used to learn parameters of the model such as the reward function or transition probabilities. In contrast to the above mentioned end-to-end CNNs which directly map camera images to motor commands in a strict feed-forward way, VINs realize multiple iterations in an inner loop within the CNN architecture to propagate values through the representation. This enables VINs to “understand” the desired goal-directed behavior and to generalize well to unseen domains. VINs have been evaluated on small 2D grid world tasks.

VINs have been extended in several works. Niu et al. (2018) proposed Generalized Value Iteration Networks which work on arbitrary graph structures and can be applied to real-world data like street maps. QMDP-Nets (Karkus et al., 2017) handle partially observable environments and express VI through a CNN. Similar to original VINs, they are also trained on small 2D occupancy maps. During evaluation, it is shown that QMDP-Nets can generalize to solve larger 2D grid world tasks.

Finally, Universal Planning Networks (UPNs) (Srinivas et al., 2018) learn useful latent space representations from images of the current scene and the desired goal scene. Gradient descent planning and

the iteration over action sequences in the learned internal representation are used to obtain motion trajectories. In contrast to VINs and QMDP-Nets, UPNs show good performance for planning problems with more than two dimensions. Since the gradient descent planner is time consuming, planning is limited to small environments. Impressively, UPNs are capable of generalizing to modified robot morphologies.

For all learning-based planning approaches mentioned, it is true that the necessary amount of training data and the required network complexity strongly depend on the size of the considered maps and the number of dimensions of the state space. Thus, those planners are restricted to small maps or low-dimensional planning problems.

In addition, learning-based methods are often used to handle sub-problems of the navigation task such as solving local planning tasks which require limited goal-directed behavior but focus on the generation of feasible motions while the generation of an initial global path is not addressed. In (Holden et al., 2017), human-like bipedal locomotion over rough terrain towards very local goals was learned. In (Peng et al., 2017), a learned hierarchical planner for bipedal locomotion on flat surfaces with obstacle avoidance was learned. A high-level planner generated local goals to reach a global goal while considering obstacles. A low-level planner provided robust bipedal motions to reach these local goals.

It applies to all of the presented learning-based planners that more complex planning tasks require larger amounts of training data which usually cannot be generated from real-world demonstration. Hence, besides the method itself, it is necessary to provide methods to generate the required amount of artificial training data and to ensure applicability of the trained behaviors to real world problems.

2.3 PLANNING ON MULTIPLE REPRESENTATION

While 2D and 3D planning problems of moderate size can be seen as solved with traditional search-based and sampling-based planning methods, motion planning for larger and higher-dimensional tasks is one of the main research topics in robotics. The state space grows linearly with the covered area and exponentially with the considered planning dimensions which pushes those planning methods quickly to their limits. Large state spaces lead to extensive searches resulting in long planning times. Memory requirements increase simultaneously.

To solve this problem, a logical extension of the aforementioned planning methods is the introduction of multiple planning representations. An additional, coarser representation describes the same problem in a smaller state space and can be used, e.g., for coarse-to-fine planning: an initial path is quickly planned in the coarse represen-

tation. In a second planning step, this plan is refined, meaning that a second planner, which is restricted to those states covered by the coarse path, plans a detailed solution in the fine representation. However, if the coarse representation assesses a scene incorrectly, this causes problems for the detailed planner. A second method to utilize an additional representation which avoids this problem is the incorporation of all representations in one planning step. While, ideally, a large part of the scene can be represented in the coarse representation, regions of special interest are represented in the fine representation. Depending on the region, the planner incorporates the corresponding representation. Finally, a coarse representation can also be used to generate a heuristic. Since planning times are short in this representation, it can be used to quickly provide estimates for the shortest path length between two states in the detailed representation. All three methods can be iteratively extended to incorporate more than two representations.

To generate such a coarse representation, one can either decrease the planning resolution or introduce a lower-dimensional robot or action representation. Behnke (2003) proposed a general concept for A*-based multiresolution planning with a decreasing resolution with increasing distance from the robot. The KPIECE algorithm (Şucan and Kavraki, 2009) introduces multiple levels of environment discretization to dynamically feasible sampling-based planning. González-Sieira et al. (2016) applied high resolution in areas of high environment complexity. Resolution decreased with increasing distance from these areas. Similarly, Pivtoraiko and Kelly (2008) applied different sets of state transitions to different areas of the environment. Bohlin (2001) generated an initial plan in a coarse resolution first and refined this plan into a finer resolution. High-resolution planning was only applied to parts of the map. Hence, the state space decreased, and planning performance increased, compared to pure high-resolution planning. Botea et al. (2004) generated a coarse representation by separating the map into clusters with defined entry points and precomputed paths within these clusters. During search queries, it was sufficient to plan in this coarse representation and search for connections to the start and goal state in the respective clusters. Multiple levels with decreasing resolution are also utilized in the learning-based planning domain. Hierarchical Value Iteration Networks (HVINs) (Tamar et al., 2016) extend VINs to start computation on coarser maps and subsequently locally refine them (see Figure 2.3) which enables the algorithm to solve significantly larger tasks.

Similar to multiresolution planning, several approaches utilize multiple representations with different planning dimensionalities to decrease planning complexity. Kohrt et al. (2012) generated an initial plan in a low-dimensional search space and re-planned in the high-dimensional search space by only considering those states that are

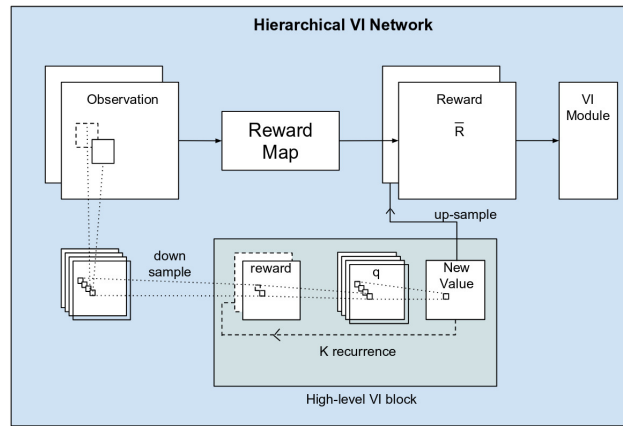


Figure 2.3: Hierarchical Value Iteration Networks (HVINs) with two levels. From Tamar et al. (2016).

part of the low-dimensional plan. Gochev et al. (2011) planned paths in a low-dimensional state space and only switched to high-dimensional planning in those areas where low-dimensional planning could not find a solution. Similarly, Zhang et al. (2012) planned in 2D and only switched to high-dimensional planning in the robot vicinity and at key points.

To achieve further planning acceleration, it is an obvious idea to combine multiresolution and multidimensional planning. However, only few works, such as Petereit et al. (2013) have addressed this. Different planning dimensionalities and resolutions were utilized by using different sets of motion primitives. A fine resolution was only considered close to the start and goal state and close to obstacles. A high planning dimensionality was considered for states which would be reached within a given time interval. This allowed the planner to provide detailed plans close to the robot while planning times stayed feasible.

All the presented methods have in common, that a coarser representation is obtained by purposely discarding information. This information loss is limited before it leads to wrong situation assessments resulting in wrong paths. An example is a 2D height map representation including a flight of stairs and a ramp: two obstacles which require different locomotion strategies. With decreasing resolution, those two objects become indistinguishable which demonstrates that essential information is discarded. This problem has been addressed by abstraction: instead of only discarding information, coarser representations are enriched with additional descriptive features. Hence, while spatial preciseness or dimensionality is decreased, the semantics are increased. Li et al. (2006) have formulated a unified theory of state abstraction for MDPs in general and Holte et al. (1995) have given a theoretical foundation for abstraction for search-based planning.

One of the main challenges in planning with multiple representations (with or without abstraction) is the definition of effective representations which preserve all important information for the respective planning domain. In addition, feasible transitions between the different representations are required. This is challenging and requires a lot of manual tuning efforts, which is a motivation to employ machine learning for this task. Kurutach et al. (2018) recently proposed the InfoGAN architecture which learns an efficient abstract representation and uses, e.g., the Dijkstra algorithm to plan in this abstract representation. Subsequently, plans are projected back to the original representation.

2.4 ROBOT LOCOMOTION PLANNING APPROACHES

The choice of a suitable planning approach is dependent on the robot platform, considered environments, and further constraints (e.g., energy efficiency or acceleration minimization). This section presents a collection of relevant robot locomotion planning approaches in the literature to provide an understanding of established methods.

Since most mobile ground robots provide pure driving with wheels or tracks, or pure legged locomotion, there exist several works that address pure driving locomotion or pure legged locomotion planning in unstructured terrain.

Planning of drivable paths in unstructured environments is dependent on the DoF of the platform. Simple robot designs with a constant shape offer longitudinal and rotational movements. Gerkey and Konolige (2008) used a gradient technique to plan globally optimal paths in a variety of outdoor environments for a differentially driven robot with four wheels. In a recent work, Faust et al. (2018) solved long-range navigation tasks for a differentially driven robot by combining Probabilistic Roadmaps (PRMs) with Reinforcement Learning (RL): a PRM containing waypoints was precomputed for a given map. During path planning, a coarse path was searched in this PRM. Subsequently, a RL-based planner generated dynamically feasible connections between the corresponding waypoints. For search and rescue scenarios, some tracked robots have been extended by tracked flippers. These allow the robots to climb stairs and thus increase capabilities but also planning complexity due to additional shape shifting DoFs. Colas et al. (2013) and Menna et al. (2014) proposed a search-based planning approaches to plan paths for the robot base. Flipper positions were derived from the resulting path and the environment representation in a second hierarchical planning step. A coarse-to-fine approach was proposed by Brunner et al. (2012). At first, a low-dimensional search generated a path for the robot base while neglecting its flippers. A second search in the high-dimensional state space, which included flipper positions, was restricted to the area around

the low-dimensional path. Platforms which offer omnidirectional locomotion increase the state space by another dimension. Ziaei et al. (2014) proposed a potential field-based approach for this problem. However, driving is generally restricted by terrain characteristics such as height differences and slopes which makes it not suitable for very rough terrain and for overcoming obstacles.

Legged locomotion is capable of traversing more difficult terrain because it only requires isolated footholds. The drawback of this locomotion mode is, that motion planning is more complex. Since legs are lifted from the ground repeatedly, the robot has to constantly ensure that it remains stable. Due to the high motion complexity of stepping, path planning is often performed in at least two hierarchical levels. We focus on quadrupedal robots here due to the many similarities to the platforms considered in this work. An approach for dexterous quadruped locomotion planning with RoboSimian (Hebert et al., 2015) was proposed by (Satzinger et al., 2016). Feasible footholds were connected to a foothold plan by graph search. Subsequently, body states were generated from these footholds and connect by RRT-Connect. Kolter et al. (2008) proposed a hierarchical Apprenticeship Learning approach to plan quadrupedal locomotion over small patches of rough terrain for the LittleDog robot (Murphy et al., 2011). Planning was split into body path planning and footstep planning along this path. Both levels were realized through Apprenticeship Learning approaches in which the operator could input the desired behavior. LittleDog was also used by Kalakrishnan et al. (2011a) to demonstrate their quadrupedal locomotion planner. In this work, an approximate body plan was generated by a Dijkstra algorithm and optimized during preprocessing. A footstep planner, which was trained on expert demonstrations, generated feasible trajectories along this approximate body path during queries. A suitable state path was derived from those footholds. Again, considered environments were small terrain patches. Wermelinger et al. (2016) proposed a navigation planning approach for the quadrupedal robot StarLETH. The robot footprint was simplified to either a circle or a rectangle, depending on the situation complexity. An RRT* planner was used to plan path with these footprints on traversability maps. Footstep planning was not performed but a predefined statically stable walking gate was incorporated. This method allowed Wermelinger et al. to plan for maps of considerable size while the missing footstep planning limited the complexity of manageable situations. Perrin et al. (2016) transformed the discrete problem of stepping motion planning into a continuous planning problem by continuously moving disks with the shape of the robot leg reachability through the environment and checking for collisions for each of these disks. RRT or PRM were used to solve the 3D path planning problem. The detailed search for individual footholds and planning of the corresponding motions to reach them

were done in additional planning steps. The method was applied to a biped and a hexapod robot and hence, can easily be transferred to quadruped platforms. Experiments demonstrated planning for small areas of challenging terrain.

In the last years, multiple wheeled-legged robot platforms have been proposed (see Section 2.1). Those platforms are promising since they combine the advantages of both locomotion modes. This comes at the price of an increasing complexity which also has to be represented in the planning of corresponding locomotion. Thus, only few works addressed locomotion planning for such platforms. Fleckenstein et al. (2017) proposed a planning approach for the agricultural BoniRob platform (Biber et al., 2012). The method incorporated omnidirectional driving as well as an additional DoF for each wheel describing its rotational configuration around the hip yaw joint. However, the capability of changing the leg height was not addressed in this work and driving was restricted to flat terrain. A feasible state space size was obtained by introducing a set of motion primitives and by merging multiple wheel positions to configuration classes. A search-based Anytime Repairing A* (ARA*) approach was utilized for path generation. It was evaluated in combination with different heuristics. Especially the incorporation of a heuristic which quickly precomputes shortest path for individual wheels significantly increased the performance. Chen et al. (2018) proposed a RL method to plan short-distance driving locomotion on flat terrain for the Centauro robot. Besides omnidirectional driving, the planner considered changes to the base height and footprint width. Stepping was not considered. In contrast, de Viragh et al. (2019) proposed a controller which includes driving and stepping for ANYmal on Wheels (Bjelonic et al., 2019). The method was only evaluated for short queries on flat, obstacle-free terrain but it is the only method incorporating the system dynamics. A work which addressed wheeled-legged locomotion planning including obstacles was done by Reid et al. (2016b) for the MAMMOTH robot (Reid et al., 2014). They proposed a hierarchical version of the Fast Marching Tree* (FMT*) algorithm: the full state space is decomposed into a lower-dimensional sub-space for which the decomposition method must be manually defined. Subsequently, the method finds a set of paths in this sub-space. Those results are used to bias sampling in the full state space. The method incorporates individual foot positions in the transversal plane, but foot heights are not considered during planning. Hence, steps are not discovered naturally but are inserted as discrete actions into the search tree, if static stability is given. Finally, Dornbush et al. (2018) proposed a search-based approach for multi-modal locomotion planning of a humanoid robot. Although this work did not include driving locomotion, it incorporated different locomotion strategies—namely bipedal walking, crawling, and climbing—into the planning process. A representation

with adaptive dimensionality was utilized and searched by the Multi-heuristic A* algorithm which solved planning tasks of considerable size.

To draw a conclusion from these works in the literature, it can be summarized that there exists only very few works which addressed the problem of hybrid driving-stepping locomotion planning. It can be further seen that works which focus on locomotion planning for complex robot platforms usually use multiple hierarchical planning representations to handle the large state spaces. However, a clear trend how to design and incorporate such representations is not discernible. The literature contains approaches for coarse-to-fine planning, the incorporation of multiple planning representations by a single planner, as well as the generation of an informed heuristic which was based on an additional representation. There is also no clear preference for a certain underlying planning method discernible. The presented works used advanced methods of both search-based and sampling-based planners. Learning-based planners were rarely used and if they were, they were only applied to local sub-problems of limited complexity.

A SEARCH-BASED APPROACH TO HYBRID DRIVING-STEPPING LOCOMOTION PLANNING

This chapter describes an approach to the problem of hybrid driving-stepping locomotion planning. As pointed out in Section 2.4, only few works addressed this topic so far. The proposed methods differ in their underlying algorithms depending on the problem specifications such that no preference for a specific approach is observable. In this work, we address planning for the platforms Momaro and Centauro which are depicted in Figure 3.1. Nevertheless, the method is easily adaptable to similar platforms. Both robots possess a quadruped lower body with legs ending in 360° steerable, actively driven wheels allowing for omnidirectional driving as well as stepping motions. Since no posture change is needed to switch between the two locomotion types, those robots are in addition capable of performing actions which neither pure driving robots, nor pure stepping robots can perform. One example is the motion of individual feet relative to the robot body while maintaining ground contact and being under load. This is an important feature for, e.g., modifying the support polygon shape to facilitate robot stabilization which should be included in the planning. Hence, it is not sufficient to combine isolated driving and stepping locomotion planners.

We consider environments which are typical for disaster response missions. Those include mostly, but not exclusively, human-made structures such as corridors, ramps, staircases, or doors. Due to the disaster, environments might be affected by partially collapsed structures, objects lying around, and other forms of debris. The characteristics of such terrains have many similarities with other domains



Figure 3.1: Hybrid driving-stepping robots employed with the described approach. *Left: Momaro. Right: Centauro.*

of robotic application such as urban delivery services or planetary exploration, which enlarges the potential application width of the developed method.

Section 3.1 gives an overview of the developed method. Section 3.2 to Section 3.5 explain the individual components in detail. The evaluation of the method is described in Section 3.6 while Section 3.7 discusses the results and draws a conclusion.

3.1 SYSTEM OVERVIEW

The system architecture possesses multiple components, as depicted in Figure 3.2. Multiple inputs are given to the system at different frequencies. The goal state input only needs to be set once and describes the desired 3-DoF (x, y, θ) state to which the robot base shall move. In this work, the goal state is intuitively defined by an operator using *RViz*¹ but in other contexts it can also be set from, e.g., a higher-level task planner. A second input to the system are aggregated point clouds which represent the environment. The employed robot platforms Momaro and Centauro are both equipped with a rotating Velodyne Puck 3D laser scanner with spherical field-of-view at their robot head. We use the Simultaneous Localization and Mapping (SLAM) method of Droschel et al. (2017) who employ multiresolution surfel grids to generate a dense 3D map of the environment from laser scanner range measurements. In the used setting, one full 3D scan is acquired every five seconds while inertial measurement unit (IMU) measurements and wheel odometry are incorporated to compensate for sensor motion. The employed SLAM method is also capable of handling other laser scanner setups but the chosen design of a 3D rotating laser scanner is beneficial since it provides information about the ground structure at areas close to the robot, especially at the robot front and sides, facilitating precise locomotion. Finally, information about the current robot state is input to the system. Those include a 6-DoF state of the robot base with respect to a global frame, which is also output by the aforementioned SLAM method, and individual foot configurations relative to the robot base computed from joint states. This information arrives at a high frequency of 50 Hz. All inputs as well as internal communication between modules and the system outputs are realized by messages in the Robot Operating System (ROS) (Quigley et al., 2009).

The system itself is split into four modules. The Map Generator (see Section 3.2) processes aggregated point clouds to suitable map representations for the planning. This is done for each incoming point cloud. The Path Planner (see Section 3.3) generates a path from the current robot state to the desired goal state. It is sufficient to generate a path once. However, path planning can be repeated after an

¹<http://wiki.ros.org/rviz>

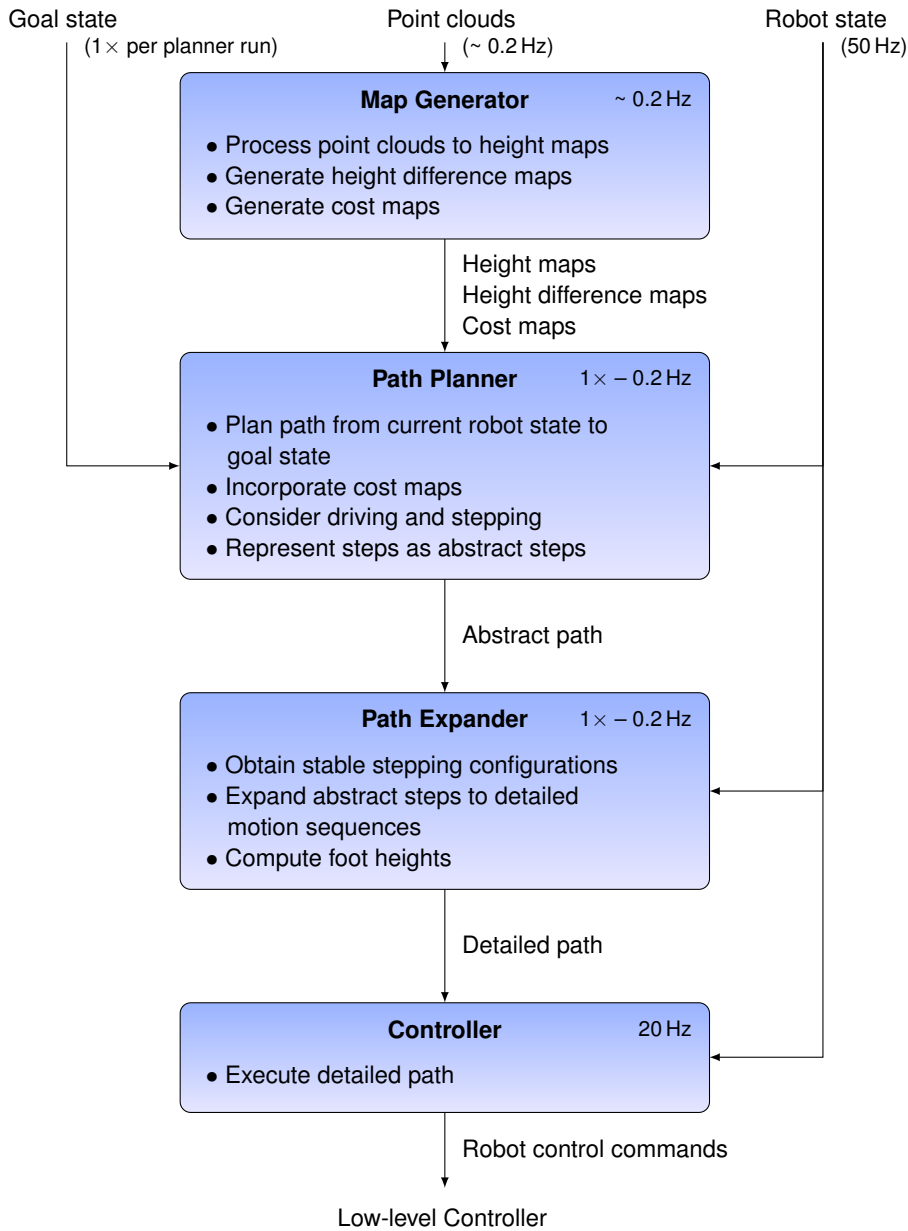


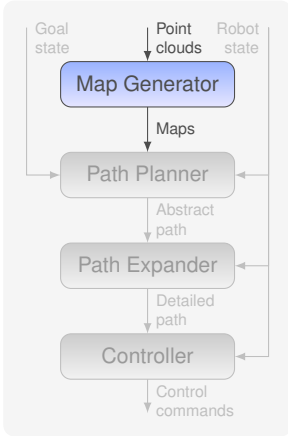
Figure 3.2: Overview of the system architecture. Intermediate modules are executed either once or up to 0.2 Hz.

arbitrary time interval, which is longer than the planning time itself, to incorporate updated environment information in the planning. Each generated path is expanded to an executable motion sequence in the Path Expander (see Section 3.4). Finally, the Controller (see Section 3.5) executes these motion sequences. It runs at a frequency of 20 Hz.

Outputs of the proposed system are robot control commands. For omnidirectional driving, those consist of a 3-DoF (v_x, v_y, v_θ) vector for the desired velocity of the base. The utilized robot control interface by

Schwarz et al. (2017) is capable of computing individual wheel orientations and velocities from these vectors. For foot motions, desired Cartesian 3D foot positions relative to the robot base frame are output for each individual foot. The same robot control interface solves the corresponding inverse kinematics (IK) task and sends joint commands.

3.2 MAP GENERATOR



The Map Generator produces a suitable environment representation for the Path Planner. Such a representation has to incorporate all information which is required to solve the planning problem in a sufficient quality while it needs to allow for efficient computation to obtain fast planning times. The type and resolution of required information is very dependent on the characteristics of the planning domain.

We present 2-dimensional state cost maps as a suitable planning representation. State costs combine individual costs for each robot foot and for the robot base. While this is sophisticated in comparison to often used circular or rectangular robot representations, it considerably extends locomotion capabilities in complex, demanding environments, where precise navigation between obstacles is required. The developed representation allows robots to, e.g., take obstacles between their legs. While the environment has 3-dimensional characteristics, we choose a 2-dimensional representation due to computational efficiency reasons. Other representation methods, such as OctoMaps by Hornung et al. (2013), offer extended capabilities to describe 3-dimensional structures but due to the fact that we only focus on ground heights, their advantages could not be fully exploited. The detailed computation procedure of these cost maps is described in the remainder of this section.

Input to the Map Generator are dense, aggregated point clouds which are generated from laser scanner range measurements. In a first step, those are processed to 2D height maps describing the ground height profile with a resolution of 2.5 cm. This is detailed enough to precisely position feet on obstacles or steps while the representation size still allows for efficient computation. Next, height difference maps are derived from height maps by searching for the maximum unsigned height difference $\Delta h(c_i)$ of each map cell c_i to one of the eight adjacent neighbor cells.

Foot costs are derived from unsigned height differences. For a foot position \vec{f} which is in the map cell $c_{\vec{f}}$, the foot costs

$$C_F(\vec{f}) = C_F(c_{\vec{f}}) = 1 + k_{C_F,1} \cdot \sum_{c_i \in \text{map}} \Delta h(c_i) \cdot w(c_i) \quad (3.1)$$

are computed, where $k_{C_F,1} = 100$ and

$$w(c_i) = \begin{cases} \infty & \text{if } \|c_i - c_{\vec{f}}\| < r_F \wedge \Delta h(c_i) > 0.05 \text{ m,} \\ 1 - \frac{\|c_i - c_{\vec{f}}\|}{r_N} & \text{if } \|c_i - c_{\vec{f}}\| < r_{SA}, \\ 0 & \text{otherwise,} \end{cases} \quad (3.2)$$

with the foot radius r_F and a safety area radius r_{SA} . Foot costs are assigned an infinite value if height differences > 0.05 m occur under the foot, such that it cannot establish reliable ground contact at that position. A foot is assumed to have a circular ground contact area with the radius r_F ($r_{F,Momaro} = 0.12$ m, $r_{F,Centauro} = 0.078$ m). A hard safety margin to obstacles, which is never entered by a foot, can be incorporated by increasing r_F . However, a large safety margin restricts the applicability in cluttered areas where precise locomotion is required. Hence, in a safety area of radius r_{SA} , height differences are accumulated weighted by their distance to $c_{\vec{f}}$. This results in increasing foot costs close to obstacle such that those areas are avoided if possible, but still can be entered if necessary. We set $r_{SA} = 0.3$ m. To summarize, C_F describes the terrain traversability under a robot foot and in its surrounding. It is 1 on flat grounds, increases for uneven terrain and in the vicinity of obstacles, and is infinite if safe foot positioning is not possible. Since foot costs are computed for a 2D foot position, a 2D foot cost map can be quickly precomputed which accelerates planning at run-time. An example is visualized in Figure 3.3.

In addition to foot costs, base costs are computed. For a robot state \vec{r} , the base costs

$$C_B(\vec{r}) = 1 + k_{C_B,1} \cdot \max(-\Delta h_{B,env}(\vec{r}), 0) + k_{C_B,2} \cdot \Delta h_{F,max}(\vec{r}), \quad (3.3)$$

where $k_{C_B,1} = 1$ and $k_{C_B,2} = 0.5$, and with

$$\Delta h_{B,env}(\vec{r}) = h_{uB,max}(\vec{r}) - h_{F,min}(\vec{r}) - \Delta h_{clear,min} \quad (3.4)$$

and

$$\Delta h_{F,max}(\vec{r}) = h_{F,max}(\vec{r}) - h_{F,min}(\vec{r}), \quad (3.5)$$

combine two weighted cost terms: in the first term, the height difference $\Delta h_{B,env}(\vec{r})$ between the robot base and the environment under it is considered. If this height difference is negative, the robot needs to lift its base to overcome the obstacle, and additional costs are induced for this lifting action. $\Delta h_{B,env}(\vec{r})$ is computed by taking the maximum height under the robot base $h_{uB,max}(\vec{r})$ and subtracting the terrain height of the lowest of the four feet $h_{F,min}(\vec{r})$. If this height difference is smaller than the minimum driving clearance $\Delta h_{clear,min}$ ($\Delta h_{clear,min,Momaro} = 0.225$ m, $\Delta h_{clear,min,Centauro} = 0.5$ m), the robot needs to lift its base. The second term induces costs if the

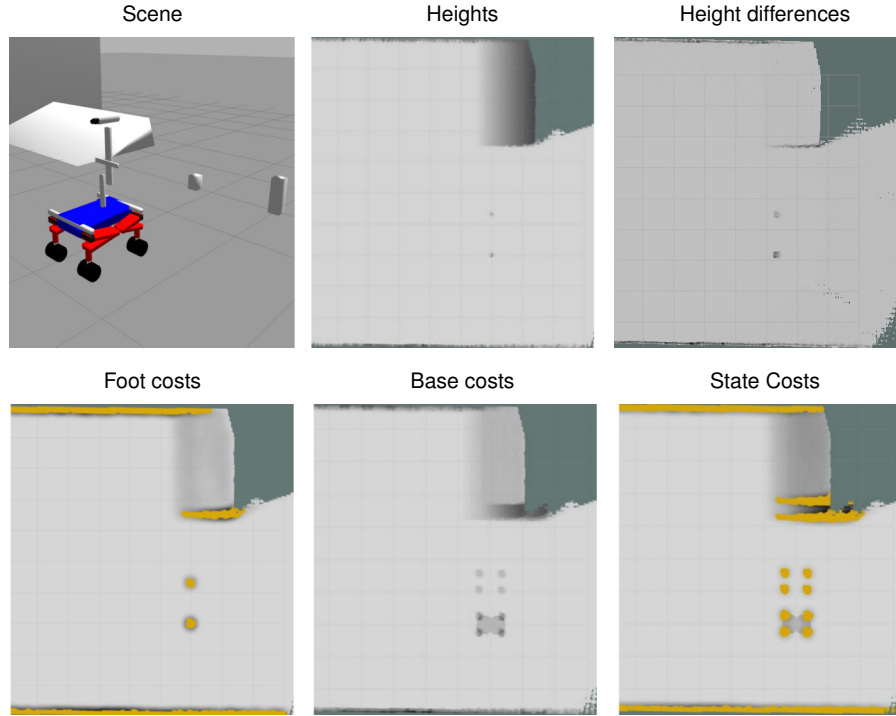


Figure 3.3: Driving cost computation for a simulated scenario in which Momo stands in front of a ramp, a small, and a tall pole. Yellow areas are not traversable by driving, olive areas are unknown. It can be seen in the base costs visualization that the robot can take the small pole between its legs while the tall pole generates costs for lifting the robot body. Base costs are shown for the current robot orientation. State costs combine base costs and foot costs at the four corresponding positions.

robot stands on uneven terrain, such as height differences or slopes. It considers the maximum height difference between the four feet $\Delta h_{F,\max}(\vec{r})$, which is computed by subtracting the terrain height under the lowest foot $h_{F,\min}(\vec{r})$ from the terrain height under the highest foot $h_{F,\max}(\vec{r})$. To summarize, base costs are 1 in even terrain and increase if the robot needs to lift its base to overcome an obstacle or if the robot stands on uneven terrain. In contrast to foot costs, base costs do not only depend on the 2D base position but the orientation of the noncircular-shaped base has to be considered and individual foot positions have to be known. This makes the precomputation of all base costs infeasible. Instead, we approximate the base shape by two disks of radius r_B as depicted in Figure 3.4. A 2D map for $h_{uB,\max}$ can be easily precomputed by inflating the 2D height map with r_B . Similarly, a 2D map for h_F is precomputed by inflating the 2D height map with r_F . To obtain an efficient architecture, those maps are the outputs of the Map Generator while the remaining cost computation is performed at run-time. For the sake of a complete description, this

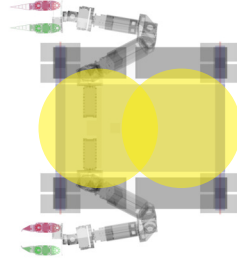


Figure 3.4: Approximation of the base shape. The image shows a model of Momaro in top view. The base shape is approximated by two disks (yellow) to facilitate base costs precomputation.

further computation is described in this section, as well. At run-time, for a given robot state \vec{r} , the center positions of the two disks are computed from the robot base center position and orientation, and the two corresponding $h_{\text{UB,max}}$ -values are extracted from the precomputed map. Next, the four absolute foot positions are computed from \vec{r} , the corresponding h_{F} -values are extracted, and $C_{\text{B}}(\vec{r})$ can be computed.

Finally, for a robot state \vec{r} with its four absolute foot positions $\vec{f}_0 \dots \vec{f}_3$, foot costs and base costs are concatenated to state costs

$$C(\vec{r}) = k_{C,1} \cdot C_{\text{B}}(\vec{r}) + k_{C,2} \cdot \sum_{i=0}^3 C_{\text{F}}(\vec{f}_i) + k_{C,3} \cdot \max_j(C_{\text{F}}(\vec{f}_j)), \quad (3.6)$$

where $k_{C,1} = 0.5$ and $k_{C,2} = k_{C,3} = 0.1$. Foot costs are added in two terms. The first foot cost term computes the average over all four foot cost values. However, this does not distinguish a situation in which all four feet experience slightly increased costs, e.g., when standing on a ramp, from a situation in which three feet are on flat terrain and one foot is at a very challenging and expensive position, although the first situation is preferable. The second foot cost term only considers the most expensive foot costs but neglects costs of the three remaining feet which is neither desirable. A combination of both foot cost terms achieves the desired situation assessment. In summary, $C(\vec{r})$ is 1 on perfectly even terrain and describes the costs to place a robot in the given state \vec{r} on the map. Again, this computation is too expensive to be precomputed, but it is performed during run-time only for the required robot states.

Although the robot has to navigate on three-dimensional ground contours, a 2D environment representation has been chosen on purpose as it is considerably more computational efficient. Since the robot always has ground contact and the considered disaster response environments have mostly vertical walls, the chosen 2D representation suffices to include all required features. If the ceiling contour needs to be incorporated, a second height map describing the ceiling

height could be extracted from the input point clouds and a corresponding cost term could be added to the base cost computation.

Finally, this environment representation allows for modular incorporation of additional information sources. This could include further processing of the sensor measurements, such as planar region segmentation which is, e.g., presented by Karkowski and Bennewitz (2016). Moreover, measurements from additional sensor modalities could be included. In the CENTAURO project², this pipeline has been, e.g., extended to incorporate terrain classification information which is generated from laser scanner point clouds and RGB camera images. Multiple features are extracted and processed to a traversability classification of the terrain (Schilling et al., 2017). This traversability information has been added as another weighted cost term to C_F , as depicted in Figure 3.5. One can think about many other possibilities to enrich this representation with information. Examples are the incorporation of existing maps, experienced costs from previous runs, or measurements from additional sensors providing, e.g., temperature or radiation information.

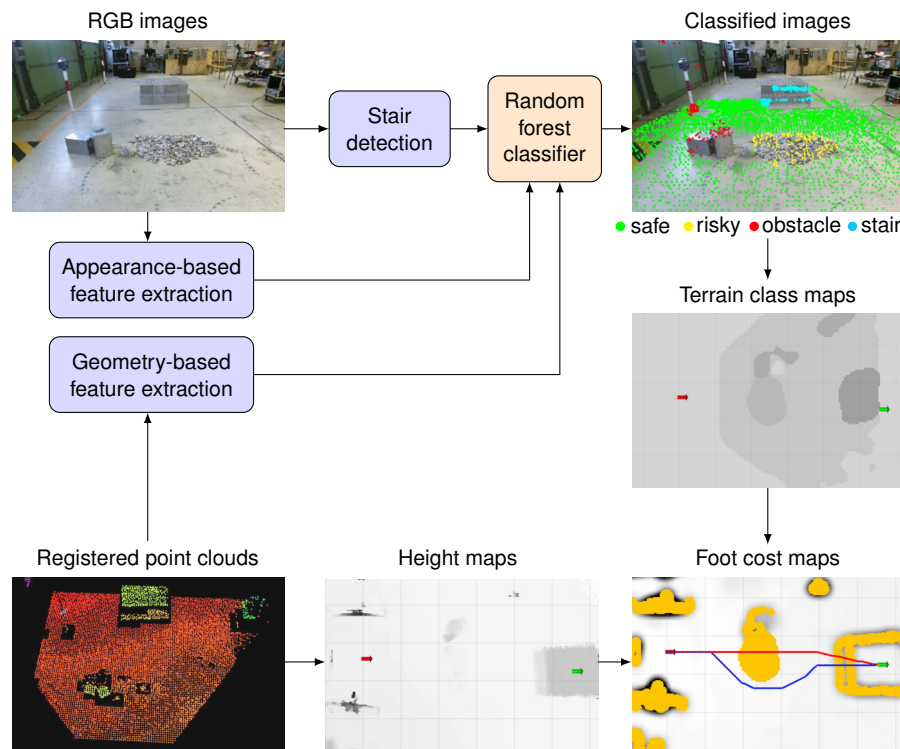


Figure 3.5: Enriching height map-based foot cost map generation with terrain class information. Multiple features are extracted from RGB images and registered point clouds, and are processed to traversability classification of the terrain. Those are included into the original pipeline during foot cost map computation. From Klamt et al. (2019a).

²<https://centauro-project.eu>

3.3 PATH PLANNER

The Path Planner generates a path from the current robot state to a desired goal state which is defined by the operator. It incorporates an environment representation which is provided by the Map Generator, a robot representation, an action set to connect individual robot states, and a cost function which assigns costs to each action and, thus, makes path alternatives assessable and comparable. In the context of hybrid driving-stepping locomotion planning, the planner must consider both locomotion types.

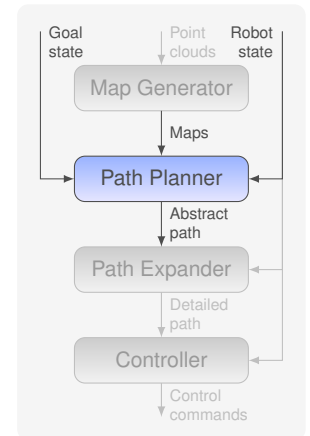
While the generation of the environment representation is described in Section 3.2, an adequate robot representation is described in Section 3.3.1. In Section 3.3.2, a suitable planning method is derived from the state-of-the-art and its implementation is described in detail. Finally, an action set which takes the planner characteristics into account is proposed in Section 3.3.3.

3.3.1 Robot Representation

The Path Planner considers robot platforms which allow for driving locomotion as well as stepping locomotion. Since pure driving or pure stepping locomotion planning is a subset of the described planning problem, the planner is also capable of generating respective plans. However, the focus of this work is on hybrid driving-stepping locomotion.

We consider quadruped robot platforms whose articulated legs end in actively steerable, powered wheels such that omnidirectional driving is possible. Furthermore, wheels as end-effectors allow for immediate switching between the two locomotion types and even enable unique motions such as moving individual feet relative to the robot base while keeping ground contact and being under load. In contrast, robots like RoboSimian (see Section 2.1) require posture changes to switch between the two locomotion types which inhibits such motions and allows for handling driving and stepping locomotion planning isolated. However, driving locomotion in disaster response environments is only feasible with a certain wheel diameter enabling driving over small gravel and ground irregularities. On the other hand, when wheels represent end-effectors, large wheel diameters result in large footholds limiting stepping locomotion in challenging environments. This trade-off has to be faced for the robot hardware design.

The proposed method addresses the two mobile manipulation robots Momaro and Centauro, as depicted in Figure 3.1. Both robots possess a centaur-like body plan with four legs. However, the robots differ in their kinematic details. Momaro's leg design incorporates three pitch joints—allowing for leg motions in the sagittal plane—



followed by the ankle yaw joint controlling the wheel orientation. The legs show compliant behavior due to their elastic carbon composite links, which work as a passive suspension system on rough terrain. Each foot ends in a pair of steerable wheels with a wheel diameter of 0.163 m and an overall foot width of 0.179 m. Hence, the foot orientation can be modified through both the ankle yaw joint and differential drive of the two wheels. Centauro’s leg design incorporates an additional hip yaw joint resulting in a spider-like configuration (yaw-pitch-pitch) of the upper joints, followed by the ankle pitch and ankle yaw joints (Kashiri et al., 2016). This additional hip yaw joint allows for 3-dimensional leg motions. Centauro possesses a single, torus-shaped wheel with a diameter of 0.156 m at the end of each leg. Given the constraint that the wheel steering axes are always perpendicular to the ground to allow for omnidirectional driving in any configuration, 18 DoF are required to sufficiently describe an arbitrary robot locomotion state for these two platforms. This includes the 6 DoF of the robot base ($b_x, b_y, b_z, b_{\text{roll}}, b_{\text{pitch}}, b_{\text{yaw}}$) and 3 DoF for the position of each foot relative to the base ($f_{j,x,\text{rel}}, f_{j,y,\text{rel}}, f_{j,z,\text{rel}}$). The two kinematic leg structures can be compared in Figure 3.6.

Both kinematic structures allow for inward (see Figure 3.7 left) as well as outward (see Figure 3.7 right) knee configurations. While a desired initial knee configuration can be chosen, switching between knee configuration during operations is challenging and should be avoided. This would either require the robot to completely stand up until legs are fully extended or to completely relieve load from the respective foot, lift it and switch to the other knee configuration in a folding maneuver. In the case of Momaro, the kinematic design provides a considerably larger reachability of all legs for an inward knee configuration. In the case of Centauro, both knee configurations provide a large reachability. However, for the front legs, an outward knee configuration bears the risk of collisions with the environment in front of the robot, e.g., when climbing stairs or approaching workspaces for manipulation. Hence, an inward knee configuration is chosen. While Momaro’s design allows for an inward

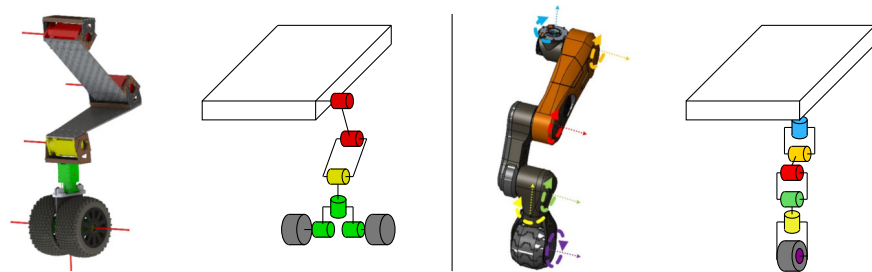


Figure 3.6: Kinematic leg structures. Joint axes are marked with colored lines. For clarity, only one leg is depicted per robot. Proportions are not to scale. *Left*: Momaro. *Right*: Centauro. CAD visualizations from (Schwarz et al., 2017) and (Klamt et al., 2019b).

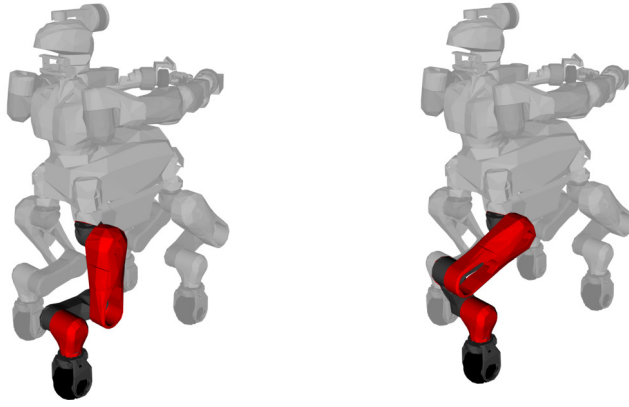


Figure 3.7: Different possible knee configurations for the front left leg of Centauro. *Left*: Inward knee configuration. *Right*: Outward knee configuration.

configuration for both, the front and rear knees without collisions between them, this is not the case for Centauro. Thus, to avoid collisions between knees and limitations to foot reachability, an outward knee configuration is chosen for Centauro's rear legs.

Given the chosen constraints, the foot reachability can be analyzed. Figure 3.8 shows foot reachability maps for both robots.

The reachability maps in the x - y -plane underline the additional capabilities Centauro obtains through its hip yaw joint in comparison to Momaro. While the lateral foot position is fixed for Momaro, Centauro possesses a large lateral reachability. However, this reachability is not equally distributed over the overall reachable area. It rather consists of two circle sectors which are connected under the hip yaw joint through a narrow section. The foot reachability maps further show that Momaro's reachability in the x - z -plane is larger due to longer legs. Due to a symmetrical design, right feet have a symmetric reachability. For Momaro, the rear feet reachability is slightly larger due to longer legs while Centauro uses identical leg kinematics for front and rear feet.

We choose a robot representation with fixed lateral foot positions such that only foot motions in the sagittal plane are considered because of the following reasons:

1. both robot platforms can be described with the same robot representation,
2. in case of Centauro, a dependency between the longitudinal foot position and feasible lateral foot positions is avoided, and
3. in case of Centauro, the planning problem is reduced by four DoF which significantly improves planning efficiency.

For Momaro, this fixed relative lateral foot position is defined to be the only kinematically feasible value of -0.25 m (0.25 m for the right side, respectively). For Centauro this value is defined to be -0.225 m

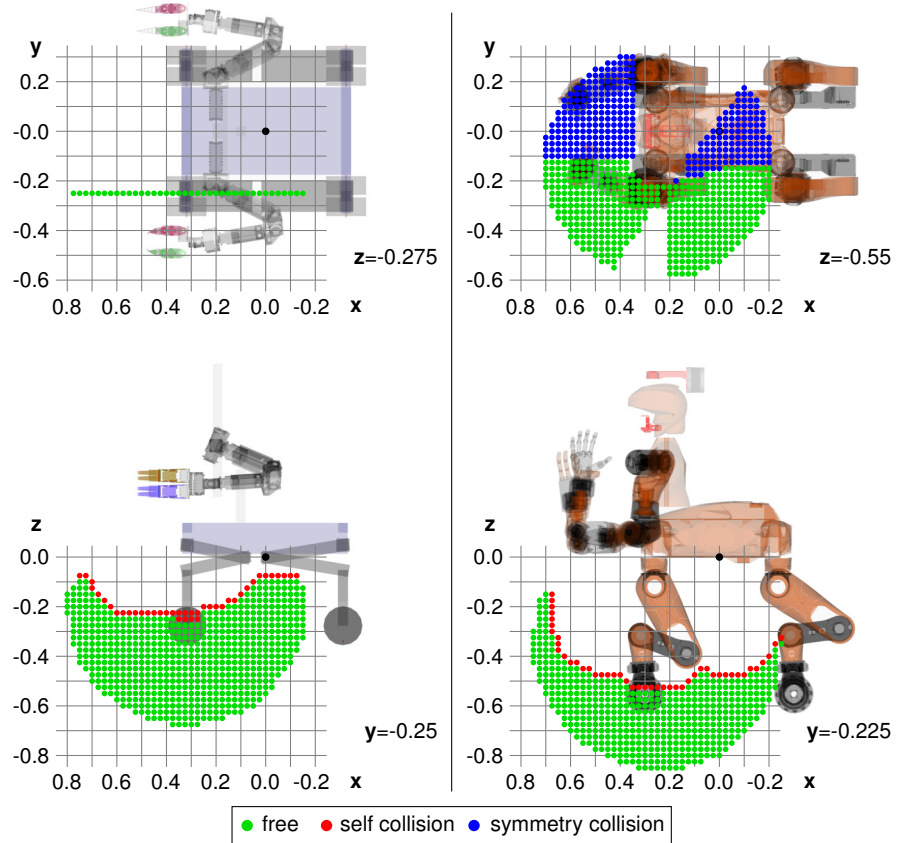


Figure 3.8: Foot reachability maps for Momaro (*left*) and Centauro (*right*). Front legs have an inward knee configuration. Symmetry collisions describe configurations which would result in self collisions if both front legs have a symmetric configuration.

(0.225 m, respectively) which is the only lateral value that connects the two circle sectors in the x - y -plane and provides maximum lateral reachability. Moreover, we neglect the z -dimension during planning, which includes foot and base heights as well as base roll and pitch information, because of the following reasons:

1. except for stepping motions, feet are required to have ground contact and foot heights are dependent on the occurring terrain heights,
2. the base height, roll and pitch orientation are mostly dependent on individual foot heights and cannot be chosen freely, and
3. the planning problem is reduced by another seven DoF.

All of these seven DoF which are neglected during planning are computed for the resulting path during path expansion from occurring terrain heights. This considerably increases planning performance since these DoF are only considered for few robot states instead of being included in every state which is expanded during planning.

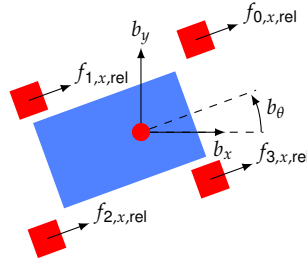


Figure 3.9: Robot representation for locomotion planning. The blue rectangle represents the robot base, red squares represent individual feet, and the red circle represents the robot CoM.

The resulting robot state representation which is employed in the planner $\vec{r} = (b_x, b_y, b_\theta, f_{0,x,rel}, f_{1,x,rel}, f_{2,x,rel}, f_{3,x,rel})$ includes seven DoF to describe an arbitrary robot state and is visualized in Figure 3.9.

The representation does not describe robot dynamics. Momaro's flexible leg design bears the risk of unpredictable dynamic behaviors. Hence, motions are chosen to be sufficiently slow, such that dynamic effects can be neglected for this first approach. However, a later extension to include dynamics in the representation would be desirable to increase operations speed and real-world applicability.

3.3.2 Planning Method

The planner generates a sequence of robot states to move the robot from its current state to a desired goal state, which we call a path. Input to the planner are environment maps from the Map Generator (see Section 3.2), the desired goal state which is defined by the operator, and information about the current robot state which is used to generate the start state for the planning problem. In this section, planning methods are compared, and one particular method is chosen for the problem of hybrid driving-stepping locomotion planning.

In a first step, the application is analyzed in detail to derive requirements to the employed method. The whole planning pipeline shall address the operation of hybrid driving-stepping robots in challenging environments, e.g., in the disaster response domain. First, the operator defines a locomotion goal state which might be in a considerable distance to the current robot position. To get there, the robot might need to omnidirectionally drive on flat and slightly rough terrain, climb obstacles and stairs, take detours to avoid obstacles and challenging regions, or move to other rooms. The planner shall be capable of comparing different path alternatives and prefer those which require less execution effort. Hence, the planner should, e.g., prefer a detour over flat terrain to a shorter path alternative on challenging terrain. Due to the characteristics of disaster response missions, environments are only traversed once or at most a few times. Plan-

ning times are desired to be in the range of a few seconds. Longer planning times would imply waiting times during robot missions resulting in longer mission times and restricting mission goals since on-board power supply is limited. Short planning times increase the possibility of frequent re-planning which allows the robot to quickly react to dynamic environment changes. Since a stable and fast data link between the robot and the operator station cannot be assumed, the planner needs to be executed on-board the robot hardware.

Given these requirements, the existing robot motion planning approaches described in Section 2.2 are assessed in terms of their applicability to the described planning problem.

Optimization- and Potential Field-based Planning

The applicability of optimization- and potential field-based planners to the described planning problem is limited. While these methods are suitable to plan in high-dimensional state spaces, those spaces usually include dynamic DoF such as velocity or acceleration. However, potential fields are vulnerable to local minima which may definitely occur in challenging cluttered environments. Optimization-based methods, if used as a stand-alone planner, are dependent to an initial feasible solution which can be optimized. If such an initial solution is already challenging to find, the strengths of these methods cannot be fully exploited (see Section 2.2.3). Kamedula et al. (2018) and de Viragh et al. (2019) used optimization-based methods to control short sequences of hybrid driving-stepping locomotion. However, those methods were only demonstrated on obstacle-free, flat terrain.

Learning-based Planning

Applying learning-based methods to the described planning problem is also difficult. At the current state-of-the-art, learning-based methods are suitable to solve potentially high-dimensional local sub-problems of a planning task. However, the described problem with its large, complex environments and high-dimensional robot representation results in a state space whose size is significantly larger than what has been solved with such planners so far and goes beyond the current limitations regarding network complexity, training time, and required amounts of training data. One example is the reinforcement learning approach by Chen et al. (2018) which generates short-distance omnidirectional driving commands for the Centauro robot on flat terrain. The method considers obstacles, and besides driving, the robot is capable of changing its base height and footprint width. Nevertheless, since we are convinced that this family of planning methods has a high potential of evolving with further research on network architecture and continuous improvements in corresponding hardware, a stand-alone learning-based planner is developed as

part of this thesis and detailed in Chapter 6 to investigate the current limitations of such approaches.

Sampling-based Planning

Regarding sampling-based planners, it has to be distinguished between single-query algorithms such as RRT and multi-query algorithms such as PRM. Since in the considered domains, most environments only have to be traversed once or few times as elaborated before, the strength of multi-query algorithms to costly generate a roadmap of the whole environment once which can be quickly accessed in multiple queries cannot be exploited. This roadmap generation is rather inefficient since many states are included which are not required to solve the current planning problem. This is different for sampling-based single-query algorithms. It is a general paradigm in the planning community that such algorithms can efficiently solve high-dimensional planning problems.

An example is the work by Reid et al. (2016b) who developed a FMT*-based hierarchical planner for the quadruped omnidirectional-driving MAMMOTH robot with actuated limbs which is also capable of climbing over obstacles. The planner provides paths for queries in challenging environments of up to 15 m length in less than 20 seconds with a success rate of 80 to 90%. Environments represent flat grounds with multiple smaller isolated obstacles as well as walls with gaps in them. Complex stepping maneuvers such as stair climbing are not considered. State transition costs are based on the estimated mechanical work which is required for the transition. This is calculated from state differences and masses of the robot base and legs. Stepping motions are not directly included in the planning method but are derived from terrain heights under the individual feet.

However, the hybrid driving-stepping locomotion planning problem addressed in this work imposes multiple challenges to the application of sampling-based single-query planners as explained in the following.

1. A generally known weakness of sampling-based planners is planning through narrow passages. There exist approaches which focus the sampling strategy on such areas (e.g., Zhang and Manocha (2008)) but those require analysis of the environment and come along with considerable computational effort. In the addressed environments, stair climbing is a frequently occurring example for narrow passages in the 7-dimensional state space. Since feet have to be positioned on individual steps precisely and leg length limitations have to be considered, only a few action sequences with low variance—forming a high-dimensional narrow passage—are feasible to solve such a task and maintain robot stability. Reid et al. (2016b) only consider the overcoming of small isolated obstacles such that this effect is limited in their planning domain.

2. Sampling-based planners require methods to determine the costs between two arbitrary states. Those methods are frequently used to, e.g., determine nearest neighbor states and, consequently, need to be computationally efficient. In many cases, costs are assumed to be proportional or equal with path lengths such that the Euclidean distance can be used as an efficient method. Reid et al. (2016b) formulate a cost function describing mechanical work. As stated above, this is only dependent on the robot states and not on the terrain that needs to be traversed. In our planning domain, it should be, e.g., considerably more expensive to climb a stair instead of driving up a ramp since the latter is faster, safer, and requires less energy. It should also be cheaper to drive over flat terrain instead of gravel. Both examples illustrate that a pure comparison of two states is not sufficient to reliably estimate transition costs between them, but the respective environment has to be analyzed. However, this requires considerable computational effort and reduces the efficiency of the planner.
3. Sampling-based planners interpolate between states and assume that a direct connection is possible. An example is the computation of action costs where multiple states are sampled between the action start and action goal state to determine the individual costs for these sampled states and generate accumulated action costs. Another example is the generation of a new state which shall be added to the state tree of RRT. First, a random state is sampled. If the distance to its nearest neighbor in the tree exceeds a specified maximum distance, an intermediate state is interpolated between the nearest neighbor and the sampled state at this specified distance to the nearest neighbor. This is feasible for omnidirectional motions but imposes challenges to, e.g., stepping motions where the feasibility of intermediate states cannot be guaranteed. Positive and negative examples are depicted in Figure 3.10.

Search-based Planning

In contrast to sampling-based planners, many of the described disadvantages do not occur for search-based planners. The main advantages of search-based planners, in the context of hybrid driving-stepping locomotion planning, are described in the following.

1. The definition of an explicit set of feasible actions is supported by search-based planners. In contrast to sampling-based planners, where a direct connection between different states is assumed, this is helpful to address the kinematic capabilities and limitations of the considered platforms. It is, e.g., no problem to consider that foot motions are only feasible in the longitudinal direction and that steps need to be added to the paths as explicit maneuvers.

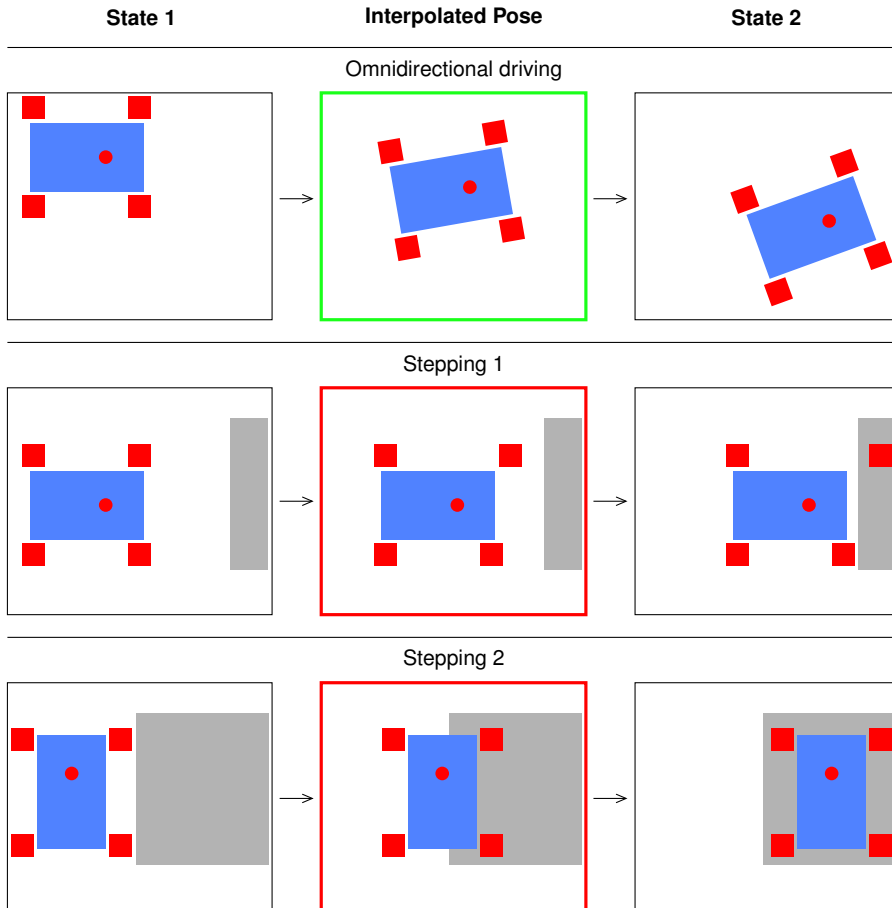


Figure 3.10: State interpolation as used in sampling-based planning methods. *Top:* State interpolation is feasible for omnidirectional driving. *Middle:* For stepping, the depicted interpolated state is undesired, since the robot would approach the platform with an already extended leg and the leg length limitation would prohibit further longitudinal extension which is required for the step. *Bottom:* Since the robot representation does not allow for sideways stepping maneuvers, the interpolated state is not on the path between the two states. Instead, the robot would have to turn, longitudinally climb the step and turn again.

This was also exploited in other works (see Section 2.4 for details). Fleckenstein et al. (2017) defined a set of explicit leg motions and omnidirectional driving maneuvers for an ARA*-based planner for the agricultural BoniRob platform. Particular emphasis was placed on multiple driving forward-actions, which is the preferred driving direction. Dornbush et al. (2018) defined different action sets for different locomotion modes for a humanoid platform. Depending on the surrounding environment requirements, only certain action sets were considered for respective regions.

2. The utilization of customized cost functions does not impose challenges. While for sampling-based methods, costs have to be frequently computed between arbitrary states, in search-based planning, costs are only computed once for each expanded action.

However, search-based planners do also introduce some challenges when applied to high-dimensional planning problems.

1. Planning problems which require detailed planning (e.g., to precisely navigate around obstacles or precisely step on them) need a fine planning representation which comes along with large state spaces to be searched. The utilization of multiresolution approaches is promising to diminish this effect (see Section 2.3).
2. In situations in which the path to the goal state requires actions carrying high costs, the planner would first search large areas around the already explored regions before considering such actions. Using sophisticated methods, such as ARA* or the employment of an informed heuristic, can minimize these extensive searches but the design of such a heuristic is challenging. As an example, Fleckenstein et al. (2017) proposed a combination of a free-space heuristic—assessing the robot motion capabilities—and a wheel Dijkstra heuristic—incorporating environment information while only considering for individual wheels.

While both, sampling- and search-based planners, have weaknesses and strengths in the context of hybrid driving-stepping locomotion planning, we choose a search-based approach for the problem of hybrid driving-stepping locomotion planning since many of the general domain characteristics can be easily addressed in search-based methods while they would require costly adaptations or extension for sampling-based methods. In particular, we employ the Anytime Repairing A* (ARA*) algorithm since the considered problem yields a large state space increasing the risk for the planner to get stuck in extensive searches. By initially searching with large heuristic weights, sub-optimal paths can be found quickly and are subsequently refined. This is desirable since it is preferred to obtain a sub-optimal solution within a given time interval—which is called the anytime property—instead of waiting for an optimal solution.

3.3.3 *Action Set and Cost Function*

The search-based planner employs an action set which is based on the robot representation described in Section 3.3.1 and which contains state transitions representing the kinematic capabilities of the considered robots. Actions need to respect the state discretization introduced by the planning method.

Driving Actions

Omnidirectional driving actions include driving with fixed orientation to one of 20 neighbor states, as depicted in Figure 3.11 (a). Eight of these states are in the direct state grid vicinity (Moore neighborhood) of the robot. Eight additional states can be reached by moving two longitudinal resolution steps and one lateral resolution step or vice versa to mitigate the angular discretization effects. However, employing these 16 actions in combination with the ARA* planner results in an undesired effect: When planning with high heuristic weights, the planner prefers those actions which bring the robot closer to the goal state. Since the maximum traversable distance in a single action is achieved through those eight actions which do not end in the direct vicinity of the action start state, those are preferred by the planner. However, as depicted in Figure 3.11 (c), this can result in zig-zag-shaped paths. Adding four additional actions to the action set, which move the robot by either two longitudinal or two lateral resolution steps (see Figure 3.11 a), prevents this undesired behavior. In addition to translational movements, omnidirectional driving actions include turning to the next discrete orientation with fixed position, as depicted in Figure 3.11 (b).

The omnidirectional driving action costs are computed from the given state costs along the edge. For driving with fixed orientation, intermediate robot states are sampled along each action edge such that the distance between two successive sampled states does not exceed half of the action resolution. This fine sampling enables precise collision checking in challenging environments. Robot states are sampled using interpolation. As explained in Section 3.3.2 and depicted in Figure 3.10, sampling through interpolation is feasible for continuous actions such as omnidirectional driving. Subsequently, state costs are computed for each sampled state. Action costs are computed by averaging over those state costs and multiplying with the action length. Since state costs are defined to be 1 in flat terrain, driving action costs

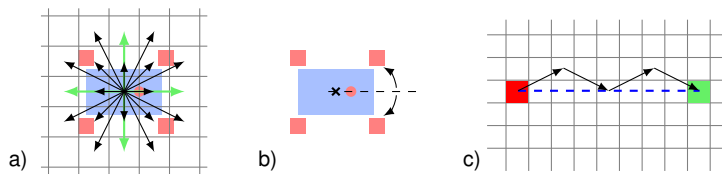


Figure 3.11: Omnidirectional driving action set. *a)* Driving to a neighbor state with fixed orientation. *b)* Turning to the next discrete orientation with fixed position. *c)* ARA* outputs for large heuristic weights results in a zig-zag-shape. To obtain the desired behavior (dashed blue line), four additional actions are added to the action set (green arrows in *a)*). Grid and orientation resolution are enlarged for illustrative purposes.

are equal to the covered distance in flat terrain and increase for actions in more challenging terrain.

For turning to the next discrete orientation, an intermediate state is sampled too, such that the maximum distance between two states is halve of the rotation resolution. This allows for precise collision checking in challenging environments. Subsequently, state costs are computed and averaged, and multiplied with the covered distance. In this case, the distance is computed from the average distance each foot covers during the turning motion. To facilitate faster computation, we assume that all feet are in standard configuration, as depicted in Figure 3.11 (a) and (b).

Stepping Actions

Besides omnidirectional driving, the planner considers stepping motions. However, since those are only feasible in certain situations, a set of criteria is checked before generating a stepping motion for a foot at the position \vec{f}_j . This criteria set is visualized in Figure 3.12.

- \vec{f}_j is close to an obstacle, i.e. ,
 $\exists c \in \text{map}$ with $C_F(c) = \infty \wedge \|\vec{c} - \vec{f}_j\| \leq d_{\text{obst,max}}$,
 with foot costs C_F and the maximum obstacle distance $d_{\text{obst,max}}$.
 Checking of this condition is realized computationally efficient through precomputed inflated foot cost maps.
- A feasible foothold in the map cell c_h with $C_F(c_h) < \infty$ can be found in front of the foot in its sagittal plane that respects a maximum leg length.
- The height difference to the foothold Δh_{step} is feasible, i.e. ,
 $\Delta h_{\text{step}} = |h(\vec{f}_j) - h(c_h)| \leq \Delta h_{\text{step,max}}$.
- The distance between the two feet on the “non-stepping” robot side is $\geq \Delta f_{x,\text{min,ns}}$ to facilitate a safe stand while stepping.

$d_{\text{obst,max}}$ is chosen to be 0.1 m, $\Delta h_{\text{step,max}}$ is 0.3 m and $\Delta f_{x,\text{min,ns}}$ is 0.5 m for Momaro and 0.3 m for Centauro. While the two robots seem similar in their kinematic capabilities, Momaro’s foot reachability is larger resulting in different thresholds.

It is important to understand that the step which is added to the search is an abstract step. We define an abstract step to be the direct transition from a pre-step state to a post-step state. Neither a guaranteed robot stability nor the detailed motion sequence to perform the step are considered during the search. This is obtained during path expansion after the search.

Each step with the target foothold in the map cell c_h is assigned the corresponding costs

$$C_{\text{step}} = k_{C_{\text{step},1}} \cdot l_{\text{step}} + k_{C_{\text{step},2}} \cdot \Delta h_{\text{step}} + k_{C_{\text{step},3}} \cdot (C_F(c_h) - 1), \quad (3.7)$$

Abstract Step

An abstract step is the direct transition from a pre-step to a post-step state. Neither the robot stability nor the detailed motion sequence to perform the step are considered.

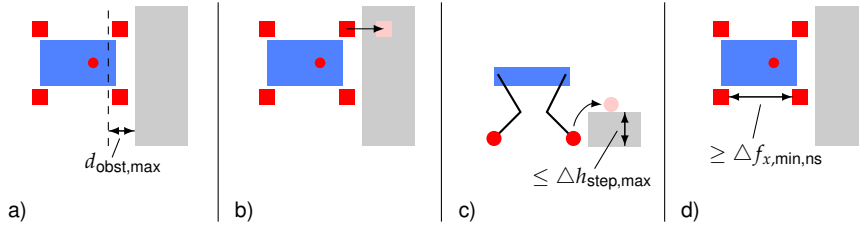


Figure 3.12: Step criteria to be considered by the planner. Figures visualize criteria for a step with the front left foot. *a)* The foot is close to an obstacle. *b)* A feasible foothold exists in front of the foot. *c)* The height difference to overcome is feasible. *d)* The distance between the two feet on the non-stepping robot side exceeds a threshold to facilitate sufficient stability.

where $k_{C_{\text{step},1}} = 0.5$, $k_{C_{\text{step},2}} = 2.3$, and $k_{C_{\text{step},3}} = 0.1$ including the step length l_{step} , the stepping height difference Δh_{step} , and a terrain difficulty assessment for the target foothold. Since foot costs are defined to be 1 on perfectly flat terrain and increase for more difficult terrain, a terrain difficulty assessment is obtained by subtracting 1 from the foot costs in c_h . If multiple feasible target footholds for a step exist, only the cheapest solution is added to the search.

The locally cheapest step is not necessarily part of the globally cheapest path. However, the consideration of multiple feasible steps during the search leads to significantly larger state spaces—especially if multiple successive steps are required. We are aware of this trade-off between computational efficiency and result optimality and opt for computational efficiency since several sources for sub-optimality are present anyway and efficiency has a high priority.

Besides the step itself, further maneuvers are required to perform stepping locomotion and to navigate in cluttered terrain (see Figure 3.13). For this, we define a neutral foot configuration describing the configuration which is depicted in the info box. It provides a low CoM, a compact but stable footprint which is desirable to, e.g., move through doors, and is the preferred foot configuration for driving.

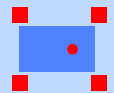
If both front feet are positioned in front of their neutral position, the robot may perform a longitudinal base shift. The base is shifted forward relative to the feet while the feet keep their position relative to the ground. The shifting length is limited by either rear legs which reach their maximum length or front feet which reach their neutral configuration (see Figure 3.13 b). Longitudinal base shifts of length l_{bs} are assigned the costs

$$C_{\text{bs}} = k_{C_{\text{bs}}} \cdot l_{\text{bs}} \cdot C_{\text{B,avg}}, \quad (3.8)$$

where $k_{\text{bs}} = 0.5$ and $C_{\text{B,avg}}$ describes the average discovered base costs during the shift.

Neutral Foot Configuration

The neutral foot configuration describes the depicted configuration. It provides a low CoM, a compact but stable footprint, and is the preferred configuration for driving.



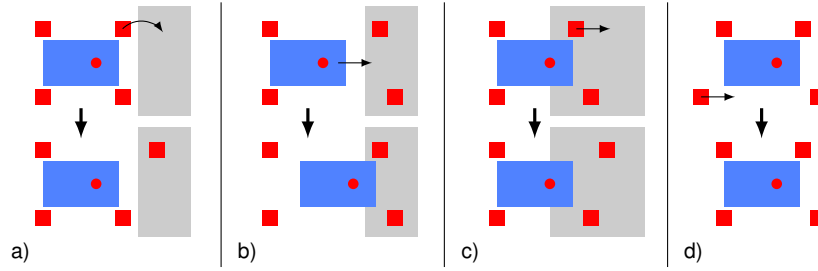


Figure 3.13: Stepping related planning maneuvers. *a)* Step. *b)* Longitudinal base shift. *c)* Driving an individual front foot forward. *d)* Driving any foot back towards its neutral configuration.

If a rear foot is close to an obstacle, the front feet may be driven forward (see Figure 3.13 c). This enables a subsequent base shift maneuver and is the preparation for a rear foot step.

Finally, whenever an individual foot is not in its neutral configuration, the robot is given the possibility to drive this foot towards this configuration (see Figure 3.13 d). If an obstacle hinders the complete execution until the neutral foot configuration is obtained, the foot is only moved towards the neutral configuration until reaching the obstacle. Driving a foot relative to the robot base with a motion length of l_{df} is assigned the costs

$$C_{df} = k_{C_{df}} \cdot l_{df} \cdot C_{F,avg}, \quad (3.9)$$

where $k_{C_{df}} = 0.125$ and $C_{F,avg}$ describes the average foot costs discovered during the motion.

Additional Cost Terms

Since driving is faster, safer, and more energy efficient compared to stepping, we want the planner to prefer driving locomotion whenever feasible and consider drivable detours of an acceptable length instead of including steps in the resulting plan. This can be addressed by a corresponding cost parametrization. We define that if the robot stands in front of an elevated platform of 0.2 m height, it should just prefer an 1.5 m long detour over a ramp instead of stepping up to this platform, as visualized in Figure 3.14. All cost terms of stepping-related maneuvers are weighted by an additional factor such that the desired behavior is obtained.

Although the considered robots are capable of omnidirectional driving, there are multiple reasons to prefer driving forward.

1. The sensor setup is designed to provide the best measurement quality for the area in front of the robot since it is also used for manipulation. This results in the most reliable and detailed environment representation in front of the robot. The rotating laser

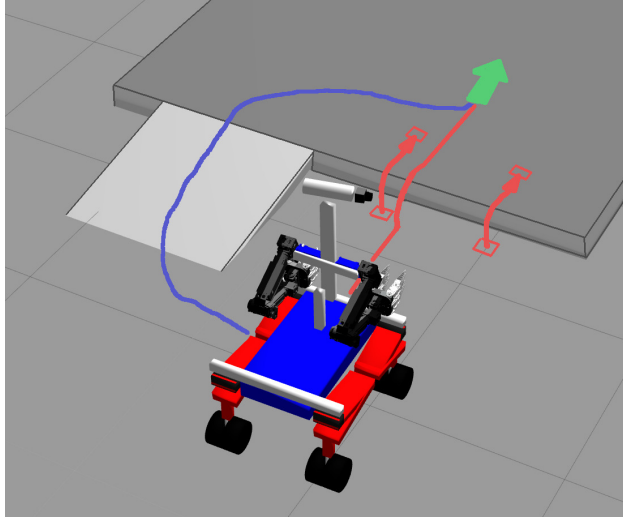


Figure 3.14: Stepping-related maneuver costs are weighted such that the planner just prefers taking a 1.5 m detour over a ramp (blue path) instead of choosing stepping (red path) to achieve a goal state (green arrow) on an elevated platform.

scanner is, e.g., mounted such that it can measure the terrain directly in front of the feet. This is not the case for the robot sides and back.

2. The required width clearance is minimal when driving in the longitudinal direction which facilitates navigation in narrow sections such as doors.
3. The kinematic leg design and the chosen robot representation restrict stepping motions to the sagittal plane. Thus, approaching corresponding situations in the right orientation without the necessity for turning maneuvers is desirable.

We express this desire of preferring certain orientations by adding an orientation cost factor $k_{\Delta\theta}$ to the costs for driving actions. $k_{\Delta\theta}$ is dependent on the orientation difference between the robot orientation and the base motion orientation and follows the scheme depicted in Figure 3.15. The factor is minimal for driving forward ($k_{\Delta\theta} = 0$ plus a margin of one orientation step) and linearly increases to a maximum value. When driving sideways, $k_{\Delta\theta}$ is assigned this maximum value. Driving backward ($k_{\Delta\theta} = \pi$) is assigned a lower value, since the required width clearance is also minimal when driving backwards. The parametrization of $k_{\Delta\theta}$ is evaluated in Section 3.6.1.

Finally, as stated above, the planner shall prefer driving in the neutral configuration. Path search is motivated to find respective paths in the state space by weighing action costs for driving in a non-neutral configuration by the factor $k_{\text{non-neutral,fp}} = 1.1$, which was parametrized through experiments.

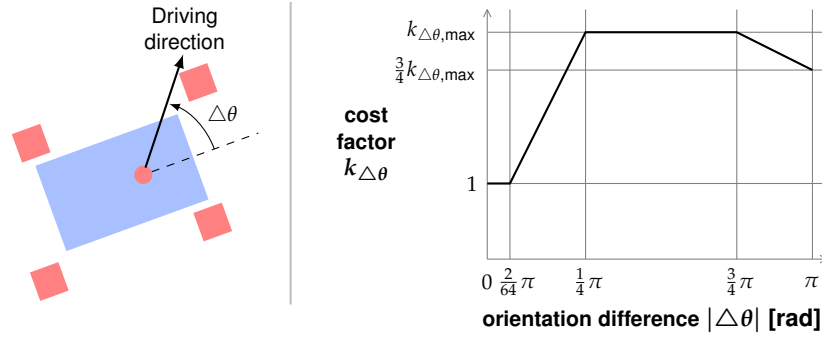


Figure 3.15: Orientation difference cost factor for driving actions. For an orientation difference $\Delta\theta$ between the robot orientation and the driving direction, a corresponding action cost factor is determined following the depicted scheme.

3.3.4 Heuristic

The performance of A*-based planners strongly depends on the employed heuristic. It is used to estimate costs from an arbitrary state to the goal state. The heuristic should steer the search towards the goal while needing as few node expansions as possible but expanding as many nodes as necessary to find a solution of the desired quality. To obtain solutions which are optimal with respect to the chosen state space discretization, the heuristic always has to underestimate the costs for the cheapest path between the two states. This is called an admissible heuristic. Furthermore, good cost estimates, which are as close to the cheapest path costs as possible, minimize the number of state expansions. Since heuristic values are frequently required during path search, its computation needs to be efficient.

We combine the often-used Euclidean distance with a heuristic term for the orientation difference and call this the *geometric heuristic*. The value between the arbitrary robot states \vec{r}_1 and \vec{r}_2 is computed as follows:

$$h(\vec{r}_1, \vec{r}_2) = k_{h,1} \cdot \left\| \begin{pmatrix} b_{x,1} \\ b_{y,1} \end{pmatrix} - \begin{pmatrix} b_{x,2} \\ b_{y,2} \end{pmatrix} \right\| + k_{h,2} \cdot \mathcal{O}d_{f,\text{neut.}} \cdot \Delta b_\theta, \quad (3.10)$$

where $\mathcal{O}d_{f,\text{neut.}}$ describes the average distance from the robot base center to a foot in neutral configuration and Δb_θ describes the minimum orientation difference between \vec{r}_1 and \vec{r}_2 . Hence, the product describes the minimum required distance feet have to move to turn the robot on the spot for the required rotational difference. $k_{h,1}$ and $k_{h,2}$ are weighting factors. As stated in Section 3.2, the Euclidean distance is weighted with $k_{h,1} = 1$ since, to underestimate costs, it assumes driving in neutral configuration on perfectly flat terrain. The orientation difference cost term is weighted with $k_{h,2} = 0.5$ since it describes a turning motion with fixed base position such that only the driving costs of the feet but not of the base are considered.

Since we employ the ARA* algorithm, the whole heuristic term is weighted with a factor $\mathcal{W} \geq 1.0$ such that result optimality cannot be immediately guaranteed but bounded sub-optimal results are generated quickly.

3.3.5 Implementation

The described planner is implemented using C++ and ROS Kinetic Kame. We employ a fine action resolution of 2.5 cm to enable the planner to position the robot precisely in challenging environments. In addition, we choose an orientation resolution of 5.625° which results in 64 discrete orientations.

Efficient implementations for the priority queue and the data structure storing node information are crucial as both are frequently used by the planner. These are described in the remainder of this section.

Priority Queue

The priority queue is used by the planner to handle all discovered nodes and to determine the order of node expansions. In this planning context, nodes represent robot states. The ARA* planner uses the f -value (see Section 2.2.1) to compare individual nodes with each other and compute their priority. The lower the f -value, the higher the states priority. The following priority queue operations are required:

push: Pushes a node with a given f -value into the queue with the correct position with respect to the f -value. During planning, each newly discovered feasible node is pushed into the queue.

top: Returns the node with the highest priority. This is used to determine the next node to expand.

pop: Deletes the node with the highest priority from the queue. After a node is expanded, it is deleted from the priority queue using this function.

increase: Increases the priority of an already included node according to a new, lower f -value. During path expansion, if for an already known node an f -value is computed which is lower than the previously known f -value, its position in the priority queue is updated..

The complexity of these operations is compared in Table 3.1 for a set of widely-used, heap-based implementations.

A Fibonacci heap-based priority queue is chosen due to the lowest overall complexity. We employ the boost open-source implementation³.

³www.boost.org/doc/libs/1_54_0/doc/html/heap/data_structures.html

Table 3.1: Complexity comparison of priority queue implementations (Corman et al., 1990).
[†] amortized

Data structure	push	top	pop	increase
Binary heap	$O(\log(N))$	$O(1)$	$O(\log(N))$	$O(\log(N))$
Binominal heap	$O(1)^\dagger$	$O(\log(N))$	$O(\log(N))$	$O(\log(N))$
Fibonacci heap	$O(1)$	$O(1)$	$O(\log(N))$	$O(1)^\dagger$

Node Data Storage

During the search, the following data has to be stored for each discovered node:

- g -cost,
- f -cost,
- pointer to predecessor node,
- a flag if the node is in the open list of the search,
- a flag if the node is in the closed list of the search, and
- a handle to the nodes position in the priority queue which is required by the *increase* function of the priority queue (see Section 3.3.5).

Since node data is accessed and updated frequently during the search, an efficient data structure is required.

For constant data access time, a multidimensional array with unique indices would be required for every possible 7-DoF robot state $\vec{r} = (b_x, b_y, b_\theta, f_{0,x,rel}, f_{1,x,rel}, f_{2,x,rel}, f_{3,x,rel})$. This would allow direct access to the corresponding data without the need for any search within the data structure. However, this would require memory preallocation for each index. A 10×10 m map, with the chosen resolution and foot reachability, contains approximately $6.7 \cdot 10^{11}$ robot states. Assuming that not the whole data but only a pointer is preallocated for each state and assuming a pointer size of 64 bit, this would result in approximately 4.8 TB of required memory, which is obviously far above the capabilities of current hardware on-board mobile robots.

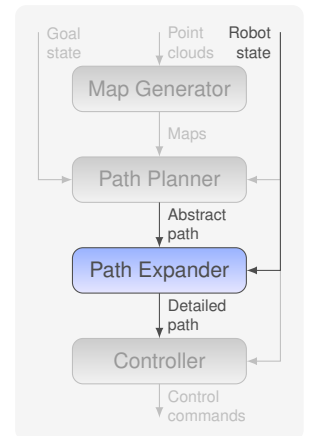
We chose a method in which a unique index is assigned to each robot base state (b_x, b_y, b_θ) and allocate memory for a pointer for each index, resulting in approximately 80 MB of required memory for the above stated example. Each pointer points to a vector storing data for all discovered foot configurations for this base state. Data of newly discovered states is added to the vector with the corresponding index. When accessing node data, the corresponding vector needs to be searched. Since mostly only few foot configurations are discovered for individual base states, this search can be performed in short times.

Depending on the available memory size, arbitrary foot configurations can be included in the index increasing the size of preallocated memory but decreasing the time to access node data. We include the configuration of the front left foot $f_{0,x,rel}$ in the index computation. For the above stated example, this results in approximately 1.25 GB of required memory for preallocation.

However, we experienced that the sizes of the resulting vectors containing pointers are unbalanced. While these vectors often only contain a single entry for regions of sufficiently flat terrain, which can be overcome by driving, vector are considerably longer for regions of complex terrains, such as staircases, where numerous foot configurations are considered. One could think about mapping the data to an abstract state space which can be searched more efficiently, as e.g., proposed by Klein (2005).

3.4 PATH EXPANDER

The output of the Path Planner is an abstract path which is a sequence of robot states to navigate the robot from its current state to the goal state. However, if steps are included in this path, they are represented as abstract steps assuming a direct transition between a pre-step to a post-step state and lacking robot stability consideration. The Path Expander expands abstract steps to detailed motion sequences (see Section 3.4.1). Moreover, the abstract path lacks information about vertical foot positions, which we refer to as foot heights. The Path Expander derives foot heights from the required motions and the terrain characteristics (see Section 3.4.2). Outputs of the Path Expander are detailed motion sequences which can be executed by the Controller.



3.4.1 Step Expansion

Abstract steps are expanded to detailed motion sequences which guarantee robot stability at all times. In the case of Momaro, its compliant leg design hinders knowledge about the exact foot positions. Those have to be estimated from joint states. Since actuator speeds are rather slow, dynamic effects are neglected. Hence, we limit stability consideration to static stability, meaning that the robot can stop its motion in every configuration while staying stable. Stability assessment is done through the support polygon which is spanned between the horizontal positions of all feet with ground contact (see Figure 3.16). During stepping, this polygon changes its shape from a quadrangle to a triangle. If the horizontal robot CoM projection is inside the support polygon, the robot state is statically stable. The closer the CoM is to the support triangle centroid (STC), the higher is the static robot stability.

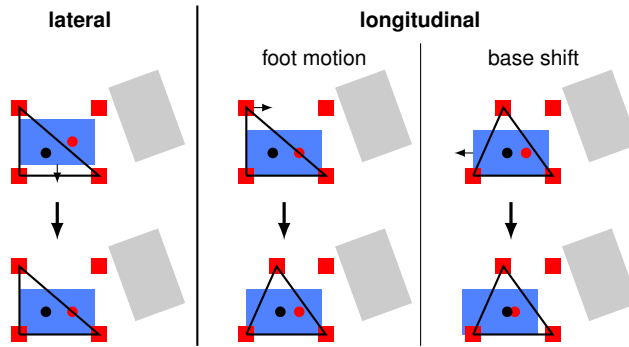


Figure 3.16: Obtaining a statically stable stepping configuration for a step with the front left foot. The support triangle spanned by the feet which maintain ground contact and its STC are depicted in black. The robot CoM is depicted as a red dot. Lateral alignment is realized through base roll motions. Longitudinal alignment is either done by driving the remaining foot on the stepping side towards the robot center or through a longitudinal base shift.

Since lateral foot movement is not supported by Momaro’s kinematics and the lateral foot movement range of Centauro strongly depends on the longitudinal foot position (see Figure 3.8), those motions are not employed during path expansion. Instead, we use base roll motions for lateral alignment of the CoM and STC, as depicted in Figure 3.16. Those rotations around the longitudinal robot axis can be realized by changing the leg lengths on one side of the robot. In the case of Momaro, the resulting angle between the wheel axes and the ground plane is compensated by the compliant legs and the soft-foam filled wheels. In the case of Centauro, its torus-shaped wheels facilitate base roll motions. Roll motion parametrization is described in the next section.

Longitudinal alignment of the CoM and STC is realized by driving the remaining foot on the stepping side (e.g., the rear left foot when stepping with the front left foot) towards the robot center. If this does not suffice—considering foot reachability limitations and obstacles hindering the motion—the remaining longitudinal alignment is realized through longitudinal base shift maneuvers. Both maneuvers are depicted in Figure 3.16. In the case of Centauro, due to a smaller foot reachability, additional arm motions are introduced facilitating balancing. Arms are moved to the robot back when stepping with the front feet while they are moved to a front configuration for rear foot steps. The longitudinal CoM position is also dependent on the robot base pitch angle described in the next section. The developed motions generate a stable robot configuration to perform the planned step. If no stable configuration can be obtained, path expansion is aborted. However, due to sufficient flexibility, this behavior was not observed in our experiments. After stepping, base shift, foot displacement, and base roll maneuvers are reverted to retrieve the initial configuration.

3.4.2 Vertical Foot Positions

Abstract paths as output by the Path Planner do only contain information about the longitudinal foot position for each robot state. While, as explained in Section 3.3.1, lateral foot positions are defined to be constant, vertical foot positions (foot heights), are required for a path to be executable by the Controller. Foot height computation needs to realize base roll motions, to provide adaptation to the ground, to avoid collisions between the robot base and the underlying terrain, and to avoid self-collisions as well as collisions between the feet and the terrain during stepping.

When driving with neutral footprint, a low foot height is established providing a low CoM and, hence, a high stability. The terrain heights under the robot base are checked to determine when the robot needs to lift its base to overcome obstacles between its legs. For maneuvers other than driving a higher base height is chosen providing sufficient clearance for individual foot motions. The terrain height under each foot is looked up in the terrain height map to determine the highest foot which is determining for the chosen base height. For each stepping-related motion it is checked for the motion goal state, if the current foot height is sufficient. It is also checked, if none of the legs exceeds its maximum leg length. If adaptations are required, an intermediate motion is inserted lifting the robot to a sufficient height before executing this motion or lowering the robot base after executing the motion. Steps itself are expanded to a foot-lift motion, are foot-extend motion, and a foot-lowering motion considering the corresponding terrain height and a small safety margin.

On terrain with sufficient slope, the robot base is pitched to an angle which provides a good trade-off between ground clearance for leg motions and a good CoM position. Since the CoM is close to the base center for both robots, pitching has only small influence on the CoM position and the corresponding robot stability. Due to the different kinematics, the base pitch angle for Momaro is chosen to be 70% of the ground slope while Centauro's base is pitched to 100% of the ground slope.

Base roll motions are realized by changing the foot heights on one robot side resulting in a lateral CoM motion. Given a desired lateral CoM position, the required foot height change is computed as follows: We assume the center of rotation $\mathcal{R}(y'_{\text{rot}}, z'_{\text{rot}})$ to be at a fixed position. In case of Momaro, this position is chosen to be between the two wheels of a foot, as depicted in Figure 3.17. Due to the soft-foam filled wheels, this assumption is sufficiently close to the real robot behavior. For Centauro, the center of rotation is determined to be at the bottom center point of the wheel while slight lateral shifts on the torus shape

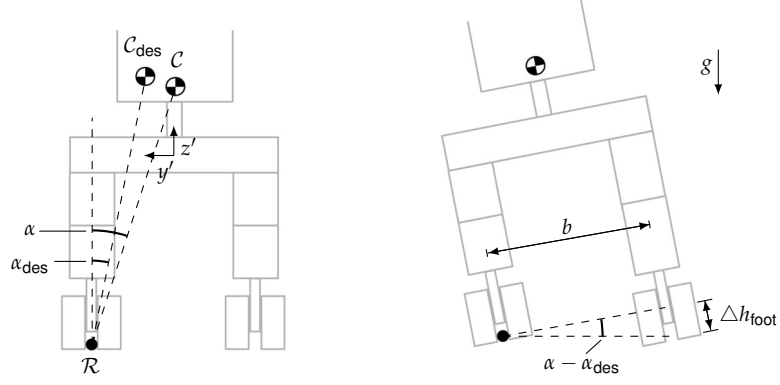


Figure 3.17: Base roll computation. Momaro’s lower body with footprint width b in back view. Lateral CoM shifts are achieved through foot height changes Δh_{foot} on one side of the robot. \mathcal{R} describes the assumed center of rotation. \mathcal{C} and \mathcal{C}_{des} describe the current and the desired CoM position. α and α_{des} describe the according angles to the vertical plane.

are neglected. In addition, the CoM position $\mathcal{C}(y'_{\text{CoM}}, z'_{\text{CoM}})$, the angle

$$\alpha = \arctan\left(\frac{y'_{\text{rot}} - y'_{\text{CoM}}}{z'_{\text{CoM}} - z'_{\text{rot}}}\right) \quad (3.11)$$

between $\overrightarrow{\mathcal{R}\mathcal{C}}$ and the vertical axis, and the desired lateral CoM position $y'_{\text{CoM,des}}$ are given.

Using $\|\overrightarrow{\mathcal{R}\mathcal{C}}\| = \|\overrightarrow{\mathcal{R}\mathcal{C}_{\text{des}}}\|$, we compute the desired angle between $\overrightarrow{\mathcal{R}\mathcal{C}_{\text{des}}}$ and the vertical axis

$$\alpha_{\text{des}} = \arcsin\left(\frac{y'_{\text{rot}} - y'_{\text{CoM,des}}}{\|\overrightarrow{\mathcal{R}\mathcal{C}_{\text{des}}}\|}\right). \quad (3.12)$$

With a given footprint width b , we compute the desired foot height difference

$$\Delta h_{\text{foot}} = b \cdot \tan(\alpha - \alpha_{\text{des}}). \quad (3.13)$$

This height difference is added to both legs on the corresponding robot side inducing the desired base roll motion.

3.5 CONTROLLER

We developed a hybrid driving-stepping controller. Input are detailed paths represented as sequences of robot states. Using the current robot position and orientation, and its foot configuration, the next robot state \vec{r}_{i+1} on the path is searched. Running at a frequency of 20 Hz, the controller determines whether omnidirectional driving or limb motions are required to obtain this next state.

If omnidirectional driving is required, we employ the following method to smooth the robot behavior: A B-Spline (Prautzsch et al., 2013) is computed through the robot base states $(b_{x,i}, b_{y,i}, b_{\theta,i})$, $(b_{x,i+1}, b_{y,i+1}, b_{\theta,i+1}), \dots, (b_{x,i+n}, b_{y,i+n}, b_{\theta,i+n})$. Depending on how many of the next states require omnidirectional driving, up to five base states are included in the B-Spline. We define a controller goal state in a given distance in front of the current robot base state on this B-Spline. Given a desired driving velocity and the base state difference between the controller goal state and the current robot state, we compute a velocity command $\vec{w} = (v_{x'}, v_{y'}, \omega)$ with horizontal linear velocities $v_{x'}$ and $v_{y'}$ in robot coordinates and a rotational velocity ω around the vertical robot axis. We set the desired driving velocity to 0.2 m/s. If the robot is closer to the B-Spline goal than 0.2 m, the desired driving velocity decreases to 0.05 m/s. This enables precise approaching of desired states for subsequent stepping maneuvers or of the overall goal. The velocity command is handled by the low-level driving controller described in Schwarz et al. (2017). Considering the current foot configuration, it computes rotational velocities and orientations for each individual foot. It further considers waiting times to steer individual wheels and obeys joint specific velocity and acceleration limits.

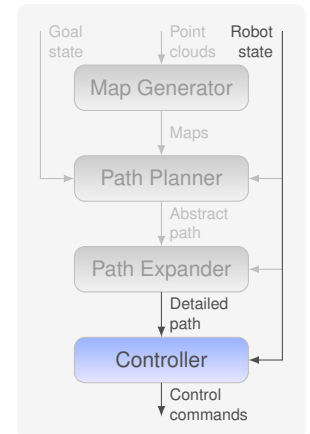
If leg motions are required to obtain the next robot state in the path, those are directly executed. Foot motions are sent to the motion player which is also described in Schwarz et al. (2017). It solves the inverse kinematic and outputs individual joint velocity commands.

3.6 EVALUATION

The developed method was evaluated in several experiments describing the parametrization of the orientation cost factor (Section 3.6.1), the investigation of the heuristic weight influence (Section 3.6.2), and a demonstration of the method's capabilities and limitations (Section 3.6.3). While these experiments were performed in the Gazebo simulation environment (Koenig and Howard, 2004), additional experiments demonstrated the application to the real Momaro and Centauro platforms (Section 3.6.4). A video explaining the method and including additional footage of the experiments is available online⁴.

3.6.1 Robot Orientation Cost Factor

In a first experiment, we evaluated the parametrization of the robot orientation cost factor, which weights driving maneuver costs depending on the occurring angle between robot orientation and driving direction (Section 3.3.3). In particular, the influence of the max-



⁴<https://doi.org/10.5281/zenodo.3628732>

imum value $k_{\Delta\theta,\max}$ was evaluated. We employed a simulated environment in which Momaro stood in a corridor in front of an elevated platform with some cluttered obstacles, as depicted in Figure 3.18. The planner was to find a path to a goal state on top of this platform. We compared the planner performance for different values for $k_{\Delta\theta,\max}$.

Experiments were performed on one core of a 2.6GHz Intel i7-6700HQ processor using 16GB of RAM. Instead of ARA*, we employed a standard A* planner with heuristic weight $\mathcal{W} = 1$ generating optimal results. Hence, as explained in Section 3.3.3, using a set of 16 instead of 20 driving actions was sufficient. The results are given in Figure 3.19. To keep path costs comparable, we explicitly do not include the orientation cost weighting factor in the stated output costs of the planner. Instead, stated path costs are those costs the resulting path would induce without the orientation cost factor. Resulting paths for different $k_{\Delta\theta,\max}$ are visualized in Figure 3.20 (top) and (middle).

It can be seen that an increasing value for $k_{\Delta\theta,\max}$ results in a decreasing difference between the robot orientation and driving direction, which is the expected behavior. It can be further seen that values of $k_{\Delta\theta,\max} > 2$ have no considerable influence on the orientation difference compared to $k_{\Delta\theta,\max} = 2$. Regarding path costs, higher values for $k_{\Delta\theta,\max}$ lead to slightly increased path costs. Comparing $k_{\Delta\theta,\max} = 1$ with $k_{\Delta\theta,\max} = 5$, path costs increase by $\sim 9\%$. A possible explanation for this behavior is that a higher value for $k_{\Delta\theta,\max}$ introduced a higher priority for some states over others to the path search. This led to fewer expanded nodes while some of the discarded nodes

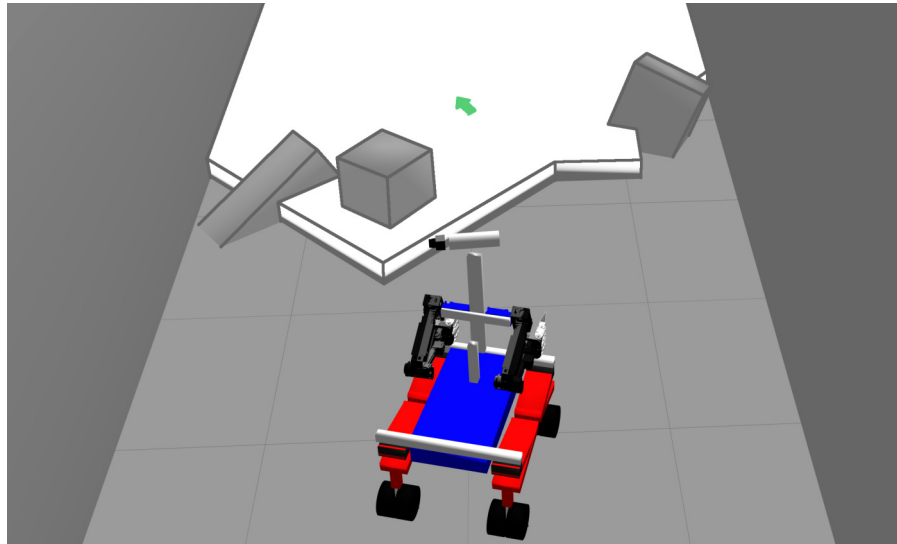


Figure 3.18: Gazebo scenario for planner evaluation. Momaro stands in front of an elevated platform cluttered with obstacles. The goal state is on top of this platform (green arrow).

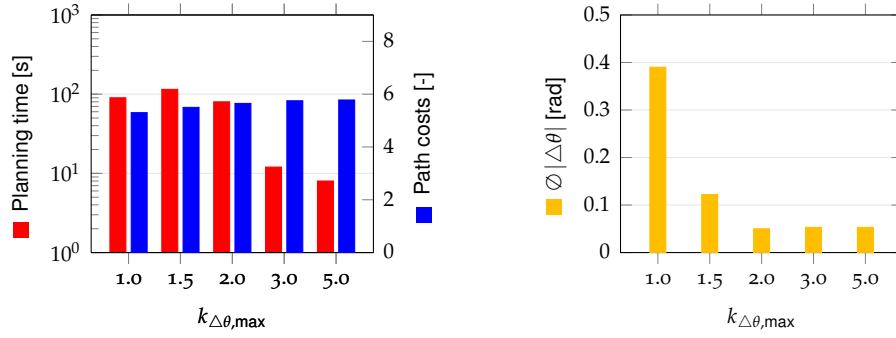


Figure 3.19: Planner performance for different maximum orientation cost factors $k_{\Delta\theta, \max}$. To enable comparability, stated path costs are those costs the resulting path would induce without weighing through the orientation cost factor. The stated orientation difference $|\Delta\theta|$ is averaged over all states in the resulting path.

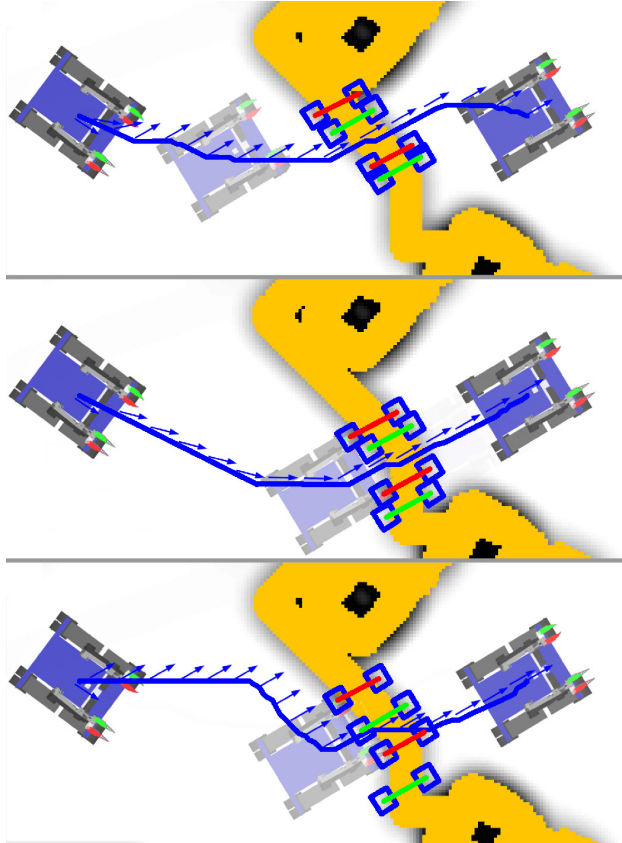


Figure 3.20: Resulting paths on foot cost maps. Yellow areas are not traversable by driving. Blue paths show the robot center position, arrows show the orientation. Blue rectangles show used footholds. Red lines represent front foot steps, green lines represent rear foot steps. *Top*: Result of the plain A* algorithm ($\mathcal{W} = 1$) without orientation cost factor. *Middle*: Optimal solution ($\mathcal{W} = 1$) for $k_{\Delta\theta, \max} = 2$. *Bottom*: First sub-optimal result of the ARA* algorithm with $\mathcal{W} = 3$ and $k_{\Delta\theta, \max} = 2$.

might have been part of the optimal solution. The same effect can be observed when comparing planning times. An increasing value for $k_{\Delta\theta,\max}$ resulted in considerably shorter planning times. This is especially the case for $k_{\Delta\theta,\max} > 2$ for which planning was roughly one order of magnitude faster compared to $k_{\Delta\theta,\max} = 1.5$. Interestingly, when comparing $k_{\Delta\theta,\max} = 1$ with $k_{\Delta\theta,\max} = 1.5$, planning took longer for a higher orientation cost factor. This might be explained by the fact that the higher $k_{\Delta\theta,\max}$ value resulted in wrong search priorities in some cases leading to more extensive searches in this region.

3.6.2 Heuristic Weight Comparison

In a second experiment, we evaluated the ARA* performance. Again, the simulated planning task from the previous experiment was employed and the same computational hardware was used. We chose exponentially decaying heuristic weights, starting at $\mathcal{W} = 3$. For this experiment, $k_{\Delta\theta,\max}$ was set to 2. The planner performance is shown in Figure 3.21. Please note that the stated planning times are accumulated due to the nature of the algorithm: It quickly generates an initial, bounded sub-optimal path with a high heuristic weight. Sub-

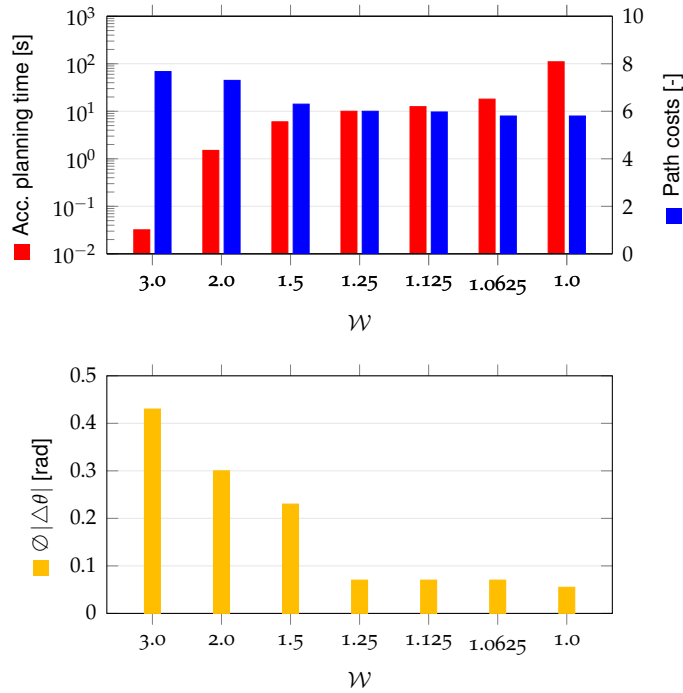


Figure 3.21: ARA* performance for the employed heuristic weights \mathcal{W} with $k_{\Delta\theta,\max} = 2.0$. Planning times are accumulated. To enable comparability, stated path costs are those costs the resulting path would induce without the orientation cost factor. The stated orientation difference $|\Delta\theta|$ is averaged over all states in the resulting path.

sequently, the heuristic weight iteratively decreases, and the planner generates results of better quality while utilizing previously generated solutions. Hence, the stated accumulated planning time for, e.g., $\mathcal{W} = 1.5$ includes consecutive planning runs for $\mathcal{W} = 3$, $\mathcal{W} = 2$, and $\mathcal{W} = 1.5$.

ARA* provided the first result with $\mathcal{W} = 3.0$, which is 31% more expensive than the optimal solution with $\mathcal{W} = 1.0$, in 32 ms. A result with only 2% higher costs was generated in ~ 10 s for $\mathcal{W} = 1.25$. This is sufficiently fast compared to the required time for path execution. Waiting for the planner to provide the optimal solution ($\mathcal{W} = 1.0$) took > 100 s, which is undesirable long for timely mission execution. In comparison to the previous experiment, finding a solution with $k_{\Delta\theta, \max} = 2$ and $\mathcal{W} = 1$ took longer. This is due to the fact that ARA* first generates sub-optimal solutions. In addition, in this experiment a set of 20 driving actions was employed instead of only 16 actions as used in the previous experiment (see explanation in Section 3.3.3). Moreover, it can be seen that the effect of the robot orientation cost factor increases with decreasing heuristic weights. A possible explanation for this behavior is that both, the orientation cost factor as well as the higher heuristic weight, introduce a preference for state expansion to the planner. While for large heuristic weights, their preference dominates, the influence of the orientation cost factor emerges for smaller heuristic weights. A comparison of two paths with different heuristic weights can be seen in Figure 3.20 (middle) and (bottom).

3.6.3 Cluttered Staircase Scenario

A second simulated scenario was generated to demonstrate the capabilities of the developed method. Momaro stood in front of a staircase with five steps which was partly blocked by obstacles, as shown in Figure 3.22. The robot was to reach a goal state on top of this staircase. Tracked vehicles would experience difficulties in such situations. Walls were used to limit the area in front of the staircase and, thus, limit the corresponding number of states the planner would potentially visit before considering expensive stair climbing. Scenarios with large free areas in front of the staircase led to unfeasible long planning times. An initial solution with $k_{\Delta\theta, \max} = 2$ and $\mathcal{W} = 3$ was generated in 1.02 s while solutions with lower heuristic weights required unfeasible long planning times.

As depicted in Figure 3.23, Momaro approached the staircase and climbed the first steps. It subsequently drove sideways while taking the obstacle on the staircase between its legs. The robot finally climbed the remaining steps. Figure 3.24 shows, how the base pitch angle adapts to the ground slope while executing this task.

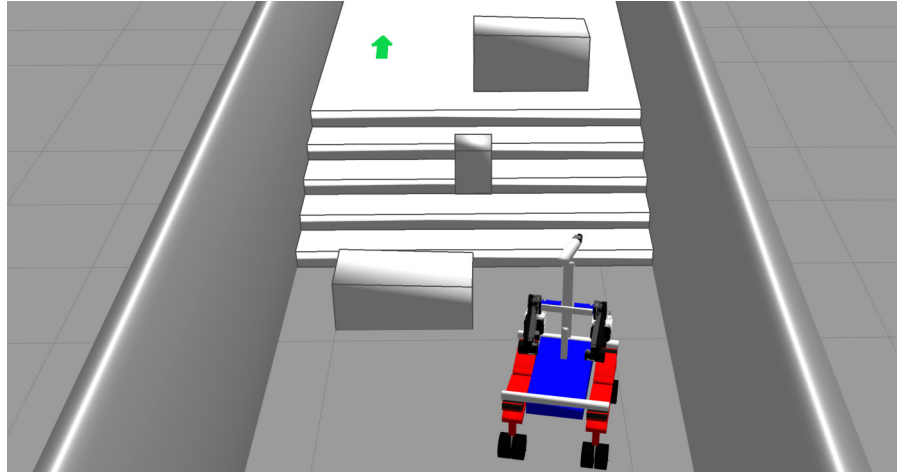


Figure 3.22: Challenging Gazebo scenario to demonstrate the planner capabilities. Momaro has to climb a staircase to reach the goal state (green arrow). Obstacles in front, behind and on the stairs hinder straight climbing but require a combination of stepping and driving maneuvers.

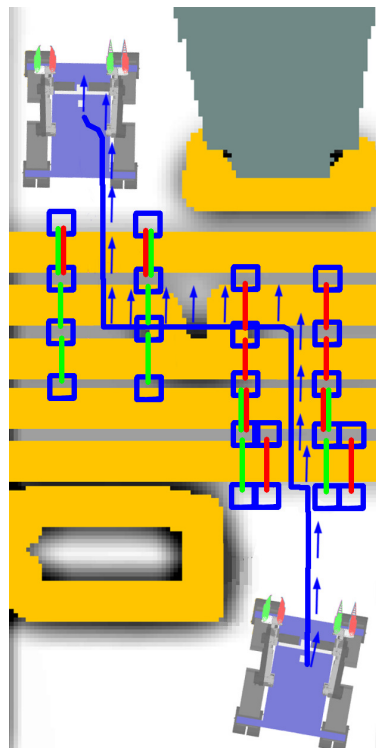


Figure 3.23: Generated path for the staircase scenario on a foot cost map with $\mathcal{W} = 3$ and $k_{\Delta\theta, \max} = 2$. The blue path shows the robot center position. Arrows show the robot orientation. Blue squares show used footholds. Red lines represent front foot steps; green lines represent rear foot steps. The center obstacle is taken between the robot legs when driving sideways.

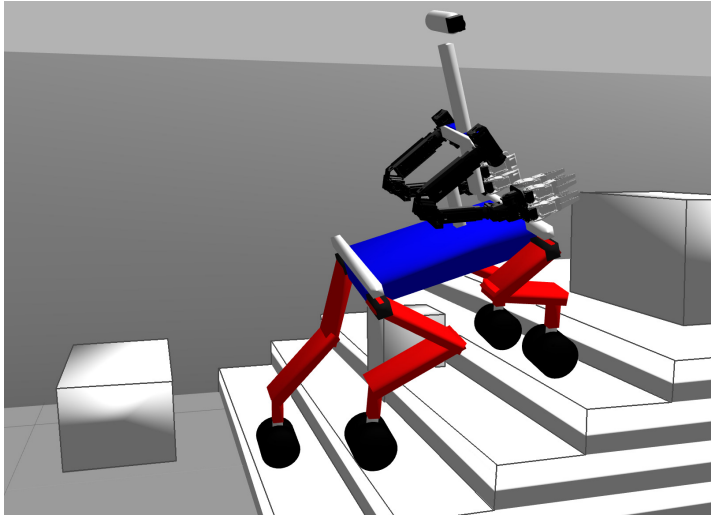


Figure 3.24: Momaro climbing the staircase. The base is rolled to the right to perform a step with the rear left foot. The robot base pitch angle adapts to 70% of the terrain slope.

3.6.4 Real Robot Application

The developed method was applied to the real Momaro and Centauro robots in different experiments.

In a first experiment, Momaro was positioned in some distance to a patch of grass, a patch of gravel, and an elevated platform which was built from concrete blocks and was 13.5 cm high. Figure 3.25 shows the scenario. We employed the environment representation depicted in Figure 3.5, which included additional terrain classification information obtained through the method by Schilling et al. (2017). This made the robot avoid the gravel and grass sections since those were assigned higher costs. Figure 3.26 shows the path execution.

We investigated that the chosen leg design with elastic carbon fiber links induces challenges to the autonomous navigation. Since foot positions were computed from joint angles, the real positions varied considerably from the estimated positions depending on the foot load. This effect was not present in the simulated experiments since links were represented rigidly. Two modifications allowed the method to cope with that behavior. First, a large safety margin was introduced to the stability assessment. Second, detailed step motions were modified by lifting the respective foot higher by a constant value and lowering the foot by this value during the stepping down phase. This avoided feet from getting stuck at stair edges when not lifted high enough.

In another experiment, the method was applied to Centauro. The robot was positioned in front of a patch of debris and some obstacles. A staircase consisting of two steps and ending in an elevated platform was positioned behind the debris. Stairs were 20 cm high and 30 cm long, which is similar to typical staircases in human-made environ-



Figure 3.25: Scenario for the real robot experiment with Momaro including a field of gravel, a patch of grass, and an elevated platform. The shown robot state is its start state.

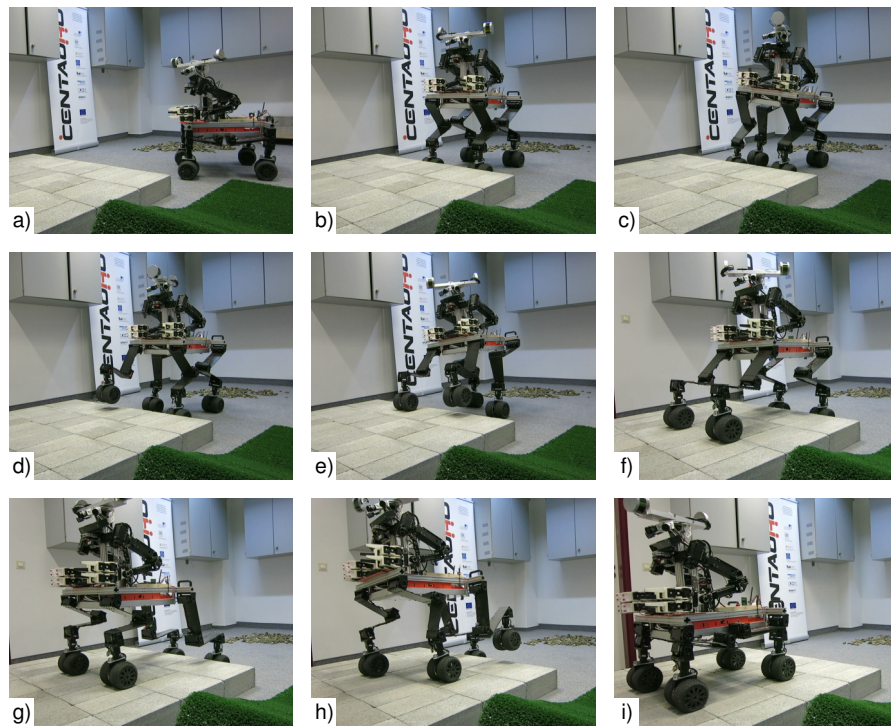


Figure 3.26: Momaro stepping up an elevated platform. *a)* It approaches the platform using omnidirectional driving, *b)* stands up, *c)* rolls its base to the left and drives its back right foot forward to establish a stable configuration, and *d)* performs a front right step. *e)* It then reverts the base roll and foot shift and repeats the procedure for a front left step. *f)* Subsequently, it drives forward *g)* performs a longitudinal base shift, and *h)* performs the two rear steps *i)* to reach the goal state.

ments. The goal state was defined to be on the elevated platform. Again, the above-mentioned environment representation including terrain classification was employed (see Figure 3.5). The developed method was extended to output two paths giving the operator alternatives to choose from. The scenario, the corresponding foot cost map, and the two generated path alternatives are depicted in Figure 3.27. Map generation, robot localization, goal state definition, and planning took around 30s on the internal robot hardware (Intel Core i5-7500T with 2.7GHz, 32 GB RAM). Figure 3.28 shows the execution.

The above-mentioned modifications to cope with the flexible leg design of Momaro were not required in this case since Centauro's leg links were rigid. However, since Centauro's foot reachability is smaller compared to Momaro's, and since its CoM is higher, additional arm motions were introduced to facilitate balancing (see Section 3.4.1). After climbing the two steps with the front feet and when attempting to perform the second step with the rear foot, the robot lost balance for a short moment and required a slight push from a person at location to regain balance. The reason was that the stability assessment used a robot model with a wrong CoM. Recent hardware modifications, such as the integration of the battery and a different position for some PCs and the router, were not included in this model leading to wrong stability assessments. This underlines the importance of a well-organized integration pipeline and just is an example for the many dependencies to other system components.

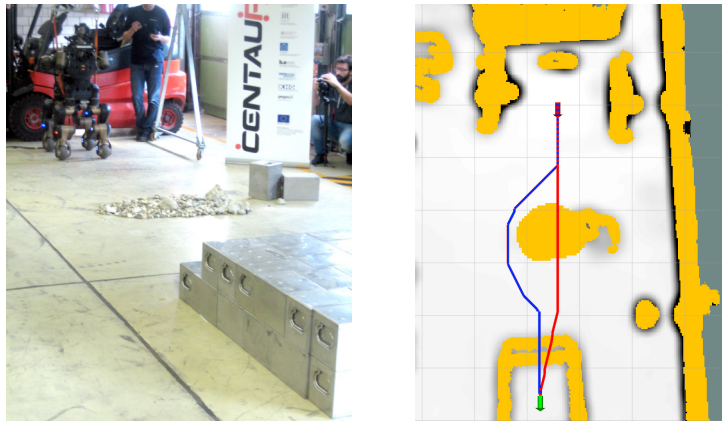


Figure 3.27: *Left*: Scenario for Centauro including a field of gravel, some obstacles and a flight of stairs ending in an elevated platform. The shown robot state is the start state. The goal state is on top of the platform. *Right*: Foot cost map of the scenario. The start state is depicted as a red arrow, the goal state is a green arrow. Two path alternatives are visualized for different terrain classification influences. The red path (low influence) includes driving over the patch of gravel which is feasible but challenging. The blue path (higher influence) contains a detour to avoid the gravel.

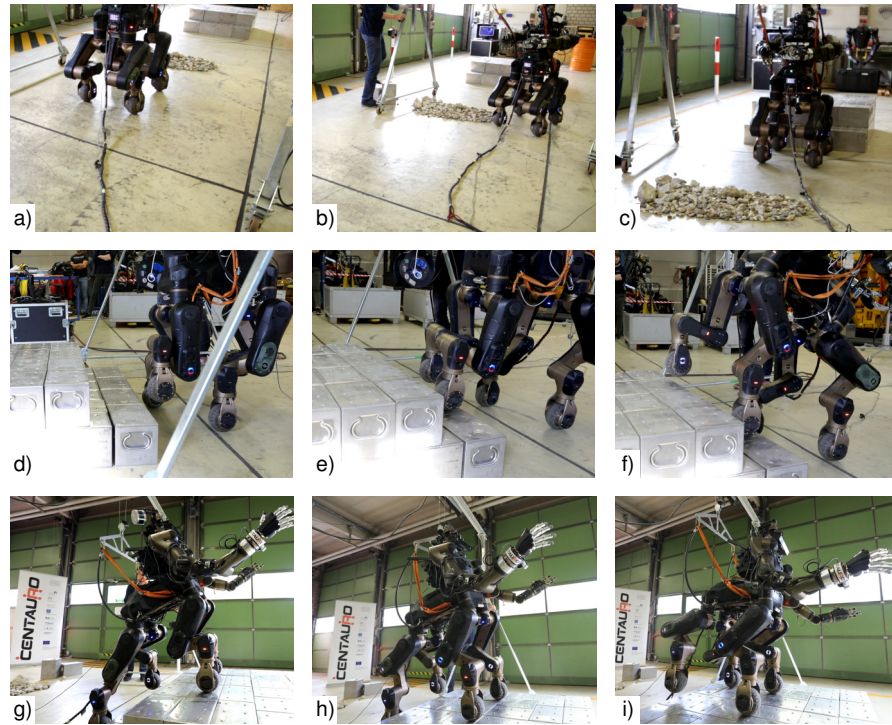


Figure 3.28: Centauro executing the real-world task. *a)* It approaches the field of gravel, *b)* omnidirectionally drives around it, and *c)* positions itself in front of the stairs. *d)* It uses base roll motions, longitudinal base shifts, and footprint changes to establish a stable configuration for a front left step. *e)* The same is done for a front right step and *f)* two more subsequent front foot steps. *g,h)* For rear foot steps, the arms are moved to a front configuration for a better CoM position. The robot alternately performs left and right rear foot steps to *i)* finally reach the goal state.

3.7 CONCLUSION

In this chapter, a search-based navigation planning approach for hybrid driving-stepping locomotion was presented. It describes all robot capabilities in a holistic planning problem including unique locomotion features such driving individual feet relative to the base while keeping ground contact and being under load. A suitable environment representation was developed addressing the challenges the considered environments induce. Costs for the robot base and individual feet are computed and enable precise locomotion planning in challenging scenarios. The 7-DoF robot representation was developed under consideration of the trade-off between representing the high platform flexibility and limiting the state space size to be suitable for efficient planning. It is applicable to multiple robots, such as Momaro and Centauro. Multiple relevant planning algorithms were discussed and the choice for a search-based planning approach was justified. A

corresponding action set, cost function, and heuristic were presented. The planner prefers omnidirectional driving and considers stepping maneuvers in the vicinity of obstacles. Multiple cost function modifications were introduced to adapt the robot behavior.

The approach follows a hierarchical architecture. Vertical foot positions and detailed step motion sequences are not considered during planning and are only generated for those robot states contained in the resulting path. This results in state space sizes which are small enough to be handled by the planner in feasible time. Path expansion strategies to establish stable robot configurations and to compute vertical foot positions were presented. A controller to execute the generated paths was presented as well.

The method was evaluated in simulation and real robot experiments with two different robot platforms. Simulation experiments provided the assessment of the planner capabilities, its performance, and the influence of selected parameters. The planner generated paths with bounded sub-optimality in feasible time and was capable of path planning in challenging environments, for which pure driving or pure walking robots would experience problems. Robot experiments demonstrated the applicability to real world planning problems. The challenges occurring during the transfer from the simulation to the real world and corresponding solutions were presented. The experiments also gave an assessment of the planner's limitations regarding manageable state space sizes and the role of the planner in a complex robotic software architecture.

PLANNING HYBRID DRIVING-STEPPING LOCOMOTION ON MULTIPLE LEVELS OF ABSTRACTION

In Chapter 3, we presented a search-based approach to plan hybrid driving-stepping locomotion paths for the robots Momaro and Centauro, even for challenging terrain such as staircases with additional obstacles on it. The cost function has been designed such that the planner prefers omnidirectional driving whenever possible and considers stepping maneuvers in the vicinity of obstacles. The robot footprint is considered at any time enabling precise locomotion in challenging environments. While path search does not include information about foot heights, robot stability, and detailed stepping motion sequences, resulting paths are expanded to include this information and to be executable by the controller. Furthermore, the influence of a heuristic weight $\mathcal{W} > 1$ has been investigated. It considerably accelerates planning while decreasing the result quality to bounded suboptimality. Experiments show that this method generates high quality paths for small but challenging environments in feasible time.

However, it is only applicable to environments of limited size. The many DoF of the robot representation and the fine planning resolution result in a large state space which grows rapidly for larger scenarios and makes planning expensive. This is even intensified by the nature of the cost function. Since steps are slower, less stable, and less energy efficient compared to driving, the planner is designed to prefer driving detours of a certain length before considering steps. Consequently, in situations where steps are required to reach the goal, the planner still evaluates many possibilities for detours before considering these steps which leads to long planning times. This effect even increases if multiple steps are required, such as it is the case when climbing staircases. Nevertheless, to ensure the applicability in a wide range of realistic tasks, the method has to be capable of handling such scenes.

This effect is not unique for hybrid driving-stepping locomotion but affects high-dimensional planning in many applications such as locomotion planning for robots with tracked flippers or manipulation planning. As described in Section 2.3, an effective solution is the utilization of multiple planning representations. Those can provide precise planning—accompanied by a large state space—in challenging map regions while planning is done in a coarser way in regions where this is sufficient. This results in considerably smaller state spaces and, hence, in considerably faster planning times enabling the planner

to cope with larger maps. Many works achieve these multiple planning representations either through multiresolution planning or by employing robot representations with varying dimensionality. Few works combine both ideas. However, all these approaches only discard information in their coarser representations. This bears the risk of wrong situation assessment resulting in wrong paths if valuable information is neglected. Therefore, instead of only discarding information, it is a promising idea to enrich coarser representation with additional semantics increasing the scene “understanding”—which is called abstraction. While abstraction is widely used in e.g., CNN architectures for perception tasks, it is rarely applied to locomotion planning representations.

We developed multiple planning representations with different level of abstraction for both the environment and the robot representation. This approach is motivated by the manner how humans efficiently plan their locomotion (see Figure 4.1). If a human would be e.g., asked to leave a room, to subsequently traverse a corridor, and to finally climb a staircase, it would only generate a precise plan for the next few actions. To leave the room, the human would only plan its next few steps towards the door. In a medium planning horizon, it would e.g., consider a path around a table blocking the direct way to the door without thinking about the exact footholds to use. Finally, in

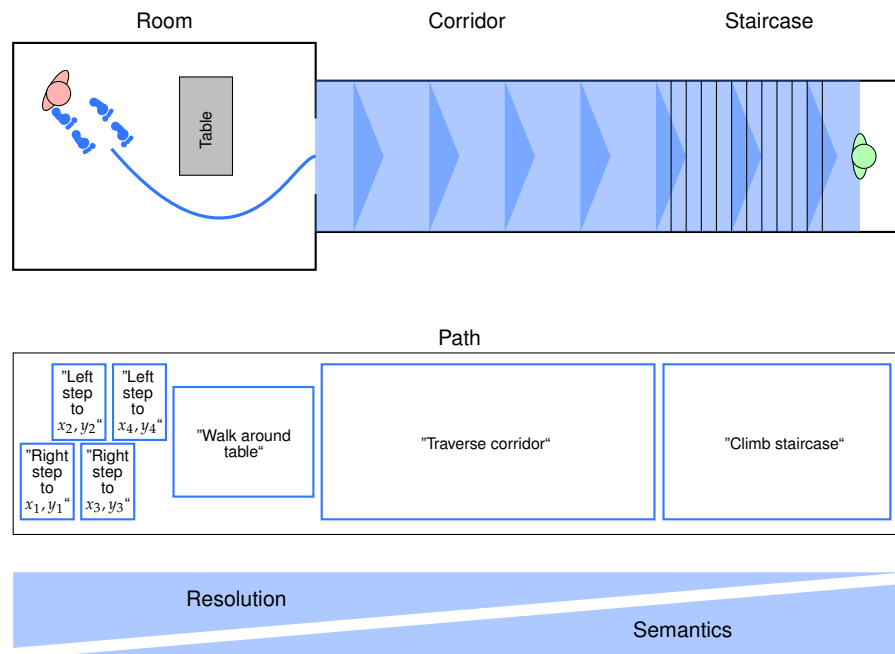




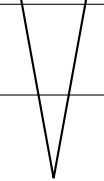

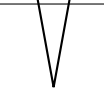
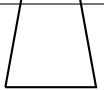
Figure 4.1: Example of a human efficiently planning a path from its current state (red) to a goal state (green). With increasing distance from the current position, the plan becomes coarser but semantics increase.

a large planning horizon, the human would only think about traversing the corridor and climbing the staircase. At this point, it would not plan its paths through the corridor and would not even consider its detailed length. Regarding the environment representation, the human would only notice “*there is a corridor*” and “*there is a staircase*” without considering further spatial details. Regarding the considered actions, its plan would be as coarse as “*traverse the corridor*” or “*climb the staircase*”, while being confident that it is able to perform these actions. When approaching these areas, the human would refine the respective path segments. This approach allows humans to efficiently plan their locomotion over long horizons and for challenging environments. Section 4.1 gives a method overview and explains the structure of the rest of this chapter.

4.1 SYSTEM OVERVIEW

We present a method which plans hybrid driving-stepping locomotion on three different levels of representation, namely *Level 1*, *Level 2*, and *Level 3*. A representation with a high planning resolution and a high number of DoF is employed in the vicinity of the current robot position. This is similar to the representation described in Chapter 3 and is defined to be the *Level 1* representation. With increasing distance from the robot, both the environment and the robot representation become more abstract resulting in the robot-centered, medium-sized *Level 2* representation and the most abstract *Level 3* representation covering the whole map. More abstract path segments are situated further in the future which comes along with a higher degree of uncertainty and less accurate sensor measurements. We compensate this less precise information for higher levels of representation by enriching the representation with additional semantics. Regarding the environment representation, its resolution gets coarser while the number of features describing each map cell increases. Regarding the robot representation, it is represented with less DoF and the action resolution gets coarser. The robot representation adapts to this lower-dimensional description by e.g., describing areas of potential foot positions instead of discrete foot positions. In addition, the action set and cost function follow the abstraction approach considering the coarser robot representation while utilizing the additional semantics of the environment representation. Figure 4.2 visualizes the different representations and Figure 4.3 depicts the corresponding areas of utilization.

All three levels are unified in a single planner. This is superior to coarse-to-fine planning approaches, as utilized in many works in the literature, for the following reason: In the vicinity of the robot, precise sensor measurements are available. In challenging situations, this precise information might be required to find the correct next

Level	Map Resolution	Map Features
1	 <ul style="list-style-type: none"> • 2.5 cm • 64 orientations 	 <ul style="list-style-type: none"> • Height
2	 <ul style="list-style-type: none"> • 5.0 cm • 32 orientations 	 <ul style="list-style-type: none"> • Height • Height Difference
3	 <ul style="list-style-type: none"> • 10 cm • 16 orientations 	 <ul style="list-style-type: none"> • Height • Height Difference • Terrain Class

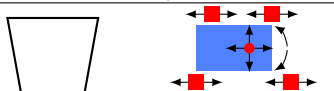

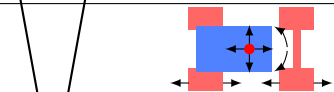
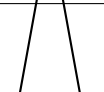
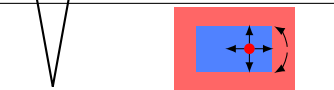
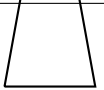
Level	Robot Representation	Action Semantics
1		 <ul style="list-style-type: none"> • Individual Foot Actions
2		 <ul style="list-style-type: none"> • Foot Pair Actions
3		 <ul style="list-style-type: none"> • Whole Robot Actions

Figure 4.2: Representation level overview. *Top*: The environment is represented in three levels or representation with increasing abstraction: while the resolution gets coarser this information loss is compensated by an increasing number of map features. *Bottom*: The robot is represented with less DoF with increasing abstraction while the corresponding actions make use of additional environment semantics.

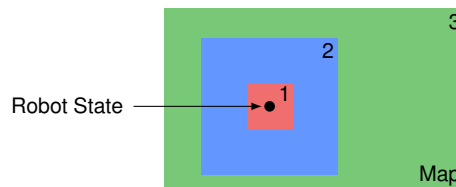


Figure 4.3: Size and position of the different levels of representation. *Level 1* covers the vicinity of the robot. *Level 2* is also robot centered and medium sized. *Level 3* covers the whole map.

actions for the robot to execute. However, coarse-to-fine planning approaches only utilize a coarse representation for the entire map in a first planning step. This coarse representation is obtained by discarding information bearing the risk of losing some of the required precise measurements. Hence, wrong decisions might be made which would result in sub-optimal or wrong paths and which would be subsequently refined without considering alternating decisions the more detailed representations would have made. By directly employing the detailed representation in the vicinity of the robot, where detailed measurements are available, this risk of information loss is overcome.

The developed method is integrated into the overall planning pipeline developed in Chapter 3 and depicted in Figure 3.2. It only af-

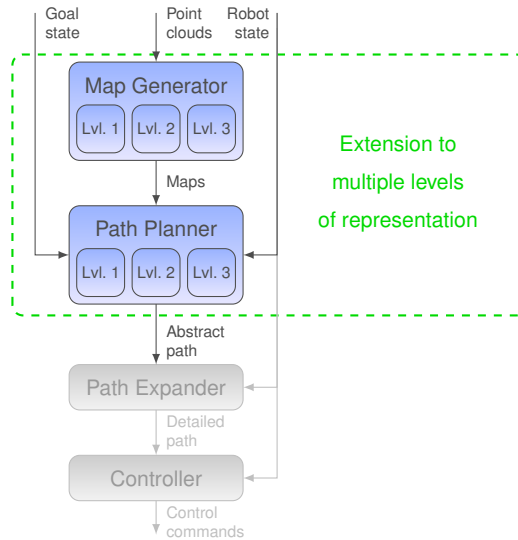


Figure 4.4: Extending the planning pipeline to employ multiple levels of representation. Only the Map Generator and the Path Planner are affected while all other components are identical to Chapter 3.

affects the Map Generator and the Path Planner, while other components of the pipeline do not require changes (see Figure 4.4). The input still consists of point clouds, the current robot state, and the desired goal state. Map generation is extended to generate three environment representations instead of one. Planning is still realized through the A*-based approach with a weighted heuristic and works on state costs. Suitable methods are presented to compute state costs from each environment representation. The resulting path consists of segments represented with different levels of abstractions depending on the distance from the current robot position. As described in Chapter 3, planner outputs need to be expanded to detailed motion sequences to be executable by the Controller. In this approach, path expansion is only applied to those path segments which include the next actions in the vicinity of the robot and, hence, are represented in *Level 1*. Since this representation is similar to the representation in the previous chapter, the same Path Expander component and subsequently the same Controller can be used. During path execution, continuous map updates, and refining or replanning procedures are used to ensure that the next actions are always represented in *Level 1* and can be expanded.

The rest of this chapter is structured as follows. Section 4.2 and Section 4.3 explain the planning representations in detail. We further developed an informed heuristic based on the most abstract representation, which is explained in Section 4.4. Moreover, a method was developed which continuously refines coarse plan segments to more detailed representations during path execution (see Section 4.5). This decreases the number of necessary replanning steps. However,

to avoid the above-described problem of wrongly assessed scenes in more abstract representations, replanning is triggered as soon as costs indicate that a situation might be assessed wrongly. Experiments show that the presented method enables the planner, compared to the method in Chapter 3, to solve path queries for significantly larger maps in feasible time while the path quality stays comparable. Details on the experiments can be found in Section 4.6 while a conclusion is drawn in Section 4.7.

4.2 ENVIRONMENT AND ROBOT REPRESENTATION

The environment representation is extended to three levels of representation with increasing abstraction. While the map resolution gets coarser for more abstract representations, the information loss is compensated by additional map features increasing the semantics. Figure 4.2 (top) gives an overview of the characteristics of the different environment representations.

Similarly, the robot representation is extended to three levels of representation. With increasing abstraction, the robot is represented with less DoF while the corresponding state cost computation accesses the additional map features, as visualized in Figure 4.2 (bottom).

Cost Similarity

We define two representations to be cost similar if the same robot actions induce similar costs in both representations.

All environment representations are used to generate state costs for a given robot state which are used by the planner. Since all representations are used simultaneously during planning, it is important that the same robot actions induce similar costs in all representations. Otherwise, the planner would prefer planning in some representations over others which is not desirable. Action costs are derived from state costs. Hence, the same robot state should induce similar costs in all representations. We call this property *cost similarity*.

In general, more abstract representations can be derived from more detailed representations. This allows for the generation of the individual environment representations in the following manner: At first, the *Level 1* environment representation is generated for the whole map. Subsequently, this is used to compute the *Level 2* environment representation for the whole map as well. Next, the *Level 3* representation is derived from the *Level 2* representation. Finally, map patches of the corresponding positions and sizes (see Figure 4.3) are cropped for each level, forming the basis for subsequent state cost computation. The individual details can be found in the following subsections.

4.2.1 Level 1 Representation

The *Level 1* environment representation is similar to the environment representation described in Section 3.2. A height map with a resolution of 2.5 cm is generated from point clouds originated from laser scanner range measurements. For the *Level 1* maps, we crop a squared

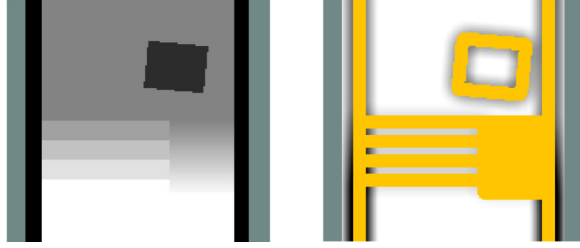


Figure 4.5: *Level 1* environment representation. *a)* Height map showing a corridor with a flight of stairs, an untraversable steep ramp and an obstacle. *b)* Corresponding foot cost map (yellow = untraversable by driving, olive = unknown).

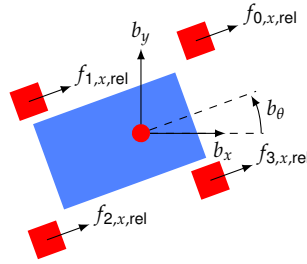


Figure 4.6: *Level 1* robot representation with seven DoF.

map patch with a size of 3×3 m around the current robot position. This is sufficiently large to plan the next robot maneuvers in high detail, but still small enough to avoid high-dimensional planning in large areas. Please note that the map patch is cropped around the current robot position, but its orientation is independent from the robot orientation.

The *Level 1* robot representation describes 7-DoF robot states $\vec{r}_1 = (b_x, b_y, b_\theta, f_{0,x,rel}, f_{1,x,rel}, f_{2,x,rel}, f_{3,x,rel})$ consisting of the robot base center position b_x, b_y and orientation b_θ , and the individual, relative, longitudinal foot positions $f_{0,x,rel}, \dots, f_{3,x,rel}$. At each position the robot can have 64 different discrete orientations.

For state cost computation, foot costs $C_F(\vec{f}_i)$ and base costs $C_B(\vec{r})$ are combined. Foot costs describe the cost to position an individual robot foot in the environment and include information about the terrain surface and obstacles in the surrounding. They are based on local unsigned height differences derived from the height map. Base costs describe the costs to place the robot base in a given state on the map. They include information about obstacles under the robot base and the terrain slope under the robot. Finally, foot costs for the four individual foot positions and base costs are combined to state costs. More details can be found in Section 3.2. Figure 4.5 depicts a *Level 1* height map and foot cost map, Figure 4.6 visualizes the *Level 1* robot representation.

4.2.2 Level 2 Representation

Input to the *Level 2* environment representation is the *Level 1* height map with a resolution of 2.5 cm covering the whole map size. To generate the *Level 2* height map and height difference map with a resolution of 5 cm, subsampling is performed. This is designed with respect to the Nyquist-Shannon sampling theorem which implies that subsampling has to come along with smoothing (Shannon, 1998). We subsample the *Level 1* height map as shown in Figure 4.7. Each *Level 2* height value is computed from the normalized, weighted sum of a 4×4 -region of *Level 1* height values. A binomial distribution is employed for weighing. The *Level 2* height difference map is generated in the same manner: local unsigned height differences are computed from the *Level 1* height map. Again, a 4×4 -region is used to subsample each *Level 2* height difference map cell using the same subsampling scheme. For the *Level 2* maps, we crop a squared map patch with a size of 9×9 m around the current robot position such that *Level 2* path segments are approximately twice as long as *Level 1* path segments. Figure 4.8 (a) and (b) visualize patches of *Level 2* height and height difference maps.

In order to obtain a robot representation with less DoF, individual foot positions are accumulated to foot pairs with a single longitudinal position. This is intuitive, since we observed a tendency to pairwise foot maneuvers in *Level 1* paths. While one can think of many situations in which an identical longitudinal coordinate for the two front or two rear feet is not feasible, we use foot position areas instead of

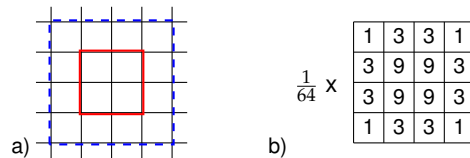


Figure 4.7: Subsampling method. *a)* For a *Level 2* cell (red square) a 4×4 -region (blue square) of *Level 1* cells is considered. *b)* Normalized binomial distribution to weigh heights and height differences.



Figure 4.8: *Level 2* environment representation. *a)* Height map. *b)* Height difference map. *c)* Foot pair cost map for the orientation indicated by the red arrow.

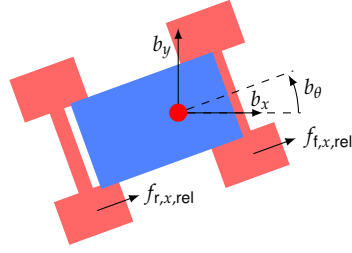


Figure 4.9: *Level 2* robot representation with five DoF.

foot positions to provide the approach the required flexibility. For a foot position area, it is known that the corresponding foot is positioned somewhere in the respective area while knowledge about the exact foot position is not required and not available. This supports the general assumption that the *Level 2* representation is employed in a certain distance to the current robot positions and, hence, the knowledge about these areas might be affected by some time-related and measurement-related imprecision. Therefore, a precise representation of the individual foot positions is not valuable. A *Level 2* robot state $\vec{r}_2 = (b_x, b_y, b_\theta, f_{f,x,rel}, f_{r,x,rel})$ is consequently represented through the base state and the relative longitudinal front foot pair and rear foot pair coordinates $f_{f,x,rel}$ and $f_{r,x,rel}$. At each position, the robot can have 32 different discrete orientations. The resulting robot representation is visualized in Figure 4.9.

For state cost computation, body costs are combined with foot pair costs. Computation of the body costs $C_{B,2}(\vec{r}_2)$ is similar to *Level 1* and only relies on height information. The foot pair costs

$$C_{FP,2}(\vec{fp}_i) = 1 + k_{C_{FP,1}} \cdot \Delta h_{FP,avg}(\vec{fp}_i) \quad (4.1)$$

are induced when a foot pair with the state $\vec{fp}_i = (fp_{i,x}, fp_{i,y}, fp_{i,\theta})$ is placed on the map. $\Delta h_{FP,avg}(\vec{fp}_i)$ describes the average height difference of the covered area. We define a set of small planning tasks in different environments for which the average path costs difference between the *Level 1* and *Level 2* solution is minimized. This results in $k_{C_{FP,1}} = 107$. Figure 4.8 (c) visualizes a foot pair cost map. Finally, the *Level 2* state costs are computed with

$$C_2(\vec{r}_2) = k_{C_{2,1}} \cdot C_{B,2}(\vec{r}_2) + k_{C_{2,2}} \cdot \left(C_{FP,2}(\vec{fp}_f) + C_{FP,2}(\vec{fp}_r) \right), \quad (4.2)$$

where $k_{C_{2,1}} = 0.5$ and $k_{C_{2,2}} = 0.25$.

4.2.3 *Level 3* Representation

We apply the subsampling method which is described in the previous subsection to generate *Level 3* height and height difference maps with a resolution of 10 cm from their *Level 2* counterparts. To increase the semantics, we categorize each of the *Level 2* map cells into one of the following terrain classes:

- *flat*: easily traversable by driving,
- *rough*: traversable by driving with high effort,
- *step*: includes height differences which are too large to be traversed by driving but can be traversed by stepping,
- *wall*: occurring height differences are too large to be traversed by stepping, and
- *unknown*: cell cannot be classified.

First, all *step* cells are identified. This is done by searching for cell pairs (c_i, c_j) meeting the following criteria:

- $\Delta h_{lv12}(c_i) < \Delta h_{\max, \text{drive}}$: c_i is on a drivable surface,
- $\Delta h_{lv12}(c_j) < \Delta h_{\max, \text{drive}}$: c_j is on a drivable surface,
- $\|c_i - c_j\| < l_{\text{step}, \max}$: the distance between c_i and c_j is within a maximum step length, and
- for the set T of cells c_k on the straight line between c_i and c_j , $C_F(c_k) = \infty$ counts for all cells $c_k \in T$: a direct foot motion from c_i to c_j requires a step.

$l_{\text{step}, \max}$ is tuned to 0.5 m. For all cell pairs (c_i, c_j) which meet the above-listed criteria, each cell $c_s \in c_i \cup c_j \cup T$ is assigned the terrain class *step*. Moreover, we compute the angle α_s between $\vec{c_i c_j}$ and the horizontal axis and save it for each cell c_s . However, most *step* cells are detected several times with different angles α_s . Hence, we save all these angles for each cell and finally compute the *mean of circular quantities* ($\alpha_{\text{avg}, s}$). $\alpha_{\text{avg}, s}$ describes the estimated step orientation of a *step* cell.

Second, the remaining map cells are classified according to their *Level 2* height difference value. Please note that these height difference values are subsampled and smoothed from the original *Level 1* height differences and cannot be directly connected to occurring height differences in the terrain:

- *flat* if $\Delta h_{lv12}(c_i) \in [0 \text{ m}, 2 \cdot 10^{-4} \text{ m})$,
- *rough* if $\Delta h_{lv12}(c_i) \in [2 \cdot 10^{-4} \text{ m}, 0.05 \text{ m})$,
- *wall* if $\Delta h_{lv12}(c_i) \in [0.05 \text{ m}, \infty)$, and
- *unknown* if $\Delta h_{lv12}(c_i)$ is unknown.

The stated height difference intervals have been tuned manually with respect to a maximum terrain height difference of 4 cm which can be overcome by driving and a maximum terrain height difference of 30 cm which can be overcome by stepping. The terrain class of a *Level 3* map cell is generated from the respective four *Level 2* cells by either choosing the terrain class with most members or, if this cannot be identified, the least difficult occurring terrain class. Figure 4.10 (a) and (b) give examples for a *Level 3* height map and terrain class map.

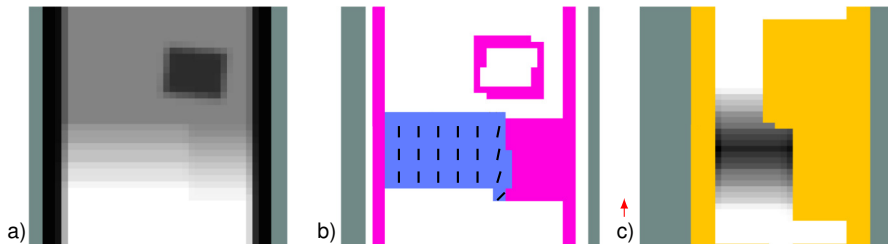


Figure 4.10: *Level 3* environment representation. *a)* Height map. *b)* Terrain class map (white = flat, blue = stepping, pink = wall, black lines = step orientations). *c)* Robot area cost map for the orientation indicated by the red arrow.

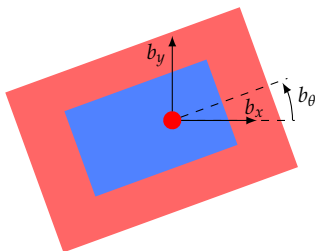


Figure 4.11: *Level 3* robot representation with three DoF.

The *Level 3* robot representation $\vec{r}_3 = (b_x, b_y, b_\theta)$ only contains information about the 3-DoF robot base state (see Figure 4.11). Instead of considering individual foot or foot pair positions, we assume that the robot feet are somewhere in a ground contact area a_r around and under the robot. Hence, this representation hinders the consideration of individual robot limb motions. The robot together with its ground contact area are rather moved over the terrain while traversing different terrain classes inducing corresponding costs. In *Level 3* the robot may have one of 16 different orientations at each position.

The *Level 3* state cost computation differs from the methods presented for the previous levels. Instead of distinguishing between base costs and costs induced by ground contacts with feet, the *Level 3* state cost computation is more abstract and only relies on terrain classes under a_r . Each cell c_i is assigned a cell costs value $C_c(c_i)$ depending on its terrain class:

- *flat*: $C_c(c_i) = 1.0$,
- *rough*: $C_c(c_i) = 1.4$,
- *step*: $C_c(c_i) = 76.0 + 2.95 \cdot \Delta h_{lv13}(c_i)$,
- *wall*: $C_c(c_i) = \infty$, and
- *unknown*: $C_c(c_i) = \text{nan}$.

The described terrain class specific cell costs are manually tuned: again, the set of small planning tasks is used to compare the *Level 1*

and *Level 3* path costs and minimize the cost difference. While constant terrain class cell costs are sufficient for *flat* and *rough* cells, costs for stepping maneuvers heavily depend on the height difference to overcome. This dependency is incorporated in the cell cost computation. The presented method obtains a cost difference of $< 5\%$ for the considered planning task set. The patch of a resulting robot area cost map can be seen in Figure 4.10 (c).

4.3 ACTION SET, AND COST FUNCTION

The employed A*-based algorithm requires a set of discrete actions to generate neighbor states and perform graph search. In our approach, only required neighbor states are generated online during state expansion since the precomputation of the whole state space is too expensive. An individual action set is required for each level, since the robot representation varies.

Actions are accompanied by a corresponding cost function which is individually described for each level in this section as well. Since the planner utilizes all representations during a single planning run, cost functions of the different levels need to be *cost similar* preventing the planner from preferring planning in some representation levels over others.

Section 4.3.1 - Section 4.3.3 describe the action sets and cost functions for the individual representation levels. In addition, Section 4.3.4 describes the level transition.

4.3.1 *Level 1*

The *Level 1* action set is identical to the actions described in Section 3.3.3. Neighbor states can be either reached through omnidirectional driving or through stepping maneuvers. Feasible driving actions can be found within a 20-position neighborhood with fixed orientation, or by turning to the next discrete orientation with fixed position, as visualized in Figure 4.12 (a) and (b). Similar to the map resolution, the action resolution is chosen to be 2.5 cm. Orientations are discretized to 64 orientation steps.

Stepping-related maneuvers are only considered in the vicinity of obstacles where driving might be infeasible. Those can be an abstract step, a longitudinal base shift, driving individual front feet forward, or driving individual feet towards their neutral configuration, as visualized in Figure 4.12 (c - f). During path search, steps are described as abstract steps which are defined to be the direct transition from a pre-step to a post-step state. The robot stability as well as the detailed motion sequence to perform the step are not considered during path planning. Subsequently, only the states in the resulting path are expanded, as described in Section 3.4. Since the *Level 1* representa-

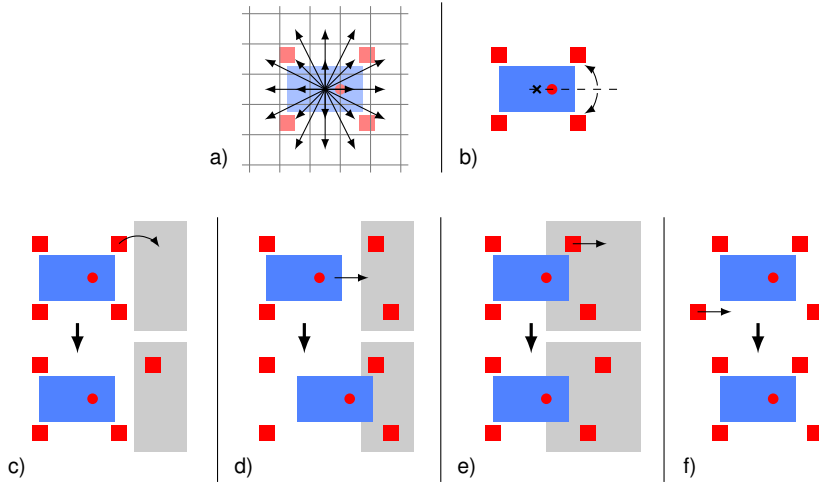


Figure 4.12: *Level 1* action set. Driving actions include *a)* omnidirectional driving within a 20-position neighborhood with fixed orientation or *b)* turning to the next discrete orientation with fixed position. Stepping-related actions include *c)* abstract steps with individual feet, *d)* longitudinal base shifts, *e)* driving an individual front foot forward, or *f)* driving an individual foot towards its neutral configuration.

tion is identical to the representation in the previous chapter, path expansion of *Level 1* path segments does not require modification.

The costs for the described actions are generated from the foot and base costs the individual robot elements induce during the respective maneuver. Stepping costs are more sophisticated and include information about the step length and step height. More details can be found in Section 3.3.3.

As an extension of the previous work, we want the robot to align its orientation with the orientation of stairs, when climbing those. This is desirable since Momaro's kinematic and the chosen robot representation only allow for foot motions in the sagittal plane (see Section 3.3.1). Moreover, alignment with the stair orientation can be observed, when humans climb staircases by themselves or teleoperate robots to do so. If, after a stepping maneuver, the two front/rear feet have the same longitudinal position but stand on different heights, this indicates that the robot is not aligned with the stairs it climbs. By punishing such configurations by an additional cost term, we achieve the desired behavior.

4.3.2 *Level 2*

The *Level 2* action set and cost function are designed with respect to the corresponding robot representation which unifies both front/rear feet to foot pairs. Similar to *Level 1*, driving actions include omnidirectional driving with fixed orientation to a neighbor state within a

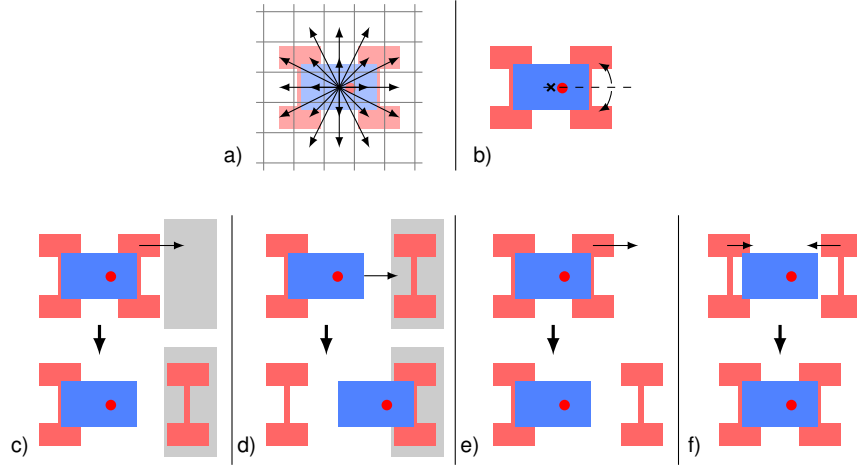


Figure 4.13: *Level 2* action set. Driving actions include *a)* omnidirectional driving within a 20-position neighborhood with fixed orientation or *b)* turning to the next discrete orientation with fixed position. Stepping-related actions include *c)* steps with a foot pair, *d)* longitudinal base shifts, *e)* driving the front foot pair forward, or *f)* driving a foot pair towards its neutral configuration.

20-position neighborhood and turning to the next discrete orientation with fixed position (see Figure 4.13 a, b). In comparison to *Level 1* and similar to the map resolution, the action resolution halves to 5 cm and 32 discrete orientation steps.

Stepping-related maneuvers differ from *Level 1* since the robot representation only allows for foot pair motions instead of moving individual feet. Again, those maneuvers are only considered in the vicinity of obstacles. In this representation, stepping maneuvers are performed with the whole foot pair. To motivate stepping maneuvers, we define a maximum height difference $\Delta h_{\max, \text{drive}}$ for the foot area center coordinate which can be overcome by driving. Larger height differences only can be traversed by stepping. In addition, the robot may perform a longitudinal base shift, move the front foot pair forward, or move any foot pair towards its neutral configuration (see Figure 4.13 c - f). Please note that resulting path segments represented in *Level 2* cannot be directly expanded by the Path Expander without refining them to *Level 1*.

The costs for driving actions are computed from the state costs of the corresponding states. Costs for foot pair maneuvers are the concatenated costs of the individual foot actions as described for *Level 1*. If, for example, the robot steps with its front foot pair, the costs for this maneuver are the sum of the costs for a step with the front left and a step with the front right foot. To obtain *cost similarity* between *Level 2* actions and corresponding *Level 1* actions, *Level 2* foot pair costs are tuned. This is done by comparing costs for a set of basic planning tasks (e.g., drive/turn on a patch of flat/rough underground, step up different height differences, do a base shift,...) in both repre-

sentations and manually tuning of the *Level 2* cost parameters until the cost difference for all those maneuvers is $< 5\%$. Again, a punishing cost term is introduced for post-stepping states with different average heights under the individual foot areas of the stepping foot pair.

4.3.3 *Level 3*

The *Level 3* action set only consists of actions which move the whole robot over the terrain while it is explicitly not distinguished between locomotion modes. Similar to omnidirectional driving, the robot may move towards a state within a 20-position neighborhood with fixed orientation or may turn to the next discrete orientation with fixed position, as visualized in Figure 4.14. In this case, the action resolution is 10 cm and the robot can have one of 16 discrete orientations. Similar to *Level 2*, path segments which are represented in *Level 3* cannot be directly expanded to detailed motion sequences by the Path Expander. Instead, an intermediate refinement to *Level 1* is required.

When moving over *step* cells, a robot state is only feasible if the difference between the robot orientation and the step orientation of all *step* cells covered by the robot area is less than one discrete orientation step. Moreover, the robot is only allowed to move parallel and orthogonal to step orientations. These constraints enforce a behavior, which is induced by the robot kinematic in lower representation levels but would be not represented in *Level 3*, otherwise.

Action costs are derived from the state costs of the corresponding states. As described in Section 4.2.3, a set of basic planning tasks is used to tune state costs and obtain *cost similarity* between *Level 3* and *Level 1*.

4.3.4 *Level Transition*

Since all three representation levels are combined in a single planner, it frequently occurs during path search that the planner reaches the border of a representation and needs to continue its search in the next, more abstract representation. To limit the state space size, the

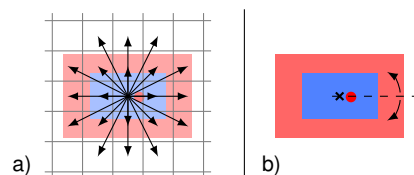


Figure 4.14: *Level 3* action set. The robot is moved above the surface either *a)* by moving to a neighbor position with fixed orientation or *b)* by turning to the next discrete orientation with fixed position.

Level 1 representation is only available in the vicinity of the robot while *Level 2* is also robot-centered and medium sized. *Level 3* covers the whole map. The planner checks for each action (e.g., drive to a neighbor position, perform an abstract step, ...) if the most detailed available representation level is identical for the start and goal state of this action. If this is the case, the action is applied. If the most detailed available representation level of the action goal state is more abstract than for the action start state, the start state is transformed into the goal state's representation level. Subsequently, the action is considered in this more abstract representation if it is still available in the corresponding action set.

However, the transformation from the action start state to the more abstract representation might induce additional costs. Due to different map resolutions, the robot might be horizontally shifted and turned to fit the coarser resolution of the more abstract representation. Foot configuration constraints might require additional foot motions to fit the next level representation (e.g., individual feet have to align within foot pair areas to enable a transformation from *Level 1* to *Level 2*). Each transformation is checked for feasibility and corresponding costs are included in the respective action costs.

4.4 ABSTRACT REPRESENTATION-BASED HEURISTIC

In Chapter 3, we presented a heuristic for the planner which combined the Euclidean distance with an orientation difference (*geometric heuristic*). However, this heuristic has no knowledge about the environment although the occurring environment has large influence on the resulting path. Incorporating environment information is promising in the considered planning domains to reduce the number of visited states before reaching the goal state. We developed an informed heuristic which is based on the *Level 3* representation and, hence, includes environment information. We call this the *abstract representation-based heuristic*.

After the goal state $\vec{r}_{g,i}$ is set, it is transformed to *Level 3* ($\vec{r}_{g,3}$). A one-to-any Dijkstra search is performed in *Level 3* starting from $\vec{r}_{g,3}$ and covering the whole map. Since the *Level 3* representation only has three DoF and a coarse resolution, this search is quick. Subsequently, a *Level 3* cost estimation to reach the goal state is available for each *Level 3* state in the map. While this search searches from the goal to any state, the heuristic requires an estimation from any state to the goal state. We employ an action set which consist of the *Level 3* action set but each action is reverted to obtain feasible results. During path planning, a cost estimation to reach the goal state can be get for an arbitrary state by transforming it to *Level 3* and reading the precomputed cost estimation. This is visualized in Figure 4.15.

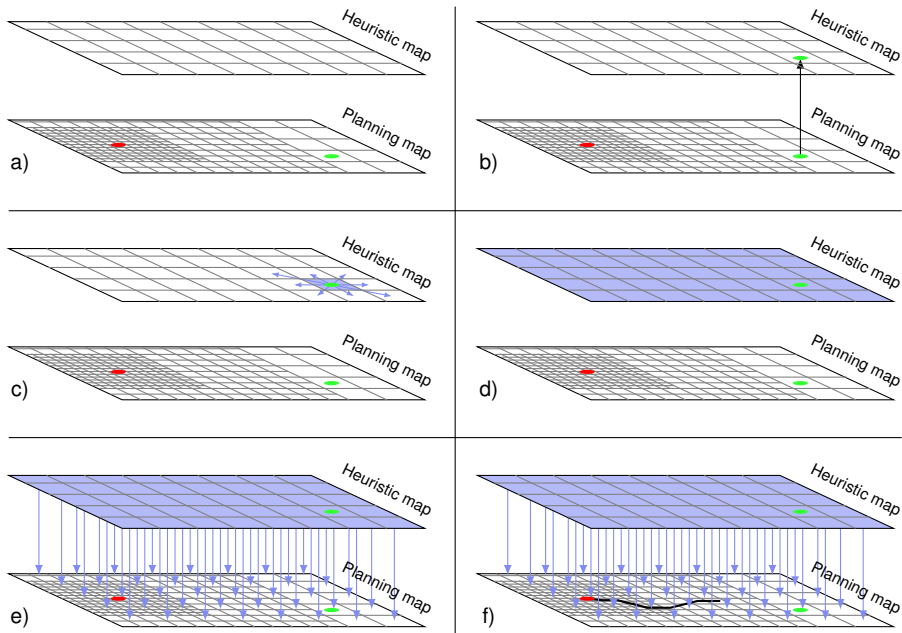


Figure 4.15: Computation of the *abstract representation-based heuristic*. *a)* For a given planning problem from a start state (red dot) to a goal state (green dot), *b)* the goal state is transformed to the *Level 3* heuristic map. *c)* A one-to-any Dijkstra search is performed in the heuristic map, starting from the goal state until *d)* all states have been visited. *e)* Subsequently, a cost estimation to reach the goal is available for each *Level 3* state and *f)* can be accessed as an informed heuristic during planning.

The quality of this heuristic strongly depends on the *Level 3*'s degree of *cost similarity*. In other words: how close does the *Level 3* cost function assess costs for maneuvers in comparison to the computed costs of the same maneuvers in other representation levels. Furthermore, we cannot prove that this heuristic always underestimates costs, which would be necessary to prove admissibility for the generation of optimal paths. Nevertheless, since we also employ the bounded sub-optimal A*-based algorithm with weighted heuristic terms, we do not aim at generating optimal solutions for a given problem. We rather focus on generating paths of sufficient quality in feasible time. The performance of this heuristic regarding path quality and computation time is evaluated in Section 4.6.2.

4.5 CONTINUOUS PATH REFINEMENT

After an initial path has been found and execution started, the robot moves along this path. Since only *Level 1* path segments can be expanded and subsequently executed by the Controller and since the initial path is only represented in *Level 1* in the vicinity of the robot, execution of the initial path is limited to a short segment. To enable

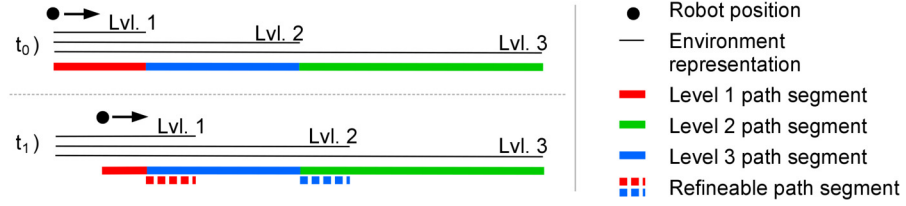


Figure 4.16: Continuous refinement method. As the robot moves along the path, the *Level 1* and *Level 2* representations move with it. Consequently, those path segments which are represented in a higher level and for which a more detailed representation becomes available, can be refined to this more detailed representation.

the robot to reach the goal, one could either trigger frequent replanning of the whole path or refine respective path segments to more detailed representations. The latter is promising since it considerably reduces the computational effort. While the robot moves, its sensors provide new measurements and environment representations are updated according to these new measurements and the current robot position. Those updates are included in the path by continuously refining those path segments for which a more detailed environment representation becomes available, as visualized in Figure 4.16. If a cost difference $> 25\%$ between the original and the refined path segments indicates that the higher-level assessment for a situation is wrong, we trigger a completely new planning run. This ensures that path segments in the vicinity of the robot are always represented in *Level 1* and thus, can be expanded and executed by the Controller. The corresponding refinement methods are described in the following.

Level 2 path segments can be refined to *Level 1* in the following manner: For a path segment between two successive *Level 2* states $\vec{r}_{i,2}$ and $\vec{r}_{i+1,2}$, both states are transformed to the *Level 1* states $\vec{r}_{i,1}$ and $\vec{r}_{i+1,1}$. By interpolating between the 3-DoF robot base states (b_x, b_y, b_θ) of these two *Level 1* states, a set of feasible base states S is generated. S is subsequently inflated by two position steps and one orientation step, as visualized in Figure 4.17. Finally, the planner, which is restricted to S , searches for a *Level 1* path between $\vec{r}_{i,1}$ and $\vec{r}_{i+1,1}$. If

- either one of the two states becomes infeasible when transformed to *Level 1* because *Level 2* assessed the given situation wrongly or
- the costs for the refined *Level 1* path differs by $> 25\%$ from the original costs for the path segment,

the respective path segment is defined to be not refinable and replanning is triggered.

The refinement of a *Level 3* path segment to *Level 2* differs in some aspects. Again, a set of feasible robot base states S is generated. In contrast to the previously described method for *Level 2* refinement, not

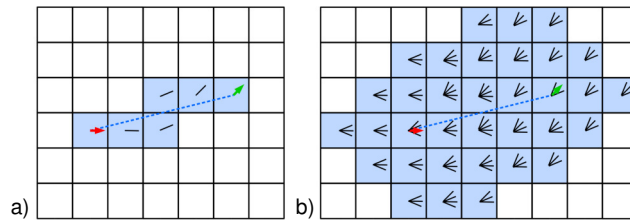


Figure 4.17: Generation of a set of feasible robot base states for path refinement. *a)* For the given start (red arrow) and goal (green arrow) robot base states of a path segment, we generate a set of feasible robot base states (black lines) by interpolating between the two. *b)* Inflation by two position steps and one orientation step.

only the path between two successive states but the whole *Level 3* path segment $\vec{r}_{i,3}, \vec{r}_{i+1,3}, \dots, \vec{r}_{j,3}$ is refined at once. The first and last robot state of this path segment are transformed to the *Level 2* states $\vec{r}_{i,2}$ and $\vec{r}_{j,2}$. Subsequently, the planner searches for a path between $\vec{r}_{i,2}$ and $\vec{r}_{j,2}$ while being restricted to S . To refine a *Level 3* path to *Level 1*, an intermediate refinement to *Level 2* is required.

4.6 EVALUATION

We evaluated the developed method in three experiments on artificially generated height maps. In a first experiment, the performance of the planner on all three representation levels was evaluated and compared (Section 4.6.1). A second experiment was designed to evaluate the developed *abstract representation-based heuristic* (Section 4.6.2). Finally, the evaluation of the influence of the robot start state is described in Section 4.6.3.

The employed hardware was one core of a 2.6 GHz Intel i7-6700HQ processor with 16 GB RAM to provide a realistic performance assessment for real robot applications, in which energy efficient processing units are often used and other computations have to be done in parallel. A video explaining the approach and showing the experiments is available online¹. It also contains an experiment in the Gazebo simulation environment demonstrating the continuous refinement strategy.

4.6.1 Representation Level Performance

To assess the quality of the different representation levels, the scenario depicted in Figure 4.18 was designed. It describes an indoor scene including obstacles of different sizes—of which some can be traversed by stepping while others are too high—, different rooms, door openings, and a step. To evaluate the performance of each representation individually, the whole map was represented in the respective repre-

¹<https://doi.org/10.5281/zenodo.3628732>

sensation and the planner was restricted to plan on this representation. In addition, planning on a combination of all three levels, which is one of the main contributions of this method, was evaluated. For this planner run, a *Level 1* size of 3×3 m and a *Level 2* size of 9×9 m were chosen while *Level 3* covered the whole map. The representation level positions followed Figure 4.3 such that *Level 1* and *Level 2* were centered to the robot start state. The planner used the *geometric heuristic* (see Section 4.4). Since this was also used in the experiments in Section 3.6, a comparison of the results is feasible.

While we used the ARA* algorithm in experiments of the previous section, we discovered that within a given planning time the planner usually outputs a solution with a constant heuristic weight. However, when directly planning with this heuristic weight, previous planner runs with higher heuristic weights can be omitted. Hence, we employed a weighted A* algorithm for these experiments, while comparing the results for different heuristic weights. The quantitative results can be seen in Figure 4.19 while a resulting path for planning with all three representation levels combined is depicted in Figure 4.18.

When planning with abstract representation levels > 1 , those path segments cannot be directly executed by the controller but need to be refined to *Level 1*. Hence, the costs of *Level 2* and *Level 3* path segments can be seen as cost estimates for the corresponding *Level 1* path segments. To evaluate the quality of these cost estimates, we refined all *Level 2* and *Level 3* path segments to *Level 1* and compared the costs, as shown in Figure 4.19.

The results indicate that planning on representation levels > 1 and with combined levels is faster by at least one order of magnitude compared to pure *Level 1* planning. Due to memory limitations, the *Level 1* path for $\mathcal{W} = 1.0$ could not be generated. A comparison between the estimated costs by the planner and the corresponding costs when refined to *Level 1* gives an assessment of the cost estimation in each level. Comparing these refined costs to the costs of pure *Level 1* planning gives an assessment of the resulting path quality. It can be seen that the estimated costs always underestimate the corresponding refined *Level 1* costs. This is an important observation for employing an abstract representation as a heuristic which needs to underestimate costs to guarantee optimality. Especially for $\mathcal{W} \leq 1.5$ the cost estimation of all levels is close to the real costs with a difference of $\leq 7.7\%$. Furthermore, for the *Level 2*, *Level 3*, and *combined* results, the corresponding refined *Level 1* costs for $\mathcal{W} \leq 1.5$ differ to the original *Level 1* path costs by $\leq 15\%$.

4.6.2 Heuristic Comparison

In a second experiment, we compared the developed *abstract representation-based heuristic* to the *geometric heuristic* (see Section 4.4 for de-

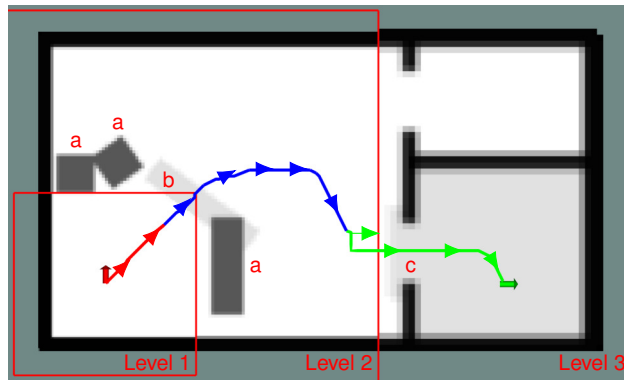


Figure 4.18: Height map of the first experiment scenario containing obstacles which are too high to be traversed (*a*), obstacles which can be traversed by stepping (*b*), and a step and door opening (*c*). From its start state (red arrow), the robot needs to navigate between these obstacles, climb a bar obstacle, navigate to the step, climb the step, and pass the door opening to finally reach the goal state (green arrow) in another room. The resulting path for $\mathcal{W} = 1.125$ and combined levels of representation is depicted. *Level 1* path segment = red, *Level 2* segment = blue, *Level 3* segments = green. Arrows visualize the robot orientation. The path visualizes the robot base center. As soon as a part of the robot leaves a representation, the planner employs the next higher representation level which is the reason why, e.g., the *Level 2* path segment already starts within the *Level 1* borders.

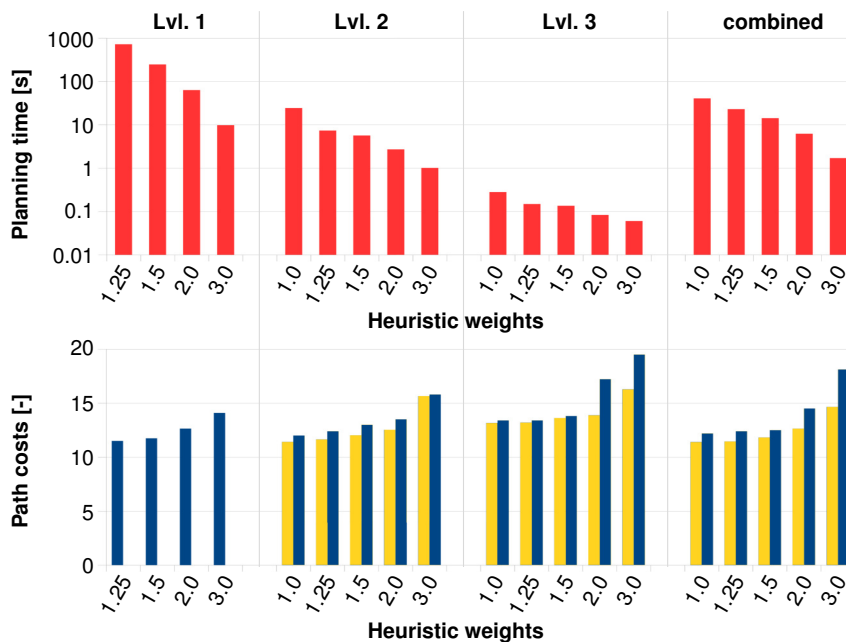


Figure 4.19: Planning performance for the first experiment for the three levels of representation isolated and combined. The *geometric heuristic* is employed. The influence of the heuristic weight \mathcal{W} is evaluated. Estimated costs by the planner = yellow, costs of the solutions when refined to *Level 1* = blue.

tails). A second scenario was designed which is larger and more challenging compared to the scenario of the first experiment (see Figure 4.20). It consists of two corridors of up to 10 m length which are connected through a door opening. The first corridor contains a bar obstacle and a field of rough terrain. The second corridor contains a flight of four stairs and two obstacles hindering stair climbing.

Planning is performed with all three levels combined, as described in Section 4.6.1. A resulting path is depicted in Figure 4.21. Quantitative planning time and path costs results are given in Figure 4.22. Preprocessing of the *abstract representation-based heuristic* took 0.52 s and is already included in the stated planning times.

It can be seen that the *abstract representation-based heuristic* significantly accelerates planning compared to the *geometric heuristic*. Since the *abstract representation-based heuristic* incorporates information of the terrain, the initial heuristic value is considerably closer to the resulting costs while still underestimating. A comparison of the resulting path costs gives an assessment of the heuristic quality. While the *geometric heuristic* guarantees feasibility, this guarantee cannot be given for the *abstract representation-based heuristic*. Nevertheless, it can be seen that path costs are comparable, especially for $\mathcal{W} \leq 1.5$. As an example, for $\mathcal{W} = 1.25$, planning is accelerated by more than two orders of magnitude while the refined path costs only differ by 3.3%. Moreover, the resulting path illustrates how the robot aligns with the stairs and only moves parallel and orthogonal to them.

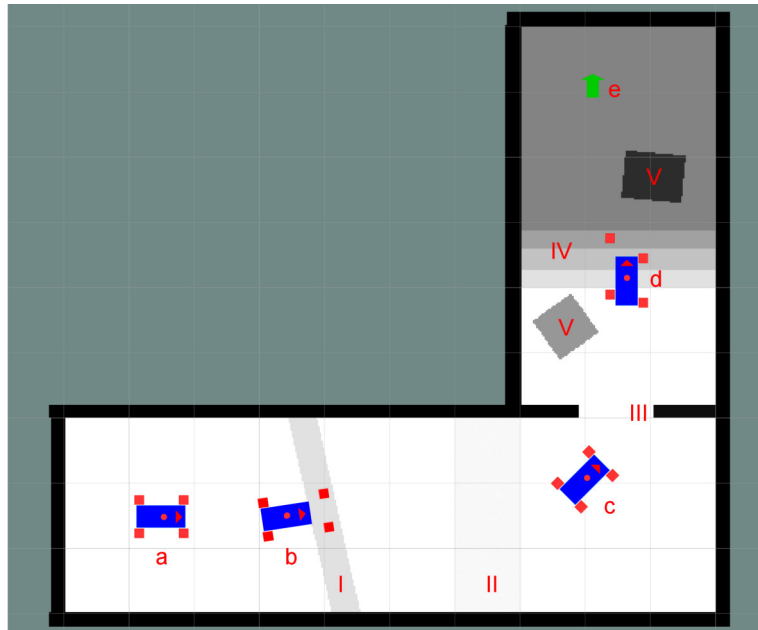


Figure 4.20: Height map of the second experiment containing a bar obstacle (I), a field of rough terrain (II), a door opening (III), a flight of stairs (IV), and two obstacles (V). $a - d$ are start states, e is the goal state. Start state a is used for heuristic comparison.

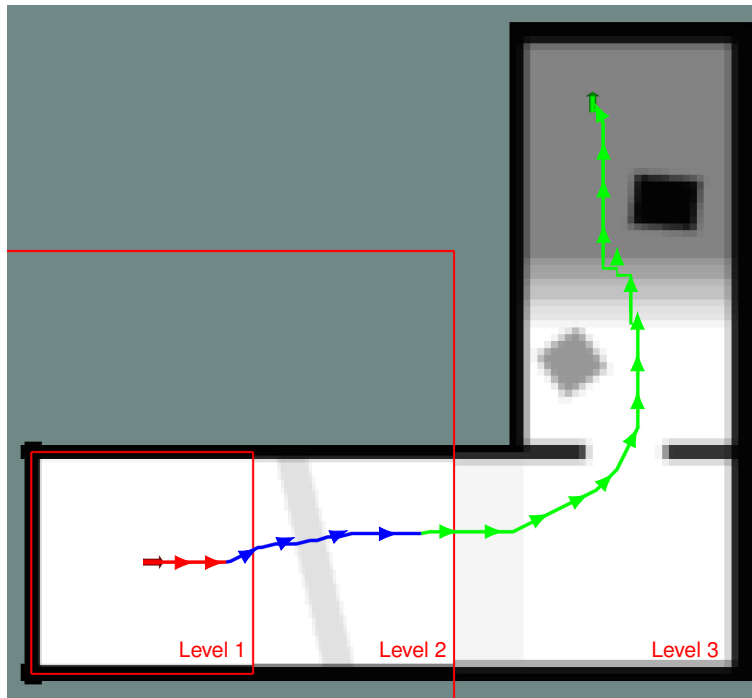


Figure 4.21: Resulting path for planning with the *abstract representation-based heuristic* and combined levels with $\mathcal{W} = 1.25$.

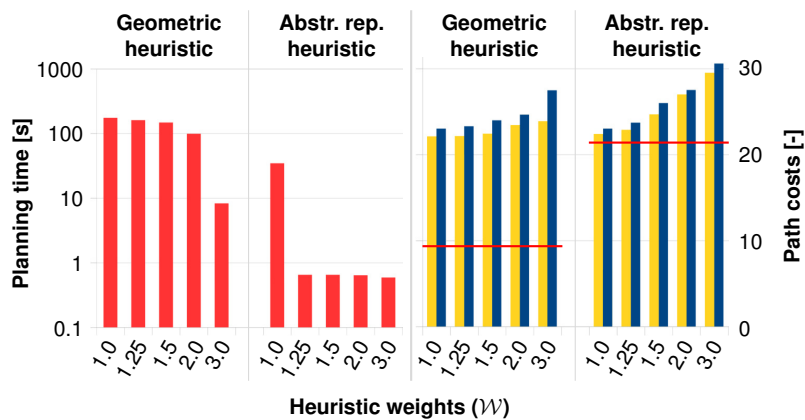


Figure 4.22: Planning performance for combined levels of representation to compare the *geometric heuristic* with the *abstract representation-based heuristic*. Red lines indicate the initial cost estimate of the heuristic.

4.6.3 Start State Influence

In a third experiment, we evaluated the influence of the start state on the planner performance. We employed the scenario described in Figure 4.20. The planner planned on all three representation levels combined and used the *abstract representation-based heuristic*. Quantitative results of the planning times are given in Figure 4.23. Since resulting

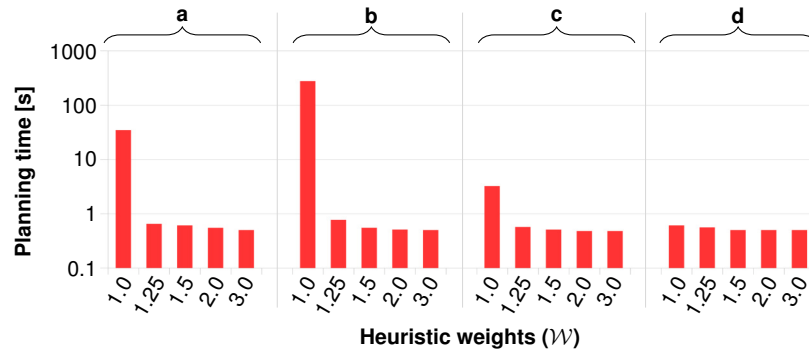


Figure 4.23: Planning times for different start states (see Figure 4.20) and different \mathcal{W} , using the *abstract representation-based heuristic*.

path costs differ between different start states, a comparison of those has no value.

The results indicate that an important factor for the planning performance is not only the distance between the start and goal state but also the complexity within the *Level 1* representation. This can be explained by the fact that complex scenarios represented in *Level 1* lead to large state spaces which have to be searched. Nevertheless, higher heuristic weights lead to feasible planner performances in any case.

4.7 CONCLUSION

In this chapter, an extension to the search-based hybrid driving-stepping locomotion planning approach of Chapter 3 to plan on three levels of representation was presented. With increasing distance to the robot position, the robot and environment representations get coarser and the robot is represented with less degrees of freedom. Hence, planning problems can be represented with smaller state space sizes accelerating search. To compensate this loss of information which comes along with the coarser and lower-dimensional representations, the corresponding representation semantics increase resulting in more abstract representations. The planner uses all representations in a single planning run avoiding the information loss of hierarchical coarse-to-fine approaches.

Experimental results indicated that this method provides resulting paths for challenging environments, which are qualitatively comparable to the detailed planning of Chapter 3, but for which the generation is faster by multiple orders of magnitude. Hence, significantly larger planning problems can be solved in feasible time which increases the real-world applicability of the approach.

Moreover, a method to continuously refine path segments from abstract representations to more detailed representations was presented. This could be used during path execution to reduce the number of required replanning procedures.

Finally, a powerful, informed heuristic was presented, which incorporates knowledge of the environment and the robot capabilities and which is based on the most abstract representation. While a quick preprocessing is required, the path search is guided by this heuristic causing further acceleration with comparable result quality.

TOWARDS LEARNING ABSTRACT REPRESENTATIONS FOR LOCOMOTION PLANNING IN HIGH-DIMENSIONAL STATE SPACES

In Chapter 4, we presented an extension to the search-based hybrid driving-stepping locomotion planner of Chapter 3. Instead of using a uniform representation for the whole planning problem, multiple representations with different levels of abstraction were developed. An abstract representation uses a coarser environment and action resolution while the robot is represented with less DoF. An increasing number of features compensates this information loss by adding additional semantics to the representation. Especially the employment of an informed heuristic based on the most abstract representation led to considerable planning acceleration with comparable result quality.

However, the generation of such abstract representations is challenging. Since the heuristic's task is the efficient cost estimation for shortest paths between states, its performance depends on the cost estimation quality of the abstract representation. In other words, the abstract representation needs to be *cost similar* with the detailed representation, i.e. the cost difference for the same planning task between the detailed and abstract representation should be minimized.

In the previous chapter, the abstract representations were manually designed. While a robot representation and corresponding action set could easily be defined, the definition of a suitable environment representation with suitable features was challenging. Moreover, a significant manual tuning effort was required for the cost function which incorporated these features. To obtain *cost similarity*, a set of local planning tasks was defined, and the abstract representation was manually tuned to minimize the path costs difference to the detailed representation on this task set. However, the corresponding tuning strongly depends on the chosen task set. Furthermore, manual tuning needs to be repeated after each modification either of the detailed or of the abstract representation, which is exhausting.

A domain where abstraction is well-established are CNNs. Convolutions decrease the input resolution while computing additional features. Moreover, abstraction through CNNs can be learned given sufficient training data. In this chapter, an approach is presented to support the implementation of an abstract representations by a CNN representing the abstract cost function. Since training is realized on generated artificial data, manual design and tuning efforts are minimized and dependencies on the task set are eliminated.

5.1 SYSTEM OVERVIEW

In general, the approach consists of a detailed and an abstract planning representation. The detailed representation has a sufficient environment and action resolution and describes the robot with sufficient DoF to make use of all required robot capabilities, to be applicable to the considered planning domain, and to generate paths which can be executed by a successive controller. However, in challenging environments and with sophisticated robot kinematics, this detailed representation usually results in large state spaces posing challenges for efficient path generation.

A second abstract representation describes the same planning problem with a smaller state space allowing for efficient planning. A smaller state space can be obtained through, e.g., a coarser resolution, a lower-dimensional robot representation, or a combination of both. The abstract representation does neither need to represent all kinematic robot capabilities nor to output paths which can be executed by the controller. It rather needs to provide feasible cost estimates to guide the planning in the detailed representation. Hence, the abstract representation needs to be *cost similar* with the detailed representation. The implementation of the abstract representation is supported by a CNN that maps spatially small planning tasks to corresponding cost estimates for the shortest path. These small planning tasks form a coarse, low-dimensional set of actions. The CNN represents a corresponding *cost similar* cost function. Subsequently, this abstract representation can be used to efficiently precompute heuristic values for the whole map accelerating path search in the detailed representation. A more precise problem statement is given in Section 5.2.

The approach is applied to the problem of hybrid driving-stepping locomotion planning, but it can be easily transferred to other domains, e.g., walking locomotion. The method architecture is visualized in Figure 6.1.

We choose the detailed representation to be the representation described in Chapter 3. It describes the environment as height maps with a resolution of 2.5 cm. 7-DoF states describe the robot while a corresponding action set contains omnidirectional driving and stepping related maneuvers.

The implementation of the abstract representation is motivated by the *Level 3* representation in Chapter 4. The robot is represented by 3-DoF base states with a resolution of 10 cm while individual foot positions are neglected. A corresponding action set describes robot motions to neighbor states. In the approach presented in this chapter, there is no explicit abstract environment representation, but the CNN directly maps abstract actions and the corresponding map patch of the underlying detailed height map to cost estimates. More details on the representations are given in Section 5.3. We developed a method

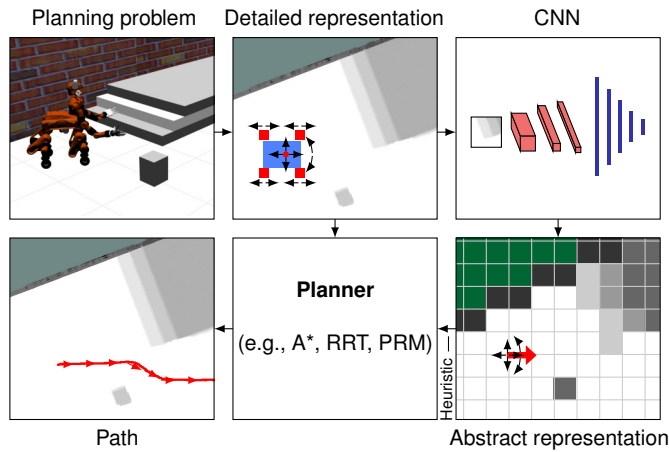


Figure 5.1: Method overview. For a complex planning problem, a detailed representation is generated representing all required environment information and robot capabilities. This representation is employed by a planning algorithm to generate paths. The pipeline is supported by an additional abstract representation. A CNN represents the abstract cost function and maps spatially small planning tasks to cost estimates. The abstract representation used to generate an informed heuristic which supports planning in the detailed representation.

to generate artificial data sets which are used for network training. During planning, the abstract representation is used to generate an informed powerful heuristic. A coarse-to-fine planning approach would also be feasible with these two representations. However, to avoid cases in which wrong situation assessments of the abstract representation lead to wrong paths that are subsequently refined to the detailed representation, we prefer to directly plan in the detailed representation while the *abstract representation-based heuristic* provides additional information and guides the search towards the goal. To generate the heuristic, a search in the abstract representation is carried out from the goal state to all states in the map. The result provides cost estimates for the shortest path from an arbitrary state to the goal state. The CNN and the heuristic are described in Section 5.4.

Experimental results are presented in Section 5.5. Our method generalizes well to real-world scenes although it is trained on generated artificial data. The results further show that the proposed method outperforms our manually tuned approach from Chapter 4 in terms of abstraction quality, while eliminating tuning efforts. Moreover, the proposed heuristic accelerates path planning by multiple orders of magnitude, compared to popular heuristics.

5.2 PROBLEM STATEMENT

Given is a planner which uses an environment representation \mathcal{E}_d , a robot representation \mathcal{R}_d , a corresponding action set \mathcal{A}_d , and a cost function \mathcal{C}_d . \mathcal{E}_d is a map with an arbitrary number of features describing each cell. \mathcal{R}_d represents all required DoF of the robot kinematic which are necessary to address the planning problem. \mathcal{A}_d contains all actions which can be executed by the robot such that $\vec{r}_{d,i} + \vec{a}_{d,j} = \vec{r}_{d,i+1}$, an action $\vec{a}_{d,j} \in \mathcal{A}_d$ connects two successive robot states $\vec{r}_{d,i}, \vec{r}_{d,i+1} \in \mathcal{R}_d$ while inducing the costs $\mathcal{C}_d(\vec{r}_{d,i}, \vec{a}_{d,j})$.

A second representation is the abstract representation consisting of $\mathcal{E}_a, \mathcal{R}_a, \mathcal{A}_a$, and \mathcal{C}_a . It can be used to support the planning. \mathcal{R}_a describes the robot state in a low-dimensional state space although this might not suffice to describe the robot state in enough detail for execution. The correspondence between an abstract robot state $\vec{r}_{a,i} \in \mathcal{R}_a$ and a detailed robot state $\vec{r}_{d,i} \in \mathcal{R}_d$ is given through the transformation

$$\vec{r}_{a,i} = \mathcal{T}_{d \mapsto a}(\vec{r}_{d,i}) \quad (5.1)$$

and vice versa with

$$\vec{r}_{d,i} = \mathcal{T}_{a \mapsto d}(\vec{r}_{a,i}). \quad (5.2)$$

\mathcal{A}_a describes actions $\vec{a}_{a,j}$ to move an abstract robot state $\vec{r}_{a,i} \in \mathcal{R}_a$ to a successive state $\vec{r}_{a,i+1} \in \mathcal{R}_a$. The resolution of \mathcal{A}_a is coarser compared to \mathcal{A}_d , such that an action sequence

$$\mathcal{T}_{a \mapsto d}(\vec{r}_{a,i}) + \vec{a}_{d,j} + \vec{a}_{d,j+1} + \dots + \vec{a}_{d,j+k} = \mathcal{T}_{a \mapsto d}(\vec{r}_{a,i+1}) \quad (5.3)$$

is necessary in the detailed representation to perform the least cost transition between two successive robot states in the abstract representation, while the difference between the abstract costs $\mathcal{C}_a(\vec{r}_{a,i}, \vec{a}_{a,j})$ and the detailed costs $\mathcal{C}_d(\mathcal{T}_{a \mapsto d}(\vec{r}_{a,i}), \vec{a}_{d,j}, \dots, \vec{a}_{d,j+k})$ should be minimized to obtain *cost similarity*.

While \mathcal{R}_a and \mathcal{A}_a can be easily defined, \mathcal{E}_a and \mathcal{C}_a needed extensive manual tuning in Chapter 4 to obtain *cost similarity*. \mathcal{E}_a was derived from \mathcal{E}_d . \mathcal{C}_a subsequently employed \mathcal{E}_a to generate costs. In this chapter, we present a CNN which directly maps from \mathcal{E}_d to \mathcal{C}_a . The design of an additional abstract environment representation is not required and tuning efforts are minimized.

5.3 PLANNING REPRESENTATIONS

The approach is applied to the problem of hybrid driving-stepping locomotion planning for the robot platforms Momaro and Centauro (see Figure 5.2). The corresponding detailed planning representation is described in Section 5.3.1. Details on the abstract representation are given in Section 5.3.2.



Figure 5.2: Hybrid driving-stepping locomotion robots addressed by the presented planning method. *Left: Momaro. Right: Centauro*

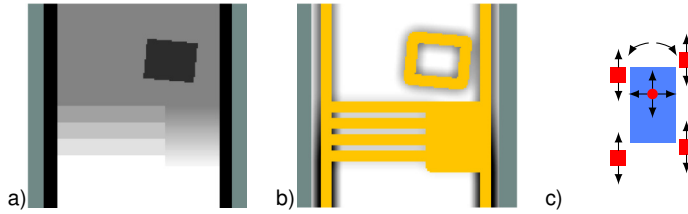


Figure 5.3: Detailed planning representation. *a) Height map. b) Foot cost map (yellow = untraversable by driving, olive = unknown). c) Robot representation.*

5.3.1 Detailed Representation

As the detailed planning representation, the representation developed in Chapter 3 is employed. The environment representation \mathcal{E}_d is a height map with a resolution of 2.5 cm. Foot costs and base costs are derived from this height map. They describe the costs to place an individual robot foot/the robot base at a given state in the map and are combined into whole robot state costs. A height map and foot cost map example is depicted in Figure 5.3 (a) and (b).

The robot representation \mathcal{R}_d is implemented as 7-DoF states $\vec{r}_d = (b_x, b_y, b_\theta, f_{0,x,rel}, f_{1,x,rel}, f_{2,x,rel}, f_{3,x,rel})$ with (b_x, b_y, b_θ) describing the robot base state and $f_{0,x,rel}, \dots, f_{3,x,rel}$ describing the relative longitudinal position of each foot, as shown in Figure 5.3 (c). Lateral foot positions are fixed, as explained in Section 3.3.1. Foot heights are computed only for those states which are contained in the resulting path. Positions have a resolution of 2.5 cm while there are 64 discrete orientations for the robot base.

The corresponding action set \mathcal{A}_d includes the following actions which are also depicted in Figure 5.4:

- omnidirectional driving within a 20-neighborhood with fixed orientation,
- turning to the next discrete orientation with fixed position,

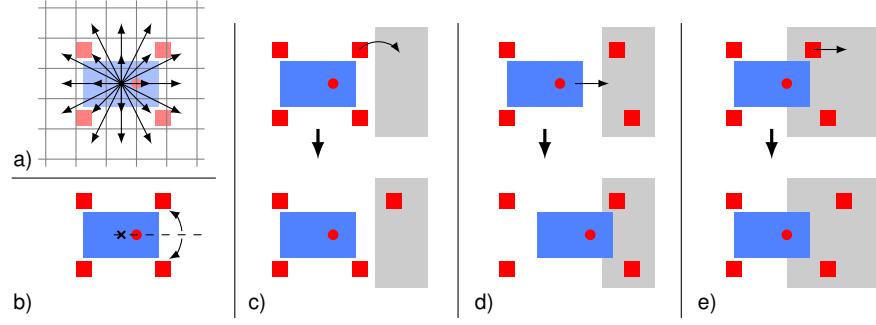


Figure 5.4: Action set of the detailed representation (\mathcal{A}_d): *a)* Omnidirectional driving with fixed orientation. *b)* Turning to the next discrete orientation with fixed position. *c)* Step. *d)* Longitudinal base shift. *e)* Driving a foot relative to the base while keeping ground contact. Grid and orientation resolution are enlarged for illustrative purposes.

- performing a step with a single foot,
- moving the base longitudinal relative to the feet, and
- driving an individual foot relative to the robot base while keeping ground contact.

Steps are represented as the direct transition from a pre-step state to a post-step state. Only those steps in the resulting path are refined to detailed motion sequences considering robot stability, detailed stepping motions, and foot heights.

The detailed cost function \mathcal{C}_d calculates costs for each action and is based on the foot and body costs the robot experiences when performing the action. Costs are subsequently weighted to, e.g., prefer driving over stepping, or to give the planner a preference of moving with a forward orientation. More details can be found in Section 3.3.3.

5.3.2 Abstract Representation

The abstract representation is inspired by the *Level 3* representation of the previous chapter (see Section 4.3.3). The robot is represented with 3-DoF states $\vec{r}_a = (b_x, b_y, b_\theta)$ which describe the robot base state with a resolution of 10 cm and 16 discrete orientation steps. Individual foot positions are neglected.

The corresponding abstract action set \mathcal{A}_a contains the following actions to move the robot over the terrain while it is not distinguished between driving and stepping locomotion:

- move the robot within a 20-neighborhood with fixed orientation (see Figure 5.4 a) and
- turn to the next discrete orientation with fixed position (see Figure 5.4 b).

An explicit environment representation which is the basis for cost computation is not required. Instead, a CNN directly maps local planning tasks with the detailed environment representation to cost estimates. Details on the network are given in Section 5.4.

The transformation from detailed robot states to abstract robot states ($\mathcal{T}_{d \mapsto a}$) is realized by neglecting the foot positions and matching the position and orientation to the coarse resolution of the abstract state space.

The transformation from an abstract robot state to the corresponding detailed state ($\mathcal{T}_{a \mapsto d}$) is more complicated. For all detailed robot base states that match the abstract state, the one with the least cost foot configuration is searched. To obtain a behavior that matches the resulting paths in the detailed representation, a cost term is added to prefer foot positions which are close to the neutral configuration (see Figure 5.4 a). The detailed robot state with the minimum state costs is the result of the transformation.

5.4 ABSTRACT COST NETWORK

The cost function of the abstract representation is represented by a CNN which is trained with planning results of the detailed representation to obtain *cost similarity*. Input to the network are a local planning task consisting of a map patch of the detailed environment representation and the goal state for a local planning task which is completely in the map patch. Since the map patch is always robot centered, it is not required to explicitly input the start state of the local planning task. These local planning tasks represent \mathcal{A}_a . The network maps those inputs to a cost estimate for this local planning task. Section 5.4.1 gives details on the developed CNN architecture while Section 5.4.2 explains the network training. Section 5.4.3 describes the generation and usage of a heuristic which is based on the abstract representation.

5.4.1 Network Architecture

The developed CNN follows a regular feed-forward architecture consisting of convolutional layers and successive fully connected layers. Figure 5.5 depicts the network architecture. The input height map patches have a size of 72×72 pixels. For a given resolution of 2.5 cm, this size ensures that for every combination of abstract start and goal states $\vec{r}_{a,g} = \vec{r}_{a,s} + \vec{a}_{a,j} \in \mathcal{A}_a$, the corresponding detailed goal states $\vec{r}_{d,s} = \mathcal{T}_{a \mapsto d}(\vec{r}_{a,s})$ and $\vec{r}_{d,g} = \mathcal{T}_{a \mapsto d}(\vec{r}_{a,g})$ with any feasible leg configuration are completely inside this map patch. The height map is normalized such that the lowest contained value is zero.

The start state $\vec{r}_{a,s}$ is defined to be always in the map patch center with a fixed orientation pointing to the right of the image. Hence, it

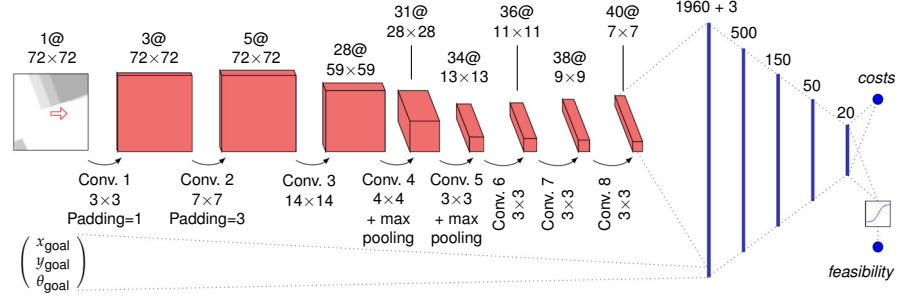


Figure 5.5: Architecture of the developed CNN. Input are a height map patch and the goal state. Although it is not fed into the network, the start state is depicted as a red arrow for better understanding. Outputs are the *feasibility* and *costs* values. Convolutional layers are visualized as red cuboids; blue lines show fully connected layers. If not stated different, convolutions have a padding of 0 and a stride of 1. Notation for maps of size $x \times x$ with y feature channels: $y @ x \times x$.

is not fed into the network. The 3-DoF goal state $\vec{r}_{a,g}$ is described in resolution steps relative to $\vec{r}_{a,s}$ (e.g., to move 20 cm in x -direction and -10 cm in y -direction with constant orientation, $\vec{r}_{a,g} = (2, -1, 0)$).

The network outputs cost estimates for planning tasks. In general, costs are assigned infinite, if no path can be found. However, infinite values come along with undesired effects during backpropagation. Instead, we split the outputs between *feasibility* and *costs*. The Boolean *feasibility* value describes, if a path between $\vec{r}_{a,s}$ and $\vec{r}_{a,g}$ exists. Only if this is the case, the *costs* value describes the corresponding costs. For a false *feasibility*, the *costs* loss is not backpropagated.

We discovered that key to a good abstraction performance are some convolutions with large kernel sizes. This might be explained as follows: a first convolution with a small kernel size (3×3) extracts descriptive map features from the input height map patch. Next, a second convolution possesses a kernel size (7×7) which is similar to the size of a robot foot and, thus, detects feasible footholds. The kernel size of the third convolution (14×14) is chosen such that it covers the maximum action length of an individual foot. Hence, it is capable of finding feasible connections between footholds which is, e.g., valuable for stepping actions. The subsequent convolutions and max pooling operations with small kernels (3×3 and 4×4) do further processing on the actions and are followed by six fully connected layers.

The last fully connected layer is split to obtain the desired outputs. While *costs* are output directly, the *feasibility* output is processed by a sigmoid function since it is Boolean.

5.4.2 Training

The network is trained on generated artificial data. On the one hand, this has the advantage that large amounts of data can be generated without considerable effort. On the other hand, the generation of such data is challenging since it has to imitate real-world data in sufficient quality such that the network generalizes well when inferred on real-world data. We developed the following method for data generation. A map generator produces height maps of the desired network input size. The following objects are placed at a random position with a random yaw orientation in those height maps:

- cuboid shaped obstacles of random size ($l_x, l_y \in [0.2 \text{ m}, 2 \text{ m}], l_z \in [0.02 \text{ m}, 0.4 \text{ m}]$),
- walls of random length and height ($l_x \in [0.4 \text{ m}, 3.0 \text{ m}], l_y \in [0.05 \text{ m}, 0.35 \text{ m}], l_z \in [0.7 \text{ m}, 2.0 \text{ m}]$), and
- staircases of random width ($s_y \in [0.9 \text{ m}, 2.0 \text{ m}]$) with a random number of steps ($s_{\#} \in [2, 4]$) with random height ($s_z \in [0.15 \text{ m}, 0.3 \text{ m}]$) and length ($s_x \in [0.25 \text{ m}, 0.35 \text{ m}]$).

As a training data set, we generate 2000 height maps of each of the following categories:

- one cuboid obstacle,
- two cuboid obstacles,
- three cuboid obstacles,
- one wall,
- two walls,
- one cuboid obstacle and one wall,
- one staircase,
- one staircase and one wall, and
- one staircase whose orientation is in the interval $[-\frac{\pi}{16}, \frac{\pi}{16}]$ around the robot orientation. Those maps are used to set a learning focus on stair climbing whose cost function is significantly more complex than driving locomotion.

Based on \mathcal{A}_a , we define 22 abstract goal states $\vec{r}_{a,gi}$ for each map. The start state $\vec{r}_{a,s}$ is always in the map center with a fixed orientation heading to the right. Hence, 22 planning tasks are available for each map. Subsequently, $\vec{r}_{a,s}$ and $\vec{r}_{a,gi}$ are transformed to \mathcal{R}_d using $\mathcal{T}_{a \mapsto d}$. If the transformation cannot find a corresponding detailed start state $\vec{r}_{d,s}$, those maps are deleted from the data set. This results in a total of 11,327 maps with 249,194 tasks. For each planning task from $\vec{r}_{d,s}$ to $\vec{r}_{d,gi}$, we search for the shortest path using the detailed A* planner (see Chapter 3 for details). We store the *feasibility* flag for each result which

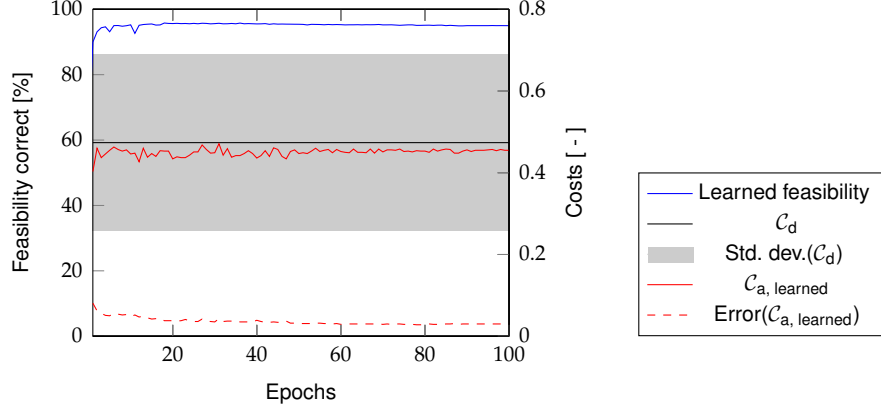


Figure 5.6: CNN training performance on the validation set after each training epoch. The detailed cost function C_d is shown as ground truth. The stated error describes the cost difference between the detailed and the abstract cost functions.

describes if a path could be found. *Costs* are saved for all feasible tasks. These two values form the desired outputs for the network training while the height map and the goal state form the input.

We use the Stochastic Gradient Descent (SGD) optimizer for network training. The learning rate is set to 0.0001 and a momentum of 0.9 is chosen. We use a Binary Cross Entropy (BCE) loss function for the *feasibility* and a Least Absolute Deviations (L1) loss function for the *costs*. The *costs* loss is only considered during backpropagation for true *feasibility* flags. Losses are weighted with $\mathcal{W}_{\text{feasible}}$ and $\mathcal{W}_{\text{costs}}$, both starting at 1.0. A dynamic loss weight adaptation procedure is applied to both losses individually: if no improvement by means of a decreasing loss is achieved in three consecutive training epochs, the corresponding loss weight is divided by 5.0.

To transform the network’s *feasibility* output to Boolean during evaluation, a threshold of 0.5 is used. The training performance is evaluated on a separate validation set which is obtained by repeating the above-explained map generation procedure with 100 maps of each category. This results in a validation set with 567 maps with 12,474 planning tasks. The training performance is depicted in Figure 5.6. We initialized the CNN with random seed, trained for 100 epochs and chose the network state for the experiments which obtained the best results on the validation set.

5.4.3 Abstract Representation-based Heuristic

The abstract representation is used to generate an informed heuristic for planning in the detailed representation. For a given goal state $\vec{r}_{d,g}$, this is transformed to the abstract representation using $\vec{r}_{a,g} = \mathcal{T}_{d \rightarrow a}(\vec{r}_{d,g})$. Next, a one-to-any 3D Dijkstra search is started from $\vec{r}_{a,g}$ and explores the whole map in the abstract representation.

During this search, neighbor states \vec{r}_{a,n_i} for a state $\vec{r}_{a,m}$ are generated through abstract actions $\vec{a}_{a,i}$ such that $\vec{r}_{a,n_i} + \vec{a}_{a,i} = \vec{r}_{a,m}$. The corresponding action costs are retrieved from the CNN. Since the search is running backwards, from the goal state to arbitrary states, start and goal of each action are exchanged. Height map patches are cropped such that the action start state is in the center. In addition, they are rotated such that the action start state is heading to the right. Finally, the height map patch and the goal state are fed into the CNN. Those actions for which the CNN outputs a false *feasibility* are discarded. Feasible actions are assigned the corresponding *costs* estimate.

The computation of action costs between neighbor states only needs to be done once per map. The 3D Dijkstra search needs to be executed once per goal state. However, due to the low dimensionality and low resolution of the abstract representation, this search is fast.

During path planning in the detailed representation, this heuristic is employed as follows: To obtain a heuristic value from an arbitrary detailed state $\vec{r}_{d,i}$ to the goal state, the precomputed heuristic value for $\vec{r}_{a,i} = \mathcal{T}_{d \mapsto a}(\vec{r}_{d,i})$ is read describing a cost estimate from this state to the goal state. The method is similar to the method described in Section 4.4 and is visualized in Figure 4.15.

In theory, a heuristic is only admissible for the generation of optimal paths if it is proven to always underestimate costs. It is important to understand that we cannot prove this admissibility. We rather focus on the generation of paths with a satisfying quality in feasible time and thus, accept sub-optimality to speedup planning. However, the training performance in Figure 5.6 already indicates that the learned cost function tends to underestimate costs which is desirable.

5.5 EVALUATION

We used Python 2.7 and PyTorch 0.4.1 to implement the CNN. Map generation and the planner were implemented in C++. Communication between these modules was realized via ROS. An online repository¹ contains code for the training data generator, the CNN, and the generation and usage of the corresponding heuristic.

The approach was evaluated in two experiments. A first experiment (see Section 5.5.1) evaluated the abstraction quality and enables a comparison to the previously presented manually designed and tuned abstraction method in Chapter 4. A second experiment evaluated the heuristic when planning hybrid driving-stepping locomotion paths in challenging environments and enables comparison to the performance of traditionally used heuristics. A video which explains the approach and gives additional footage of the experiments is available online².

¹<https://doi.org/10.5281/zenodo.3628727>

²<https://doi.org/10.5281/zenodo.3628732>

5.5.1 Abstraction Quality

The abstraction quality was evaluated on three data sets:

- *random*: we generated 200 random maps of each category (see Section 5.4.2) resulting in a set of 1,124 maps with 24,728 tasks.
- *simulated*: height map patches of the desired size were cut out from height maps of simulated generated planning environments. This set includes 77 maps with 1,694 tasks.
- *real*: height map patches of the desired size were cut out from height maps that were generated from laser scanner measurements during real-world experiments. This set includes 109 maps with 2,398 tasks.

Figure 5.7 shows example tasks of the different data sets. The abstraction performance of the developed CNN is shown in Table 5.1. We evaluate the *feasibility* and *costs* outputs. A correct *feasibility* assessment means that the abstract representation outputs the same *feasibility* value as the detailed representation. Only if both representations assess a situation as feasible, *costs* are considered and give an evaluation of the *cost similarity* of the two representations. *Costs* for the tasks in the detailed representation \mathcal{C}_d are stated as ground truth. We compared the performance to the manually tuned *Level 3* abstraction method, described in Chapter 4.

The results indicate that the CNN *feasibility* output is significantly better compared to the manually tuned abstraction method. While the latter has problems in simulated and real-world robot environments, the CNN assesses a correct *feasibility* for $> 92\%$ of the tasks throughout all test sets.

Regarding the *costs* estimates, the average *costs* error of the CNN is smaller compared to the manually tuned abstraction on all test sets. The error is particular small, when seen in relation to the large standard deviation of the ground truth costs of the detailed representation. While the error of the proposed CNN is $< 5.7\%$ of the absolute *costs* on the *random* and *simulated* test sets, it is 15.9% on the *real* test

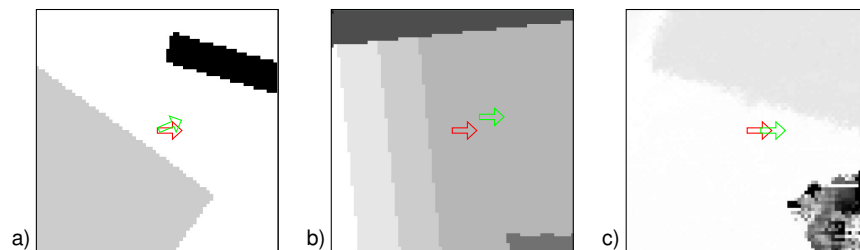


Figure 5.7: Example tasks of the test data sets. *a)* *random* test set. *b)* *simulated* test set. *c)* *real* test set. Images show the input height maps. Red arrows show start states (not fed into the CNN), green arrows show goal states.

Table 5.1: Abstraction quality on the three data sets. The costs in the detailed representation (C_d) state the ground truth. $C_{a,CNN}$ describes the learned abstract cost function of this presented method while $C_{a,man.tuned}$ describes the manually designed and tuned abstract *Level 3* cost function from Chapter 4.

	<i>random</i>	<i>simulated</i>	<i>real</i>
C_d	0.476	0.466	0.509
Std. dev. (C_d)	0.222	0.202	0.236
<i>feasibility</i> correct, CNN	95.04%	96.69%	92.62%
$C_{a,CNN}$	0.453	0.469	0.446
Error($C_{a,CNN}$)	0.027	0.013	0.081
<i>feasibility</i> correct, man.tuned	79.27%	65.35%	69.77%
$C_{a,man.tuned}$	0.435	0.402	0.429
Error($C_{a,man.tuned}$)	0.057	0.021	0.103

set. This might be explained by noisier sensor measurements in real-world experiments resulting in noisier height maps which can also be observed in Figure 5.7.

5.5.2 Application to Planning

We designed a 10×10 m arena in the Gazebo simulation which includes typical locomotion tasks for Centauro in search and rescue missions (Figure 5.8). The used system is equipped with an Intel Core i7-8700K processor at 3.70 GHz, 64 GB RAM and an NVidia GeForce GTX 1080Ti with 11 GB memory.

For all abstract states of the whole map, map patches were extracted and neighbor states with corresponding action costs were precomputed by the CNN. This took 239 s. However, this precomputation is only required once per map and can be incrementally updated if parts of the map change. The one-to-any Dijkstra search starting from the defined goal state and generating the heuristic took 0.049 s, averaged over all goal states. We used the weighted-A* planner to plan paths to all goals. It planned in the detailed representation while utilizing the *abstract representation-based heuristic*. The planning performance was compared to planning with the *geometric heuristic* which combines Euclidean distances with rotational differences and is admissible (more details in Section 3.3.4). Hence, when used with a heuristic weight $\mathcal{W} = 1$, results are optimal. Both heuristics were evaluated with multiple $\mathcal{W} \geq 1$ to also obtain fast, sub-optimal solutions.

Figure 5.9 shows the planner performance for both heuristics and different \mathcal{W} . Table 5.2 summarizes the resulting speedup and cost increase compared to the optimal solution. The results indicate that the

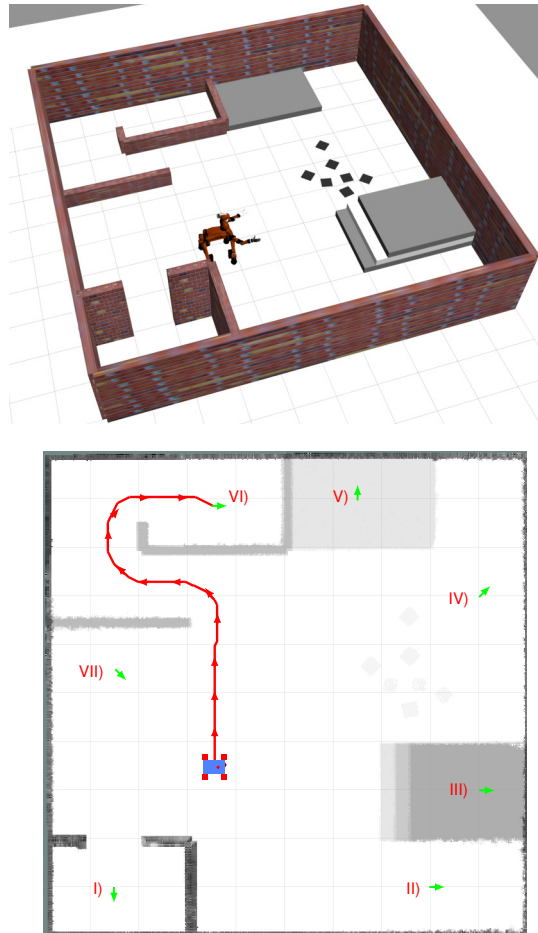


Figure 5.8: Locomotion planning experiment. *Top)* Gazebo arena with Centauro. *Bottom)* Height map with start state (blue/red) and goals (green arrows): I) Behind a narrow door opening. II) Next to stairs. III) On top of stairs. IV) Behind some debris. V) On a platform. VI) Inside a labyrinth. VII) Behind the robot. The red path is the resulting path to VI) with the developed *abstract representation-based heuristic* and $\mathcal{W} = 1.25$.

presented *abstract representation-based heuristic* accelerates planning by multiple orders of magnitude compared to the *geometric heuristic* while, in particular for $\mathcal{W} = 1.25$, path costs stay comparable. Especially for challenging tasks such as the stairs (III) and the labyrinth (VI), our heuristic was mandatory to obtain a solution in feasible time. This can be explained by the fact that the *geometric heuristic* has no information about the environment and thus the planner visits many states before considering expensive actions. In contrast, the presented *abstract representation-based heuristic* uses its cost estimates to guide the planner towards the goal while including knowledge about the environment.

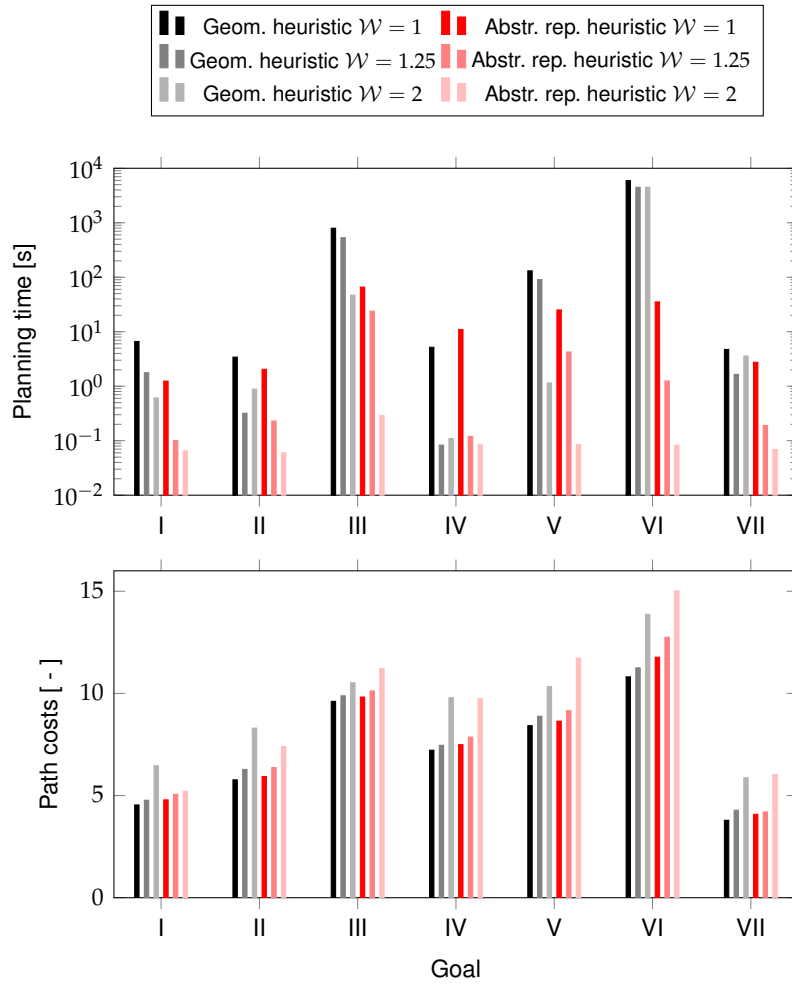


Figure 5.9: Planning times (including 3D Dijkstra search for heuristic generation) and path costs for all goal states and for both the *abstract representation-based heuristic* and the *geometric heuristic*.

Table 5.2: Planner performance for the *abstract representation-based heuristic* and the *geometric heuristic* averaged over planning queries to all seven goal states.

\mathcal{W}	Abstract representation			Geometric		
	1.0	1.25	2.0	1.0	1.25	2.0
speedup factor	27.80	708.5	10,860	1.0	12.00	27.88
costs increase	+4.77%	+10.5%	+33.1%	optimal	+6.07%	+33.9%

5.6 CONCLUSION

In this chapter, it is presented how a Convolutional Neural Network (CNN) can be employed to represent the cost function of an abstract planning representation. The CNN maps a local planning task, consisting of a map patch and local start and goal states on this map patch, to a costs estimate for the corresponding shortest path. This can be used to precompute a heuristic map of the whole environment whose values can be efficiently accessed during planning in a detailed representation.

In comparison to the previously presented approach with a manually designed and manually tuned abstract representation (Chapter 4), the utilization of a CNN minimizes those design and tuning efforts. Experiments showed that the learned cost function furthermore outperformed the manually designed version in all measures. While it was trained on generated artificial data, it generalized well to real-world data.

Moreover, the method was applied to the high-dimensional problem of hybrid driving-stepping locomotion planning. Planning in the detailed representation—when being guided by the presented informed heuristic—was faster by multiple orders of magnitude with comparable result quality compared to a previously used geometry-based heuristic. The presented heuristic was superior since it incorporates information about the environment, which has a significant influence on the planning outcomes in the considered domain.

VALUE ITERATION NETWORKS ON MULTIPLE LEVELS OF ABSTRACTION

Chapter 5 demonstrated how a learning-based method can be beneficial to support a traditional planning approach through an increased environment “understanding” in terms of an informed heuristic. Subsequently, it is an interesting idea to investigate how an even higher level of problem “understanding” affects the planning. This can be realized by solving the whole planning problem through a learning-based approach. In contrast to traditional planning approaches, this is promising since corresponding methods enable planning without extensive, iterative searches and can be parallelized efficiently on, e.g., GPUs.

An overview of the state of the art of motion planning with learning-based methods is given in Section 2.2.4. A promising work are Value Iteration Networks (VINs, Tamar et al. (2016)), which approximate the Value Iteration algorithm by rewriting it as a CNN. This enables trainability through backpropagation and massive parallelization on GPUs. In contrast to standard feed-forward CNN architectures, VINs show good planning performance in terms of goal-directed behavior and the generalization to unseen domains. However, similar to all learning-based planners, VINs can only handle small and low-dimensional planning problems with small state spaces which limits their applicability to real-world motion planning problems. Internally, VINs represent each state as one cell of a multi-dimensional grid and compute a reward and state-value for each of these grid cells. For large and high-dimensional grids, this leads to large computation graphs for the gradients during backpropagation, resulting in large amounts of required training data, long training times, and high memory consumption. Thus, VINs were so far only evaluated on 2D grid worlds up to a size of 28×28 cells. Since the number of manageable states is limited, one has to modify what each state represents to apply the method to larger planning problems. Extensions propose hierarchical architectures (HVINs, see Section 2.3) which can be summarized as a multiresolution, coarse-to-fine planning approach. This makes VINs applicable to larger state spaces, but comes at the cost of data loss through subsampling, which restricts the application to planning environments of limited complexity.

Inspired by the results of Chapter 4 and Chapter 5, we developed an extension of Value Iteration Networks on multiple levels of abstraction (AVINs) (see Figure 6.1). In other domains, especially in perceptual contexts, hierarchical convolutional neural networks (CNNs)

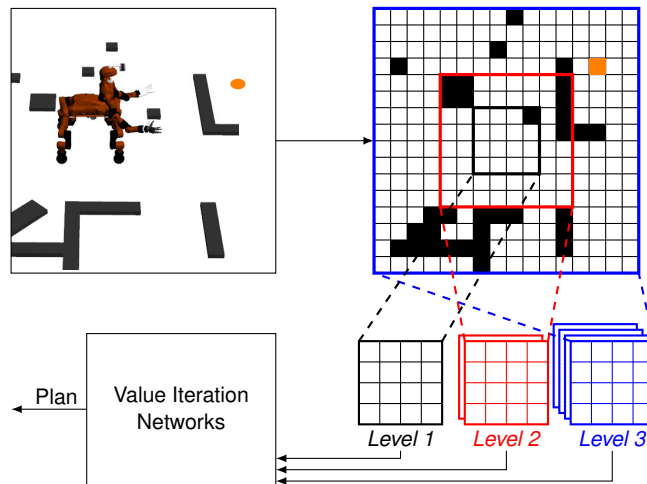


Figure 6.1: The general idea of Value Iteration Networks on multiple levels of abstraction (AVINs).

are highly successful, because they learn increasingly abstract representations of their input by decreasing the resolution and increasing the number of feature maps. This idea is applied to VINs. While the vicinity of the robot is represented in sufficient detail, the representation gets spatially coarser with increasing distance from the robot. The information loss caused by the decreasing resolution is compensated by increasing the number of features representing a cell. Those additional representation levels are integrated in the network architecture. Since they are fully differentiable, the whole network can be trained using backpropagation. Hence, a manual representation design is not required any more, but representations are learned by the network.

Section 6.1 gives a system overview while Section 6.2 describes the method in detail. In Section 6.3, the method is evaluated against original VINs and HVINs. In addition, it is applied to plan 3D omnidirectional driving with footprint consideration in cluttered environments for the Centauro robot. Finally, Section 6.4 draws a conclusion.

6.1 SYSTEM OVERVIEW

Input to the network is an occupancy map of the environment and a goal map of equal size. Both maps are always egocentric, which removes the need for explicit information about the start state—a difference to original VINs. Since the resolution of a representation level is halved to obtain the next more abstract level, the maps need to have a suitable number of cells. For two abstraction levels, the edge length of the input maps needs to be a multiple of two, for three levels, a multiple of four, for four levels, a multiple of eight, ... If no information is available for some regions of the input environment map since those regions are, e.g., not included in the original

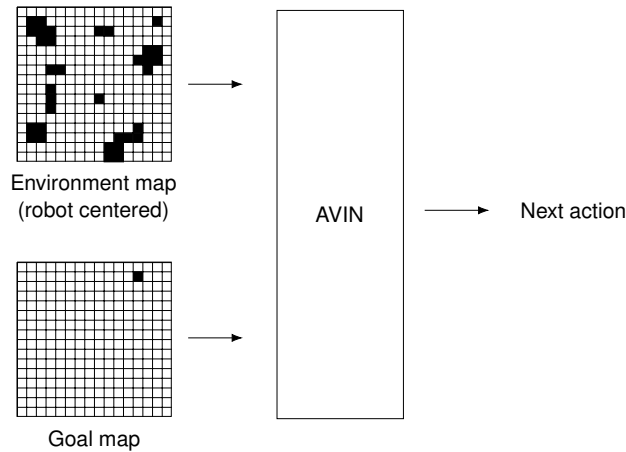


Figure 6.2: Input and output specification for AVINs.

planning environment, the corresponding cells of the input environment map can be marked as occupied to avoid the planner from considering those regions for planning. The goal map represents the same area as the environment map and is realized as a one-hot-map containing only zeros except for the goal cell.

The network outputs action probabilities for the whole action set. The action with the highest probability is chosen to be the next action. To obtain a path for solving a planning problem, we iteratively let the network predict the next action and shift the input maps according to the new robot position. Consequently, the network performance is evaluated in two measures. The *accuracy* describes if the network outputs the correct next action, while the *success* describes if the presented planner was able to find a feasible path from the start to the goal state. The input and output are visualized in Figure 6.2.

6.2 METHOD

The method combines multiple representations with increasing level of abstraction with VINs to obtain a learning-based planner that is capable of handling large and complex state spaces. With increasing distance from the robot, the level of abstraction increases. A more abstract representation has a spatially coarser resolution while the number of cells is constant for all levels. This results in larger covered areas for more abstract representations. Moreover, an increasing level of abstraction comes along with an increasing number of descriptive features for each cell.

As an example, we define three levels of abstraction. *Level 1* has the original input resolution but only covers the vicinity of the robot. For *Level 2*, the resolution is halved resulting in a covered area that is four times larger. This is repeated to obtain the *Level 3* representa-

tion. Thus, the corresponding covered area is 16 times larger than the *Level 1* area.

Additional features are introduced for each abstract cell to compensate the information loss caused by the coarsening operation. Those features are learned by the network during training. Experiments showed that one, two, and six features for *Level 1*, *Level 2*, and *Level 3*, respectively, achieved best results.

In this section, the general network architecture (see Section 6.2.1), the application to different planning problems (see Section 6.2.2), and the corresponding training procedures (see Section 6.2.3) are described.

6.2.1 Network Architecture

Input to the network is an occupancy map of the environment and an equally sized goal map. Both maps possess a fine resolution which is equal to the *Level 1* resolution. Each cell is described by a single feature. To facilitate understanding, the network architecture is described for three abstraction levels and 2D planning problems. Modifications to extend the number of abstraction levels or cope with planning problems of higher dimensions are described in later sections. An overview of the network architecture is given in Figure 6.3.

In a first step, the **Abstraction Module** processes the input maps to representations in the three abstraction levels. This module is not

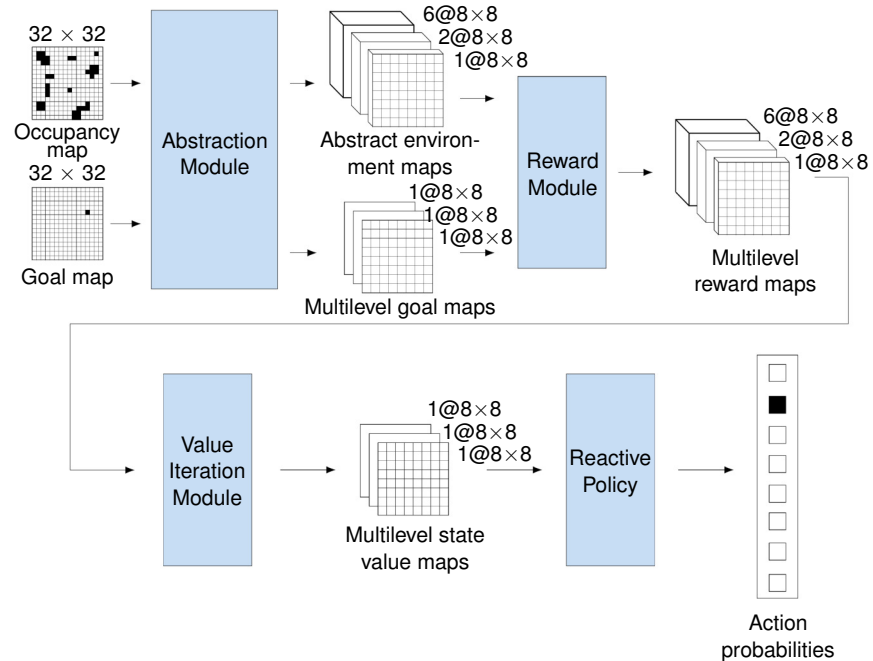


Figure 6.3: Network architecture. Depicted map sizes correspond to 32×32 input maps and 2D planning. Notation for maps of size $x \times x$ with y feature channels: $y @ x \times x$.

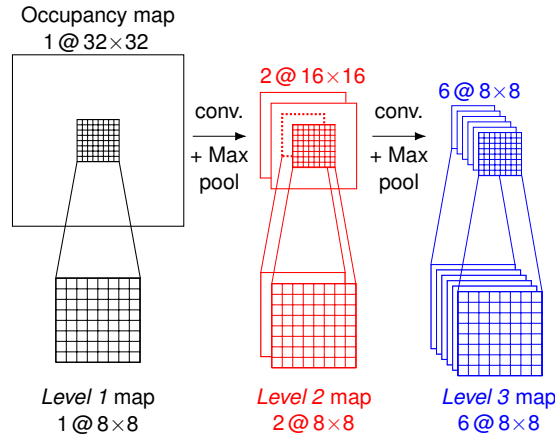


Figure 6.4: Abstraction Module. Both convolutions use kernels of size 3×3 followed by a 2×2 max pooling operation. The goal map is processed using the same max pooling operations without prior convolutions. Depicted map sizes correspond to 32×32 input maps.

present in the original VIN implementation. The detailed process is depicted in Figure 6.4. Regarding the environment maps, the *Level 1* map is extracted as a patch around the map center. The *Level 2* representation with halved resolution and two features per cell is generated from the input map through a convolution and subsequent max pooling operation. The *Level 2* map is again extracted from the map center, while the whole *Level 2* representation is processed with another convolution and max pooling operation to obtain the *Level 3* map. The goal map is processed similarly using max pooling operations without convolutions. Outputs of the Abstraction Module are three equally sized environment and goal maps representing the different levels of abstraction.

Subsequently, the abstract environment maps and goal maps are fed into the **Reward Module**. It generates rewards for each state. The architecture is shown in Figure 6.5. Similar to the original VIN implementation, the environment and goal maps are stacked, a convolution extracts 150 features and a second convolution generates the reward map. Padding keeps the map size constant and ensures that the relation between cell position and environment location stays fixed. Moreover, similar to the environment maps, the reward maps of higher abstraction levels require multiple features to compensate information loss due to coarser resolutions. For example, it might be possible that an abstract map cell can be entered from one direction but not from another, which might be encoded in the features. However, the information encoded in the real features is not manually defined but learned by the network.

Furthermore, we enable information flow from detailed levels to more abstract levels. This is desirable since, e.g., when computing rewards for *Level 2*, there is more precise information available for the center area of this maps, encoded in the *Level 1* map. At this point, it

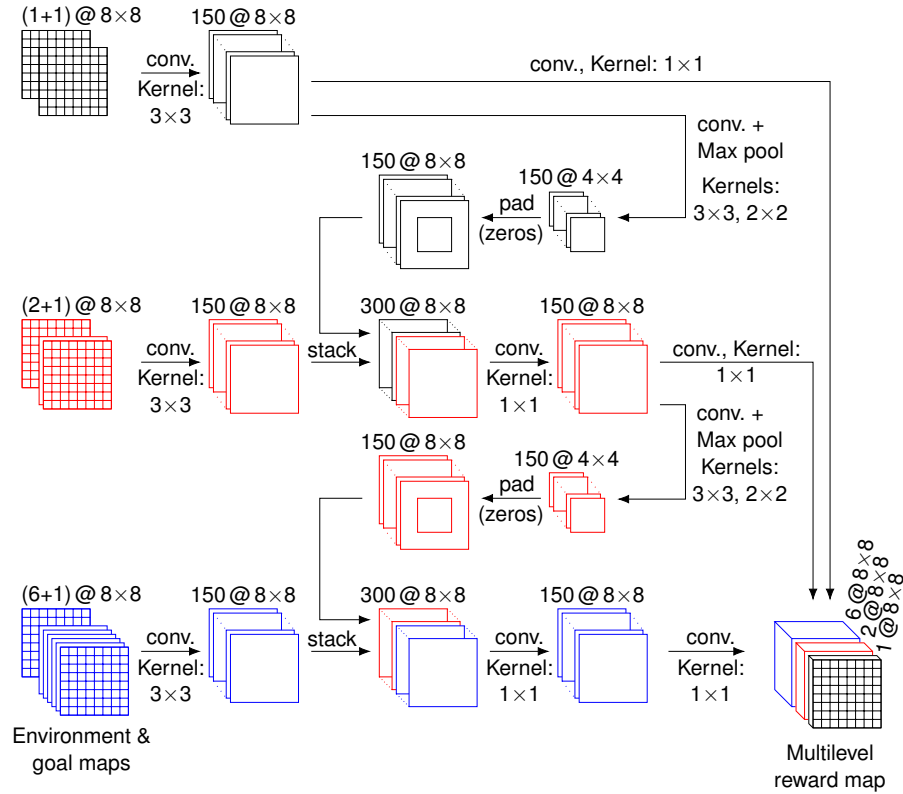


Figure 6.5: Reward Module. *Level 1* maps are shown in black, red parts belong to *Level 2*, and blue parts to *Level 3*. Depicted map sizes correspond to 32×32 input maps and 2D planning.

is important to understand that information encoded at the same cell position of different levels refers to different locations in the environment. We support the network in “understanding” this relation with the following method: The resolution of the 150 extracted *Level 1* feature maps is halved to match the *Level 2* resolution through a convolution and subsequent max pooling operation. The result is padded with zeros to match the size of the *Level 2* map. This procedure ensures that, when the *Level 1* feature maps are stacked with the *Level 2* feature maps, the relation between cell position and environment location is constant throughout all maps. Finally, a convolution is used to merge the stacked *Level 1* and *Level 2* features. This procedure is repeated to enable information flow from *Level 2* to *Level 3*.

Outputs of the Reward Module are multilevel reward maps of equal size possessing the same numbers of features as the environment maps.

Reward maps are input to the **Value Iteration Module** where they are processed to state-value maps. The module for one level is depicted in Figure 6.6. State values are iteratively propagated through the map, similar to the original VI algorithm. Each iteration of the rewritten Bellman equation, which we call Bellman update, is real-

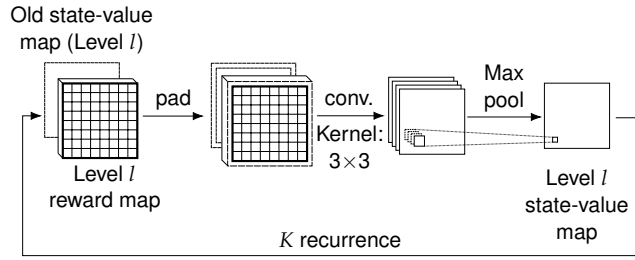


Figure 6.6: Value Iteration Module. The depicted operation is performed for each level individually. The padding operation (see Figure 6.7) enables information flow between levels.

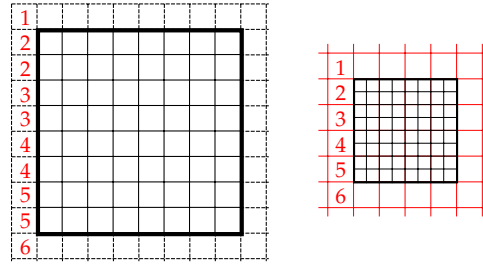


Figure 6.7: Padding the map of abstraction level l (left) to allow information flow from the map of level $l+1$ (right) to level l . The numbers indicate which values are copied where.

ized by a convolution and subsequent max pooling operation. The kernel size is chosen such that it covers the set of possible actions. Unlike the reward maps, state-value maps encode the expected long-term reward of a state, which can be encoded in a single feature.

In order to make correct long-term decisions considering the structure of the whole environment, information flow from higher to lower abstraction levels is required. We realize this information flow through the following padding method: Within each iteration, the reward maps and old state-value maps are padded with values of the neighboring cells of the next higher level, as visualized in Figure 6.7. Since the reward maps vary in their number of features, a mapping from the higher-level features of an abstract cell to the lower-level features of the corresponding less abstract cell is required. We found that the best result quality was achieved by using the average over all features of one higher level-cell as the padding value for all corresponding lower level-cells. Interestingly, learning a mapping from higher-level to lower-level features with fully connected layers performed worse. The number of recurrence iterations K for level maps of a given size is defined to be 1.5 larger than the K value proposed in the VIN implementation for the identical map size. The additional iterations enable information flow steps between levels. Nevertheless, due to the multiresolution characteristics, significantly less iterations are needed for the presented method compared to VINs, as can be seen in Table 6.1.

Table 6.1: Number of iterations depending on the input map size. E.g., for an input map size of 64×64 , all three AVINs levels possess a map size of 16×16 . Hence, the number of iterations is 1.5 times as high as for VINs with this map size.

	Input map size			
	16×16	32×32	64×64	128×128
VINs	20	40	80	160
AVINs (3 Level)	7	15	30	60
AVINs (4 Level)	-	-	-	30

Finally, for all neighbors of the start state, their state-values are mapped to probabilities over actions through a **Reactive Policy**, which simply is a fully connected layer.

6.2.2 Application

The presented method is applied to two different planning problems. Path planning in 2D grid worlds is valuable to compare the planning performance against VINs and HVINs which were designed and evaluated in this task. However, real-world planning problems usually possess more complex planning representations. Hence, we additionally apply AVINs to plan omnidirectional driving locomotion of a robot while considering its footprint, which is significantly more complex than 2D grid worlds. We intended to apply the method to even more complex problems, but, as can be seen in the experiment section, the method reached its limitations on currently available hardware.

The network is implemented using Python 2.7 and PyTorch 0.4.1. Corresponding source code is available online¹. The following subsections describe modifications and the application of AVINs to the two planning domains.

6.2.2.1 2D Grid Worlds

The planner is given queries for a point-like agent in 2D grid worlds. As actions, the agent can move to one of the eight adjacent neighbor cells, which is depicted in Figure 6.8 (a). The agent's orientation is not considered. For this domain, the network architecture, as described in Section 6.2.1, can be used without modification. However, for large maps, we introduce a fourth abstraction level to the network. Its integration follows the already described principles while *Level 4* employs ten features to describe the environment and reward maps.

¹<https://doi.org/10.5281/zenodo.3628729>

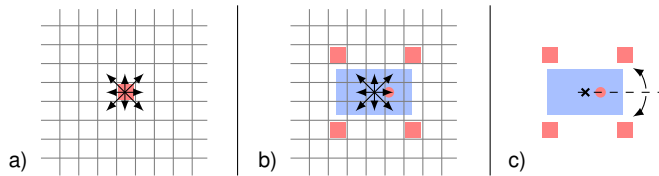


Figure 6.8: Possible actions for both planning domains. *a)* Move to an adjacent neighbor cell in 2D grid worlds. *b)* Omnidirectionally drive to an adjacent neighbor state with fixed orientation in 3D robot locomotion planning. *c)* Turn to the next discrete orientation with fixed position in 3D robot locomotion planning.

6.2.2.2 3D Robot Locomotion

Given is a robot that can perform omnidirectional driving and has a fixed footprint which shall be considered by the planner. This planning problem requires multiple modifications to the network architecture. The orientation introduces an additional DoF to the two-dimensional position state space. Furthermore, the robot footprint needs to be taught to the planner such that only the ground contact areas of the individual feet are considered. This enables the robot to, e.g., take obstacles between its legs. At this point, the robot footprint is fixed. Therefore, possible actions are:

- move the robot center to one of the eight adjacent neighbor states with fixed orientation and
- turn the robot to the next discrete orientation with fixed position.

The actions are also visualized in Figure 6.8 (b) and (c).

To enable the network to handle 3D state spaces, reward and value maps are extended by an additional dimension for the orientation. The environment maps stay two-dimensional. In preliminary experiments, we tried architectures which also possessed individual environment maps for each robot orientation but those performed worse. Similar to the spatial resolution, the orientation resolution decreases for higher levels of abstraction. We represent the robot orientation for *Level 1* in 16, for *Level 2* in eight, and for *Level 3* in four discrete orientations of equal angular distance. Due to the increased complexity of the states, we increase the number of features for *Level 2* to five and for *Level 3* to ten.

In the Reward Module, we increase the number of convolutions to extract the descriptive features from the environment maps. Two additional convolutions are used for the *Level 1* map and one additional convolution is used for the *Level 2* map. The reward computation is done on two-dimensional maps representing rewards to position individual robot feet in the environment. In order to introduce the robot footprint, reward maps are transformed correspondingly: For each robot base position, we sum over the four cells of the corresponding

foot positions and assign the result to the robot base center cell, as visualized in Figure 6.9. This operation is repeated for each robot orientation resulting in 16 *Level 1* reward maps with one feature per cell, eight *Level 2* reward maps with five features per cell, and four *Level 3* reward maps with ten features per cell.

Further adaptation is required in the Value Iteration Module. It needs to process the input reward maps of the described new size and to consider all actions of the three-dimensional action set. The latter is realized through a 3D convolution kernel, as can be seen in Figure 6.10. Since the neighborhood relation of the orientation is cyclic, we pad the reward maps and state-value maps on each end of the orientation channel with the values of the opposite end, as visualized in Figure 6.11.

In contrast to 2D planning, the planner needs information not only about the start and goal position but also about their orientation. The start state orientation is fed into the system as an additional parameter. It is only required within the Reactive Policy to select those state-

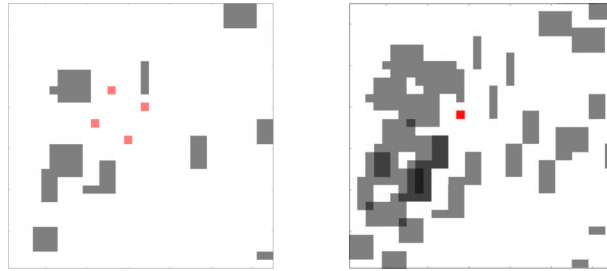


Figure 6.9: Robot footprint computation. *Left*: Two-dimensional reward map for individual foot positions and one example footprint (four red cells). *Right*: Rewards of four associated foot positioned are added together. The result is assigned to the robot base center cell (red cell for example footprint). The shown map displays the rewards for the depicted footprint orientation. This operation is repeated for each robot orientation.

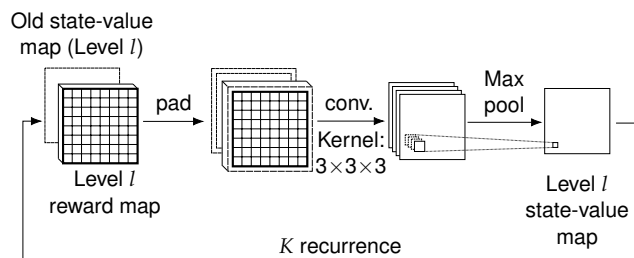


Figure 6.10: Value Iteration Module for 3D robot locomotion planning. The depicted operation is performed for each level individually. Information flow between levels is realized through padding operations at the map borders (see Figure 6.7) and at the orientation channel (see Figure 6.11).

values that belong to neighbor states of the start state. The goal state orientation is encoded in the goal map: The goal cell is assigned with the goal state’s orientation index (1-16) while all other cell entries are zero.

An overview of the resulting network architecture including all described modifications is given in Figure 6.12.

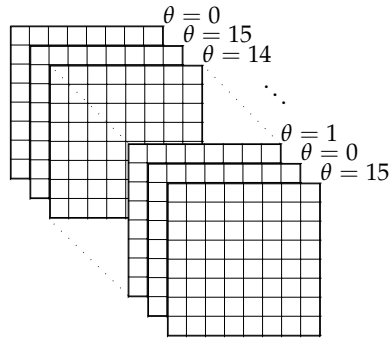


Figure 6.11: Orientation padding during 3D Value Iterations to emphasize that the orientations $\theta = 15$ and $\theta = 0$ are neighbors. The number of orientations refers to *Level 1*.

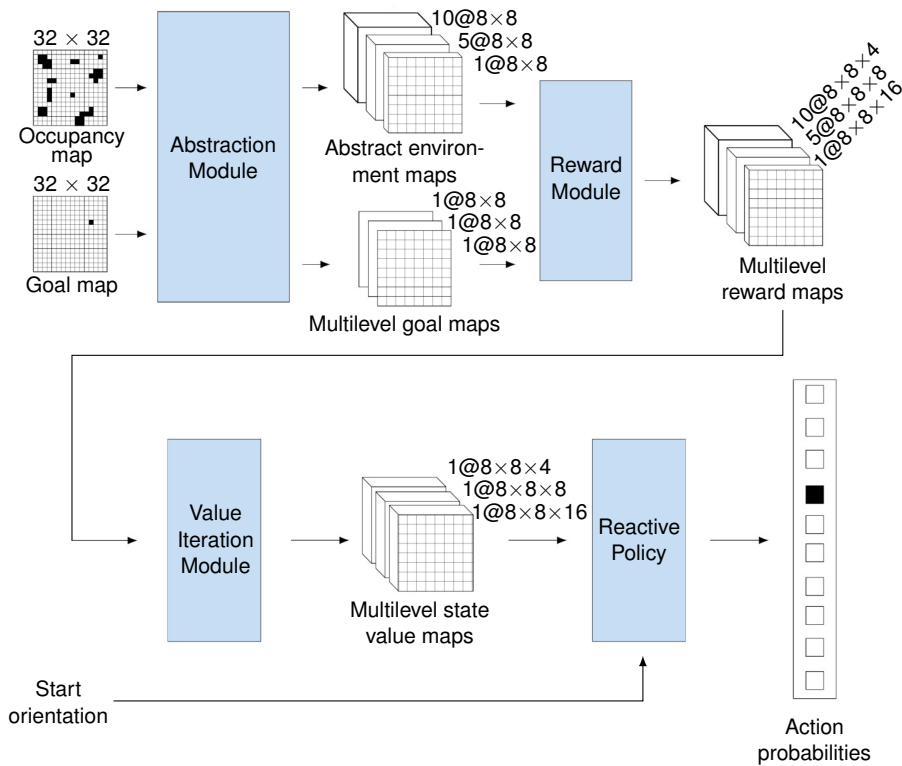


Figure 6.12: Network architecture for 3D robot locomotion planning. The depicted map sizes correspond to 32×32 input maps.

6.2.3 Training

The network is trained on generated artificial data. For 2D obstacle grid worlds, obstacles of random number and size are placed in occupancy grid maps of the desired size. We generated 5,000 of these maps. In addition, seven feasible goal states are placed randomly in each of these maps resulting in a training set with 35,000 tasks. Since similar data sets are used by Tamar et al. (2016), they offer a high comparability to the original VIN implementation. The same method is used to generate a training set with 35,000 tasks for 3D robot locomotion planning which also requires 2D environment maps as an input. To obtain a more challenging planning task, 2D maze grid worlds are generated. Again, 5,000 mazes are generated randomly, and seven feasible goal states are placed in each map. For all maps, the start state is defined to be in the map center. For 3D planning tasks, the start orientation is chosen randomly. Subsequently, we use an A* planner (see Section 2.2.1) as an expert to generate optimal paths. To increase data efficiency during training, we do not only use the whole expert paths but also sub-paths. During each training epoch, we randomly place the start and goal at some position on the expert path and only train for this sub-path.

We discovered that in the training data set, some actions were chosen more often than other actions. However, to support the training of all possible actions, we weighted the losses for the different actions by their inverse action frequencies.

Validation and test data sets for all domains are obtained from 715 additionally generated maps with seven planning tasks each, resulting in 5,005 different tasks in each data set.

In order to provide comparability to the original VIN publication, all networks are trained using the RMSprop optimizer, as proposed by Hinton et al. (2012). However, when using RMSprop without any further learning rate scheduling, the network occasionally converged to sub-optimal local minima and sometimes showed unstable training behavior. This effect was also reported by Lee et al. (2018). Therefore, we tried additional training procedures with the cyclic learning rate scheduler proposed by Loshchilov and Hutter (2016): During training, the learning rate decreases following a cosine annealing scheme. After several training epochs, the learning rate is reset following a slower decay schedule (see Figure 6.13 top right). We call the time between learning rate resets a learning rate cycle. Initially, the length of a learning rate cycle is set to 48 epochs and the learning rate is 0.001. After each cycle, the cycle length increases to 150% while the initial learning rate decreases to 95% of the previous one. Employing the cyclic learning rate scheduler results in a stabilized training performance.

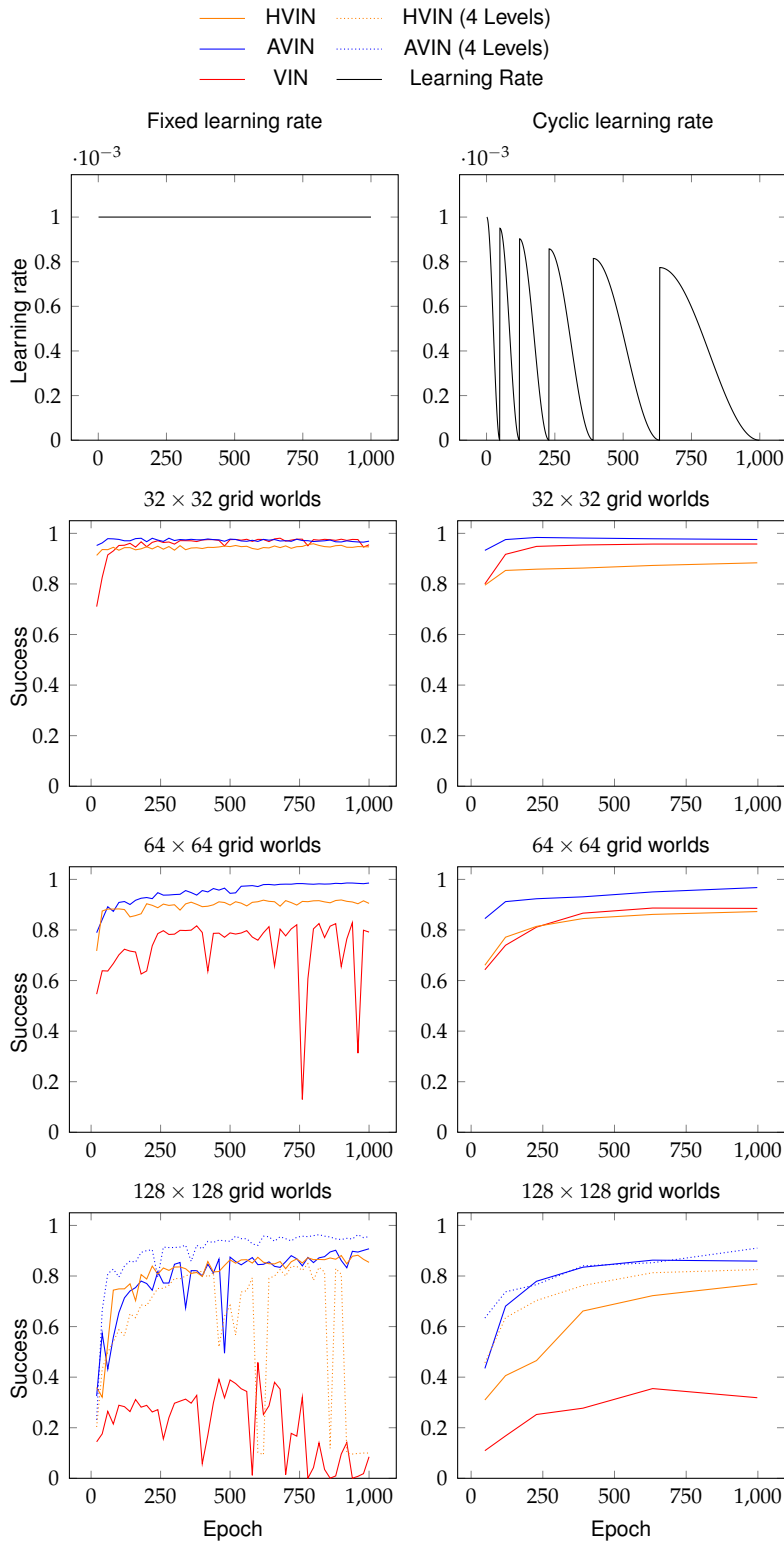


Figure 6.13: Training performance of VINs, HVINs, and AVINs on random obstacle grid worlds on validation sets. *Left*: Fixed learning rate. *Right*: Cyclic learning rate scheduling. While the fixed learning rate training was evaluated every 30 epochs, the training with cyclic learning rate scheduling was only evaluated at the end of each learning rate cycle.

The same figure also states the training performance on the 2D obstacle grid domain for VINs, HVINs, and AVINs. It can be seen that our method achieves slightly better results on the validation set with the cyclic learning rate scheduler. It can be further seen that, especially with a fixed learning rate, VINs show unstable training behavior for larger map sizes. The training on 128×128 maps indicates that HVINs and AVINs partially benefit from the introduction of a fourth level. A systematic evaluation and comparison is given in the following section.

6.3 EVALUATION

We evaluated our method in three experiments. In Section 6.3.1, we describe the application to 2D random obstacle grid world planning. Since this domain is similar to the experiments in the original publication of VINs and HVINs, a high comparability is obtained. A more challenging planning problem is presented through planning in 2D maze grid worlds, which is described in Section 6.3.2. Finally, we applied AVINs to plan 3D omnidirectional driving in cluttered terrain for the Centauro robot (see Section 6.3.3). All experiments were done on a system equipped with an Intel Core i7-8700K@3.70 GHz, 64 GB RAM, and an NVidia GeForce GTX 1080Ti with 11 GB memory. A video with additional experimental footage is available online².

We evaluated the network output quality in three measures. The *accuracy* describes whether the network output for the next action is identical to the next action computed by the expert A* planner. Regarding path planning, the *success* rate describes, if the network was able to find a collision-free path from the start state to the goal state. Furthermore, we assess the output path quality by computing the length difference relative to the optimal path length. We furthermore evaluated the hardware requirements of the networks. Besides the required GPU memory, we state training times for one training epoch providing an assessment of the network complexity. Stated planning times and memory consumption of our approach include input maps shifting on the CPU after each network inference to concatenate the next action network outputs to paths.

Regarding the *accuracy* measure, please note that in many cases there is more than one optimal next action. However, as in original VINs, the network is trained to output one next action which is compared to the output of the expert planner. Hence, there occur cases in which the output of the network is different from the output of the A* planner but the network still unrolls an optimal path although the *accuracy* measures a mistake. We also tested a version of AVINs which learns to output all optimal actions. However, this performed worse compared to the version presented here.

²<https://doi.org/10.5281/zenodo.3628732>

6.3.1 Path Planning in 2D Random Obstacle Grid Worlds

In a first experiment, AVINs were compared to VINs and HVINs on random obstacle grid worlds. We used a version with three levels for HVINs and AVINs, each level halving the resolution of the previous one. For HVINs, the coarsest level used the same number of Bellman updates K as proposed for VINs. This state-value initialization was then refined by two Bellman updates on the medium resolution map and two consecutive Bellman updates on the fine resolution map.

In the original publication, grid world sizes from 8×8 to 28×28 were considered. We performed tests on slightly larger maps with 32×32 , and significantly larger maps with 64×64 and 128×128 cells. For the largest map size, we tested an additional version of HVINs and AVINs using four levels. Table 6.2 states the results for training with the fixed learning rate while results for training with cyclic learning rate scheduling are given in Table 6.3. Figure 6.14 depicts resulting paths on a 128×128 map.

Table 6.2: Results for 2D random obstacle grid worlds with fixed learning rate training. All stated numbers are averaged over five network instances with different random seed initializations.

32×32	VIN	HVIN	AVIN		
Accuracy	80.38%	80.08%	84.52%		
Success	91.72%	93.84%	97.18%		
Path difference	2.48%	2.23%	1.63%		
GPU memory (training)	761 MB	739 MB	685 MB		
Training time per epoch	12.08 s	5.20 s	14.43 s		

64×64	VIN	HVIN	AVIN		
Accuracy	70.08%	77.42%	81.60%		
Success	71.98%	86.82%	94.02%		
Path difference	2.94%	2.55%	1.34%		
GPU memory (training)	1815 MB	1399 MB	969 MB		
Training time per epoch	55.49 s	11.31 s	26.92 s		

128×128	VIN	HVIN-3	HVIN-4	AVIN-3	AVIN-4
Accuracy	55.96%	76.50%	78.04%	77.70%	83.54%
Success	31.56%	83.24%	84.00%	78.52%	88.72%
Path difference	8.46%	2.09%	2.80%	3.60%	1.84%
GPU memory (training)	8247 MB	4085 MB	4049 MB	2189 MB	1167 MB
Training time per epoch	339.88 s	41.22 s	37.27 s	97.01 s	38.77 s

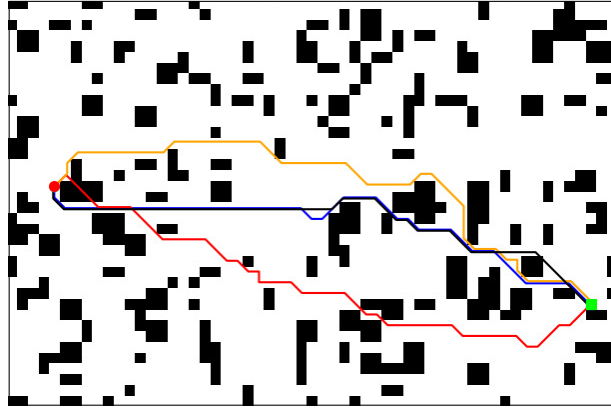


Figure 6.14: Resulting paths for fixed learning rate training on a 128×128 2D random obstacle grid map. The figure only depicts the corresponding map section. The start is marked with a red circle and the goal with a green square. An optimal path from the A* planner is depicted in black. VINs (red) fail to find a collision free path. HVINs (orange) and AVINs (blue) were able to find a path while the AVIN solution is closer to the optimal one.

Table 6.3: Results for 2D random obstacle grid worlds with cyclic learning rate training.

32×32	VIN	HVIN	AVIN		
Accuracy	84.92%	81.36%	85.00%		
Success	95.26%	88.27%	97.56%		
Path difference	1.01%	1.01%	1.56%		
GPU memory (training)	761 MB	739 MB	685 MB		
Training time per epoch	12.08 s	5.20 s	14.43 s		

64×64	VIN	HVIN	AVIN		
Accuracy	78.88%	80.46%	83.75%		
Success	89.17%	88.33%	94.99%		
Path difference	1.02%	1.02%	1.28%		
GPU memory (training)	1815 MB	1399 MB	969 MB		
Training time per epoch	55.49 s	11.31 s	26.92 s		

128×128	VIN	HVIN-3	HVIN-4	AVIN-3	AVIN-4
Accuracy	66.41%	77.46%	79.34%	84.28%	85.01%
Success	34.89%	77.40%	83.56%	86.85%	91.59%
Path diff.	1.02%	1.02%	1.02%	0.80%	1.31%
GPU memory (training)	8247 MB	4085 MB	4049 MB	2189 MB	1167 MB
Training time per epoch	339.88 s	41.22 s	37.27 s	97.01 s	38.77 s

The results indicate that our AVINs outperform VINs and HVINs on all map sizes with both learning rate behaviors in terms of *accuracy*, *success*, and memory consumption. While VINs and AVINs show a consistently better performance with the cyclic learning rate scheduling, HVINs perform better with the constant learning rate. However, the cyclic learning rate leads to shorter path differences, and, hence, a better path quality, in all cases. As also shown in Figure 6.14, we observed that AVINs showed a better long-distance “understanding” in comparison to HVINs which often reacted to obstacles directly before collisions. It can be furthermore seen that on the 128×128 maps, both HVINs and AVINs benefit from a fourth representation level in all measures.

6.3.2 Path Planning in 2D Maze Grid Worlds

In a second experiment, we aimed at investigating the limitations of our proposed method and the quality of its abstraction by evaluating the planning performance on 2D maze grid worlds. Mazes possess a larger information density in comparison to the random obstacle grid worlds since the occupancy of nearly every single grid cell is important. Hence, when generating coarser representations, the effect of information loss is large. This puts the focus on the quality of the abstraction which should learn to encode all required information in the additional features. Table 6.4 states the performance of VINs, HVINs, and AVINs on map size of 16×16 , 32×32 , and 64×64 . Training was performed with fixed learning rate. An example maze and generated paths are depicted in Figure 6.15.

Since original VINs perform no abstraction procedure, it was to expect that they obtain the best *accuracy* and *success* rates in this challenging domain. However, while the performance difference between

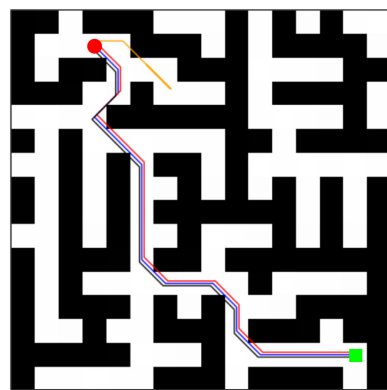


Figure 6.15: Resulting path for a 32×32 maze grid map. The figure only depicts the corresponding map sections. HVINs (orange) are not able to find a path. VINs (red), our AVINs (blue), and the A* planner (black) provide the same optimal solution.

Table 6.4: Results for 2D maze grid worlds with fixed learning rate training.

16×16	VIN	HVIN	AVIN
Accuracy	94.42%	87.20%	85.59%
Success	94.48%	87.42%	86.88%
Path difference	0.51%	2.02%	1.96%
GPU memory (training)	569 MB	575 MB	635 MB
Training time per epoch	4.68 s	3.91 s	8.70 s
32×32	VIN	HVIN	AVIN
Accuracy	85.60%	69.94%	82.17%
Success	82.10%	48.54%	71.50%
Path difference	0.88%	2.02%	1.09%
GPU memory (training)	761 MB	739 MB	685 MB
Training time per epoch	12.08 s	5.20 s	14.43 s
64×64	VIN	HVIN	AVIN
Accuracy	84.58%	58.82%	81.57%
Success	78.02%	14.22%	59.39%
Path difference	1.49%	1.99%	0.68%
GPU memory (training)	1815 MB	1399 MB	969 MB
Training time per epoch	55.49 s	11.31 s	26.92 s

HVINs and VINs was rather small in the random obstacle domain, this difference increases considerably for large maze worlds. For a map size of 64×64 cells, HVINs were only able to find a valid path in about 14% of the tasks. This can be explained by the fact that only information is discarded but no abstraction—in terms of increasing scene “understanding” while coarsening the resolution—is performed and the represented information in higher levels does not contain all required information. For the 16×16 maps, our approach performs worse than HVINs. An explanation for this might be that, for this input map size, *Level 1* only has a size of 4×4 which might be insufficient to plan next actions in the vicinity of the robot in this detailed representation. Nevertheless, our method significantly outperforms HVINs on larger maps indicating the advantage of our abstraction method—which introduces additional features to compensate information loss—compared to HVINs. Interestingly, while the path difference increases with increasing map size for VINs, paths become better with increasing map size for AVINs.

Regarding the computational performance, Figure 6.16 visualizes the GPU memory consumption during training and training times for

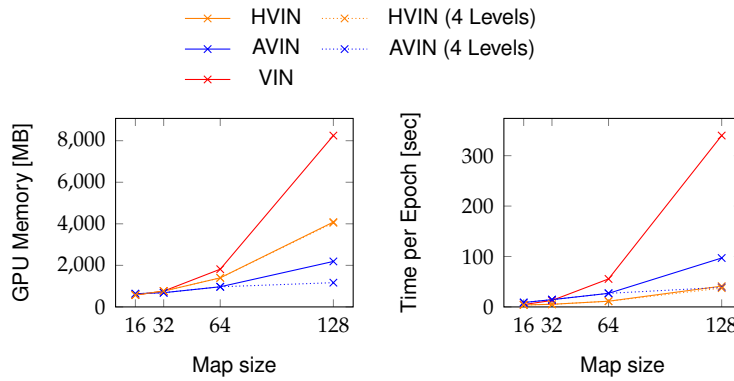


Figure 6.16: Computational performance of all three methods depending on the map size.

all map sizes. It can be seen that AVINs mostly have the lowest memory requirements which even considerably decreases for 128×128 maps, when employing a fourth level. It was to expect that VINs require the most memory since they discard no information at all. Regarding the required training time per epoch, which is an indication for the network complexity, VINs perform the worst as well, since they have to perform their value iteration in state-value maps of the original input size. HVINs and AVINs decrease the state space size result in faster computation. Since HVINs perform no abstraction procedure and do not consider additional features for each cell, they are generally faster than AVINs. However, as can be seen for the 128×128 map size, HVINs do nearly not benefit from a fourth level while AVINs even outperform them on this map size.

6.3.3 *Planning Omnidirectional Robot Driving Locomotion with Footprint Consideration*

Since in the previous experiments AVINs demonstrated to be applicable to larger state spaces, we evaluated their performance to plan omnidirectional robot driving locomotion while considering the robot footprint. An example platform is the quadrupedal disaster response robot Centauro whose legs end in 360° steerable, active wheels, as can be seen in Figure 6.17. We applied the 3D version of AVINs and chose a fixed quadratic footprint with 0.8m longitudinal and lateral distance between wheels. Environment maps had a resolution of 0.2 m.

At first, we employed our method to plan paths for the described footprint on 32×32 maps of the random obstacle grid domain. Averaged over five training runs, we achieved a success rate of 74.20% for the 5,005 tasks in the test set while our paths were 1.86% longer than the optimal solution. The method required 865 MB of GPU memory.

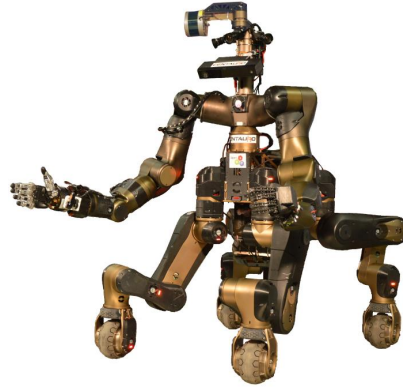


Figure 6.17: The Centauro robot.

In addition, we let the A* planner solve the same tasks as well as the 2D random obstacle tasks on 32×32 maps. A comparison of the corresponding planning times is given in Table 6.5.

It can be seen that the A* planner is in average about 23 times faster than AVINs on the 2D planning task. However, this advantage disappears for the more complex 3D planning tasks for which both planners achieved similar planning times. This observation supports our initial assumption that learning-based planners are beneficial for complex planning tasks since they do not perform extensive, iterative searches, as traditional planners do.

Finally, the developed 3D AVINs method was integrated into the locomotion planning pipeline of Centauro. The detailed pipeline description is given in Chapter 3, while stepping motions were not considered in this context. To match the desired input type, we computed occupancy grids with a resolution of 0.2m from the height maps which were used as environment representation in the previous chapters. Robot perception and control was implemented in C++. Communication with AVINs was realized using ROS. The experiment was performed in the Gazebo simulation environment. The world contained challenging obstacles of different shape and size, as shown in Figure 6.18. We placed nine different goal states in the map, as shown in Figure 6.19. Table 6.6 states the planner performance and compares the results to the performance of the A* planner. An example resulting path of AVINs is depicted in Figure 6.19.

Table 6.5: Planning time comparison for the A* planner and AVINs.

32×32	A* planner	AVINs
2D random obstacle grid worlds	0.004 s	0.093 s
3D robot locomotion with footprint	0.263 s	0.283 s

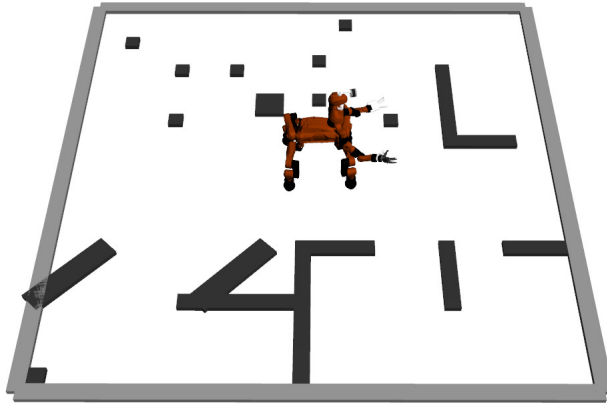


Figure 6.18: Gazebo arena of the 3D locomotion planning experiment. Obstacle heights were chosen to be rather small to prevent the laser scanner from coping with occlusions.

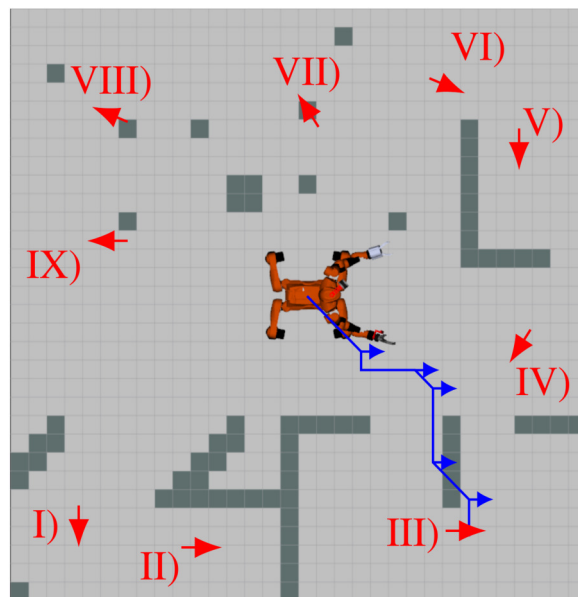


Figure 6.19: Occupancy map of the 3D locomotion planning experiment with the nine chosen goal states and one example resulting path computed by AVINs.

Table 6.6: Results of our approach and the A* planner for the tasks depicted in Figure 6.19.

Goal	AVIN		A* planner	
	Path length	Planning time	Path length	Planning time
I)	24.59	0.431 s	23.41	0.169 s
II)	Not found		24.14	0.980 s
III)	18.49	0.342 s	17.90	0.102 s
IV)	18.80	0.363 s	18.80	0.341 s
V)	Not found		27.76	2.117 s
VI)	18.65	0.321 s	17.01	0.172 s
VII)	15.55	0.321 s	15.55	0.051 s
VIII)	24.67	0.449 s	22.92	0.223 s
IX)	21.13	0.405 s	21.13	0.705 s

The results indicate that the AVIN planner provided optimal or close-to-optimal paths in most cases. Even challenging tasks, which, e.g., required the robot to take obstacles between its legs (*III* and *VIII*) could be solved. However, for goal states *II* and *VIII*, the presented planner could not find a solution but ended in oscillations between two adjacent states. In both cases, turning actions in narrow passages were required. The AVIN planner might have a problem with such situations. Moreover, AVIN planning times had a considerably smaller distribution than the A* planner. Although AVINs planning times do not show a considerable advantage over A*, this experiment demonstrates the application of AVINs to significantly more challenging tasks compared to the original VIN applications.

6.4 CONCLUSION

In this chapter, AVINs, an extension to Value Iteration Networks (VINs) to employ multiple levels of abstraction is presented. With increasing distance from the robot, the level of abstraction increases. With increasing abstraction, the spatial resolution gets coarser while the accompanying information loss is compensated through additional features increasing the situation “understanding”. The method is fully differentiable and can be learned using standard backpropagation.

Experiments showed that our approach outperformed VINs in 2D grid worlds with random obstacles, as presented in their original publication. While the success rate of AVINs was considerably better, they were capable of planning for environments which were up to 16 times larger while only requiring a fraction of the GPU memory. In compar-

ison to Hierarchical VINs (HVINs), which employ multiresolution representations in coarse-to-fine planning without the introduction of additional features, AVINs obtain a better result quality with lower memory requirements. In particular, it was shown that AVINs learn to encode useful information in their abstract representations since the performance in challenging environments, such as mazes, was considerably better in comparison to HVINs. As a demonstration of the applicability to challenging real-world problems, AVINs were applied to plan omnidirectional driving locomotion for the search-and-rescue robot Centauro while considering its footprint for precise collision checking. In summary, it was demonstrated how abstraction enables learning-based planners to handle more complex state spaces—increasing their applicability towards real-world motion planning problems.

Comparing the obtained planning times to those of an A* planner supported the initial assumption that learning-based planners are promising to outperform traditional planners in complex tasks, since learning-based planners do not need to perform extensive, iterative searches or sampling but develop an “understanding” of the planning problem. However, decreasing success rates for the more complex 3D locomotion planning task indicated the limitations of the presented method. To enable planning for considerably more challenging planning problems, such as hybrid driving-stepping locomotion planning, more complex network architectures and correspondingly large amounts of training data would be needed introducing challenges to future hardware development.

DISCUSSION

In this thesis, a planning method for hybrid driving-stepping locomotion of quadrupedal ground robots in challenging environments was presented. The approach is based on a graph search algorithm. Extensions to plan on multiple representations with increasing level of abstraction and to employ machine learning methods were developed to increase the planning performance.

The presented planner operates on costs for the individual robot feet and its base. In combination with a fine planning resolution, precise planning in challenging environments is achieved. It was described how the robot representation, action set, and corresponding cost function were derived from the considered robot platforms and environments. Moreover, different underlying planning algorithms were discussed with respect to their applicability to the described problem, and a search-based approach was chosen. Since the fine planning resolution and the high-dimensional robot representation result in large state spaces, focus was put on planning efficiency. A hierarchical representation of stepping maneuvers and the application of anytime characteristics were described besides several additional optimizations. It was shown in simulated and real-world experiments that this method is capable of generating bounded sub-optimal paths in feasible time. This even applies to environments as complex as staircases with obstacles on them. To the best of our knowledge, this is the first approach that describes hybrid driving-stepping locomotion planning for scenarios as challenging as staircases. However, it was further shown that this method is limited to relatively small environments since the corresponding state spaces become too large, otherwise.

It was subsequently presented how an extension to plan on multiple representations with different levels of abstraction empowers the planner to overcome this limitation to small environments. While the planning problem is described in high detail in the robot vicinity, the planning representation gets coarser with increasing distance from the robot. The loss of information that comes along with such coarsening operations is compensated through additional semantic features. In this manner, the scene “understanding” and, hence, the applicability to challenging environments with a high density of important information are increased. An in-depth analysis about how the different representations were manually designed and how their cost functions were parametrized to give similar situation assessments was carried out. In addition, it was described how the most abstract representa-

tion can be used to generate an informed heuristic. The experiments indicated that this method accelerates planning by multiple orders of magnitude while the result quality stays comparable.

In order to minimize extensive manual tuning efforts when designing abstract representations, it was presented how the tuning-intensive abstract cost function can be replaced by a CNN. This can be trained on generated artificial data such that manual labeling of training data is not required. Experiments indicated that the CNN is capable of transferring its knowledge to real-world data and that the abstraction quality outperforms the manually designed abstract representations. The method was applied to the hybrid locomotion planning problem as the basis for the above-described informed heuristic. The resulting pipeline requires preprocessing that can be carried out in justifiable time and achieves similar planning performances to the manually tuned representations without the need for extensive tuning efforts.

It was finally presented in this thesis, how the idea of planning on multiple representations with different levels of abstraction was applied to the learning-based planning approach Value Iteration Networks (VINs). The presented method is fully differentiable and is integrated in the network architecture. Experiments prove that this method enables VINs to solve significantly larger 2D grid world tasks and that the abstract representations possess an increased scene “understanding”. When applied to plan omnidirectional driving in challenging environments, the method demonstrated its capability to solve significantly more complex planning tasks compared to 2D grid worlds which, hence, increases its real-world applicability. Nevertheless, experiments also revealed limitations of the method with this task and, thus, the additional integration of stepping motions would probably not achieve satisfying results.

The presented research in this thesis supports a current trend in the planning community: traditional planning methods are not capable of solving complex, high-dimensional planning problems with large state spaces in feasible time. Novel, learning-based planning methods are limited to even smaller state spaces. However, the combination of the complementary strengths of these different planning approaches achieves good results. Traditional planners are well-suited to consider complex kinematic capabilities, to perform reliable obstacle avoidance, and show a good goal-directed behavior and high success rates. In contrast, learning-based planners are advantageous to efficiently draw planning decisions in complex situations by leveraging their scene “understanding”, and thus they do not require extensive, iterative searches or sampling.

Regarding the generalizability of the presented methods, planning on multiple representations with different levels of abstraction is applicable to a wide range of high-dimensional planning problems and

is promising to considerably accelerate planning. This applies to both versions, either with manually tuned or learned cost functions which are both flexible with respect to the provided planning representation. Moreover, the presented extension of VINs that employs multiple levels of abstraction is flexible to the nature of the planning problem. If the manageable state space size suffices to describe the planning problem, this is an interesting alternative to traditional planning approaches. In addition, the idea to extend learning-based planners to employ multiple levels of abstraction can be transferred to other learning-based planning approaches, e.g., QMDP-Nets by Karkus et al. (2017), and is promising to increase their real-world applicability.

OUTLOOK AND FUTURE WORK

The methods presented in this thesis open several directions for future work with the main motivation of increasing the performance, the generalizability, and the real-world applicability.

Dynamic locomotion: The locomotion speed, especially of stepping motions, of the presented controller was set such that static stability computation can be applied. To increase the method's real-world applicability, faster execution is desirable. This inevitably comes along with the need to consider robot dynamics, and thus, a different system stability computation. Moreover, acceleration needs to be explicitly handled by the planner in order to achieve dynamic maneuvers. Medeiros et al. (2019), de Viragh et al. (2019), and Bellegarda and Byl (2019) propose in their recent works dynamic approaches to the control of wheeled-legged robots. A combination with the planning methods presented in this thesis is promising to yield interesting results.

Dynamic environments: While disaster response environments mostly possess static obstacles, other potential application domains of hybrid driving-stepping locomotion, such as urban delivery services, exhibit more dynamic characteristics. In order to increase the method's applicability to those domains, dynamic environments need to be considered. This would require the introduction of time to the planning problem. Moreover, the integration with behavior prediction methods for dynamic environment parts would pose an interesting research problem.

Learning abstract representations: This thesis presented how abstract representations can considerably support traditional planning and how machine learning methods can be used to represent the tuning intensive parts of those representations. Nevertheless, the representation still needs to be manually designed limiting the generalizability to other planning domains. In contrast,

abstract representations for planning problems of limited complexity were fully represented as a CNN in the presented VIN extension. It is a challenging and fascinating idea to investigate if machine learning methods are capable of representing entire abstract representations for more complex planning problems such as hybrid driving-stepping locomotion.

Further integration of traditional planners and machine learning: The results in this thesis indicated that the combination of traditional planners and machine learning outperforms isolated planning approaches. Hence, it is a promising idea to conduct further research in this direction and to investigate how machine learning methods can support traditional planners and mitigate their weaknesses. Regarding search-based planning, a learning-based method might support the planner in its decision about the next node to expand by using scene “understanding” of the environment. Corresponding basic principles are, e.g., presented in Frontzek et al. (2001). Regarding sampling-based planning, a learning-based method might guide the planner’s sampling to regions of special interest.

Exploit massive parallelization on GPUs: One of the reasons for the popularity of machine learning methods in recent years is considerable hardware improvements for GPUs and other parallel processing units. While there exist multiple frameworks for machine learning methods that exploit this massive parallelization, this is not true for traditional planners. Sampling-based algorithms with massive parallel state sampling or search-based algorithms with massive parallel node expansion might be an exciting research field. Recently, e.g., Battaglia et al. (2018) proposed a framework to represent graphs as neural networks which is a promising foundation in this direction.

LIST OF FIGURES

Figure 1.1	Overview of the thesis structure.	7
Figure 2.1	Examples of hybrid driving-stepping robot plat- forms.	10
Figure 2.2	Value Iteration Networks (VINs) architecture. .	17
Figure 2.3	Hierarchical Value Iteration Networks (HVINs)	20
Figure 3.1	Hybrid driving-stepping robots employed with the described approach.	25
Figure 3.2	Overview of the system architecture.	27
Figure 3.3	Driving cost computation example.	30
Figure 3.4	Approximation of the base shape.	31
Figure 3.5	Enriching height map-based foot cost map gen- eration with terrain class information.	32
Figure 3.6	Kinematic leg structures of Momaro and Cen- tauro.	34
Figure 3.7	Different possible knee configurations.	35
Figure 3.8	Foot reachability maps for Momaro and Cen- tauro.	36
Figure 3.9	Robot representation for locomotion planning.	37
Figure 3.10	State interpolation used in sampling-based plan- ning methods.	41
Figure 3.11	Omnidirectional driving action set.	43
Figure 3.12	Step criteria to be considered by the planner. .	45
Figure 3.13	Stepping related planning maneuvers.	46
Figure 3.14	Driving vs. stepping cost weighting.	47
Figure 3.15	Orientation difference cost factor.	48
Figure 3.16	Obtaining a stable stepping configuration. . . .	52
Figure 3.17	Base roll computation.	54
Figure 3.18	Gazebo scenario for planner evaluation.	56
Figure 3.19	Planner performance for different maximum orientation cost factors.	57
Figure 3.20	Resulting paths on foot cost maps.	57
Figure 3.21	ARA* performance for different employed heuris- tic weights.	58
Figure 3.22	Challenging Gazebo scenario to demonstrate the planner capabilities.	60
Figure 3.23	Generated path for the staircase scenario. . . .	60
Figure 3.24	Momaro climbing the staircase.	61
Figure 3.25	Scenario for the real robot experiment with Mo- maro.	62
Figure 3.26	Momaro stepping up an elevated platform. . .	62

Figure 3.27	Scenario for real robot experiment with Centauro.	63
Figure 3.28	Centauro executing the real-world task.	64
Figure 4.1	Example of a human efficiently planning a path.	68
Figure 4.2	Representation level overview.	70
Figure 4.3	Size and position of the different levels of representation.	70
Figure 4.4	Extending the planning pipeline to employ multiple levels of representation.	71
Figure 4.5	<i>Level 1</i> environment representation.	73
Figure 4.6	<i>Level 1</i> robot representation.	73
Figure 4.7	Subsampling method.	74
Figure 4.8	<i>Level 2</i> environment representation.	74
Figure 4.9	<i>Level 2</i> robot representation.	75
Figure 4.10	<i>Level 3</i> environment representation.	77
Figure 4.11	<i>Level 3</i> robot representation.	77
Figure 4.12	<i>Level 1</i> action set.	79
Figure 4.13	<i>Level 2</i> action set.	80
Figure 4.14	<i>Level 3</i> action set.	81
Figure 4.15	Computation of the <i>abstract representation-based heuristic</i>	83
Figure 4.16	Continuous refinement method.	84
Figure 4.17	Generation of a set of feasible robot base states for path refinement.	85
Figure 4.18	Height map of the first experiment scenario.	87
Figure 4.19	Planning performance for the first experiment.	87
Figure 4.20	Height map of the second experiment.	88
Figure 4.21	Resulting path for planning with the <i>abstract representation-based heuristic</i> and combined levels with $\mathcal{W} = 1.25$	89
Figure 4.22	Planning performance comparison for different heuristics.	89
Figure 4.23	Planning times for different start states.	90
Figure 5.1	Method overview.	95
Figure 5.2	Hybrid driving-stepping locomotion robots addressed by the presented planning method.	97
Figure 5.3	Detailed planning representation.	97
Figure 5.4	Action set of the detailed representation.	98
Figure 5.5	Architecture of the developed CNN.	100
Figure 5.6	CNN training performance.	102
Figure 5.7	Example tasks of the test data sets.	104
Figure 5.8	Locomotion planning experiment.	106
Figure 5.9	Planning times and path costs for all goal states and both heuristics.	107
Figure 6.1	General idea of Value Iteration Networks on multiple levels of abstraction (AVINs).	110

Figure 6.2	Input and output specification for AVINs. . . .	111
Figure 6.3	Network architecture.	112
Figure 6.4	Abstraction Module.	113
Figure 6.5	Reward Module.	114
Figure 6.6	Value Iteration Module.	115
Figure 6.7	Information flow between levels through map padding.	115
Figure 6.8	Possible actions for both planning domains. . .	117
Figure 6.9	Robot footprint computation.	118
Figure 6.10	Value Iteration Module for 3D robot locomotion planning.	118
Figure 6.11	Orientation padding during 3D Value Iterations.	119
Figure 6.12	Network architecture for 3D robot locomotion planning.	119
Figure 6.13	Training performance on validation sets.	121
Figure 6.14	Resulting paths for fixed learning rate training on a 128×128 2D random obstacle grid map.	124
Figure 6.15	Resulting path for a 32×32 maze grid map. . .	125
Figure 6.16	Computational performance depending on the map size.	127
Figure 6.17	The Centauro robot.	128
Figure 6.18	Gazebo arena of the 3D locomotion planning experiment.	129
Figure 6.19	Occupancy map of the 3D locomotion planning experiment	129

LIST OF TABLES

Table 3.1	Complexity comparison of priority queue implementations.	50
Table 5.1	Abstraction quality on the three data sets.	105
Table 5.2	Planner performance for both heuristics.	107
Table 6.1	Number of iterations depending on the input map size.	116
Table 6.2	Results for 2D random obstacle grid worlds with fixed learning rate training.	123
Table 6.3	Results for 2D random obstacle grid worlds with cyclic learning rate training.	124
Table 6.4	Results for 2D maze grid worlds with fixed learning rate training.	126
Table 6.5	Planning time comparison for the A* planner and AVINs.	128
Table 6.6	Results of our approach and the A* planner.	130

ACRONYMS

ARA*	Anytime Repairing A*
AVINs	Value Iteration Networks on multiple levels of abstraction
BCE	Binary Cross Entropy
CHOMP	Covariant Hamiltonian Optimization for Motion Planning
CNN	Convolutional Neural Network
CoM	center of mass
DoF	degrees of freedom
DRC	DARPA Robotics Challenge
FMT*	Fast Marching Tree
GPU	graphics processing unit
HVINs	Hierarchical Value Iteration Networks
IMU	inertial measurement unit
IK	inverse kinematics
L1	Least Absolute Deviations
MDP	Markov Decision Process
PRM	Probabilistic Roadmap
RL	reinforcement learning
ROS	Robot Operating System
RRT	Rapidly-exploring Random Tree
SLAM	Simultaneous Localization and Mapping
SGD	Stochastic Gradient Descent
STC	support triangle centroid
STOMP	Stochastic Trajectory Optimization for Motion Planning
UPNs	Universal Planning Networks
VI	Value Iteration
VINs	Value Iteration Networks

BIBLIOGRAPHY

- Aine, S., S. Swaminathan, V. Narayanan, V. Hwang, and M. Likhachev (2016). "Multi-heuristic A*." In: *The International Journal of Robotics Research (IJRR)* 35.1-3, pp. 224–243.
- Battaglia, P. W., J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu (2018). "Relational inductive biases, deep learning, and graph networks." In: *arXiv preprint: 1806.01261*.
- Behnke, S. (2003). "Local multiresolution path planning." In: *RoboCup 2003: Robot Soccer World Cup VII*. Springer, pp. 332–343.
- Bellegarda, G. and K. Byl (2019). "Trajectory optimization for a wheel-legged system for dynamic maneuvers that allow for wheel slip." In: IEEE International Conference on Decision and Control (CDC).
- Bellman, R. (1957). "A Markovian decision process." In: *Journal of Mathematics and Mechanics* 6.5, pp. 679–684.
- Bellman, R. (2013 [1957]). *Dynamic programming*. Courier Corporation.
- Biber, P., U. Weiss, M. Dorna, and A. Albert (2012). "Navigation system of the autonomous agricultural robot BoniRob." In: *Workshop on Agricultural Robotics: Enabling Safe, Efficient, and Affordable Robots for Food Production*.
- Bjelonic, M., C. D. Bellicoso, Y. de Viragh, D. Sako, F. D. Tresoldi, F. Jenelten, and M. Hutter (2019). "Keep rollin'-Whole-body motion control and planning for wheeled quadrupedal robots." In: *IEEE Robotics and Automation Letters (RA-L)*.
- Bohlin, R. (2001). "Path planning in practice; Lazy evaluation on a multi-resolution grid." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Bojarski, M., D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, and J. Zhao (2016). "End to end learning for self-driving cars." In: *arXiv preprint:1604.07316*.
- Boston Dynamics (2019). *Handle*. <https://www.bostondynamics.com/handle>. [Online; accessed 22-June-2019].
- Botea, A., M. Müller, and J. Schaeffer (2004). "Near optimal hierarchical path-finding." In: *Journal of Game Development* 1.1, pp. 7–28.
- Brock, O. and O. Khatib (2002). "Elastic strips: A framework for motion generation in human environments." In: *The International Journal of Robotics Research (IJRR)* 21.12, pp. 1031–1052.

- Brunner, M., B. Brüggemann, and D. Schulz (2012). "Motion planning for actively reconfigurable mobile robots in search and rescue scenarios." In: *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*.
- Burget, F., M. Bennewitz, and W. Burgard (2016). "BI²RRT: An efficient sampling-based path planning framework for task constrained mobile manipulation." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Chen, X., A. Ghadirzadeh, J. Folkesson, M. Björkman, and P. Jensfelt (2018). "Deep reinforcement learning to acquire navigation skills for wheel-legged robots in complex environments." In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Colas, F., S. Mahesh, F. Pomerleau, M. Liu, and R. Siegwart (2013). "3D path planning and execution for search and rescue ground robots." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Cordes, F., C. Oekermann, A. Babu, D. Kuehn, T. Stark, F. Kirchner, and DFKI Bremen Robotics Innovation Center (2014). "An active suspension system for a planetary rover." In: *International Symposium on Artificial Intelligence, Robotics and Automation in Space (SAIRAS)*.
- Corman, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (1990). *Introduction to algorithms*. MIT press.
- Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs." In: *Numerische Mathematik* 1.1, pp. 269–271.
- Dornbush, A., K. Vijayakumar, S. Bardapurkar, F. Islam, M. Ito, and M. Likhachev (2018). "A single-planner approach to multi-modal humanoid mobility." In: *IEEE International Conference on Robotics and Automation (ICRA)*.
- Droeschel, D., M. Schwarz, and S. Behnke (2017). "Continuous mapping and localization for autonomous navigation in rough terrain using a 3D laser scanner." In: *Robotics and Autonomous Systems* 88, pp. 104–115.
- Faust, A., K. Oslund, O. Ramirez, A. Francis, L. Tapia, M. Fiser, and J. Davidson (2018). "PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning." In: *IEEE International Conference on Robotics and Automation (ICRA)*.
- Fleckenstein, F., C. Dornhege, and W. Burgard (2017). "Efficient path planning for mobile robots with adjustable wheel positions." In: *IEEE International Conference on Robotics and Automation (ICRA)*.
- Fontaine, T. (2016). *CMU robotics center rebuilding staff after 'Uber crisis'*. <https://triblive.com/news/adminpage/10101108-74/million-research-nrec>. [Online; accessed 22-June-2019].

- Frontzek, T., T. N. Lal, and R. Eckmiller (2001). "Towards learning path planning for solving complex robot tasks." In: *International Conference on Artificial Neural Networks (ICANN)*.
- Gammell, J. D., S. S. Srinivasa, and T. D. Barfoot (2014). "Informed RRT: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Geraerts, R. and M. H. Overmars (2004). "A comparative study of probabilistic roadmap planners." In: *Algorithmic Foundations of Robotics V*. Springer, pp. 43–57.
- Gerkey, B. P. and K. Konolige (2008). "Planning and control in unstructured terrain." In: *ICRA Workshop on Path Planning on Costmaps*.
- Gochev, K., B. Cohen, J. Butzke, A. Safonova, and M. Likhachev (2011). "Path planning with adaptive dimensionality." In: *Fourth Annual Symposium on Combinatorial Search*.
- González-Sieira, A., M. Mucientes, and A. Bugarín (2016). "An adaptive multi-resolution state lattice approach for motion planning with uncertainty." In: *Robot 2015: Second Iberian Robotics Conference*. Springer, pp. 257–268.
- Halme, A., I. Leppnen, M. Montonen, and S. Ylmen (2001). "Robot motion by simultaneously wheel and leg propulsion." In: *International Conference on Climbing and Walking Robots and Support Technologies for Mobile Machines (CLAWAR)*.
- Halme, A., I. Leppänen, J. Suomela, S. Ylönen, and I. Kettunen (2003). "WorkPartner: Interactive human-like service robot for outdoor applications." In: *The International Journal of Robotics Research (IJRR)* 22.7-8, pp. 627–640.
- Hart, P. E., N. J. Nilsson, and B. Raphael (1968). "A formal basis for the heuristic determination of minimum cost paths." In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.
- Hashimoto, K., T. Hosobata, Y. Sugahara, Y. Mikuriya, H. Sunazuka, M. Kawase, H. Lim, and A. Takanishi (2005). "Realization by biped leg-wheeled robot of biped walking and wheel-driven locomotion." In: *IEEE International Conference on Robotics and Automation (ICRA)*.
- Hebert, P., M. Bajracharya, J. Ma, N. Hudson, A. Aydemir, J. Reid, C. Bergh, J. Borders, M. Frost, M. Hagman, J. Leichty, P. Backes, and B. Kennedy (2015). "Mobile manipulation and mobility as manipulation—Design and algorithms of RoboSimian." In: *Journal of Field Robotics (JFR)* 32.2, pp. 255–274.
- Hinton, G., N. Srivastava, and K. Swersky (2012). *Neural Networks for Machine Learning - Lecture 6e - rmsprop: Divide the gradient by a running average of its recent magnitude*.
- Holden, D., T. Komura, and J. Saito (2017). "Phase-functioned neural networks for character control." In: *ACM Transactions on Graphics (TOG)* 36.4, p. 42.

- Holte, R. C., M. B. Perez, R. M. Zimmer, and A. J. MacDonald (1995). "Hierarchical A*: Searching abstraction hierarchies efficiently." In: *Symposium on Abstraction, Reformulation, and Approximation*.
- Hornung, A., K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard (2013). "OctoMap: An efficient probabilistic 3D mapping framework based on octrees." In: *Autonomous Robots* 34.3, pp. 189–206.
- Hsu, D., T. Jiang, J. Reif, and Z. Sun (2003). "The bridge test for sampling narrow passages with probabilistic roadmap planners." In: *IEEE International Conference on Robotics and Automation (ICRA)*.
- Janson, L., E. Schmerling, A. Clark, and M. Pavone (2015). "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions." In: *The International Journal of Robotics Research (IJRR)* 34.7, pp. 883–921.
- Kalakrishnan, M., J. Buchli, P. Pastor, M. Mistry, and S. Schaal (2011a). "Learning, planning, and control for quadruped locomotion over challenging terrain." In: *The International Journal of Robotics Research* 30.2, pp. 236–258.
- Kalakrishnan, M., S. Chitta, E. Theodorou, P. Pastor, and S. Schaal (2011b). "STOMP: Stochastic trajectory optimization for motion planning." In: *IEEE International Conference on Robotics and Automation (ICRA)*.
- Kamedula, M., N. Kashiri, and N. G. Tsagarakis (2018). "On the kinematics of wheeled motion control of a hybrid wheeled-legged centauro robot." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Karaman, S. and E. Frazzoli (2011). "Sampling-based algorithms for optimal motion planning." In: *International Journal of Robotics Research (IJRR)* 30.7, pp. 846–894.
- Karkowski, P. and M. Bennewitz (2016). "Real-time footstep planning using a geometric approach." In: *IEEE International Conference on Robotics and Automation (ICRA)*.
- Karkus, P., D. Hsu, and W. S. Lee (2017). "QMDP-Net: Deep learning for planning under partial observability." In: *Advances in Neural Information Processing Systems (NIPS)*.
- Kashiri, N., A. Ajoudani, D. G. Caldwell, and N. G. Tsagarakis (2016). "Evaluation of hip kinematics influence on the performance of a quadrupedal robot leg." In: *International Conference on Informatics in Control, Automation and Robotics (ICINCO)*.
- Kavraki, L. E., P. Svestka, J.-C. Latombe, and M. H. Overmars (1996). "Probabilistic roadmaps for path planning in high-dimensional configuration spaces." In: *IEEE Transactions on Robotics and Automation (T-RA)* 12.4, pp. 566–580.
- Khatib, O. (1986). "Real-time obstacle avoidance for manipulators and mobile robots." In: *Autonomous Robot Vehicles*. Springer, pp. 396–404.

- Klamt, T., D. Rodriguez, L. Baccelliere, X. Chen, D. Chiaradia, T. Cichon, M. Gabardi, P. Guria, K. Holmquist, M. Kamedula, H. Karagoz, N. Kashiri, A. Laurenzi, C. Lenz, D. Leonardis, E. Mingo Hoffman, L. Muratore, D. Pavlichenko, F. Porcini, Z. Ren, F. Schilling, M. Schwarz, M. Solazzi, M. Felsberg, A. Frisoli, M. Gustmann, P. Jensfelt, K. Nordberg, J. Roßmann, U. Süss, N. G. Tsagarakis, and S. Behnke (2019a). "Flexible disaster response of tomorrow – Final presentation and evaluation of the CENTAURO system." Version 26. In: *IEEE Robotics and Automation Magazine (RAM)* (4), pp. 59–72.
- Klamt, T., M. Schwarz, C. Lenz, L. Baccelliere, D. Buongiorno, T. Cichon, A. Di Guardo, D. Droschel, M. Gabardi, M. Kamedula, N. Kashiri, A. Laurenzi, D. Leonardis, L. Muratore, D. Pavlichenko, A. S. Periyasamy, D. Rodriguez, M. Solazzi, A. Frisoli, M. Gustmann, J. Roßmann, U. Süss, N. G. Tsagarakis, and S. Behnke (2019b). "Remote mobile manipulation with the Centauro robot: Full-body telepresence and autonomous operator assistance." In: *Journal of Field Robotics (JFR)*.
- Klein, R. (2005). "Geometrische datenstrukturen." In: *Algorithmische Geometrie: Grundlagen, Methoden, Anwendungen*, pp. 107–154.
- Koenig, N. and A. Howard (2004). "Design and use paradigms for Gazebo, an open-source multi-robot simulator." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Kohrt, C., A. G. Pipe, J. Kiely, R. Stamp, and G. Schiedermeier (2012). "A cell based Voronoi roadmap for motion planning of articulated robots using movement primitives." In: *IEEE International Conference on Robotics and Biomimetics (ROBIO)*.
- Kolter, J. Z., M. P. Rodgers, and A. Y. Ng (2008). "A control architecture for quadruped locomotion over rough terrain." In: *IEEE International Conference on Robotics and Automation (ICRA)*.
- Kurutach, T., A. Tamar, G. Yang, S. J. Russell, and P. Abbeel (2018). "Learning plannable representations with Causal InfoGAN." In: *Advances in Neural Information Processing Systems (NIPS)*.
- Lavalle, S. M. (1998). "Rapidly-exploring random trees: A new tool for path planning." In: *Computer Science Department, Iowa State University Technical Report 98.11*.
- Lee, L., E. Parisotto, D. S. Chaplot, E. Xing, and R. Salakhutdinov (2018). "Gated path planning networks." In: *35th International Conference on Machine Learning (ICML)*.
- Levine, S., C. Finn, T. Darrell, and P. Abbeel (2016). "End-to-end training of deep visuomotor policies." In: *The Journal of Machine Learning Research* 17.1, pp. 1334–1373.
- Li, L., T. J. Walsh, and M. L. Littman (2006). "Towards a unified theory of state abstraction for MDPs." In: *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*.

- Likhachev, M., G. J. Gordon, and S. Thrun (2004). "ARA*: Anytime A* with provable bounds on sub-optimality." In: *Advances in Neural Information Processing Systems (NIPS)*.
- Loshchilov, I. and F. Hutter (2016). "SGDR: Stochastic gradient descent with warm restarts." In: *arXiv preprint:1608.03983*.
- Medeiros, V. S., M. Bjelonic, E. Jelavic, R. Siegwart, M. A. Meggiolaro, and M. Hutter (2019). "Trajectory optimization for wheeled quadrupedal robots driving in challenging terrain." In: *9th International Symposium on Adaptive Motion of Animals and Machines*.
- Menna, M., M. Gianni, F. Ferri, and F. Pirri (2014). "Real-time autonomous 3D navigation for tracked vehicles in rescue environments." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Motiv Robotics (2019). *RoboMantis*. <http://www.motivrobotics.com/products/#/products/robots/>. [Online; accessed 22-June-2019].
- Murphy, M. P., A. Saunders, C. Moreira, A. A. Rizzi, and M. Raibert (2011). "The LittleDog robot." In: *The International Journal of Robotics Research (IJRR)* 30.2, pp. 145–149.
- Nannicini, G., D. Delling, L. Liberti, and D. Schultes (2008). "Bidirectional A* search for time-dependent fast paths." In: *International Workshop on Experimental and Efficient Algorithms*. Springer.
- Niu, S., S. Chen, H. Guo, C. Targonski, M. C. Smith, and J. Kovačević (2018). "Generalized Value Iteration Networks: Life beyond lattices." In: *Thirty-Second AAAI Conference on Artificial Intelligence*.
- Pavlichenko, D. and S. Behnke (2017). "Efficient stochastic multicriteria arm trajectory optimization." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Peng, X. B., G. Berseth, K. Yin, and M. Van De Panne (2017). "DeepLoco: Dynamic locomotion skills using hierarchical deep reinforcement learning." In: *ACM Transactions on Graphics (TOG)* 36.4, p. 41.
- Perrin, N., C. Ott, J. Engelsberger, O. Stasse, F. Lamiroux, and D. Caldwell (2016). "Continuous legged locomotion planning." In: *IEEE Transactions on Robotics* 33.1, pp. 234–239.
- Petereit, J., T. Emter, and C. W. Frey (2013). "Mobile robot motion planning in multi-resolution lattices with hybrid dimensionality." In: *IFAC Intelligent Autonomous Vehicles Symposium* 46.10, pp. 158–163.
- Pivtoraiko, M. and A. Kelly (2008). "Differentially constrained motion replanning using state lattices with graduated fidelity." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Prautzsch, H., W. Boehm, and M. Paluszny (2013). *Bézier and B-spline techniques*. Springer.
- Quigley, M., K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng (2009). "ROS: An open-source Robot Operating System." In: *ICRA workshop on open source software*.

- Quinlan, S. and O. Khatib (1993). "Elastic bands: Connecting path planning and control." In: *IEEE International Conference on Robotics and Automation (ICRA)*.
- Ratliff, N., M. Zucker, J. A. Bagnell, and S. Srinivasa (2009). "CHOMP: Gradient optimization techniques for efficient motion planning." In: *IEEE International Conference on Robotics and Automation (ICRA)*.
- Reid, W., A. H. Goktogan, and S. Sukkarieh (2014). "Moving MAMMOTH: Stable motion for a reconfigurable wheel-on-leg rover." In: *Australasian Conference on Robotics and Automation (ACRA)*.
- Reid, W., F. J. Pérez-Grau, A. H. Göktoğan, and S. Sukkarieh (2016a). "Actively articulated suspension for a wheel-on-leg rover operating on a Martian analog surface." In: *IEEE International Conference on Robotics and Automation (ICRA)*.
- Reid, W., R. Fitch, A. H. Göktoğan, and S. Sukkarieh (2016b). "Motion planning for reconfigurable mobile robots using hierarchical fast marching trees." In: *Workshop on the Algorithmic Foundations of Robotics*.
- Ruckelshausen, A., P. Biber, M. Dorna, H. Gremmes, R. Klose, A. Linz, R. Rahe, R. Resch, M. Thiel, D. Trautz, and U. Weiss (2009). "BoniRob: An autonomous field robot platform for individual plant phenotyping." In: *Precision Agriculture* 9.841, p. 1.
- Satzinger, B. W., C. Lau, M. Byl, and K. Byl (2016). "Experimental results for dexterous quadruped locomotion planning with RoboSimian." In: *Experimental Robotics*. Springer, pp. 33–46.
- Schilling, F., X. Chen, J. Folkesson, and P. Jensfelt (2017). "Geometric and visual terrain classification for autonomous mobile navigation." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Schwarz, M., T. Rodehutsors, D. Droeschel, M. Beul, M. Schreiber, N. Araslanov, I. Ivanov, C. Lenz, J. Razlaw, S. Schüller, D. Schwarz, A. Topalidou-Kyniazopoulou, and S. Behnke (2017). "NimbRo rescue: Solving disaster-response tasks with the mobile manipulation robot Momaro." In: *Journal of Field Robotics (JFR)* 34.2, pp. 400–425.
- Shannon, C. E. (1998). "Communication in the presence of noise." In: *Proceedings of the IEEE* 86.2, pp. 447–457.
- Siegwart, R., P. Lamon, T. Estier, M. Lauria, and R. Piguët (2002). "Innovative design for wheeled locomotion in rough terrain." In: *Robotics and Autonomous Systems* 40.2-3, pp. 151–162.
- Srinivas, A., A. Jabri, P. Abbeel, S. Levine, and C. Finn (2018). "Universal Planning Networks: Learning generalizable representations for visuomotor control." In: *International Conference on Machine Learning (ICML)*.
- Stentz, A. (1995). "The focussed D* algorithm for real-time replanning." In: *International Joint Conferences on Artificial Intelligence (IJCAI)*.

- Stentz, A., H. Herman, A. Kelly, E. Meyhofer, G. C. Haynes, D. Stager, B. Zajac, J. A. Bagnell, J. Brindza, C. Dellin, M. George, Gonzales-Mora J., S. Hyde, M. Jones, M. Laverne, M. Likhachev, L. Lister, M. Powers, O. Ramos, J. Ray, D. Rice, J. Scheifflee, R. Sidki, S. Srinivasa, K. Strabala, J.-P. Tardif, J.-S. Valois, J. M. Vandeweghe, M. Wagner, and C. Wellington (2015). "CHIMP, the CMU highly intelligent mobile platform." In: *Journal of Field Robotics (JFR)* 32.2, pp. 209–228.
- Şucan, I. A. and L. E. Kavraki (2009). "Kinodynamic motion planning by interior-exterior cell exploration." In: *Algorithmic Foundation of Robotics VIII*. Springer, pp. 449–464.
- Takahashi, M., K. Yoneda, and S. Hirose (2006). "Rough terrain locomotion of a leg-wheel hybrid quadruped robot." In: *IEEE International Conference on Robotics and Automation (ICRA)*.
- Tamar, A., Y. Wu, G. Thomas, S. Levine, and P. Abbeel (2016). "Value Iteration Networks." In: *Advances in Neural Information Processing Systems (NIPS)*.
- The Wheel Network (2019). *Hyundai Cradle Walking Car Concept – Robot Demo CES 2019*. <https://youtu.be/hQ5Xib6sFp4>. [Online; accessed 22-June-2019].
- Wermelinger, M., P. Fankhauser, R. Diethelm, P. Krüsi, R. Siegwart, and M. Hutter (2016). "Navigation planning for legged robots in challenging terrain." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Wilcox, B. H., T. Litwin, J. Biesiadecki, J. Matthews, M. Heverly, J. Morrison, J. Townsend, N. Ahmad, A. Sirota, and B. Cooper (2007). "ATHLETE: A cargo handling and manipulation robot for the moon." In: *Journal of Field Robotics* 24.5, pp. 421–434.
- Zhang, H., J. Butzke, and M. Likhachev (2012). "Combining global and local planning with guarantees on completeness." In: *IEEE International Conference on Robotics and Automation (ICRA)*.
- Zhang, L. and D. Manocha (2008). "An efficient retraction-based RRT planner." In: *IEEE International Conference on Robotics and Automation (ICRA)*.
- Zhou, Y. and J. Zeng (2015). "Massively parallel A* Search on a GPU." In: *AAAI Conference on Artificial Intelligence*.
- Ziaei, Z., R. Oftadeh, and J. Mattila (2014). "Global path planning with obstacle avoidance for omnidirectional mobile robot using overhead camera." In: *IEEE International Conference on Mechatronics and Automation (ICMA)*.
- Zucker, M., S. Joo, M. X. Grey, C. Rasmussen, E. Huang, M. Stilman, and A. Bobick (2015). "A General-purpose system for teleoperation of the DRC-HUBO humanoid robot." In: *Journal of Field Robotics (JFR)* 32.3, pp. 336–351.
- de Viragh, Y., M. Bjelonic, C. D. Bellicoso, F. Jenelten, and M. Hutter (2019). "Trajectory optimization for wheeled-legged quadrupedal

robots using linearized ZMP constraints." In: *IEEE Robotics and Automation Letters (RA-L)* 4.2, pp. 1633–1640.