The EU Framework Programme for Research and Innovation H2020
Research and Innovation Action

Deliverable D5.4 CENTAURO Walking Navigation

Dissemination Level: Public

Project acronym:	CENTAURO
Project full title:	Robust Mobility and Dexterous Manipulation in Disaster Response by Fullbody Telepresence in a Centaur-like Robot
Grant agreement no.:	644839
Lead beneficiary:	UBO – University of Bonn
Authors:	T. Klamt, X. Chen, H. Karaoguz, S. Behnke
Work package:	WP5 Navigation
Date of preparation:	2018-10-08
Type:	Report
Version number:	1.0

Document History

Version	Date	Author	Description
0.1	2018-05-16	TK	First draft
0.2	2018-06-05	TK	Add experiment sections
0.3	2018-06-25	XC	Update cloud stair detection results, update additional work
0.4	2018-06-26	TK	Spell check and minor corrections
0.5	2018-06-28	TK	Add missing section about UBO - KTH integration
0.6	2018-07-02	TK	Changes after discussion with Sven Behnke
0.7	2018-07-03	XC	Updated figures in terrain classification method and evaluation
0.8	2018-08-06	TK	Incorporate LIU review
0.9	2018-10-08	TK	Incorporate IIT review
1.0			Submitted version

Executive Summary

This deliverable describes extensions to the CENTAURO locomotion planning pipeline. One of the outstanding features of the CENTAURO platform is the hybrid driving-stepping locomotion capability. To address this in a planning approach, it is not sufficient to handle each locomotion type separately. A suitable navigation planner rather needs to address all the robot capabilities in a single planning framework. The basic concept for such a planner was presented in Deliverable D5.3 CENTAURO Driving Navigation [3]. This deliverable focuses on extensions of the approach, in particular stair detection and planning on multiple levels of abstraction. We also report additional work on learning navigation skills. The integration with the CENTAURO robot platform is described in detail. The semantic terrain perception and hybrid wheeled-legged locomotion planning are now ready for the final evaluation of the integrated CENTAURO disaster-response system.

Contents

1 Introduction

This deliverable reports on the extensions of the navigation planning system used for the Centauro robot within the CENTAURO project. One of Centauro's unique features is its hybrid wheeled-legged lower body design and the corresponding locomotion capabilities it provides: Four legs with five degrees of freedom (DoF) each, ending in 360° steerable wheels. This enables the robot to perform omnidirectional driving locomotion as well as walking locomotion. Moreover, the robot can perform manoeuvres, which can neither be performed by pure driving nor by pure walking robots, such as moving individual feet while keeping ground contact. To take those capabilities into account when planning locomotion, the planning component needs to consider and integrate the driving and walking mobility of the robot together in the locomotion planning problem. This planning component is already described in "Deliverable D5.3 CENTAURO Driving Navigation" [3]. Fig. 1 visualizes the system architecture. This "Deliverable D5.4 CENTAURO Walking Navigation" reports on modifications and extension to the components *Terrain Classification* and *Hybrid Path Planner*. The *Terrain Classification* module is extended by geometrical-based and image-based stairs detection to increase the semantics in typical indoor environments. The *Hybrid Path Planner* is improved in its capability to generate abstract representations.

Further additional work which has been performed in the context of this work package is described in C.

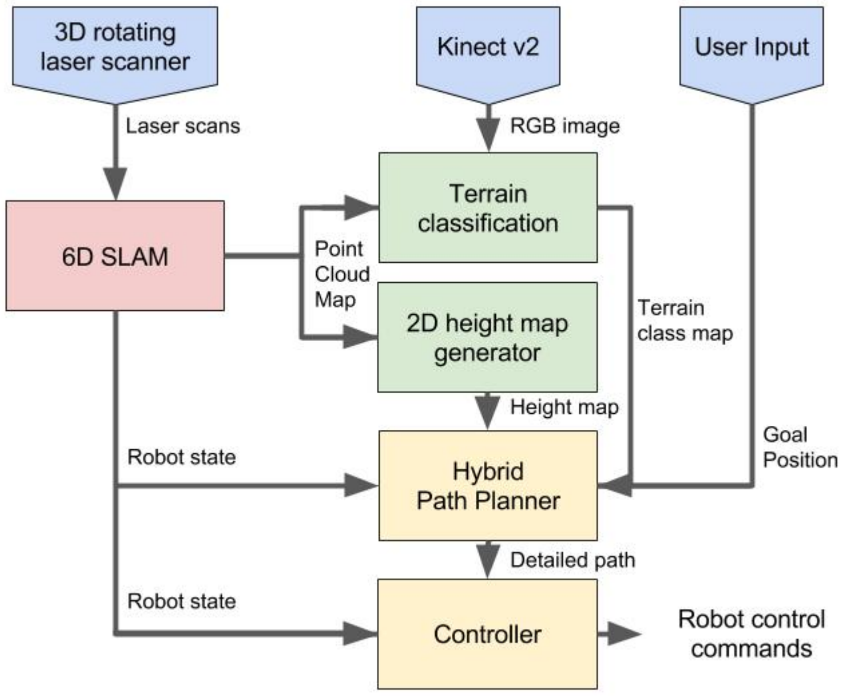


Figure 1: CENTAURO locomotion planning system architecture.

2 Method Extensions

Different parts of the locomotion planning pipeline are improved. Detection is enriched by stairs detection, both from point clouds and RGB images. This increases the semantics of the environment representation, especially for typical indoor environments. The search-based planner is improved in its capability to generate abstract representations.

2.1 Stairs Detection

To enrich the output of the *Terrain Classification* component, stairs detection modules are added. Two methods are used which rely on RGB images (see Sec. 2.1.1) and geometric point clouds (see Sec. 2.1.2). The results are merged with the other terrain classification results in a random forest classifier, as described in ”Deliverable D5.2 CENTAURO Terrain Classification” [1].

2.1.1 Image-based Stairs Detection

For detecting stairs using image data, we have employed a Dilated Residual Network (DRN) [9]. The reason for using a DRN compared to a regular CNN architecture is that DRN has more detailed activation maps compared to a regular CNN architecture. This is shown in Fig. 2. As we go deeper in the regular ResNet, the activation maps get coarser. However, in DRN architecture, thanks to the dilation operation, the activation maps do not shrink. As a result the performance of the DRN is better than the regular ResNet without a big memory penalty.

For stairs detection, we have used the 54-layer Encoder/Decoder architecture of DRN. We have used 100 images for training and 20 images for validation. Data augmentation is applied on the fly to increase the training data variability. Moreover we have added class-specific weights in the loss function to improve the segmentation result. We have setup the stairs detection problem as pixel-wise binary classification. A pixel can be labeled either as a stairs or as background.

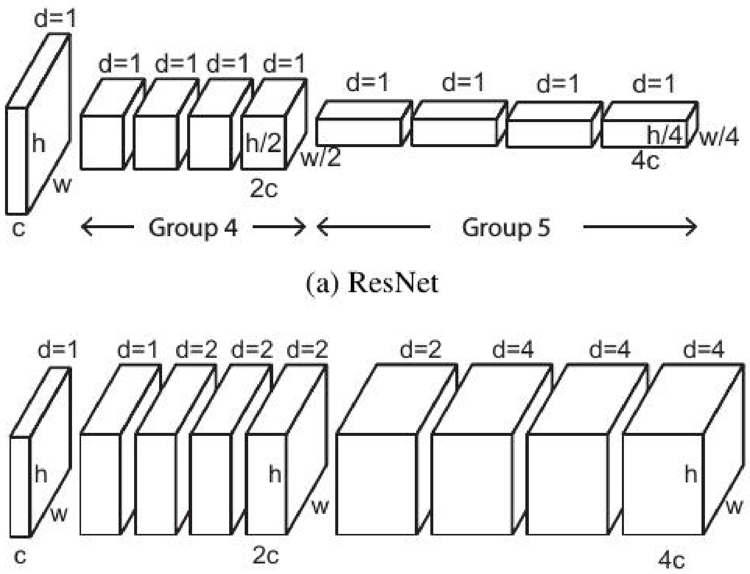


Figure 2: The architectural difference between a regular ResNet and DRN [9].

The output of this network is a pixel-wise probability map of an input image. Some of the obtained results are given in Fig. 3. Here, the pink color represents the pixel with stairs label while the purple color represents the background.

2.1.2 Point Cloud based Stairs Detection

The method receives the full registered point cloud as its input. Initial preprocessing steps include voxel grid filtering to downsample and passthrough filtering to reduce the field of view. The resulting cloud is fed into a RANSAC plane segmentation method that extracts horizontal planes from the scene. The next step is to extract certain metrics like width, depth, height and orientation in the world frame. This is achieved through Principal Component Analysis of the found planes, the results are used to filter out planes that can be ignored, e.g., planes big enough to be driven on or planes not wide for the robot. By searching through all the metrics of the planes, cases that fit the model of a staircase are labeled as staircases. The pipeline for stairs detection is shown in Fig. 4.

A fit is considered reasonable if the planes are similar in width and depth, and have constant height difference between consecutive steps. Additionally, in cases where the vertical planes on a staircase are visible, they are used to obtain and verify if the orientation of each step matches the normals of the vertical planes. Fig. 5 and Fig. 6 show the detection examples for detecting upstairs and downstairs cases. In Fig. 5 the robot is at the foot of the stairs, the resulting point cloud in these cases was found to not have reliable planes (insufficient point density) after the first few steps. To mitigate this, after a hypothesis (step metrics constituting a staircase) is generated with three horizontal steps, vertical planes are used to test it. In Fig. 6, this extra step is not necessary and the results obtained only with horizontal planes are satisfactory.

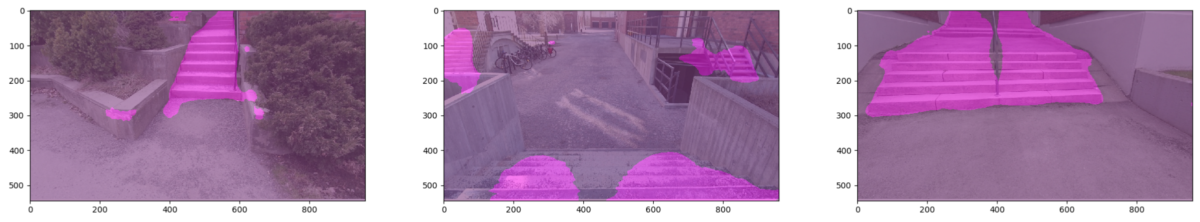


Figure 3: Example stairs detection results.

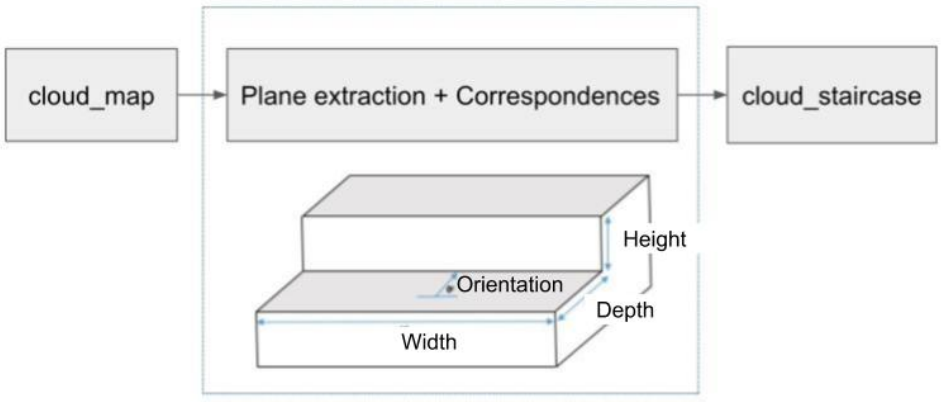


Figure 4: Pipeline for stairs detection using point cloud.

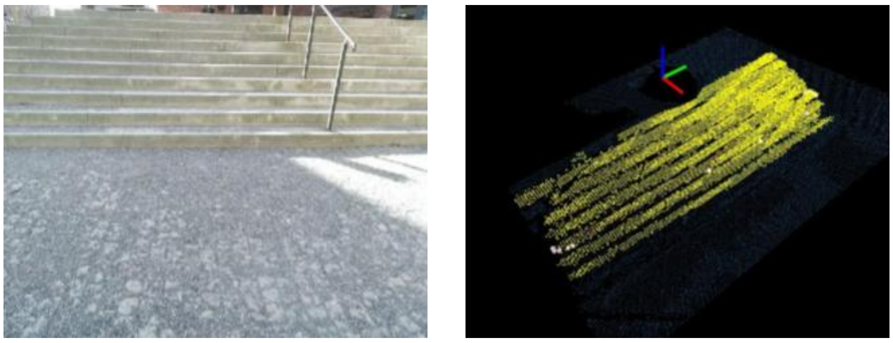


Figure 5: Example of upstairs detection. The left image shows the raw RGB image from Kinect and the right image shows the stairs detected from the point cloud.

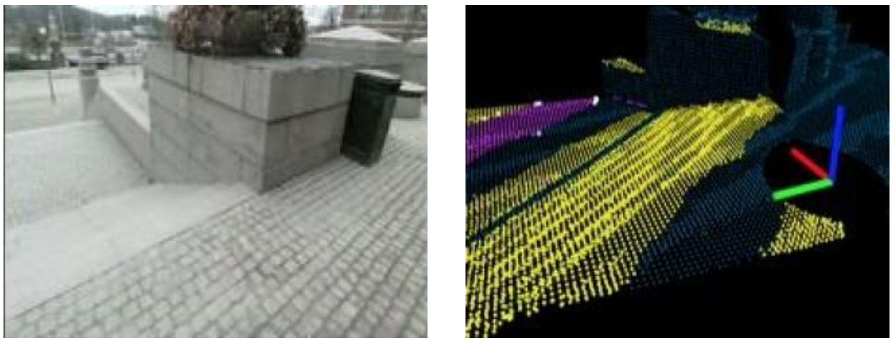


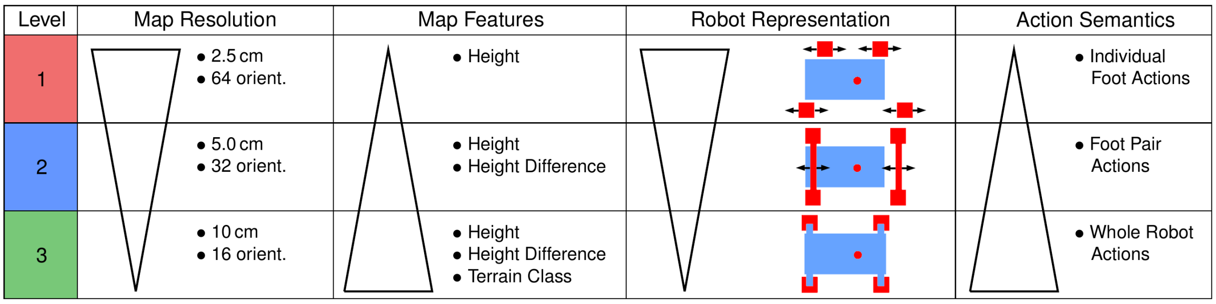
Figure 6: Example of downstairs detection. The left image shows the raw RGB image from Kinect and the right image shows the stairs detected from the point cloud.

2.2 Search-based Planning on Multiple Levels of Representation

In ”Deliverable D5.3 CENTAURO Driving Navigation” [3] we described the general approach of a search-based planner for hybrid driving-stepping locomotion. To enable this planner to plan for larger scenarios, we presented an extension to plan on multiple levels of representation: With increasing distance from the robot, the environment and action resolution gets coarser. In addition, the number of DoF of the robot representation which we call *representation dimensionality* decreases with increasing distance from the robot. Such abstraction comes along with a loss of information, which we compensate by enriching the environment representation with additional semantic features such as height differences and terrain classes. Hence, while the representation gets less precise, the understanding of the scene increases. Experiments show that such planning on multiple levels of representation significantly accelerates planning by multiple orders of magnitude and thus enables the planner to solve for larger scenarios. The concept of the different levels of representation is visualized in Fig. 7 and Fig. 8.

In the detailed planning, the robot is represented with seven DoF (degrees of freedom) (x, y, θ for the robot base, $f_1, ..., f_4$ for individual relative longitudinal foot positions). This representation is referred to as *Level 1* representation. As described in ”Deliverable D5.3 CENTAURO Driving Navigation” [3], two abstractions are applied to the robot representation to generate representations for the more abstract representations *Level 2* and *Level 3*:

- *Level 2*: Combine the two front and the two rear feet to foot pairs which results in a 5D representation (x, y, θ, f_f, f_r).
- *Level 3*: Fix all feet to the robot base and neglect their individual configuration which results in a 3D representation (x, y, θ).



Level	Map Resolution	Map Features	Robot Representation	Action Semantics
1	• 2.5 cm • 64 orient.	• Height		• Individual Foot Actions
2	• 5.0 cm • 32 orient.	• Height • Height Difference		• Foot Pair Actions
3	• 10 cm • 16 orient.	• Height • Height Difference • Terrain Class		• Whole Robot Actions

Figure 7: The search-based planner includes three levels of representation with decreasing resolution and robot configuration dimensionality. To compensate the loss of information, the semantics for both the environment representation and the robot actions increase.

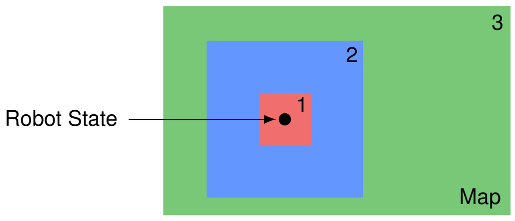


Figure 8: Size and position of the different levels of representation. *Level 1* covers the vicinity of the robot. *Level 2* is also robot centered and medium sized. *Level 3* covers the whole map.

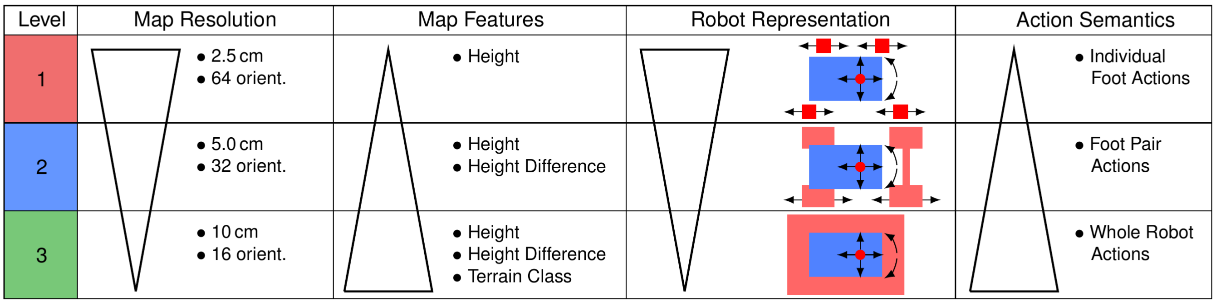
Unfortunately, those abstract representations cause restrictions to the feasibility of certain robot configurations. E.g., if the robot should step up on a height difference which is not aligned with the robot orientation, the *Level 2* representation could estimate this situation as infeasible because it could not find a solution with identical longitudinal positions for two neighboring feet while there exists a feasible solution for this scenario in *Level 1*. Similar, the *Level 3* representation neglects the detailed foot position and instead assumes them to be fixed. Again, such simplification leads to situations in which the *Level 3* representation assesses a scenario as not traversable while the *Level 1* representation finds a path. A modification of the abstract robot representations is presented in Sec. 2.2.1.

A modification of the used heuristic comes along with an updated robot representation and is described in Sec. 2.2.2. Moreover, we present a method for continues refinement which accelerates replanning during path execution (see Sec. 2.2.3).

2.2.1 Robot Representation for Search-based Planning on Multiple Levels of Representations

We propose a modification to the concept of planning on multiple levels of representation to solve the above mentioned problems. Similar to the previous work, the robot representation gets more abstract by combining the two front/rear feet to foot pairs or by completely neglecting the individual foot positions. To compensate for the loss of information which comes along with this abstraction, we introduce foot areas: Instead of planning with the detailed foot positions, we define areas in which the feet could be placed, and consider those in the planning.

In the *Level 2* robot representation, the two front and the two rear feet are still combined to foot pairs which results in a 5-dimensional robot representation. For each foot a certain foot area is determined. In the *Level 3* robot representation, one large area around and under the robot is defined in which all four feet are positioned. The action cost functions for the abstract representations are tuned manually so that they match closely to the *Level 1* costs of the same



Level	Map Resolution	Map Features	Robot Representation	Action Semantics
1	• 2.5 cm • 64 orient.	• Height		• Individual Foot Actions
2	• 5.0 cm • 32 orient.	• Height • Height Difference		• Foot Pair Actions
3	• 10 cm • 16 orient.	• Height • Height Difference • Terrain Class		• Whole Robot Actions

Figure 9: Three levels of representation with updated robot representations.

actions (described in [6]). A set of simple tasks is used for this parametrization. Fig. 9 visualizes the new proposed robot representations.

A further problem of the abstract robot representations described in ”Deliverable D5.3 CENTAURO Driving Navigation” [3] can be found when planning to overcome stairs in *Level 3*. In the current state, the planner only considers foot movements in longitudinal direction. Thus, the robot is not able to climb stairs sideways. This property is considered in *Level 1* and *Level 2* where only longitudinal steps are considered in the planning. However, *Level 3* describes such a situation more abstract. The robot moves over the terrain while costs are generated by the terrain classes in the foot area under and around the robot. Individual foot actions are not considered. Hence, it is possible to plan paths in which the robot climbs stairs laterally, which cannot be performed by the real robot due to restricted leg reachability.

This is solved by enriching the *Level 3* terrain class features with additional information about step orientation. First, we search for cells of the terrain type *step*. This is done by searching for cell pairs c_i and c_j that fulfill the following criteria:

- $\triangle H(c_i) < \triangle H_{\text{max,drive}}$: c_i is on a drivable surface,
- $\triangle H(c_j) < \triangle H_{\text{max,drive}}$: c_j is on a drivable surface,
- $\|c_{\text{i}} - c_j\| < 0.45\,\text{m}$: 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, $C_{\text{GC}}(c_k) = \infty$ counts for all cells $c_k \in T$: A direct foot movement from c_i to c_j requires a step,

where $\triangle H$ describes the height difference at a given cell and $\triangle H_{\text{max,drive}}$ is the maximum allowed height difference for a drivable surface. C_{GC} describes the ground contact costs of a given cell. For all pairs of c_i and c_j which fulfill these criteria, each cell $c_s \in c_i \cup c_j \cup T$ is assigned the terrain class *step*. In addition, we compute the angle $\alpha_{\text{i,j}}$ between c_i and c_j and save it for c_s. Since most *step* cells are detected several times, we collect several angles for each cell. $\alpha_{\text{avg,s}}$, the *mean of circular quantities* of these angles describes the estimated step orientation in c_s. Fig. 10 b shows a terrain class map with visualized step orientations.

When moving over *step* cells, a robot state is only feasible if the difference between the robot orientation and the step orientation of each *step* cell c_r is less than one discrete orientation step: $\text{abs}(\alpha_{\text{avg,r}} - r_\theta) < \frac{1}{16} \cdot 2\pi$. Moreover, the robot is only allowed to move parallel and orthogonal to step orientations. These restrictions are required to enforce a behavior, which is induced by the robot kinematic in lower representation levels but not represented in *Level 3* otherwise.

The performance of this new approach is evaluated in Section 3.2. Our earlier work [5] describes the search-based planning for hybrid driving-stepping locomotion. The concept to extend this to plan on multiple levels of representations is described in detail in [6].

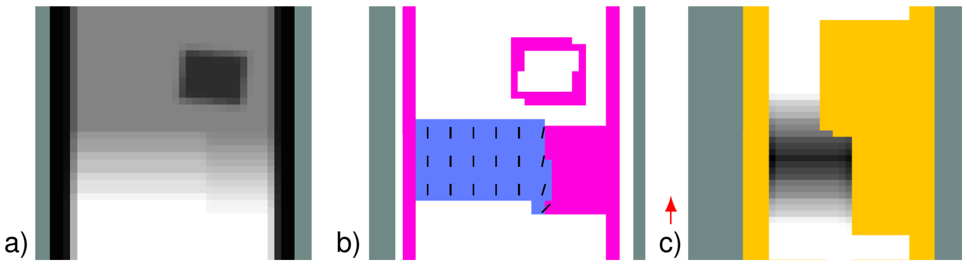


Figure 10: *Level 3*: a) height map, b) terrain class map (white = flat, blue = stepping, pink = wall, black lines = step orientations), c) robot area cost map for the orientation indicated by the red arrow.

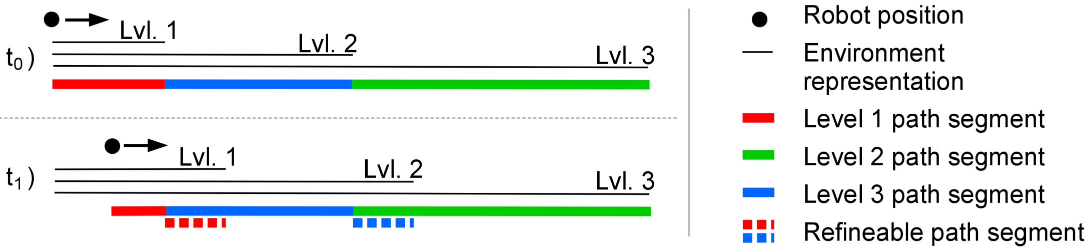


Figure 11: As the robot moves along the path, the *Level 1* and *Level 2* representations move with it. Consequently, those path segments which are represented in a higher level and for which a more detailed representation becomes available, can be refined to this more detailed representation.

2.2.2 Heuristic Based on an Abstract Representation

In ”Deliverable D5.3 CENTAURO Driving Navigation” [3] we proposed a heuristic which was based on the abstract *Level 3* representation and showed very promising results. Since the *Level 3* robot representation changed, the heuristic had to be updated as well.

After the goal pose $\vec{r}_{i,G}$ is set, it is transformed to *Level 3*. We then start a one-to-any 3D Dijkstra search in *Level 3* starting from $\vec{r}_{3,G}$. Hence, we get for each *Level 3* pose in the map an estimation of the costs to reach the goal pose. During path planning, we can estimate the costs from any robot pose to the goal by transforming it to *Level 3* and get the respective cost value.

Note that the quality of this heuristic strongly depends on the quality of the *Level 3* cost model in comparison to costs for the same manoeuvres in other levels of representation. Further note that we cannot prove that this heuristic always underestimates costs, which is necessary to prove admissibility for the generation of optimal paths. However, since we also utilize the sub-optimal ARA* algorithm, we do not aim to generate optimal paths for a given problem. In fact, we focus on generating paths with satisfying quality in feasible time. The performance of this heuristic is evaluated in Section 3.3.

2.2.3 Continuous Refinement

As the robot moves along the initial path, the sensors provide new measurements and high-detailed environment representations are generated in the vicinity of the current robot position. We include these updated representations in the path by continuously refining the respective path segments, as shown in Fig. 11. If a cost difference $> 25\%$ between the original and the refined path segments indicates that the higher-level planning assessed a situation wrongly, we initiate a new planner run. With this approach, we can guarantee that path segments in the vicinity of the robot are always represented in *Level 1* and thus, included steps can be expanded and the result can be executed by the controller.

3 Evaluation

3.1 Terrain Classification

Data collection efforts were made to evaluate the system as a whole. The trials were performed both at KTH and UBO. The terrain classification was tested on both recorded data sets.

3.1.1 Data Collection at KTH

In order to train the image and the cloud based stairs detectors, additional data was required. Therefore, the Pluto robot at KTH RPL Lab was modified to have a similar sensory setup as the Centauro robot. Fig. 12 shows some data collection efforts at KTH. After three days of data collection at different areas of Stockholm, data from 64 different scenes, which correspond to 80 gigabytes of data, has been collected. The data consists of point clouds, RGB frames and robot related information.

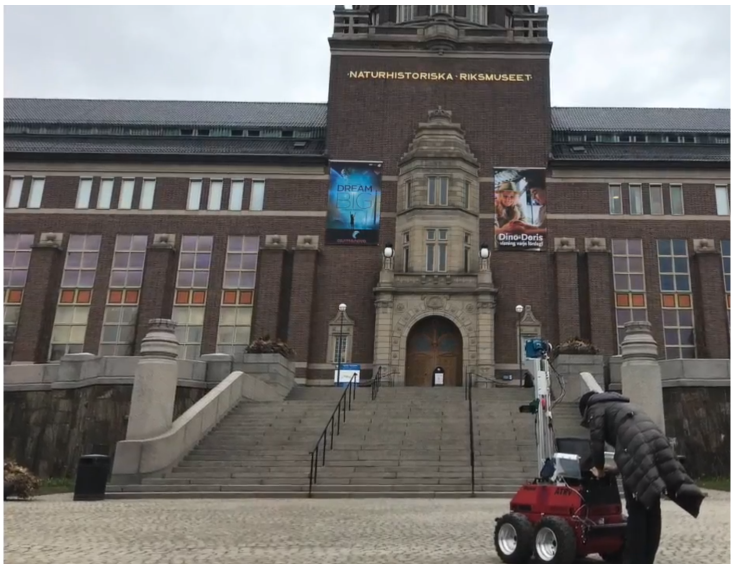


Figure 12: A scene from data collection efforts at KTH.

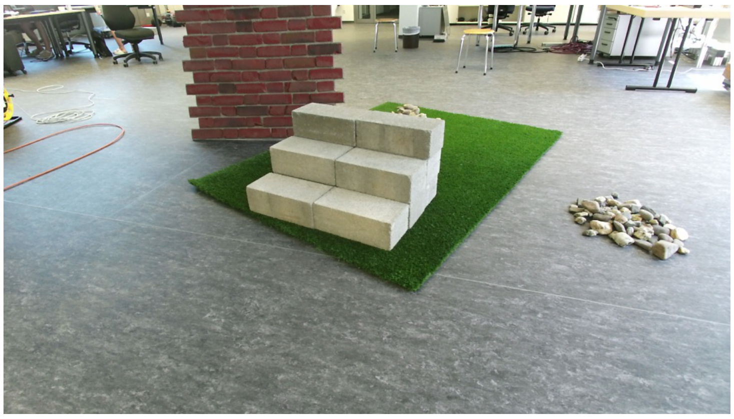


Figure 13: Example scenery at UBO.

Table 1: Obtained results for KTH data set.

No. of training images	No. of test images	Mean class accuracy on test data (%)
30	5	92
25	10	88
20	15	87
15	20	85

3.1.2 Data Collection at UBO

During the integration efforts between KTH and UBO, significant amount of data has been collected at UBO. The Momaro platform, which has an identical sensor head as the Centauro robot with a rotating *Velodyne Puck* laser scanner and a *Microsoft Kinect V2*, was used. In order to collect relevant sensory data, an artificial terrain scenario given in Fig. 13 has been created. Then the robot was positioned at different locations and distances in order to collect data from varying fields of view. In addition, data from staircases located in the building has been collected.

In total, 29 ROS bags have been recorded which corresponds to 21 gigabytes of data. The data includes laser scanner point clouds, image and depth frames from Kinect, and robot related information such as joint states.

3.1.3 Terrain Classification Results – KTH Data Set

We have evaluated our terrain classification pipeline on KTH data set. We have randomly shuffled the number of training and test data with different ratios to fully evaluate the method. The training data was always different from the test data. Our obtained results based on mean class accuracy is given in Table 1. From the table, it is observed that even though the training data ratio decreases significantly, the mean class accuracy stays over 85%. Fig. 14 shows processing steps of the terrain classification pipeline for an example image.

3.1.4 Terrain Classification Results – UBO Data Set

Our terrain classification pipeline is evaluated also on the UBO data set. In this data set, 22 scenes were selected as training data while 7 were selected as test data. The obtained mean class accuracy for this setting is 89%. Fig. 15 shows some example results obtained from this data set. From the results, it is observed that irregular stair configurations such as the one shown in Fig. 15 top row, can cause the stairs detection system to fail. Nevertheless, most of the remaining parts of the stairs are seen as obstacle, which makes sense according to the given circumstances.

3.2 Planning on Multiple Levels of Representation

The planning performance of the different levels of representation is tested in an indoor scenario. It is shown in Fig. 16. Planning queries are executed on each level of representation individually to compare the quality of each representation. In addition, planning on combined levels of representation, as shown in Fig. 8, is tested. For this, we choose the *Level 1* size to be 3×3 m. This is sufficiently large to plan the next robot manoeuvres in high detail, but still small enough to avoid long high-dimensional planning. The *Level 2* size is chosen to be 9×9 m so that

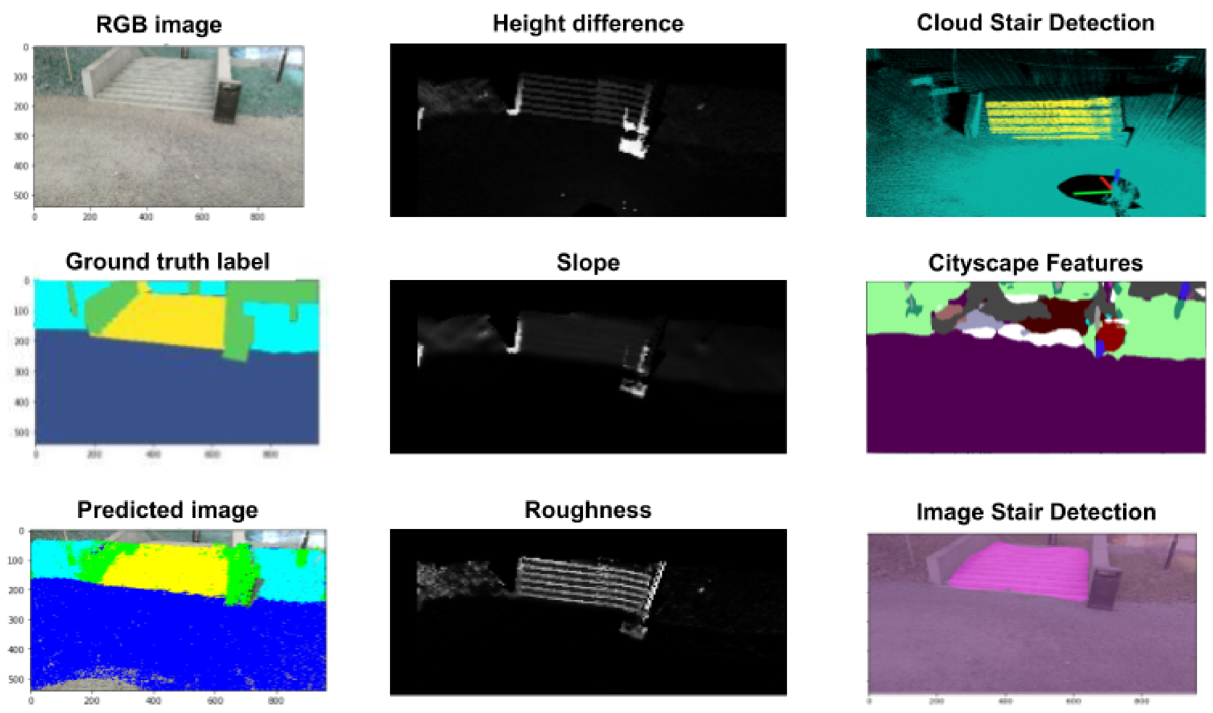
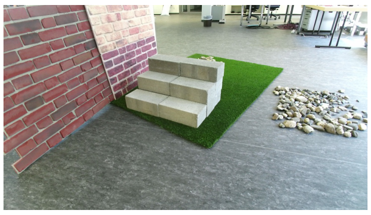
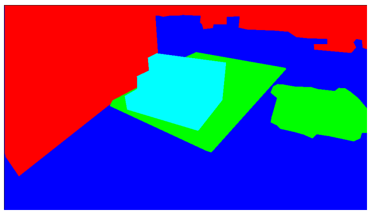


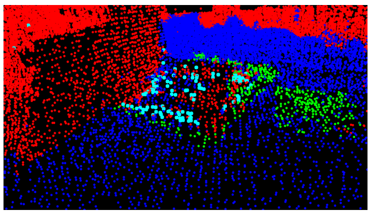
Figure 14: The processing steps of the terrain classification pipeline for an example image.



(a) Image

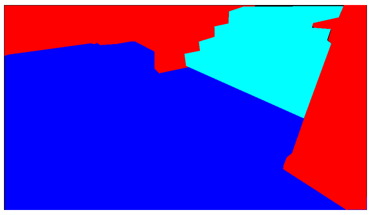


(b) Ground truth labels

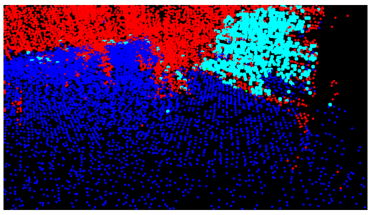


(c) Predicted labels

(d) Image



(e) Ground truth labels



(f) Predicted labels

Figure 15: Example terrain classification results from UBO data set.

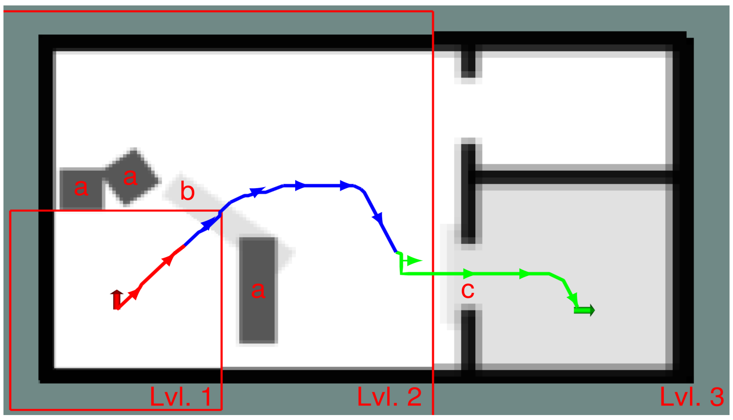


Figure 16: Height map of the first experiment scenario. From its start position (red arrow), the robot needs to navigate between multiple objects (a), over a bar obstacle (b), step up to an elevated platform and through a door (c) to the goal pose (green arrow). The resulting path for $\mathcal{W} = 1.125$ and combined levels of representation is shown. *Level 1* path segments = red, *Level 2* segments = blue, *Level 3* segments = green. Arrows show r_θ.

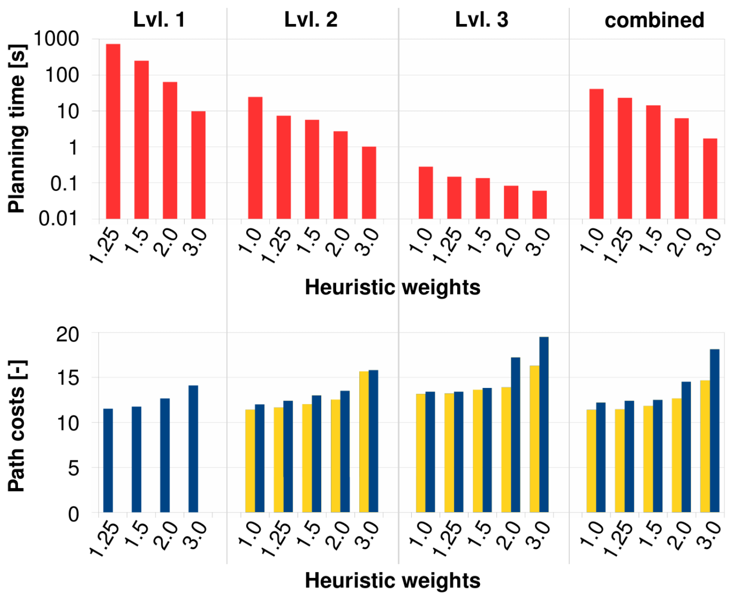


Figure 17: Planning performance for different levels and different $\mathcal{W}$ for the first experiment. estimated costs = yellow, refined *Level 1* costs = blue.

the *Level 2* path segment is about twice as long as the *Level 1* path segment. The used heuristic is the *Euclidean heuristic*, a combination of the Euclidean distance and orientation difference. Since we use an ARA* algorithm which works with several heuristic weights $\mathcal{W}$, we evaluate the influence of these. Fig. 17 shows the planner performance.

All experiments are done on one core of a 2.6 GHz Intel i7-6700HQ processor using 16 GB of memory. An additional video is available online[1] which also contains a Gazebo experiment to demonstrate the continuous refinement strategy.

It can be seen that planning on levels of representation >1, and with combined levels, is faster by at least one order of magnitude compared to pure *Level 1* planning. The *Level 1* path for $\mathcal{W} = 1.0$ could not be computed due to memory limitations. We distinguish between the

[1]`https://www.ais.uni-bonn.de/videos/ICRA_2018_Klamt/`

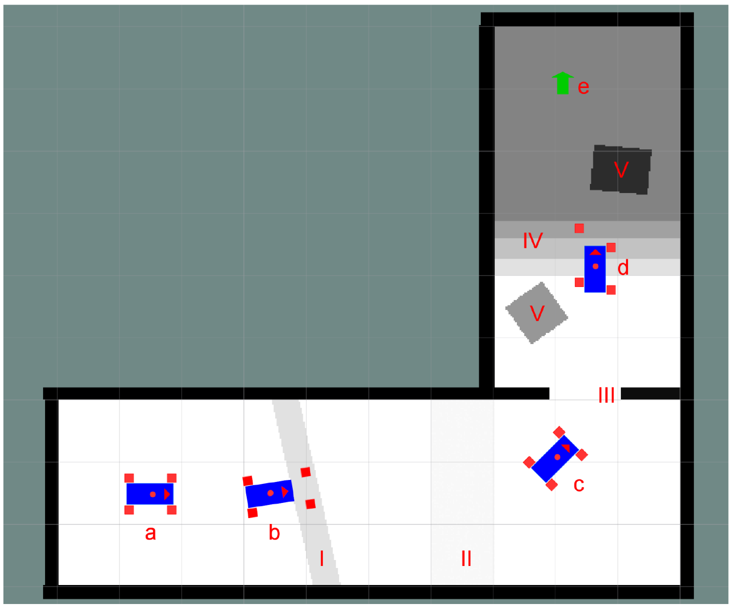


Figure 18: Height map for the second experiment containing a bar obstacle (I), a rough area (II), a door (III), a flight of stairs (VI) and two obstacles (V). a - d are different starting poses for the planner, e is the goal pose.

path costs in the respective levels of representation (estimated cost) and the costs each path carries when refined to *Level 1*. Comparing the estimated costs to the refined *Level 1* costs gives an assessment about the quality of cost generation in each level of representation. The comparison of the refined *Level 1* costs to the original *Level 1* costs indicates the quality of the resulting path. It can be seen that the estimated costs always underestimate the refined *Level 1* costs. Especially for $\mathcal{W} \leq 1.5$ the estimation is close with a difference $\leq 7.7\%$. Furthermore, the results show that for $\mathcal{W} \leq 1.5$ the refined *Level 1* costs differ to the original *Level 1* costs by $\leq 15\%$.

3.3 Abstract Representation-based Heuristic

The *Dijkstra heuristic*, presented in Sec. 2.2.2, is compared to the *Euclidean heuristic* in a larger and more challenging scenario, shown in Fig. 18. The robot starts at pose *a*. Planning is performed on combined levels of representation. A resulting path is shown in Fig. 19. Planning times and resulting costs are shown in Fig. 20. Preprocessing the *Dijkstra heuristic* took 0.52 s which is already included in the presented planning times.

It can be seen that the *Dijkstra heuristic* further accelerates planning while the resulting costs stay comparable at least for $\mathcal{W} \leq 1.5$. E.g., 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.

We finally compare the planner performance when started from different poses, as shown in Fig. 18. The results in Fig. 21 indicate that an important factor for the planner performance is the complexity of the planning within *Level 1*, but higher $\mathcal{W}$ lead to feasible performances in any case.

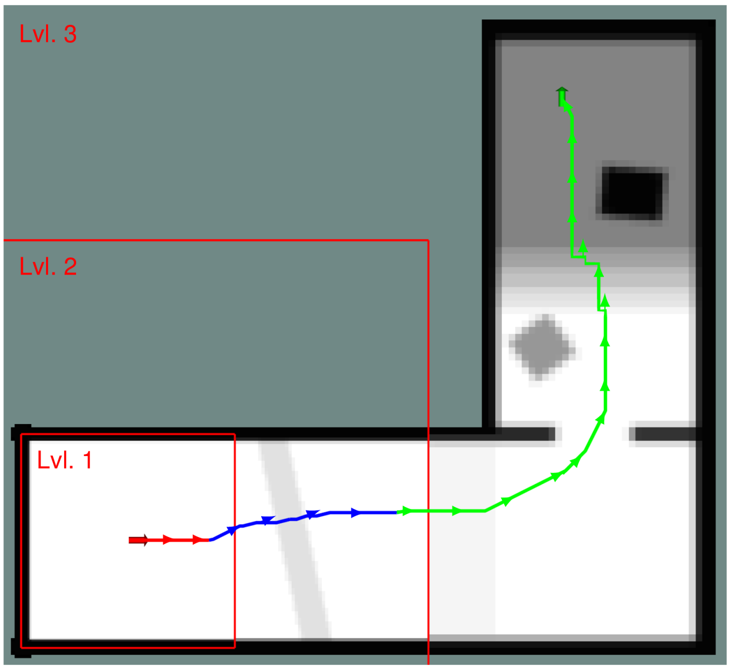


Figure 19: Resulting path for planning with the *Dijkstra heuristic* and combined levels with $\mathcal{W} = 1.25$.

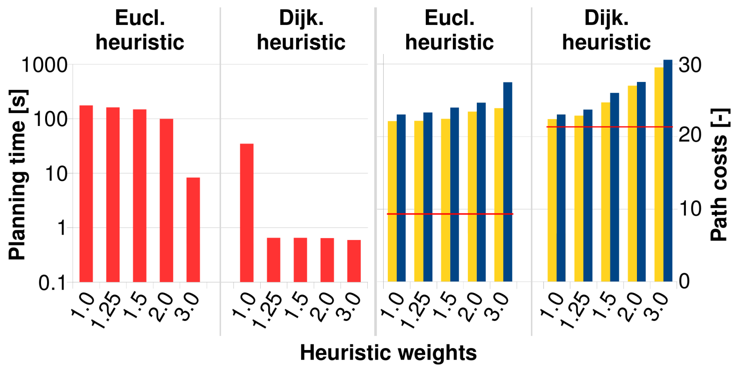


Figure 20: Planning performance for combined levels of representation to compare the *Euclidean heuristic* with the *Dijkstra heuristic*. Red lines indicate the cost estimation for the path by each heuristic.

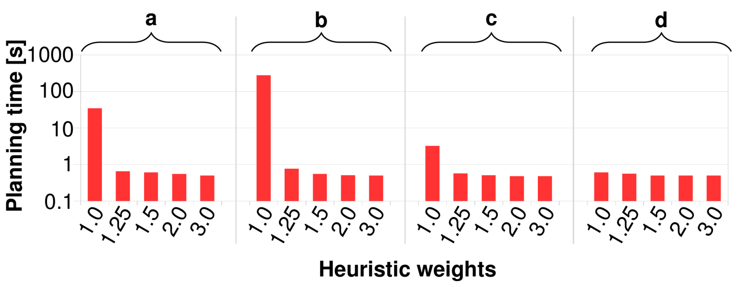


Figure 21: Planning time for different starting poses (see Fig. 18) and different $\mathcal{W}$, using the *Dijkstra heuristic*.

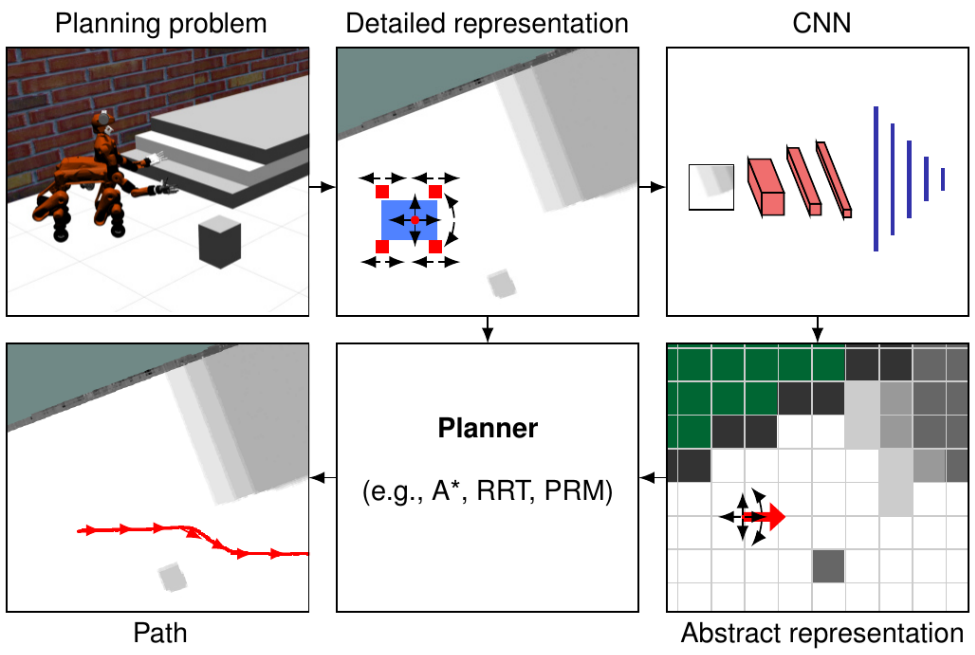


Figure 22: Pipeline overview of the approach to generate abstract representations with a CNN to support planning.

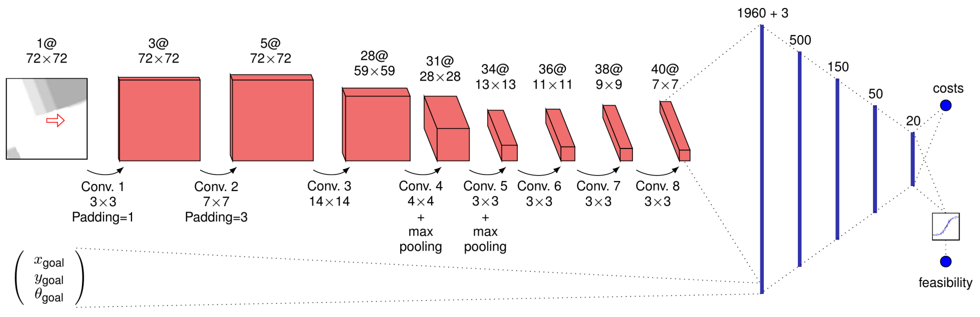


Figure 23: Architecture of the proposed CNN to represent the environment and cost function of an abstract planning representation.

3.4 Learning Abstract Representations for Locomotion Planning in High-dimensional Configuration Spaces

Planning hybrid driving-stepping locomotion on multiple levels showed promising results since planning was accelerated by multiple orders of magnitude while path quality stayed comparable. Especially the employment of the most abstract representation as an informed heuristic was helpful. However, the generation of abstract environment representations and cost functions requires extensive manual tuning. Hence, it is an idea to use CNNs to represent abstract representations. While a desired robot representation and action set can be easily defined, the tuning-intensive environment representation and cost function are represented as a CNN. Figure 22 gives an overview over the pipeline and Fig. 23 shows the CNN architecture.

Input to the CNN is a height map patch of a fixed size which represents the environment in the vicinity of the robot. The local robot start pose is defined to be always in the center of this map patch with a fixed orientation. In addition a desired local 3D goal pose (x, y, yaw of the robot base) is input to the network. The network outputs a *feasibility* value which indicates if a feasible path from the robot start pose to the robot goal pose exists. In addition, the network

Table 2: Abstraction quality evaluation

	random	***simulated***	***real***
Ø $\mathcal{C}_\mathrm{d}$	0.476	0.466	0.509
Std. dev.($\mathcal{C}_\mathrm{d}$)	0.222	0.202	0.236
feasibility correct, CNN	95.04%	96.69%	92.62%
Ø $\mathcal{C}_\mathrm{a,CNN}$	0.453	0.469	0.446
Ø Error($\mathcal{C}_\mathrm{a,CNN}$)	0.027	0.013	0.081
feasibility correct, man.tuned	79.27%	65.35%	69.77%
Ø $\mathcal{C}_\mathrm{a,man.tuned}$	0.435	0.402	0.429
Ø Error($\mathcal{C}_\mathrm{a,man.tuned}$)	0.057	0.021	0.103

outputs a *cost* estimation which describes the expected costs in case a feasible path exists.

To generate a heuristic, a 3D one-to-any Dijkstra search is started at the global goal pose set by the operator. It generates a cost estimation from each 3D base pose in the map to the global goal pose while using the coarse resolution of the abstract representation and neglecting foot configurations. This cost estimation to the goal can be used as a powerful and informed heuristic for planning in the detailed representation.

Training of the network is done on randomly generated artificial data, but it generalizes well to the abstraction of real world scenes. Table 2 compares the abstraction quality of this approach to the manually tuned abstract representation described above. The performance is measured on three datasets containing randomly generated artificial data (different from the training data), height maps from simulation, and height maps which come from real world laser scanner measurements. The CNN outperforms the manually tuned approach on all three datasets. More details can be found in [4], which is attached to this document.

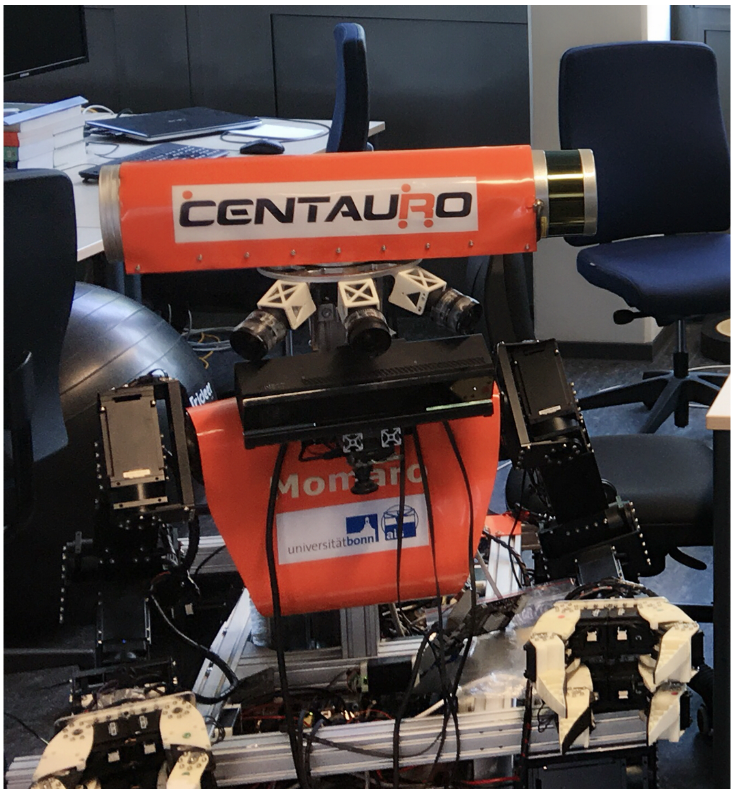


Figure 24: Sensor head of the mobile manipulation robot Momaro, built by UBO. It possesses a rotating *Velodyne Puck* laser scanner with spherical field of view, a *Microsoft Kinect V2* and three *Pointgrey* cameras. The sensor head setup is similar to the Centauro sensor head.

4 WP5 Integration

4.1 Integration of the Terrain Classification and the Locomotion Planner

During a bilateral integration meeting between UBO and KTH in Bonn, the terrain classification module was integrated with the hybrid locomotion planner module. Data was collected with the sensor head of the Momaro robot, which is similar to the Centauro sensor head (see Fig. 24).

The data pipeline for the environment representation for navigation planning is visualized in Fig. 25 with example data. Two environment representations had to be fused:

- The UBO navigation environment representation, which uses point clouds and generates a 2D height map with high resolution (2.5 cm), and
- the KTH terrain classification representation which uses RGB images from the *Kinect V2* and point clouds to generate a terrain class map with a resolution of 5 cm.

Both maps have been fused in the cost map representation successfully, which is used for navigation planning. It can be seen that the height map causes a detailed traversability assessment while the terrain class map enriches this representation with additional information (e.g., risky assessment for grass or gravel). Moreover, it can be seen that the terrain class map shows inconsistencies and does not assess given areas of the same terrain class (e.g., grass) continuously which needs to be improved.

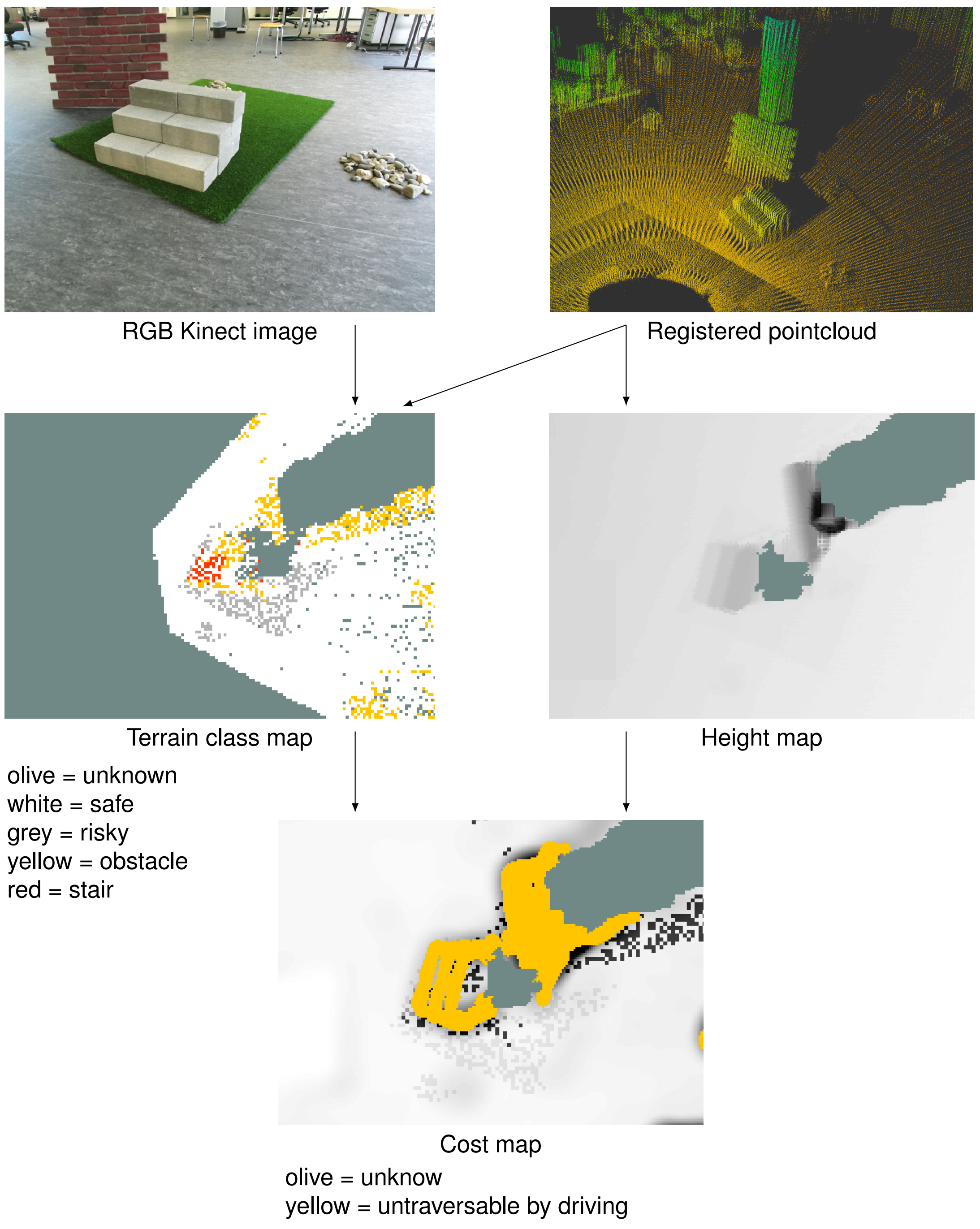


Figure 25: Data pipeline for navigation planning environment representation. Input are *Kinect V2* RGB images and registered point clouds from the Momaro sensor head. Both are fed into the terrain classification module which outputs a terrain class map. In parallel a 2D height map is generated from the point clouds. Both maps are fused to a cost map.

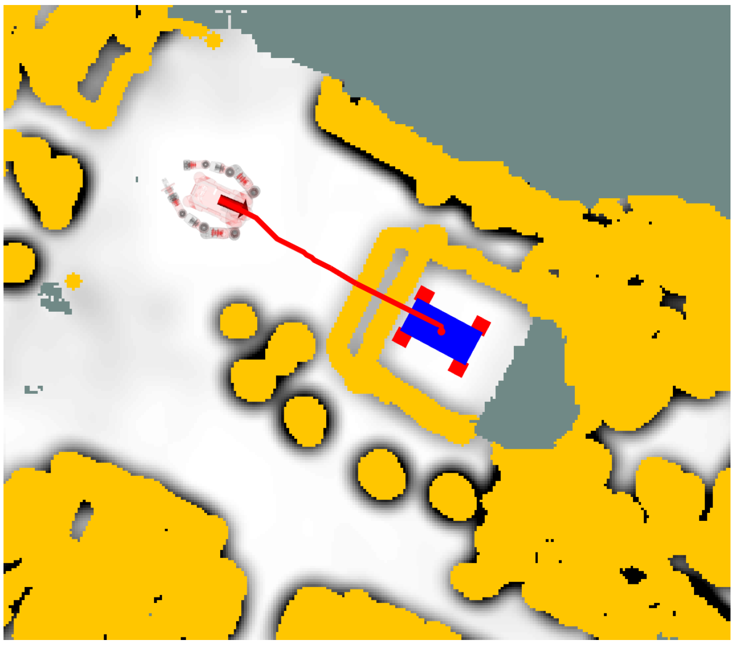


Figure 26: Current robot position (light red), goal pose (red/blue) and generated hybrid locomotion path (red), visualized on a cost map of the scenario. Olive areas are unknown, yellow areas are untraversable by driving, greyscale cells show foot costs where white is cheap and black is expensive.

4.2 Integration of the Locomotion Planner on the Centauro Platform

During a bilateral integration meeting between UBO and IIT in Genoa, the locomotion planner pipeline was tested on the Centauro platform. Data from the rotating *Velodyne Puck* laser scanner was processed to registered point clouds. These were processed to height maps which are input to the generation of cost maps. An exemplary cost map of Centauro standing in front of some stairs in the Lab in Genoa can be seen in Fig. 26. The only user input is a robot goal pose (position and orientation). A hybrid driving-stepping locomotion plan to this goal pose is planned on the cost maps. Fig. 26 also shows a given goal pose and a generated path.

Subsequently, this path is executed by the respective hybrid locomotion controller. Fig. 27 shows, how Centauro climbs these stairs autonomously.

In comparison to the Momaro platform, which was used for former experiments, there were some modifications necessary:

- The legs of Centauro are shorter compared to Momaro. Changing the respective planning parameters for maximum feasible leg length results in a different gait. Centauro takes at maximum one step between its front and rear legs ($\sim 0.6\,\mathrm{m}$) while Momaro took up to two steps between its legs ($\sim 0.9\,\mathrm{m}$). Furthermore, the ankle pitch had to be incorporated while stepping to avoid collisions with the terrain (see Fig. 27, front right foot in top right figure).

- The shorter leg length also restricts the possibility to adjust the longitudinal CoM position through base shifts to stabilize for stepping. Instead, different arm positions in different stepping phases are used for additional balancing.

In general, the experiments with the real Centauro robot showed that the real robot behaved very similar to the simulated robot, which demonstrates the high value of simulation as a development tool. When transferring the planning from the simulation to the robot, one issue

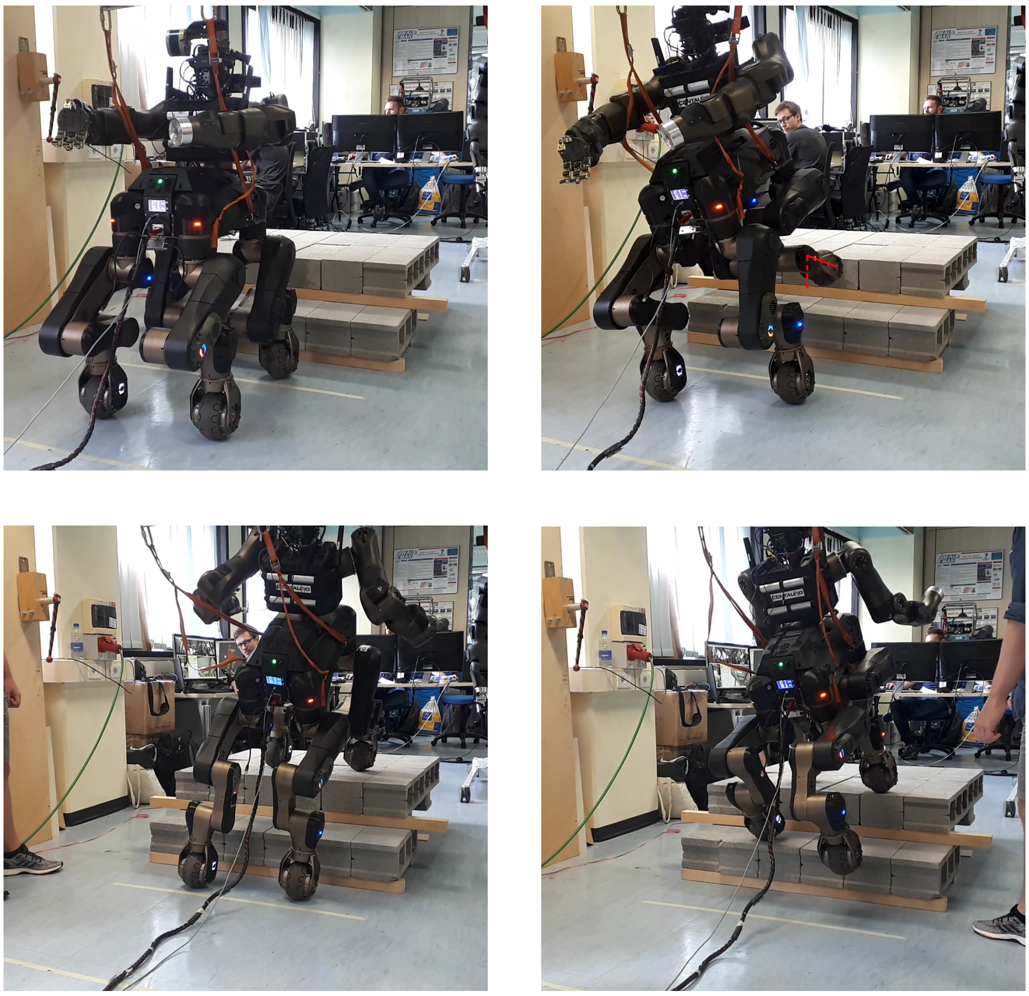


Figure 27: Centauro climbing stairs: The robot approaches the stairs by omnidirectional driving (top left) and includes stepping motions to climb the stairs while balancing through base roll and pitch motions (top right). Before reaching the top platform it moves its arms forward for a better center of mass (CoM) position (bottom left), and finally climbs with its back feet (bottom right).

occurred in the context of localization. While the simulated robot uses perfect odometry from the simulator, the real robot has to estimate its state by fusing input from laser scanner localization, wheel odometry and IMU. It is very important that the wheel odometry incorporates information about the robot pitch and roll angles as well as contact detections of each foot to provide a suitable estimate. The localization could be improved by including all these information correctly in the state estimation. In the performed experiments, some minor position corrections through joystick inputs were necessary due to slight mislocalizations.

5 Conclusion

Since the originally intended content of this deliverable —walking navigation planning— has been already reported in the previous deliverable [3], this deliverable describes extensions which improve the system performance and applicability to the considered disaster response-typical scenarios.

In particular, terrain classification has been extended to also classify stairs, based on geometric and image data. The planner module, which uses abstract representations for planning acceleration, has been modified to abstract robot representations whose characteristics are closer to the original, detailed planning representation and thus provide a better abstraction. Both modifications have been evaluated.

Moreover, we describe progress in the integration of the different modules towards a system which is running on the Centauro robot. The status of integration is close to finalization since most parts of the pipeline have been tested on the Centauro robot and all other components have been tested on a similar setup.

Overall, the developed components are ready for the final evaluation of the integrated CENTAURO disaster-response system. Remaining work in WP7 – Integration will focus on the optimization of the individual components and the final integration. In particular, we plan to

- further improve the terrain classification results for stairs and for the random forest classifier which merges all terrain class results into one terrain class map that can be used by the planner,
- optimize leg movements during stepping actions, and
- test the terrain classification within the whole locomotion planning pipeline of the Centauro robot.

A Planning Hybrid Driving-Stepping Locomotion on Multiple Levels of Abstraction

The concept of planning hybrid driving-stepping locomotion on multiple levels of abstraction is described in detail in [6] which is also attached to this document.

B Learning Abstract Representations for Locomotion Planning in High-dimensional Configuration Spaces

The concept of learning abstract representations for locomotion planning in high-dimensional configuration spaces is described in detail in [4] which is also attached to this document.

C Additional Work in Workpackage 5

C.1 Value Iteration Networks on Multiple Levels of Abstraction

While traditional search- and sample-based planning methods tend to perform extensive searches in large high-dimensional configuration spaces, learning-based approaches follow a different idea. They assess a situation and directly derive a suitable path. One method which arose large interest in the community is Value Iteration Networks (VINs) [8]. However, similar to other learning-based planners, VINs are only applicable to small low-dimensional state spaces since the required network complexity and amount of training data rapidly increases for larger spaces. We propose a method to utilize VINs to multiple levels of abstraction to enable them to handle larger scenes. This enables us to handle significantly larger queries and plan omnidirectional driving with the Centauro robot in cluttered environments. Further details can be found in [7] which is also attached to this document.

C.2 Local Planner using Deep Reinforcement Learning

In our previous work described in ”Deliverable D5.3 CENTAURO Driving Navigation”, we set up a framework which learns a local planner with a five-dimensional action space using a proximal policy optimization (PPO) algorithm developed by OpenAI. In our current work, we extend our approach to train action policies to acquire more complex navigation skills. The policy maps height-map image observations to motor commands to navigate to a target position while avoiding obstacles. We propose to acquire the multifaceted navigation skill by learning and exploiting a number of manageable navigation behaviors. We also introduce a domain randomization technique to improve the versatility of the training samples.

The algorithm is described in detail in [2].

References

[1] X. Chen, F. Schilling, T. Klamt, and P. Jensfelt. Deliverable D5.2 CENTAURO Terrain Classification.

[2] Xi Chen, Ali Ghadirzadeh, John Folkesson, and Patric Jensfelt. Deep reinforcement learning to acquire navigation skills for wheel-legged robots in complex environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[3] T. Klamt, P. Jensfelt, X. Chen, K. Nordberg, and D. Doreschel. Deliverable D5.3 CENTAURO Driving Navigation.

[4] Tobias Klamt and Sven Behnke. Learning abstract representations for locomotion planning in high-dimensional configuration spaces. Submitted to IEEE International Conference on Robotics and Automation (ICRA), 2019.

[5] Tobias Klamt and Sven Behnke. Anytime hybrid driving-stepping locomotion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[6] Tobias Klamt and Sven Behnke. Planning hybrid driving-stepping locomotion on multiple levels of abstraction. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[7] Daniel Schleich, Tobias Klamt, and Sven Behnke. Value iteration networks on multiple levels of abstraction. Submitted to IEEE International Conference on Robotics and Automation (ICRA), 2019.

[8] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[9] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. Dilated residual networks. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

Planning Hybrid Driving-Stepping Locomotion on Multiple Levels of Abstraction

Tobias Klamt and Sven Behnke

***Abstract*— Navigating in search and rescue environments is challenging, since a variety of terrains has to be considered. Hybrid driving-stepping locomotion, as provided by our robot Momaro, is a promising approach. Similar to other locomotion methods, it incorporates many degrees of freedom—offering high flexibility but making planning computationally expensive for larger environments.**

We propose a navigation planning method, which unifies different levels of representation in a single planner. In the vicinity of the robot, it provides plans with a fine resolution and a high robot state dimensionality. With increasing distance from the robot, plans become coarser and the robot state dimensionality decreases. We compensate this loss of information by enriching coarser representations with additional semantics. Experiments show that the proposed planner provides plans for large, challenging scenarios in feasible time.

I. Introduction

Hybrid driving-stepping locomotion is a flexible approach to traverse many types of terrain since it combines the advantages of both, wheeled and legged, locomotion types. However, due to its high robot state dimensionality, planning respective paths is challenging.

In our previous work [1] we presented an approach to plan hybrid driving-stepping locomotion paths for our robot Momaro [2] even for very challenging terrain such as staircases with additional obstacles on it. The planner prefers omnidirectional driving whenever possible and considers individual steps in situations where driving is not possible. The individual configuration of ground contact points (robot footprint) is considered at any time. During planning, steps are represented as abstract manoeuvres which are expanded to detailed motion sequences before executing them. For small scenarios, this method generates high quality paths in feasible time with bounded suboptimality. Due to the high dimensionality of the robot configuration, the explored state space increases rapidly for larger scenarios and makes planning expensive. 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.

The search space can be reduced by choosing a coarser resolution or describing the robot and its manoeuvres in a more abstract way with less degrees of freedom (DoF). However, a fine resolution is key to navigate the robot

All authors are with Rheinische Friedrich-Wilhelms-Universität Bonn, Computer Science Institute VI, Autonomous Intelligent Systems, Endenicher Allee 19A, 53115 Bonn, Germany `klamt@ais.uni-bonn.de, behnke@cs.uni-bonn.de`.

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

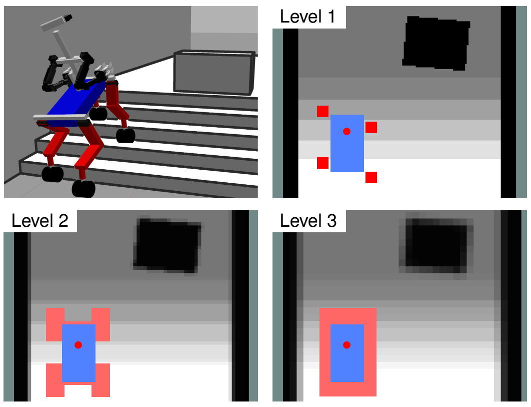


Fig. 1. Momaro on a staircase, visualized in three representation levels. Maps show terrain heights (olive = unknown).

precisely through challenging terrain. Moreover, only using a more abstract robot description is difficult, since the planning result shall be a path which can be executed by the robot with its given number of DoF.

Coarse-to-fine planning approaches [3], [4] address this problem by generating a rough plan first and refine the resulting path to the desired resolution and number of DoF in a second step. Especially in challenging, cluttered terrain, this procedure bears the risk of only finding expensive paths due to the lack of detail in the initial search.

We present a method which plans hybrid driving-stepping locomotion on three different levels of representation (see Fig. 1). In the vicinity of the robot, a representation with a high resolution and a high number of DoF is used to find paths which can be executed by the robot. With increasing distance from the robot, the resolution gets coarser and the robot is described with less DoF. These path segments are situated further in the future which comes along with a higher degree of uncertainty and less accurate sensor information. We compensate this loss of information for higher levels of representation by enriching the representation with additional semantics. All levels of representation are unified in a single planner. We further present methods to refine path segments into more detailed levels of representation. This decreases the number of necessary replanning steps. Replanning is only initiated if costs indicate that a situation is wrongly assessed in the coarser representation. In addition, we introduce a heuristic, based on the most abstract level of representation.

Experiments show that, compared to our previous work, this approach can handle much larger scenarios in feasible planning time while the path quality stays comparable.

II. Related Work

Multiple works addressed path planning for challenging environments, either by driving [5]–[7] or walking with quadruped robots [8], [9]. To our knowledge, there exist no approaches for hybrid driving-stepping path planning in challenging terrain, except our previous work [1].

A common idea to accelerate planning for larger scenarios is the usage of multiresolutional approaches. Behnke [10] proposed a general concept for A*-based multiresolution planning with a decreasing resolution with increasing distance from the robot. González-Sieira et al. [11] apply high resolution in areas of high environment complexity. Resolution decreases with increasing distance from these areas. Similarly, Pivtoraiko et al. [12] apply different sets of state transitions to different areas of the environment. Bohlin [3] generates an initial plan in a coarse resolution first and refines this plan into a finer resolution. Since high resolution planning is only applied to parts of the map, the search space decreases and planning performance increases, compared to pure high resolution planning. One of the main challenges in multiresolutional approaches is the definition of feasible transitions between the different resolutions. All of the presented approaches face the problem that a coarse resolution representation neglects information and thus is not capable of representing challenging terrain features in sufficient detail—which might lead to wrong or bad plans.

Planning for systems with high-dimensional motion flexibility quickly reaches its limits for larger environments since the search space grows exponentially. Similar to multiresolution planning, several approaches utilize multiple representations with different planning dimensionalities to decrease planning complexity. Kohrt et al. [4] generate an initial plan in a low-dimensional search space and replan in the high-dimensional search space by only considering those states that are part of the low-dimensional plan. Gochev et al. [13] plan a path in a low-dimensional search space and only switch to high-dimensional planning in those areas where low-dimensional planning cannot find a solution. Similarly, Zhang et al. [14] plan in 2D and switch to high-dimensional planning in the robot vicinity and at key points. As described for multiresolution planning, planning with multiple robot configuration dimensionalities might lead to wrong or bad plans, since a low-dimensional robot representation might assess challenging situations wrongly.

To achieve further planning acceleration, it is an obvious idea to combine multiresolution and multidimensional planning. However, only few works, such as by Petereit et al. [15] address this. Different planning dimensionalities and resolutions are applied by using different sets of motion primitives. A fine resolution is only considered close to the start and goal pose and close to obstacles. A high planning dimensionality is considered for states which will be reached within a given time interval. This allows the planner to provide detailed plans close to the robot while planning times stay feasible. The drawbacks of both, multiresolutional and multidimensional planning also apply to this work.

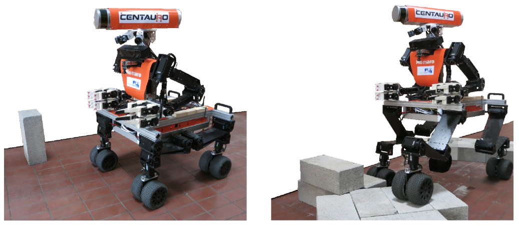


Fig. 2. Our wheeled-legged robot Momaro is capable of omnidirectional driving (left) and stepping (right).

The platforms which are used in the presented works are quite limited in their configuration capabilities, compared to our robot Momaro. Our approach applies multiresolution and multidimensional planning to the challenging problem of hybrid driving-stepping locomotion. Furthermore, we compensate the loss of information for coarser resolutions and low-dimensional robot representations by enriching those representations with additional semantic features.

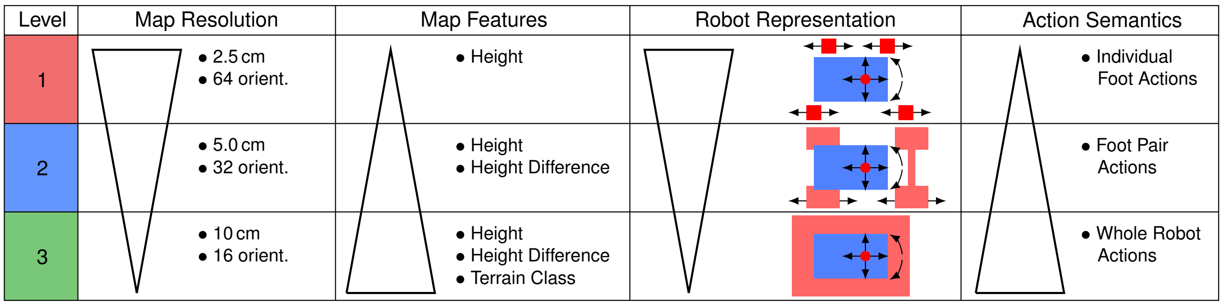
III. Hardware

We use our mobile manipulation robot Momaro [2] (see Fig. 2). It offers omnidirectional driving through its four articulated legs ending in directly driven 360° steerable pairs of wheels. The unique design enables manoeuvres which are neither realizable by pure driving nor pure walking robots such as shifting a single foot while maintaining ground contact and thus changing the robot footprint under load. Active leg movements are restricted to the sagittal plane since each leg consists of three pitch joints.

Sensor inputs come from an IMU and a continuously rotating Velodyne Puck 3D laser scanner at the robot head which provides a spherical field-of-view. The laser-range measurements are registered and aggregated to a 3D environment map using the method of Droeschel et al. [16].

IV. Approach

Input to our method is a height map with a resolution of 2.5 cm which is generated from the 3D environment map. In the vicinity of the robot, height information is very precise. With increasing distance from the robot, the accuracy decreases due to measurement errors. Planning is done on foot and body costs. The ground contact costs C_{GC} describe the costs to place an individual ground contact element (e.g., a foot or a foot pair) in a given configuration on the map. C_{GC} includes information about the terrain surface and obstacles in the vicinity. The body costs $C_{\text{B}}(\vec{r}_b)$ describe the costs to place the robot base $\vec{r}_b = (r_x, r_y, r_\theta)$ with its center position r_x, r_y and its orientation r_θ on the map. $C_B(\vec{r}_b)$ include information about obstacles under the robot base and about the terrain slope under the robot. The generation of C_{GC} and C_{B} from the height map varies between the different levels of representation and may contain several steps. Ground contact costs and body costs are combined to pose costs $C(\vec{r})$ which describe the costs to place the robot in a given configuration $\vec{r}$ on the map.



Level	Map Resolution	Map Features	Robot Representation	Action Semantics
1	• 2.5 cm • 64 orient.	• Height		• Individual Foot Actions
2	• 5.0 cm • 32 orient.	• Height • Height Difference		• Foot Pair Actions
3	• 10 cm • 16 orient.	• Height • Height Difference • Terrain Class		• Whole Robot Actions

Fig. 3. The planner includes three levels of representation with decreasing resolution and robot configuration dimensionality. To compensate the loss of information, the semantics for both the environment representation and the robot actions increase.

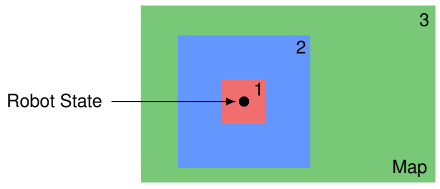


Fig. 4. Size and position of the different levels of representation. *Level 1* covers the vicinity of the robot. *Level 2* is also robot centered and medium sized. *Level 3* covers the whole map.

Path planning is realized through an A*-search with anytime characteristics (ARA* [17]) on pose costs. For the current search pose, feasible neighbour poses are generated during the search. They can either be reached by omnidirectional driving or by stepping-related motions. Stepping-related motions are only considered in the vicinity of obstacles where driving is infeasible. Steps are described as abstract steps, the direct transition between a pre-step to an after-step pose. The detailed motion sequence for steps is not considered during planning but generated before execution.

The environment and the robot are described in three different levels of representation with different sizes. In the vicinity of the robot, we use a fine resolution and a high robot configuration dimensionality for planning. We call this *Level 1* representation. With increasing distance from the current robot position, the environment and the robot are represented on higher levels with a coarser resolution and a robot representation with lower dimensionality. This is reasonable, since those parts of the plan are reached in the further future and thus are more uncertain. Moreover, sensor measurements become less precise with increasing distance from the robot. At the same time, we compensate this loss of detail by enriching the environment representation with additional features, which increase the understanding of the situation. Pose costs and robot actions use these semantic features. Higher levels of representation can be derived from lower levels of representation. The approach is visualized in Fig. 3. Level sizes and positions are shown in Fig. 4. For a planning task, the planner only performs a single planning run while including all three levels of representation. Hence, it is important that the same action carries the same costs in different levels of representation to make planning consistent over all levels. Moreover, the transition between the different levels of representation is challenging. All three levels of representation and the transition between them are described in detail in the following sections.

The resulting path consists of segments in multiple levels of representation. As described before, the contained steps are abstract manoeuvres. Abstract steps in the initial path segment are expanded to detailed motion sequences before executing them. Roll and pitch motions of the robot base as well as single foot shifts stabilize the robot to perform each step safely. In addition, foot heights are derived. See our previous work [1] for more details.

Steps are only expanded for path segments in *Level 1* which is based on our previous work. For higher levels, representations are not detailed enough to derive concrete robot motions. As the robot executes the initial path segment, more measurements are made and a more detailed environment representation becomes available for path segments which have been represented in higher levels before. The path is updated with these updated representations. This can either be done by replanning the whole path or by transforming the respective path segments into more detailed representations, as described in Section IV-F. We call this coarse-to-fine transformation *refinement*.

A. Representation Level 1

Level 1 is based on the approach which we presented in our previous work [1]. Input is a height map with a resolution of 2.5 cm. We derive local unsigned height differences between neighbour cells from this height map to generate ground contact costs for each individual foot. Base costs are derived from the height map itself. A height map and the derived foot costs can be seen in Fig. 5. In this level of representation, a robot pose $\vec{r} = (\vec{r}_b, f_1, ..., f_4)$ is represented through the robot base configuration $\vec{r}_b$ and the individual longitudinal foot positions $f_1, ..., f_4$. At each position, the robot can have 64 different discrete orientations.

Feasible driving neighbour poses can be found within a 20-position-neighbourhood and by turning on the spot to the next discrete orientation, as shown in Fig. 6. If the robot is close to an obstacle, additional stepping-related manoeuvres are considered which are visualized in Fig. 7. Those can be a discrete step, a longitudinal base shift manoeuvre,

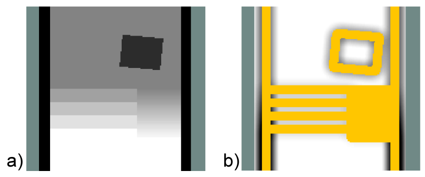


Fig. 5. a) *Level 1* height map showing a corridor with a flight of stairs, an untraversable steep ramp and an obstacle, b) respective foot cost map (yellow = untraversable by driving, olive = unknown).

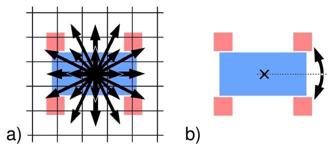


Fig. 6. Driving neighbour poses can be found by either a) straight moves with fixed orientation within a 20-position-neighbourhood or b) by turning on the spot to the next discrete orientation.

shifting individual feet forward or shifting individual feet towards their neutral position. We define the neutral robot pose as the pose visualized in Fig. 7 a, top. The costs for the presented manoeuvres are based on the foot and body costs, the individual robot elements induce during the manoeuvre.

As an extension of the previous work, we want the robot to align its orientation with the stair orientation, when climbing those. This is desirable, since the kinematic only allows for leg movements in the sagittal plane and since this behavior can also be observed when humans climb stairs by themselves or teleoperate robots to do so. If, after a stepping manoeuvre, 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 a configuration with an additional cost term, we achieve the desired behavior.

B. *Representation Level 2*

We use the input height map with a resolution of 2.5 cm to compute the *Level 2* representation consisting of a height map and a height difference map with a resolution of 5 cm (see Fig. 9 a,b). According to the Nyquist-Shannon sampling theorem, subsampling has to come along with smoothing. To satisfy this theorem, we subsample the *Level 1* height map as shown in Fig. 8. Each *Level 2* height value is computed from

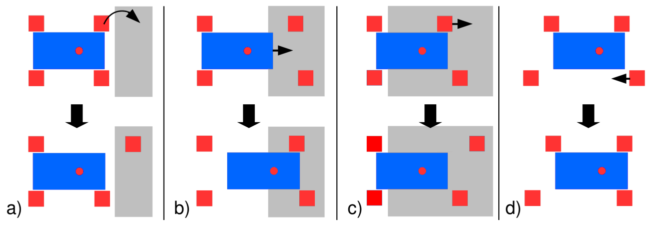


Fig. 7. *Level 1* stepping-related manoeuvres: a) Abstract step, b) longitudinal base shift, c) shifting a front foot forward, and d) shifting any foot back to its neutral position.

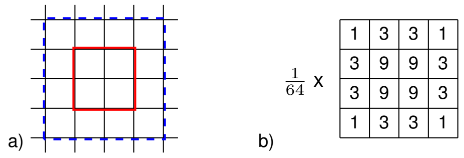


Fig. 8. Subsampling method: a) For a *Level 2* cell (red square) a 4×4-window (blue square) of *Level 1* cells is considered. b) Normalized binomial distribution to weigh heights and height differences.

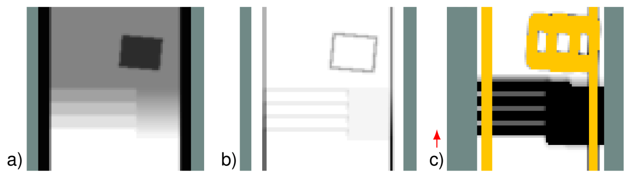


Fig. 9. *Level 2*: a) height map, b) height difference map, c) foot area pair cost map for the orientation indicated by the red arrow.

the normalized, weighted sum of a 4×4-region of *Level 1* height values. We use a binomial distribution for weighing. A *Level 2* height difference map is generated in the same manner: We generate a *Level 1* height difference map by computing local height differences on the *Level 1* height map. This height difference map is then subsampled to a *Level 2* height difference map.

To decrease the robot configuration space dimensionality, we accumulate individual feet to pairs. This is intuitive, since we observe a tendency to pairwise foot movement in *Level 1* paths. Moreover, instead of describing each foot position precisely, we use foot areas as a more abstract description. We know, that a foot will be placed somewhere in the respective area but since the representation contains some time-related and measurement-related imprecision, a knowledge of the accurate foot position is not necessary. A *Level 2* robot pose $\vec{r} = (\vec{r}_b, f_\mathrm{f}, f_\mathrm{r})$ is consequently represented by its robot base pose $\vec{r}_b$ and its relative longitudinal front and rear foot area pair coordinates f_f and f_r. Note that our platform and planner only allow sagittal leg movement. Lateral foot coordinates are fixed and thus a single variable is sufficient to describe each foot area pair.

We use the generated *Level 2* representation to compute ground contact and body costs. Body cost computation is similar to *Level 1* and only relies on height information. Ground contact costs

$$C_{\mathrm{GC},2} = 1 + k_1 \cdot \triangle H_{\mathrm{avg}}, \tag{1}$$

where $k_1 = 107$, are costs to place foot area pairs on the map and are generated from the average height differences $\triangle H_{\mathrm{avg}}$ in the respective area. A *Level 2* foot area pair cost map can be seen in Fig. 9 c. Again, a punishing cost term is introduced for after-step poses with different average heights under neighbouring foot areas.

The robot actions are defined accordingly. Driving neighbours can be found similar to *Level 1* but with a doubled action resolution of 5 cm and 32 discrete robot orientations at each position. Additional stepping-related manoeuvres differ

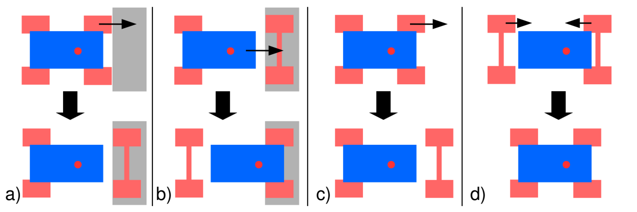


Fig. 10. *Level 2* stepping-related manoeuvres: a) Step, b) longitudinal base shift, c) move the front foot pair forward, and d) move any foot pair towards its neutral position.

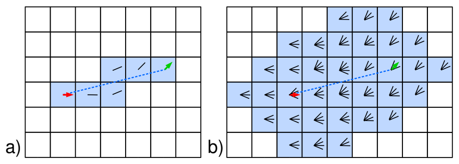


Fig. 11. Generating a set of feasible robot base poses for path refinement: a) For a given start ($\vec{r}_{1,\mathrm{i}}$, red arrow) and goal ($\vec{r}_{1,\mathrm{i+1}}$, green arrow) robot base pose, we generate a set of feasible robot base poses (black lines) by interpolating between the two. b) Inflation by two position steps and one orientation step.

from *Level 1* since the robot is only able to move foot pairs instead of individual feet. If the robot is close to an obstacle, it may step with a foot pair or perform another stepping related manoeuvre, as visualized in Fig. 10. To motivate stepping manoeuvres, we define a maximum height difference $\triangle H_{\text{max,drive}}$ for the foot area center coordinate which can be overcome by driving. Larger height differences only can be traversed by stepping.

The costs for such a foot pair manoeuvre are the concatenated costs for each individual foot action as described for *Level 1*. If, for example, the robot steps with its front foot pair as visualized in Fig. 10 a, the costs for this manoeuvre are the sum of the costs for a step with the front left foot and a step with the front right foot. Since *Level 2* foot pair area costs differ from *Level 1* foot costs, we reparametrized the manoeuvre cost computation. We do this by performing foot pair manoeuvres in a variety of basic scenarios (e.g., drive/turn on a patch of flat/rough underground, step up different height differences, do a base shift) in both representation levels and manually tune the *Level 2* cost parameters until the costs for those manoeuvres in both levels vary by $\leq 5\%$.

During planning and execution, it is an important feature to refine *Level 2* path segments into *Level 1*. To refine a *Level 2* path segment between two successive poses $\vec{r}_{2,\mathrm{i}}$ and $\vec{r}_{2,\mathrm{i+1}}$, we transform both poses into *Level 1* and generate a set S of feasible robot base poses by interpolating between $\vec{r}_{1,\mathrm{i}}$ and $\vec{r}_{1,\mathrm{i+1}}$. S is then inflated with a radius of two position steps and one orientation step as visualized in Fig. 11. A local planner, which is restricted to S, searches for a *Level 1* path between $\vec{r}_{1,\mathrm{i}}$ and $\vec{r}_{1,\mathrm{i+1}}$. If

- either one of the two poses 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,

we call this path segment *not refineable*.

C. Representation Level 3

We apply the described subsampling process (see Section IV-B) to generate a *Level 3* height map and height difference map with a resolution of 10 cm from the *Level 2* height map and height difference map. To increase the semantics of the environment representation, we categorize each *Level 2* map cell 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, we search for cells of the terrain type *step*. This is done by searching for cell pairs c_i and c_j that fulfill the following criteria:

- $\triangle H(c_i) < \triangle H_{\text{max,drive}}$: c_i is on a drivable surface,
- $\triangle H(c_j) < \triangle H_{\text{max,drive}}$: c_j is on a drivable surface,
- $\|c_\mathrm{i} - c_j\| < 0.45\,\mathrm{m}$: 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, $C_{\mathrm{GC}}(c_k) = \infty$ counts for all cells $c_k \in T$: A direct foot movement from c_i to c_j requires a step.

For all pairs of c_i and c_j which fulfill these criteria, each cell $c_s \in c_i \cup c_j \cup T$ is assigned the terrain class *step*. In addition, we compute the angle $\alpha_{\mathrm{i,j}}$ between c_i and c_j and save it for c_s. Since most *step* cells are detected several times, we collect several angles for each cell. $\alpha_{\text{avg,s}}$, the *mean of circular quantities* of these angles describes the estimated step orientation in c_s.

Second, we classify the remaining cells by their *Level 2* height difference value $\triangle H$[1]:

- *flat* if $\triangle H(c_i) \in [0\,\mathrm{m}, 2 * 10^{-4}\,\mathrm{m}]$,
- *rough* if $\triangle H(c_i) \in [2 * 10^{-4}\,\mathrm{m}, 0.05\,\mathrm{m}]$,
- *wall* if $\triangle H(c_i) \in [0.05\,\mathrm{m}, \infty]$, and
- *unknown* if $\triangle H(c_i)$ is unknown.

The height difference intervals are 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.

Another source for terrain class segmentation can be camera images as shown in [18]. Fig. 12 a,b gives an example for a *Level 3* height map and terrain class map.

The *Level 3* robot representation $\vec{r} = \vec{r}_b$ only consists of the robot base pose. Individual feet positions are not

[1]These height difference values are subsampled and smoothed and thus cannot be directly transferred to occurring height differences in the terrain.

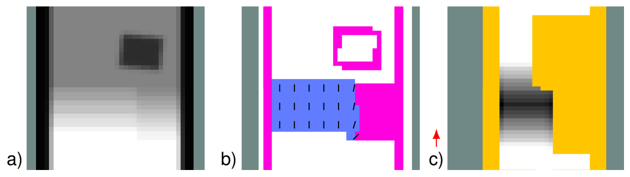


Fig. 12. *Level 3*: a) height map, b) terrain class map (white = flat, blue = stepping, pink = wall, black lines = step orientations), c) robot area cost map for the orientation indicated by the red arrow.

considered but we assume that the feet are somewhere in a ground contact area around the robot a_r (see Fig. 3). Hence, the robot is not able to perform foot or foot pair movements in this representation. The whole robot is rather moved over the terrain and traverses different terrain classes with different costs. Path search neighbour poses can be found similar to the driving neighbours described for *Level 1*. In this level of representation, the action resolution is 10 cm and the robot may have 16 different orientations at each position. When moving over *step* cells, a robot state is only feasible if the difference between the robot orientation and the step orientation of each *step* cell c_r is less than one discrete orientation step: $\text{abs}(\alpha_{\text{avg,r}} - r_\theta) < \frac{1}{16} \cdot 2\pi$. Moreover, the robot is only allowed to move parallel and orthogonal to step orientations. These restrictions are required to enforce a behavior, which is induced by the robot kinematic in lower representation levels but not represented in *Level 3* otherwise.

Regarding cost generation, each cell c_i is assigned a cost value $C_\text{c}(c_i)$ depending on its terrain class:

- *flat*: $C_\text{c}(c_i)$ = 1.0,
- *rough*: $C_\text{c}(c_i)$ = 1.4,
- *step*: $C_\text{c}(c_i)$ = 76.0 + 2.95 $\cdot \bigtriangleup H(c_i)$,
- *wall*: $C_\text{c}(c_i) = \infty$, and
- *unknown*: $C_\text{c}(c_i)$ = nan.

The pose cost $C(\vec{r})$ does not combine individual ground contact and body cost but averages the cost values of all cells in a_r. The described terrain class specific cell costs are manually tuned by comparing the cost of *Level 1* and *Level 3* manoeuvres for the same set of basic scenarios, as mentioned in Section IV-B. While constant values were sufficient for *flat* and *rough* cells, costs for stepping manoeuvres depend on the height difference to overcome. The presented computation method for *step* cells is required to keep cost differences for these basic manoeuvres $\leq 5\%$. A resulting robot area cost map can be seen in Fig. 12 c.

Level 3 paths can be refined to *Level 2* paths in the following way: As described for *Level 2*, we generate a set S of feasible robot base poses. In contrast to *Level 2*, we do not only consider two successive poses but the whole path segment $\vec{r}_{3,\text{s}}, ..., \vec{r}_{3,\text{g}}$ that needs to be refined at once. The first and last robot pose $\vec{r}_{3,\text{s}}$ and $\vec{r}_{3,\text{g}}$ of this *Level 3* path segment are transformed to a *Level 2* start and goal pose and a local *Level 2* planner, which is restricted to S, searches for a path between $\vec{r}_{2,\text{s}}$ and $\vec{r}_{2,\text{g}}$. If a *Level 3* path needs to be refined to *Level 1*, *Level 2* is taken as an intermediate refinement step.

D. Level Transition

All three levels of representation are combined in a single planner, which chooses the lowest available level for each pose to provide the most detailed planning. Since planning in a low level of representation is slower, we provide *Level 1* data only in a small area around the robot position which is sufficiently large to plan the next manoeuvres. *Level 2* data is provided for a medium-sized region around the current robot position while *Level 3* covers the whole map.

The planner checks for each manoeuvre (e.g., drive into one direction, do a step, ...) if both, start and goal pose of this manoeuvre, are part of the same level of representation. If the goal pose is not part of the start pose level of representation, the start pose is transformed to the next higher level of representation and the same manoeuvre is replanned in this level if it is still available in this level. Note that the transformation of the start pose to the next higher level of representation might induce costs. Due to different map resolutions, the robot might be shifted to fit into the next level map cell and discrete orientation. Due to increasing foot restrictions, feet might be shifted to fit the next level robot representation (e.g., individual feet have to align within foot area pairs). We check each transformation for feasibility and generate costs from the occurring manoeuvre costs.

E. Heuristic

In our previous work, a combination of the Euclidean distance and the orientation difference was used as an admissible A* heuristic (*Euclidean heuristic*). However, this heuristic does not consider the terrain which has large influence on the path costs. We propose a *Level 3*-based heuristic which includes such terrain features (*Dijkstra heuristic*).

After the goal pose $\vec{r}_{\text{i,G}}$ is set, it is transformed to *Level 3*. We then start a one-to-any 3D Dijkstra search in *Level 3* starting from $\vec{r}_{3,\text{G}}$. Hence, we get for each *Level 3* pose a cost estimation to reach the goal pose. During path planning, we can estimate the costs from any robot pose to the goal by transforming it to *Level 3* and get the respective cost value.

Note that the quality of this heuristic strongly depends on the quality of the *Level 3* cost model in comparison to costs for the same manoeuvres in other levels of representation. Further note that we cannot prove that this heuristic always underestimates costs, which is necessary to prove admissibility for the generation of optimal paths. However, since we also utilize the suboptimal ARA* algorithm, we do not aim to generate optimal paths for a given problem. In fact, we focus on generating paths with satisfying quality in feasible time. The performance of this heuristic is evaluated in Section V.

F. Continuous Refinement

As the robot moves along the initial path, the sensors provide new measurements and high-detailed environment representations are generated in the vicinity of the current robot position. We include these updated representations in the path by continuously refining the respective path segments, as shown in Fig. 13. If a cost difference $> 25\%$ between the original and the refined path segments indicates

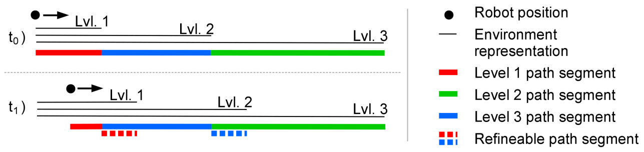


Fig. 13. As the robot moves along the path, the *Level 1* and *Level 2* representations move with it. Consequently, those path segments which are represented in a higher level and for which a more detailed representation becomes available, can be refined to this more detailed representation.

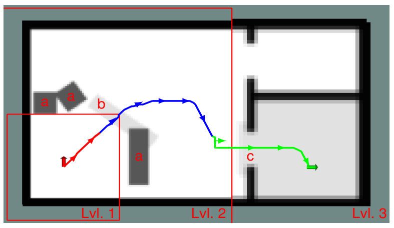


Fig. 14. Height map of the first experiment scenario. From its start position (red arrow), the robot needs to navigate between multiple objects (a), over a bar obstacle (b), step up to an elevated platform and through a door (c) to the goal pose (green arrow). The resulting path for $\mathcal{W} = 1.125$ and combined levels of representation is shown. *Level 1* path segments = red, *Level 2* segments = blue, *Level 3* segments = green. Arrows show r_θ.

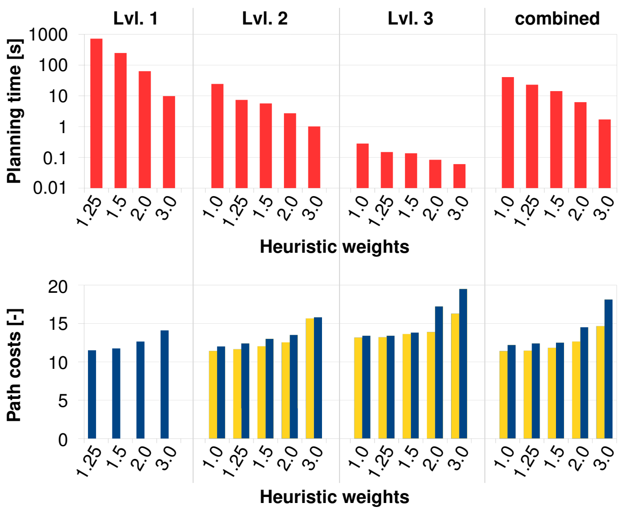


Fig. 15. Planning performance for different levels and different $\mathcal{W}$ for the first experiment. estimated costs = yellow, refined *Level 1* costs = blue.

that the higher-level planning assessed a situation wrongly, we initiate a new planner run. With this approach, we can guarantee that path segments in the vicinity of the robot are always represented in *Level 1* and thus, included steps can be expanded and the result can be executed by the controller.

V. Experiments

We evaluate the proposed approach in two experiments. Both are done on one core of a 2.6 GHz Intel i7-6700HQ processor using 16 GB of memory. An additional video is available online[2] which also contains a Gazebo experiment to demonstrate the continuous refinement strategy.

A first experiment evaluates the planning performance of the different levels of representation individually and combined, as shown in Fig. 4. For this, we choose the *Level 1* size to be 3×3 m. This is sufficiently large to plan the next robot manoeuvres in high detail, but still small enough to avoid long high-dimensional planning. The *Level 2* size is chosen to be 9×9 m so that the *Level 2* path segment is about twice as long as the *Level 1* path segment. We utilized the *Euclidean heuristic* to compare the results to our previous work. The height map and a resulting path are shown in Fig. 14. Since we use an ARA* algorithm which works with several heuristic weights $\mathcal{W}$, we evaluate the influence of these. Fig. 15 shows the planner performance.

It can be seen that planning on levels of representation >1 and with combined levels is faster by at least one order of magnitude compared to pure *Level 1* planning. The *Level 1* path for $\mathcal{W} = 1.0$ could not be computed due to memory limitations. We distinguish between the path costs in the respective levels of representation (estimated cost) and the costs each path carries when refined to *Level 1*. Comparing the estimated costs to the refined *Level 1* costs gives an assessment about the quality of cost generation in each level of representation. The comparison of the refined *Level 1* costs to the original *Level 1* costs indicates the quality of the resulting path. It can be seen that the estimated costs always underestimate the refined *Level 1* costs. Especially for $\mathcal{W} \leq 1.5$ the estimation is close with a difference $\leq 7.7\%$. Furthermore, the results show that for $\mathcal{W} \leq 1.5$ the refined *Level 1* costs differ to the original *Level 1* costs by $\leq 15\%$.

In a second experiment, we compare the presented *Dijkstra heuristic* to the *Euclidean heuristic*. The scenario shown in Fig. 16 is larger and more challenging, compared to the first scenario. The starting pose is pose *a*. Planning is performed on combined levels of representation. A resulting path is shown in Fig. 17. Planning times and resulting costs are shown in Fig. 18. Preprocessing the *Dijkstra heuristic* took 0.52 s of the presented planning times. It can be seen that the *Dijkstra heuristic* further accelerates planning while the resulting costs stay comparable at least for $\mathcal{W} \leq 1.5$. E.g., 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.

We finally compare the planner performance when started from different poses, as shown in Fig. 16. The results in Fig. 19 indicate that an important factor for the planner performance is the complexity of the planning within *Level 1* but higher $\mathcal{W}$ lead to feasible performances in any case.

VI. Conclusion

In this paper, we presented a hybrid locomotion planning approach which is able to provide plans for large scenarios with high detailing in the vicinity of the robot. We achieve this by introducing three levels of representation with de-

[2]https://www.ais.uni-bonn.de/videos/ICRA_2018_Klamt/

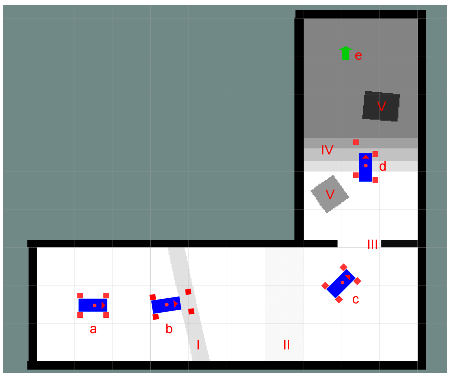


Fig. 16. Height map for the second experiment containing a bar obstacle (I), a rough area (II), a door (III), a flight of stairs (VI) and two obstacles (V). a - d are different starting poses for the planner, e is the goal pose.

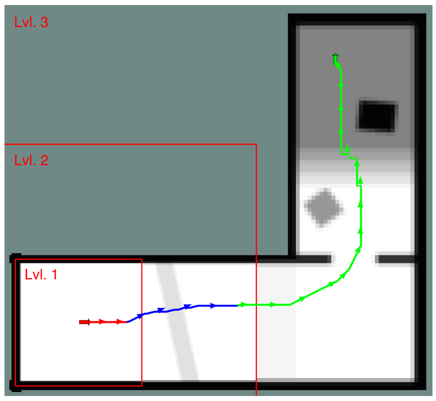


Fig. 17. Resulting path for planning with the *Dijkstra heuristic* and combined levels with $\mathcal{W} = 1.25$.

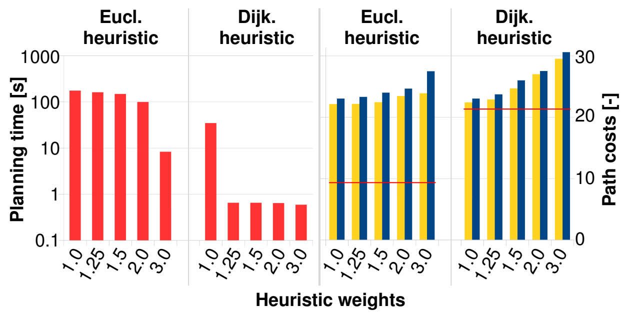


Fig. 18. Planning performance for combined levels of representation to compare the *Euclidean heuristic* with the *Dijkstra heuristic*. Red lines indicate the cost estimation for the path by each heuristic.

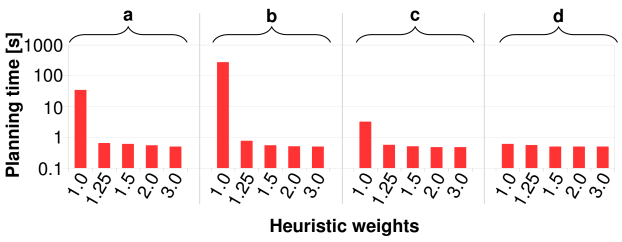


Fig. 19. Planning time for different starting poses (see Fig. 16) and different $\mathcal{W}$, using the *Dijkstra heuristic*.

creasing resolution and robot configuration dimensionality but increasing semantics of the situation. The most abstract level of representation can be used as a heuristic which poses a second acceleration strategy. Experiments show that the presented approach significantly accelerates planning while the result quality stays feasible and, hence, significantly larger scenarios can be handled in comparison to our previous work.

REFERENCES

[1] T. Klamt and S. Behnke, "Anytime hybrid driving-stepping locomotion planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[2] M. Schwarz, T. Rodehutskors, M. Schreiber, and S. Behnke, "Hybrid driving-stepping locomotion with the wheeled-legged robot Momaro," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[3] R. Bohlin, "Path planning in practice; lazy evaluation on a multi-resolution grid," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2001.

[4] C Kohrt, A. G. Pipe, J. Kiely, R Stamp, and G Schiedermeier, "A cell based voronoi roadmap for motion planning of articulated robots using movement primitives," in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2012.

[5] Z. Ziaei, R. Oftadeh, and J. Mattila, "Global path planning with obstacle avoidance for omnidirectional mobile robot using overhead camera," in *IEEE International Conference on Mechatronics and Automation (ICMA)*, 2014.

[6] T. M. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *The International Journal of Robotics Research*, vol. 26, no. 2, pp. 141–166, 2007.

[7] M. Brunner, B. Brüggemann, and D. Schulz, "Motion planning for actively reconfigurable mobile robots in search and rescue scenarios," in *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2012.

[8] M. Wermelinger, P. Fankhauser, R. Diethelm, P. Krüsi, R. Siegwart, and M. Hutter, "Navigation planning for legged robots in challenging terrain," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[9] N. Perrin, C. Ott, J. Englsberger, O. Stasse, F. Lamiraux, and D. G. Caldwell, "Continuous legged locomotion planning," *IEEE Transactions on Robotics*, vol. 33, no. 1, pp. 234–239, 2016.

[10] S. Behnke, "Local multiresolution path planning," in *RoboCup 2003: Robot Soccer World Cup VII*, Springer, 2003, pp. 332–343.

[11] A. González-Sieira, M. Mucientes, and A. Bugarín, "An adaptive multi-resolution state lattice approach for motion planning with uncertainty," in *Robot 2015: Second Iberian Robotics Conference*, Springer, 2016, pp. 257–268.

[12] M. Pivtoraiko and A. Kelly, "Differentially constrained motion replanning using state lattices with graduated fidelity," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2008.

[13] K. Gochev, B. Cohen, J. Butzke, A. Safonova, and M. Likhachev, "Path planning with adaptive dimensionality," in *Fourth Annual Symposium on Combinatorial Search*, 2011.

[14] H. Zhang, J. Butzke, and M. Likhachev, "Combining global and local planning with guarantees on completeness," in *IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2012.

[15] J. Petereit, T. Emter, and C. W. Frey, "Mobile robot motion planning in multi-resolution lattices with hybrid dimensionality," *IFAC Proceedings Volumes*, vol. 46, no. 10, pp. 158–163, 2013.

[16] D. Droeschel, M. Schwarz, and S. Behnke, "Continuous mapping and localization for autonomous navigation in rough terrain using a 3D laser scanner," *Robotics and Autonomous Systems*, vol. 88, pp. 104–115, 2017.

[17] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Conference on Neural Information Processing Systems (NIPS)*, 2003.

[18] F. Schilling, X. Chen, J. Folkesson, and P. Jensfeld, "Geometric and visual terrain classification for autonomous mobile navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

Learning Abstract Representations for Locomotion Planning in High-dimensional Configuration Spaces

Tobias Klamt and Sven Behnke

***Abstract*— Ground robots which can navigate in a variety of terrains are needed in many domains. Key is the capability to adapt to the ground structure, which can be realized through additional degrees of freedom. However, planning respective locomotion is challenging since suitable representations result in large configuration spaces. Employing an abstract, low-dimensional representation can support the planning.**

We propose a method to learn such an abstract representation. While a desired robot representation and action set can be easily defined, the tuning-intensive environment representation and cost function are represented as a CNN. Training of the network is done on generated artificial data, but it generalizes well to the abstraction of real world scenes. We further apply our method to the problem of search-based planning of hybrid driving-stepping locomotion. The learned abstract representation is used as a powerful informed heuristic which accelerates planning by multiple orders of magnitude.

I. Introduction

Most planners for robot locomotion planning feature search-based or sampling-based approaches which show great performance in 2D and 3D planning problems, e.g., driving robot locomotion [1]–[3]. However, many areas of operation, e.g., in search and rescue scenarios, have challenging properties. Suitable robots need to adapt to those environments to provide fast, safe, and energy efficient locomotion which can be realized through, e.g., tracked flippers, legs, or wheels at adjustable limbs. Including these capabilities in the planning problem results in high-dimensional configuration spaces (C-spaces) which may lead to extensive searches for search- and sampling-based approaches.

In recent years, intensive research has been performed on learning-based motion planning approaches [4]–[8] which learn to map a given situation to a reasonable action without performing extensive searches. However, the necessary amount of training data and the required network complexity strongly depend on the size of the considered maps and the number of C-space dimensions. Thus, at the current state-of-the-art, learning-based planners are restricted to small maps or low-dimensional planning problems.

A well investigated idea to accelerate planning in large C-spaces is abstraction. An abstract representation has a coarser resolution or fewer C-space dimensions and compensates this information loss through additional features to increase semantics. Given such an abstract representation, the necessity of the detailed representation during planning may be reduced to certain situations. A valuable property of an abstract representation to be used along with a detailed representation (e.g., as a heuristic or for coarse-to-fine planning) is that the same actions induce the same costs in both representations (*cost similarity*).

All authors are with Rheinische Friedrich-Wilhelms-Universität Bonn, Computer Science Institute VI, Autonomous Intelligent Systems, Bonn, Germany `klamt@ais.uni-bonn.de`, `behnke@cs.uni-bonn.de`. This work was supported by the European Union's Horizon 2020 Programme under Grant Agreement 644839 (CENTAURO).

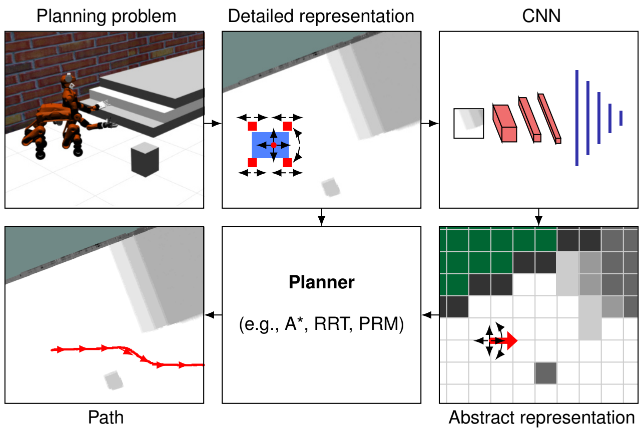


Fig. 1. A CNN is used to represent an abstract representation of a detailed planning problem which is employed to support the planner.

In [9], we propose a high-dimensional search-based planner for hybrid driving-stepping locomotion which we extend in [10] to multiple levels of abstraction. This shows promising results since planning is significantly accelerates while the resulting path quality stays comparable. However, abstract representations were manually designed and parametrized to obtain *cost similarity* which is a challenging and exhausting task.

In this paper, we propose a method to represent abstract representations as convolutional neural networks (CNNs) which map a spatially small planning task to a corresponding cost assessment for the shortest path. Since these small planning tasks represent a coarse, low-dimensional set of actions, the CNN represents a *cost similar* abstract representation for the high-dimensional planning problem (see Fig. 1).

We train the CNN on artificially generated data and evaluate it on simulated and real-world sensor data. Furthermore, the method is used to generate a powerful heuristic for hybrid driving-stepping locomotion planning, but it can be easily transferred to other domains, e.g., walking locomotion. The results indicate that the proposed method outperforms our manually tuned approach in terms of abstraction quality while eliminating tuning efforts and that the proposed heuristic accelerates path planning by multiple orders of magnitude compared to popular heuristics.

II. Related Work

Most robot motion planning approaches are either sampling-based, such as Rapidly-exploring Random Trees (RRT) [1] or Probabilistic Roadmaps (PRM) [2], search-based, such as A* [3] or a combination of those [11]. Low-dimensional motion planning in 2D or 3D C-spaces, can be seen as solved with these approaches. However, it is still challenging to solve high-dimensional, large planning problems since the required computational power and memory significantly increases with an increasing C-space size.

A solution to handle large environment sizes is multiresolution planning [12]. To handle high-dimensional C-spaces, a local adaptation of the robot representation is an option. In previous work, we proposed a search-based approach to plan hybrid driving-stepping locomotion [9]. Similarly, Dornbush et al. [13] plan multi-modal paths for a humanoid with a search-based planner. Both approaches handle the occuring high-dimensional C-spaces by separating the planning problem with respect to the locomotion mode and apply high-dimensional planning only if required. Nevertheless, both works suffer the problem of handling large scenarios in feasible time since the high-dimensional represented areas are still too large.

However, those approaches only neglect information in their coarse/low-dimensional representations which might result in wrong assessments, especially for complex terrain. This is addressed by abstraction: Representations are coarser but semantically enriched to compensate the information loss. A theoretical basis for abstraction for search-based planning is given by Holte et al. [14]. In [10], we extended hybrid driving-stepping locomotion planning to three levels of abstraction. With increasing abstraction, the environment is represented in a coarser resolution but with additional hand-crafted features such as height differences or terrain classes. In addition, the robot representation has a coarser resolution and less dimensions with increasing abstraction. The costs functions are manually tuned to obtain *cost similarity*. This is done by iteratively comparing costs on a small set of exemplary tasks and adjusting parameters. The abstract representations accelerates planning by multiple orders of magnitude while the path quality stays comparable. Especially the utilization of the most abstract representation as a heuristic lead to significant speedup. However, the design of descriptive features and tuning of cost functions required extensive manual parametrization and is very dependent on the used set of exemplary tasks.

In recent years, learning-based approaches for solving robot motion planning problems were proposed. In [4] and [5] CNNs are trained to map camera images directly to motor commands, e.g., for manipulation tasks or steering of a self-driving car. However, such approaches are missing a long-term goal-directed behavior. Tamar et al. [6] proposed a differentiable approximation of the value iteration algorithm which can be represented as a CNN—the Value Iteration Networks (VINs). Their performance was evaluated on small 2D grid worlds. Similarly, Karkus et al. [7] proposed QMDP-Net which is also capable of planning in 2D grid worlds. Srinivas et al. [8] proposed Universal Planning Networks (UPNs) which map images of the initial and goal scene to actions. In contrast to the VINs evaluation, UPNs are also evaluated for robots with more degrees of freedom (DoF) but the considered maps are rather small. These three approaches point out the general problem of learning-based approaches at the current state of the art: The required amount of training data and the required network complexity are not manageable for large, high-dimensional planning problems.

To summarize, learning-based planning approaches can handle local problems with limited state space sizes quickly without performing intensive searches. In contrast, traditional planning approaches show good goal-directed behavior but might get stuck in intensive searches for complex high-dimensional problems. Hence, it is an interesting idea to combine these approaches and merge the advantages of both. Faust et al. [15] use a reinforcement learning (RL) agent to learn short-range, point-to-point navigation policies for 2D and 3D action spaces which capture the robot dynamic and task constraint without considering the large-scale topology. Sampling-based planning is used to plan waypoints which give the planning a long-range goal-directed behavior.

In contrast to that work, we combine CNNs and search-based planning to handle 7-dimensional hybrid driving-stepping locomotion planning. The CNN represents an abstract representation of the high-dimensional planning problem which is used as a heuristic to accelerate planning.

III. Problem Statement

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

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

$$r_\text{a,i} = \mathcal{T}_{\text{d} \mapsto \text{a}}(r_\text{d,i}) \qquad (1)$$

and vice versa with

$$r_\text{d,i} = \mathcal{T}_{\text{a} \mapsto \text{d}}(r_\text{a,i}). \qquad (2)$$

$\mathcal{A}_\text{a}$ describes actions to move an abstract robot configuration $r_\text{a,i} \in \mathcal{R}_\text{a}$ to a successive configuration $r_\text{a,i+1} \in \mathcal{R}_\text{a}$. The resolution of $\mathcal{A}_\text{a}$ is coarser compared to $\mathcal{A}_\text{d}$, such that a sequence of actions

$$\mathcal{T}_{\text{a} \mapsto \text{d}}(r_\text{a,i}) + a_\text{d,j} + a_\text{d,j+1} + \dots + a_\text{d,j+k} = \mathcal{T}_{\text{a} \mapsto \text{d}}(r_\text{a,i+1}), \qquad (3)$$

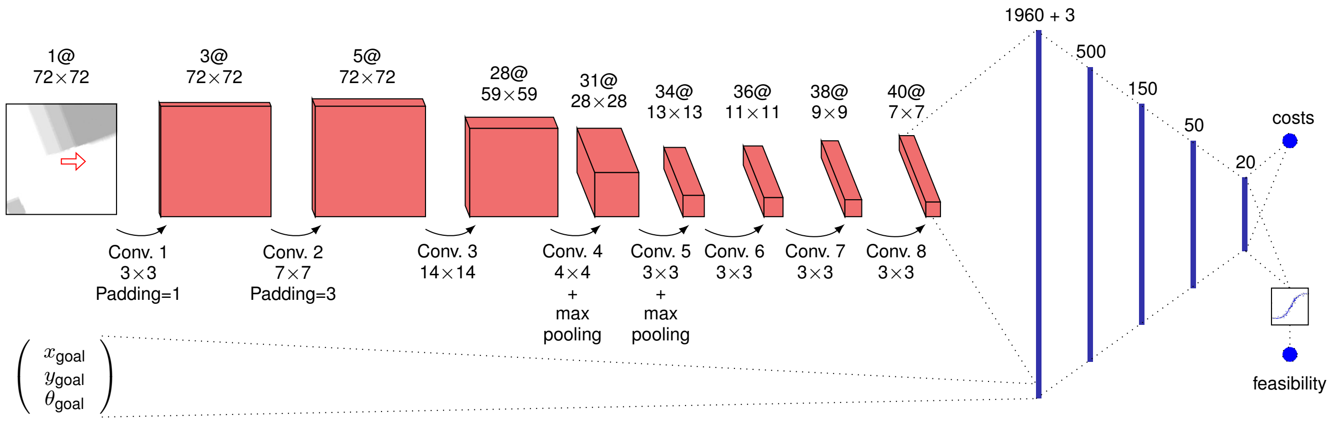


Fig. 2. Architecture of the proposed CNN. Input are a height map patch and the goal pose. Although it is not fed into the network, the start pose is depicted as a red arrow for better understanding. Output are the *feasibility* and *costs* values. Convolutional layers are visualized as red cuboids, blue lines show fully connected layers. If not stated different, convolutions have a padding of 0 and a stride of 1.

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

While $\mathcal{R}_a$ and $\mathcal{A}_a$ can be easily defined, $\mathcal{E}_a$ and $\mathcal{C}_a$ need extensive tuning. We propose to represent $\mathcal{E}_a$ and $\mathcal{C}_a$ in a CNN to avoid these tuning efforts and improve abstraction quality.

IV. Network Design

We propose a regular CNN architecture—consisting of convolutional layers and successive fully connected layers—to learn the abstract representation (see Fig. 2). Input are a height map patch with 72×72 pixels and the three-dimensional abstract goal pose $r_{a,g}$. The start pose $r_{a,s}$ is assumed to be always in the map center with a fixed orientation, but is not fed into the network. For a given resolution of 2.5 cm, the map size is chosen such that for every abstract goal pose $r_{a,g} = r_{a,s} + a_{a,j} \in \mathcal{A}_a$, the corresponding detailed goal pose $r_{d,g} = \mathcal{T}_{a\mapsto d}(r_{a,g})$ with any feasible leg configuration is completely inside this map patch. $r_{a,g}$ is defined in resolution steps relative to $r_{a,s}$.

Instead of only outputting costs which become infinite for infeasible queries, we output two values: The *feasibility* value describes whether there exists a solution for the query, and, if so, the *costs* value describes the corresponding costs.

We discovered that key to a good abstraction performance are some convolutions with large kernels. A first small convolution extracts descriptive map features from the input height map. Consequently, a second convolution possesses a kernel size which is similar to the size of a robot foot and thus can determine if foot placement is possible at a given position. The kernel size of the third convolution is chosen such that is covers the maximum action length for an individual foot. Hence, it can find connections between feasible foot positions in a certain distance which is valuable for, e.g., steps. The following convolutions and max pooling operations with small kernels do further processing on the actions and are followed by six fully connected layers.

The last fully connected layer is split: While *costs* are output directly, the *feasibility* output is processed by a sigmoid function since it is boolean.

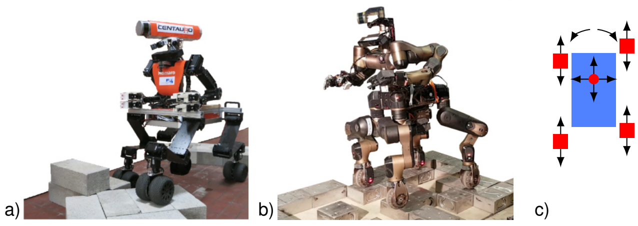


Fig. 3. Hybrid driving-stepping locomotion robots. a) Momaro, b) Centauro, c) corresponding detailed robot representation (blue = robot base, red squares = feet, arrows visualize the DoF).

V. Learning Abstraction of Hybrid Driving-Stepping Locomotion Planning

We apply the method to hybrid driving-stepping locomotion planning for our platforms e.g., Momaro [16] and Centauro [17] (see Fig. 3 a, b). Both are able to perform omnidirectional driving and stepping motions. A detailed robot representation which matches both robots is depicted in Fig. 3 c.

A. Detailed Representation

The detailed environment representation $\mathcal{E}_d$ possesses a height map which is generated from registered point clouds (Fig. 4 a). Regarding $\mathcal{C}_d$, foot costs (see Fig. 4 b) and base costs, which describe the costs to place a single foot/the base at a given position/in a given configuration on the map, are computed from this height map. Foot costs and base costs are merged to pose costs. The robot is represented in 7D configurations $r_d \in \mathcal{R}_d = (r_x, r_y, r_\theta, f_1, ..., f_4)$ with the robot base pose (r_x, r_y, r_θ) and the longitudinal position of each foot $f_1, ..., f_4$ as shown in Fig. 3 c. Positions have a resolution of 2.5 cm while there are 64 discrete orientations. Lateral foot positions are fixed and foot heights are computed after a result path is found.

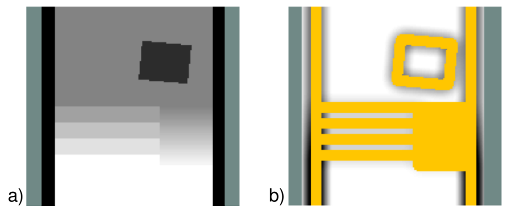


Fig. 4. Detailed environment representation: a) Input height map, b) foot cost map (yellow = untraversable by driving, olive = unknown).

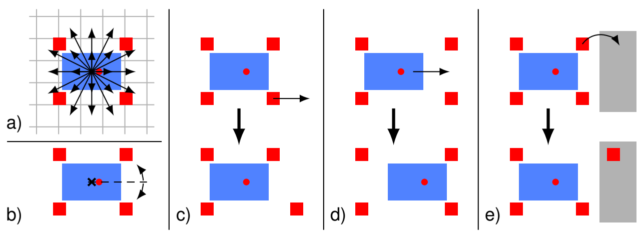


Fig. 5. Robot actions in $\mathcal{A}_{\mathrm{d}}$: a) Omnidirectional driving with fixed orientation, b) turning with fixed position, c) moving a foot relative to the base while keeping ground contact, d) longitudinal base shift, e) step. Grid and orientation resolution are enlarged to facilitate visualization.

Robot actions in $\mathcal{A}_{\mathrm{d}}$ are (see Fig. 5):

- Omnidirectional driving within a 20-neighborhood with fixed orientation,
- turning to the next discrete orientation,
- moving an individual foot relative to the robot base while keeping ground contact,
- moving the base longitudinal relative to the feet, and
- performing a step with a single foot.

Steps are represented as the direct transition from a pre-stepping to a post-stepping pose. Only those steps in the result path are refined to detailed motion sequences which consider robot stability and the detailed stepping motion. Each action carries costs with respect to the occuring foot and base costs that the individual robot elements experience. For driving locomotion, a large angular difference between the robot orientation and driving direction is punished with higher costs to prefer driving forward which brings advantages to the perception of the environment directly in front of the robot and when switching to stepping motions in the sagittal direction. An A*-based planner which uses the above presented representation is used to plan hybrid driving-stepping locomotion paths. More details can be found in [9].

B. Abstract Representation

$\mathcal{R}_{\mathrm{a}}$ contains 3D robot configurations $r_{\mathrm{a}} = (r_{\mathrm{x}}, r_{\mathrm{y}}, r_{\theta})$ which describe the robot position with a resolution of 10 cm and the orientation in 16 discrete steps. Individual foot configurations are neglected. $\mathcal{A}_{\mathrm{a}}$ contains

- moving the robot within a 20-neighborhood with fixed orientation (see Fig. 5 a) and
- turning to the next discrete orientation with fixed position (see Fig. 5 b).

Transforming a detailed robot configuration to an abstract robot configuration ($\mathcal{T}_{\mathrm{d} \mapsto \mathrm{a}}$) is done by neglecting the foot positions and matching the position and orientation to the coarse resolution of the abstract C-space. The transformation from an abstract to a detailed robot configuration $\mathcal{T}_{\mathrm{a} \mapsto \mathrm{d}}$ is more complicated: For all detailed robot base poses that match the abstract configuration, we search the least cost foot configuration while preferring configurations which are close to the neutral robot pose (Fig. 5 a). The detailed robot pose with the minimum pose costs is the transformation result.

C. Network Training

Training data is generated artificially. Hence, large amounts of training data can be produced without considerable effort. A map generator produces height maps of the desired network input size. The following obstacles are placed randomly in those height maps:

- Cuboid shaped obstacles of random size,
- walls of random length and height, and
- staircases of random width with a random number of stairs (with random height and length).

We produce 2,000 maps of each of the following categories:

- one/two/three cuboid obstacles,
- one/two walls,
- one cuboid obstacle and one wall,
- one staircase,
- one staircase and one wall, and
- one staircase whose orientation is in the interval $\left[-\frac{\pi}{16}, \frac{\pi}{16}\right]$ around the robot orientation. Those maps are used to set a learning focus on stair climbing.

For each map, we define 22 abstract goal configurations $r_{\mathrm{a,g_i}}$ with respect to $\mathcal{A}_{\mathrm{a}}$. The start configuration $r_{\mathrm{a,s}}$ is always in the map center with a fixed orientation. $r_{\mathrm{a,s}}$ and $r_{\mathrm{a,g_i}}$ are transformed to $\mathcal{R}_{\mathrm{d}}$ using $\mathcal{T}_{\mathrm{a} \mapsto \mathrm{d}}$. For some maps a valid detailed start configuration $r_{\mathrm{d,s}}$ cannot be found due to obstacles. Those maps are deleted. In total we get a set of 11,327 maps with 249,194 tasks. Subsequently, we search for a shortest path from $r_{\mathrm{d,s}}$ to $r_{\mathrm{a,g_i}}$ with our detailed A*-planner. For each task we save the *feasibility* flag which describes if a path could be found. *Costs* are saved for all feasible tasks.

The network is trained using the SGD optimizer with a learning rate of 0.0001 and a momentum of 0.9. We use a BCE loss function for the *feasibility* and a L1 loss function for the *costs*. The *costs* loss is only considered in the backpropagation if the task is feasible. Losses are weighted with $\mathcal{W}_{\mathrm{feasible}}$ and $\mathcal{W}_{\mathrm{costs}}$, both starting at 1. If no improvement by means of a decreasing loss is achieved in three successive training epochs, the corresponding loss weight is divided by 5. This dynamic is applied to both losses individually. For evaluation, a threshold of 0.5 is used to make the *feasibility* output boolean. A validation set which includes 100 maps of each mentioned category is generated and used to evaluate the training performance (Fig. 6). We train the network for 100 epochs and choose the state with the best results on the validation set for our experiments.

D. Abstract Representation as Heuristic

We utilize the learned abstract representation as a heuristic for planning in the detailed representation. For a given goal pose $r_{\mathrm{d,g}}$, a one-to-any 3D Dijkstra search is started from

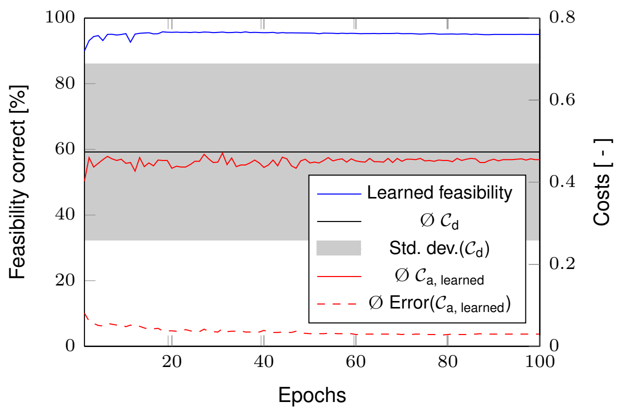


Fig. 6. CNN training performance. $\mathcal{C}_d$ is shown as a base line.

$r_{a,g} = \mathcal{T}_{d \mapsto a}(r_{d,g})$ and explores the whole map in the abstract representation. During that search, neighbor poses r_{a,n_i} for a pose $r_{a,m}$ are generated through abstract actions $a_{a,i}$ such that $r_{a,\ n_i} + a_{a,i} = r_{a,\ m}$ while respective costs are computed by the CNN which is fed with the respective height map patches. Please note that start and goal of each action are exchanged since this search is running backwards, starting at the planner goal pose. While actions which are assessed as infeasible are neglected, feasible actions are assigned the corresponding *costs* output.

Consequently, each abstract pose in the map carries the estimated costs of the shortest path to $r_{d,g}$. When planning in the detailed representation, the planner uses these cost estimations as an informed heuristic.

Please note that we cannot prove that this heuristic always underestimates costs, and thus, we cannot prove admissibility for the generation of optimal paths. We rather focus on the generation of paths with a satisfying quality in feasible time and thus accept suboptimality to speedup planning.

The CNN is implemented using Python 2.7 and PyTorch 0.4.1. The planner is implemented in C++. Communication is realized via ROS. Code for the CNN, the training data generator and the framework to use the CNN as a heuristic is available online[1].

VI. Experiments

We evaluate the proposed approach in two experiments which compare the abstraction quality to the manually tuned abstraction of our previous work and show the performance of the proposed heuristic to plan hybrid driving-stepping locomotion. A video which shows additional footage of the experiments is available online[2].

A. Abstraction Quality

The abstraction quality is evaluated on three data sets:

- *random*: We generate 200 random maps of each category resulting in a set of 1,124 maps with 24,728 tasks.

[1]`https://github.com/AIS-Bonn/planning_abstraction_net`

[2]`https://www.ais.uni-bonn.de/videos/ICRA_2019_Klamt/`

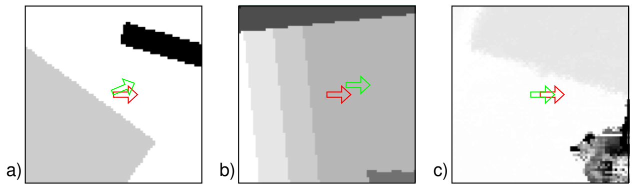


Fig. 7. Example tasks of the *random* test set (a), *simulated* test set (b), and *real* test set (c). Red arrows show start poses (not fed into the CNN), green arrows show goal poses.

TABLE I
Abstraction Quality Evaluation

	random	*simulated*	*real*
Ø $\mathcal{C}_d$	0.476	0.466	0.509
Std. dev.($\mathcal{C}_d$)	0.222	0.202	0.236
feasibility correct, CNN	95.04%	96.69%	92.62%
Ø $\mathcal{C}_{a,\text{CNN}}$	0.453	0.469	0.446
Ø Error($\mathcal{C}_{a,\text{CNN}}$)	0.027	0.013	0.081
feasibility correct, man.tuned	79.27%	65.35%	69.77%
Ø $\mathcal{C}_{a,\text{man.tuned}}$	0.435	0.402	0.429
Ø Error($\mathcal{C}_{a,\text{man.tuned}}$)	0.057	0.021	0.103

- *simulated*: Height map patches of the desired size are cut out from height maps of simulated planning scenes. This set includes 77 maps with 1,694 tasks.
- *real*: Height map patches of the desired size are 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.

We compared the performance to the manually tuned abstraction approach from our previous work. Finally, costs for the tasks in the detailed representation $\mathcal{C}_d$ are stated as a base line. We evaluate the *feasibility* and *costs* output. 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 *costs similarity* of the two representations. Figure 7 shows some example tasks. The abstraction performance of the proposed CNN is shown in Tab. I.

The results indicate that the CNN *feasibility* output is significantly better compared to the manually tuned abstraction. While the latter has problems in simulated and real world robot environments, the CNN assesses a correct *feasibility* for $> 92.62\%$ of the tasks throughout all test sets.

Regarding the *costs* assessment, 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 distribution of the base line costs. While the error of the proposed CNN is $< 5.67\%$ of the absolute *costs* on the *random* and *simulated* test sets, it is 15.9% on the *real* test set. This might be explained by noisier sensor measurements which result in noisier height maps.

B. Application to Planning

We designed an arena in Gazebo simulation which includes typical locomotion tasks for Centauro in search and rescue missions (Fig. 8). Environment perception is realized through a continuously rotating Velodyne Puck 3D laser

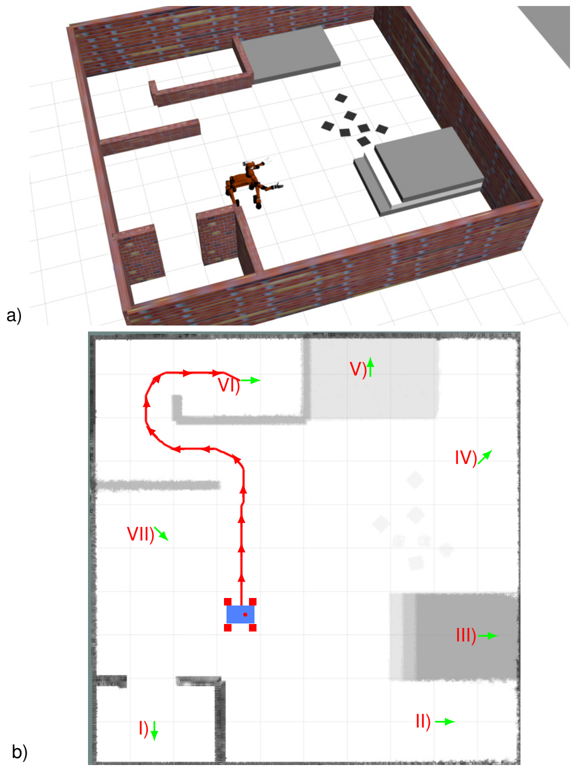


Fig. 8. Locomotion planning experiment. a) Gazebo arena with Centauro. b) Height map with start pose (blue/red) and goals (arrows): I) Behind a narrow door, II) next to stairs, III) on top of stairs, IV) behind some clutter, V) on a platform, VI) inside a labyrinth, and VII) behind the robot. The red path is the resulting path to VI with the proposed heuristic and $\mathcal{W} = 1.25$.

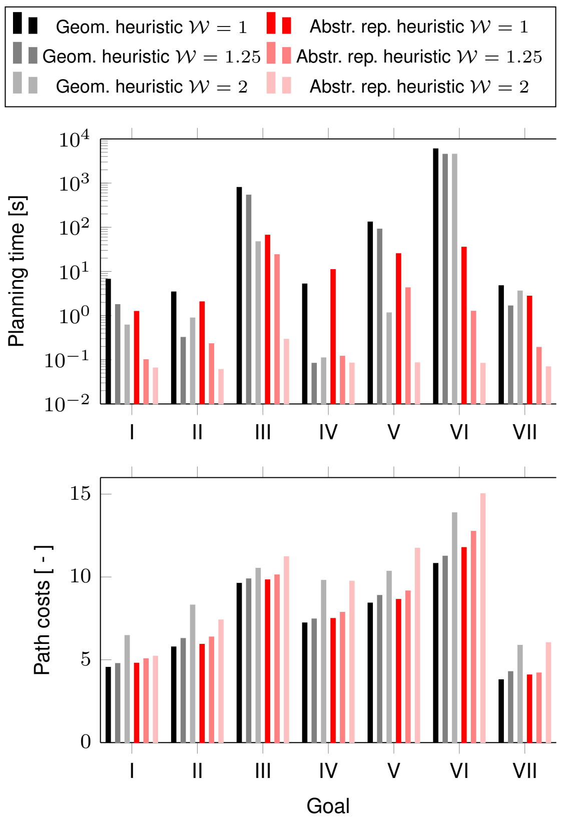


Fig. 9. Planning times (including heuristic generation) and path costs for all goal poses. The heuristic which is based on the learned abstract representation is compared to the geometric heuristic.

TABLE II

HEURISTIC PERFORMANCE

	Abstract representation			Geometric	
$\mathcal{W}$	1.0	1.25	2.0	1.25	2.0
speedup factor	27.80	708.5	10860	12.00	27.88
costs increase	+4.77%	+10.5%	+33.1%	+6.07%	+33,9%

scanner with spherical field-of-view at the robot head. Sensor measurements are processed to registered point clouds and used for localization using the method by Droeschel et al. [18]. Height maps are generated from these point clouds. The used system is equipped with an Intel Core i7-8700K@3.70 GHz, 64 GB RAM and an NVidia GeForce GTX 1080Ti with 11 GB memory.

For all abstract poses, height map patches are extracted and neighbors with corresponding costs are precomputed by the CNN which takes 239 s and is only required once per map. We plan a path to all goals while using the learned abstract representation as a heuristic. The one-to-any Dijkstra search which starts from each goal pose and generates the heuristic takes 0.049 s in average. We compare the planning performance to planning with a geometric heuristic. This combines Euclidean distances with rotational differences and is admissible. Hence, when used with a weight $\mathcal{W} = 1$, results are optimal. Both heuristics are evaluated with multiple $\mathcal{W} \geq 1$ to also obtain accelerated, sub-optimal solutions.

Figure 9 visualizes the planner performance for both heuristics and different $\mathcal{W}$. Tab. II summarizes the resulting speedup and cost increase compared to the optimal solution. The results indicate that the proposed abstraction-based heuristic accelerates planning by multiple orders of magnitude while, in particular for $\mathcal{W} = 1.25$, path costs stay comparable. This significantly outperforms the geometric heuristic. 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 may expand many poses before considering expensive actions. In contrast, the proposed abstraction-based heuristic uses its costs assessments to support the planner in its goal-directed behavior while knowledge about the environment is included.

VII. CONCLUSION

In this paper, we presented a CNN which learns the environment representation and cost function of a 3-dimensional abstract representation for a high-dimensional locomotion planning problem. The CNN maps a local planning task, consisting of a height map patch and goal pose, to a costs assessment for this task. We demonstrate, how such a CNN can generate an informed heuristic for search-based high-dimensional planning. Experiments show that such a heuristic accelerates planning by multiple orders of magnitude, especially for challenging tasks. We further show, that the CNN outperforms a manually tuned abstract representation from previous work while eliminating tuning efforts.

REFERENCES

[1] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” 1998.

[2] L. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[3] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[4] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” *The Journal of Machine Learning Research (JMLR)*, vol. 17, no. 1, pp. 1334–1373, 2016.

[5] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, *et al.*, “End to end learning for self-driving cars,” *ArXiv preprint arXiv:1604.07316*, 2016.

[6] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, “Value iteration networks,” in *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[7] P. Karkus, D. Hsu, and W. S. Lee, “QMDP-Net: Deep learning for planning under partial observability,” in *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[8] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, “Universal planning networks,” *ArXiv preprint arXiv:1804.00645*, 2018.

[9] T. Klamt and S. Behnke, “Anytime hybrid driving-stepping locomotion planning,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[10] T. Klamt and S. Behnke, “Planning hybrid driving-stepping locomotion on multiple levels of abstraction,” in *IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2018.

[11] J. Li, S. Liu, B. Zhang, and X. Zhao, “RRT-A* motion planning algorithm for non-holonomic mobile robot,” in *IEEE Society of Instrument and Control Engineers Annual Conference (SICE)*, 2014.

[12] S. Behnke, “Local multiresolution path planning,” in *RoboCup 2003: Robot Soccer World Cup VII*, Springer, 2003, pp. 332–343.

[13] A. Dornbush, K. Vijayakumar, S. Bardapurkar, F. Islam, M. Ito, and M. Likhachev, “A single-planner approach to multi-modal humanoid mobility,” in *IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2018.

[14] R. C. Holte, M. Perez, R. Zimmer, and A. MacDonald, “Hierarchical A*: searching abstraction hierarchies efficiently,” in *Symposium on Abstraction, Reformulation, and Approximation*, 1995.

[15] A. Faust, O. Ramirez, M. Fiser, K. Oslund, A. Francis, J. Davidson, and L. Tapia, “PRM-RL: long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning,” in *IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2018.

[16] M. Schwarz, T. Rodehutskors, D. Droeschel, M. Beul, M. Schreiber, N. Araslanov, I. Ivanov, C. Lenz, *et al.*, “NimbRo Rescue: Solving disaster-response tasks with the mobile manipulation robot Momaro,” *Journal of Field Robotics*, vol. 34, no. 2, pp. 400–425, 2017.

[17] T. Klamt, D. Rodriguez, M. Schwarz, C. Lenz, *et al.*, “Supervised autonomous locomotion and manipulation for disaster response with a centaur-like robot,” Accepted for IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2018, `http://www.ais.uni-bonn.de/papers/IROS_2018_Klamt.pdf`.

[18] D. Droeschel, M. Schwarz, and S. Behnke, “Continuous mapping and localization for autonomous navigation in rough terrain using a 3D laser scanner,” *Robotics and Autonomous Systems*, vol. 88, pp. 104 –115, 2017.

Value Iteration Networks on Multiple Levels of Abstraction

Daniel Schleich, Tobias Klamt, and Sven Behnke

***Abstract*— Learning-based methods are promising to plan robot motion without performing extensive searches, as done by non-learning approaches. Value Iteration Networks (VINs) received much interest since—in contrast to standard CNN-based architectures—they learn goal-directed behaviors which generalize well to unseen domains. However, VINs are restricted to small and low-dimensional domains; which limits their applicability to real-world planning problems.**

We propose VINs on multiple levels of abstraction. While the vicinity of the robot is represented in sufficient detail, the representation gets spatially coarser but more descriptive with increasing distance from the robot. This enables VINs to solve more complex planning tasks. We show that our approach outperforms original VINs in 2D grid worlds in terms of planning success and memory requirement. Further, we employ our approach to plan omnidirectional driving for a search-and-rescue robot in cluttered terrain.

I. Introduction

While search-based and sampling-based methods are well investigated for motion planning, they tend to extensive searches for complex high-dimensional tasks due to the lack of scene understanding [1]–[3].

In other domains, especially in perceptual contexts, convolutional neural networks (CNNs) are well investigated [4], [5]. An increasing number of works applied CNNs to robot motion planning in recent years. This is promising since CNNs enable planning without extensive searches. Standard CNN architectures have been employed to map system state observations directly to actions [6], [7]. However, those approaches have difficulties to understand the goal-directed behavior of planning and to generalize to unseen domains.

This issue is addressed by, e.g., Value Iteration Networks (VINs) [8] or Universal Planning Networks (UPNs) [9]. Instead of following a strict feed-forward approach, values iterate multiple times in an inner loop to be propagated through the representation. Those methods show promising results in terms of goal directed behavior and generalization to unseen domains. However, they have only been applied to small, low-dimensional problems. Planning in larger configuration spaces requires more complex network designs and significantly more training data which becomes at some point unfeasible on currently available hardware. Thus, it is challenging to apply them to most real-world planning problems.

All authors are with Rheinische Friedrich-Wilhelms-Universität Bonn, Computer Science Institute VI, Autonomous Intelligent Systems, Bonn, Germany `schleich@uni-bonn.de`, `klamt@ais.uni-bonn.de`, `behnke@cs.uni-bonn.de`.

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

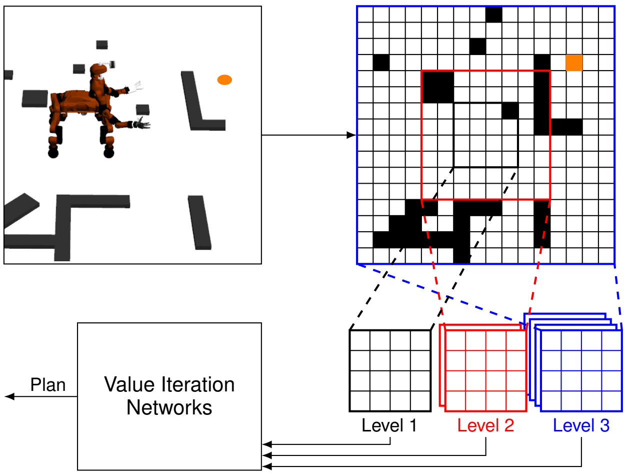


Fig. 1: The general idea of VINs on multiple levels of abstraction.

A well-established idea to handle large configuration spaces is abstraction [10], [11]. An abstract representation describes neighboring states in a spatially/temporally coarser resolution while enriching the representation with additional descriptive features.

We propose a method to combine three environment representations with increasing abstraction with VINs to obtain a learning-based planner which is capable of handling more complex tasks (Fig. 1). With increasing distance from the robot, the level of abstraction increases. While the spatial resolution decreases with an increasing level of abstraction, the number of cells is constant for all levels which results in larger covered areas for more abstract maps. In addition, an increasing level of abstraction comes along with an increasing number of descriptive features for each cell.

Experiments show that our approach outperforms VINs in 2D grid worlds in their original implementation and an extension to employ multiple hierarchical representations, namely Hierarchical VINs (HVINs). We further apply our method to plan omnidirectional driving locomotion while considering the individual configuration of ground contact areas (which we refer to as the robot footprint). Results show that planning on multiple levels of abstraction enables VIN to solve larger planning tasks while the result quality increases and memory requirements decrease.

II. Related Work

Most planning problems can be described as a Markov Decision Process (MDP) which consists of state and action spaces, transition probabilities, and rewards for each state-action pair [12]. The goal is to find a policy which results in high long-term rewards. One common algorithm to find such an optimal policy is Value Iteration (VI). By applying the Bellman equation [13], it calculates for each state the expected long-term reward. The optimal policy is obtained by always moving to the state with the highest state value.

While CNNs are well investigated for tasks such as image classification [4] and robot perception [5], their application to motion planning arose in recent years. Traditional CNN architectures have been used to learn policies and directly derive actions from state observations. In [6] and [7], CNNs are trained to map raw images to robot motor torques for real-world manipulation tasks and autonomous car steering. Although the results of these applications are impressive, such approaches lack the capability of including long-term goal directed behavior and generalization to unseen domains.

In 2016, Tamar et al.[8] proposed VINs. An explicit planning module approximates the Value Iteration Algorithm by rewriting the application of Bellman equations (which we refer to as Bellman update) as a CNN. Since this planning module is fully differentiable, standard backpropagation can be used to learn the parameters of the model, like a suitable reward function or state transition probabilities. The embedded planning operation enables VINs to generalize well to unseen environments and understand the desired goal-directed behavior. However, VINs do not scale well to larger map sizes or higher-dimensional planning tasks, since the number of required Bellman updates depends on the state space size. Larger state spaces result in longer training times and increasing memory requirements. Hence, evaluation is limited to small 2D grid worlds though. In the supplementary material of this work, HVINs are proposed to reduce the number of necessary Bellman updates. Value iteration is first performed on a down-sampled copy of the input map to generate rough state-value estimates, which are up-sampled and used as initialization for another value iteration module working on the full resolution. This model can be extended to multiple hierarchical layers. However, the information loss through down-sampling is not compensated. Furthermore, all layers operate on the whole environment size resulting in only slightly decreasing memory requirements.

In addition, VINs have been applied in other domains. Niu et al.[14], proposed generalized value iteration networks which work on arbitrary irregular graph structures and can be applied to real world data like street maps. Karkus et al.[15] proposed QMDP-nets which handle partially observable environments and express value iteration through a CNN.

UPNs by Srinivas et al.[9] learn useful latent state representations from images of the current scene and the desired goal scene. They infer motion trajectories by performing gradient descent planning and iterating over action sequences in the learned internal representations. Considered environments may have more than two dimensions but are rather small. The gradient descent planner is very time consuming and hinders scaling to larger environments. Impressively, UPNs are able to generalize to modified robot morphologies.

In other domains, abstraction is an established method to handle large state spaces. Abstract states unify multiple detailed states. This can be realized through coarser resolutions or lower-dimensional representations while the loss of information is compensated by additional features which increase the representation's semantics. In [11] the search-based approach for the high-dimensional problem of hybrid driving-stepping locomotion planning [16] is extended to plan on multiple levels of abstraction which results in significantly shorter planning times while the result quality stays comparable. In [10] temporal abstraction is applied to reinforcement learning which generates an efficient space to explore complicated environments.

We propose a method to combine VINs with the idea of planning on multiple levels of abstraction to obtain a learning-based planner which is capable of solving planning tasks on larger state spaces. The information loss in coarser representations is compensated through additional features. In addition, detailed representations are only generated for parts of the environment which decreases memory requirements. This increases the applicability of learning-based planning approaches to real-world problems.

III. Method

VINs internally represent each state as one cell of a multi-dimensional grid and compute a reward and state-value for each of these grid cells. To enable information flow from the goal to the start state, Bellman updates are performed repeatedly within the value iteration (VI) module. The number of required Bellman updates depends on the grid size. For large and high-dimensional grids, this leads to large computation graphs for the gradients during backpropagation, resulting in long training times and high memory consumption. Since the number of states within the VI module is limited, we change what each state represents. In the vicinity of the robot, which is defined to be always in the center of each map, spatial precision is needed to plan the next robot action. Regions which are further away from the robot can be described in a coarser more abstract representation.

We define three levels of abstraction with a constant number of cells but decreasing resolution. *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 four times larger covered area. This step is repeated to obtain *Level-3*. Hence, *Level-3* covers an area which is 16 times larger than the *Level-1* area. The spatial arrangement of the three representations is depicted in Fig. 1.

To compensate the information loss in coarser representations, additional features are introduced for each abstract cell and are learned during training. We define the number of features for *Level-1*, *Level-2*, and *Level-3* to be one, two, and six, respectively.

A. Network Architecture

Input to the network (Fig. 2) is an occupancy map of the environment and an equally sized goal map which only contains zeros except for the goal cell (one-hot-map). In contrast to original VINs, we do not provide the system explicit information about the start pose, but define that input maps are always robot centered.

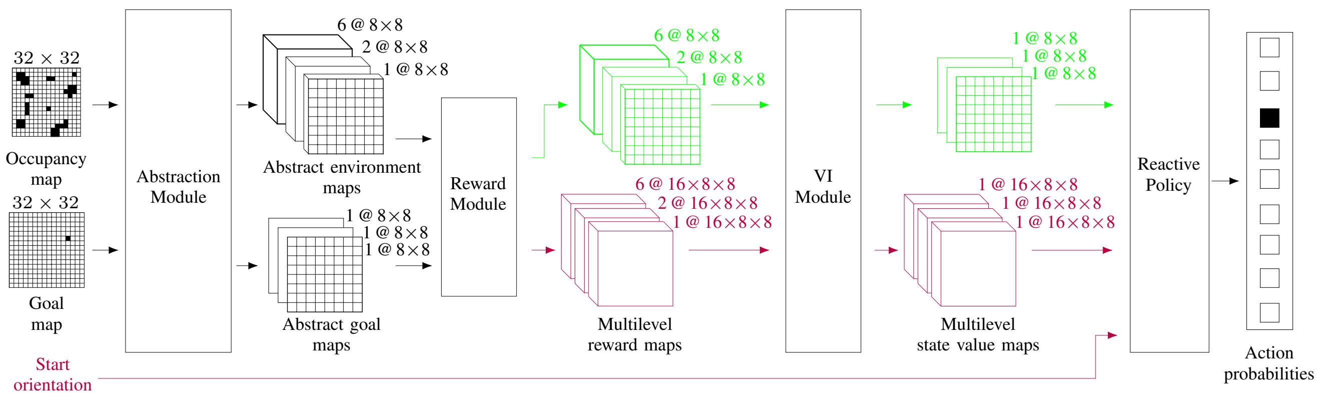


Fig. 2: Network architecture. Elements which only account to 2D grid world planning are shown in green. Elements for 3D locomotion planning are purple.

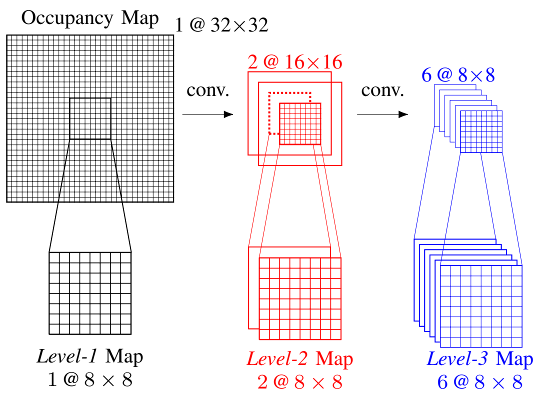


Fig. 3: Abstraction Module. Both convolutions use kernels of size 2 × 2 with a stride of 2. The goal map is processed using max pooling operations with the same parameters instead of the convolutions.

In a first step, the **Abstraction Module** (Fig. 3) processes the input environment map to three, equally sized abstract environment maps. The *Level-1* map is extracted as a patch around the center of the occupation map. A convolution generates the *Level-2* representation with halved resolution from the input map. While the *Level-2* map is again extracted from the map center, the whole *Level-2* representation is processed by another convolution to obtain the *Level-3* map. The goal map is processed similarly. However, we do not use convolutions but max pooling operations here.

Subsequently, the abstraction maps and the goal map are fed into the **Reward Module** (Fig. 4) to generate rewards for each state. Since the abstract environment maps have multiple features per cell, this needs to be considered in the reward computation. It, e.g., might be possible that an abstract map cell can be entered from one direction but not from another which is encoded in the features. Hence, multiple reward features are necessary for each cell to represent such information. It is important to understand that information encoded at the same cell of different abstraction maps refers to different locations in the environment. We support the network in understanding this relation with the following method: The *Level-1* reward map is obtained by stacking the respective environment and goal maps and processing them with two convolutions. These convolutions use a padding to keep the map size constant. Thus, the relation between cell position and environment location stays constant either.

To enable information flow between levels, the *Level-1* reward map is also used to generate the *Level-2* reward map. A max pooling operation matches the resolution of the the *Level-1* map resolution to *Level-2*. The result is padded with zeros to match the size of the *Level-2* map. This procedure ensures that information at the same cell position in both maps describe the same environment location.

Subsequently, the stacked *Level-1* and *Level-2* maps are processed as described above to obtain the *Level-2* reward map and provide the result for the *Level-3* reward map generation.

Reward maps are input to the **VI Module** (Fig. 5) where they are processed to state-values. Each iteration of the Bellman update is represented through a convolution and subsequent max pooling operation. The kernel is chosen such that it covers the set of possible actions and thus can propagate state values through the map, respectively. Unlike the reward maps, state-value maps consist of only one channel as they describe the expected long-term reward for a pose. At the beginning of each iteration, we apply a padding to the input maps as shown in Fig. 6. The padded area contains values of the neighboring cells of the next higher abstraction level. Since the reward maps vary in their number of features, we use the average over all features of one cell of the higher level map as the padding value for all respective cells in the lower level map.

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.

We apply the proposed architecture to two planning problems: 2D grid worlds and 3D robot locomotion. The former is used to compare against original VINs and HVINs while the latter demonstrates the capabilities of our approach to handle problems of higher complexity. Necessary specifications and modifications for each planning domain are described in the following. The network is implemented using Python 2.7 and PyTorch 0.4.1. Respective source code is available online[1].

1) 2D Grid Worlds: The planner is given queries for a point-like agent in 2D grid worlds. The goal map is input as a one-hot map. As actions, the agent can move to one of the eight adjacent neighbor cells (Fig. 7 a).

[1]https://github.com/AIS-Bonn/abstract_vin

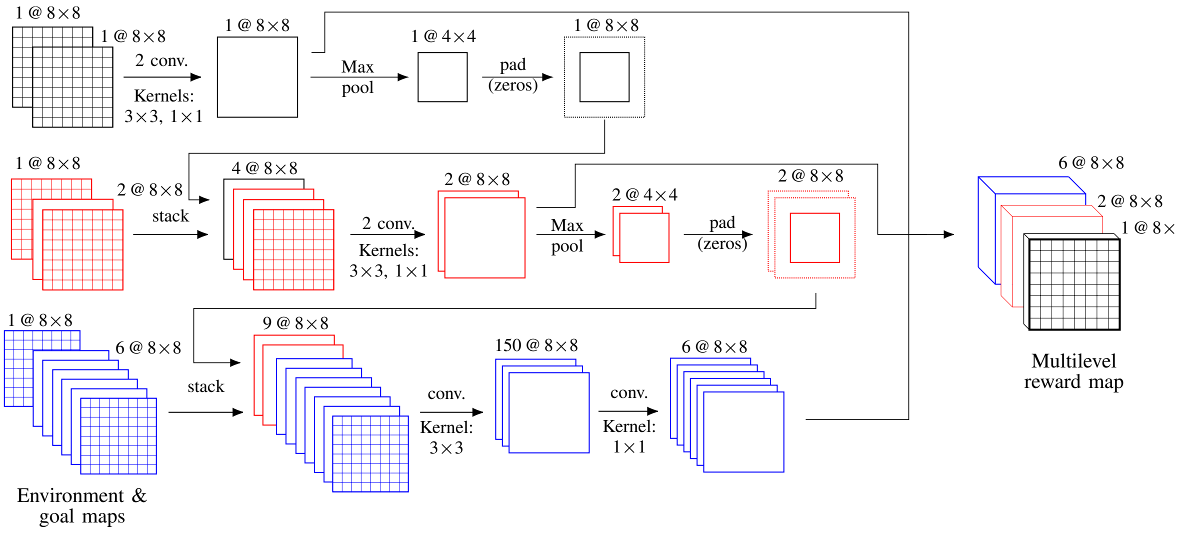


Fig. 4: Reward Module. *Level-1* maps are shown in black, red parts belong to *Level-2* and blue parts to *Level-3*. Wherever two convolutions are depicted in one step, the first maps to 150 features and the second to the depicted number of channels, as shown for *Level-3*.

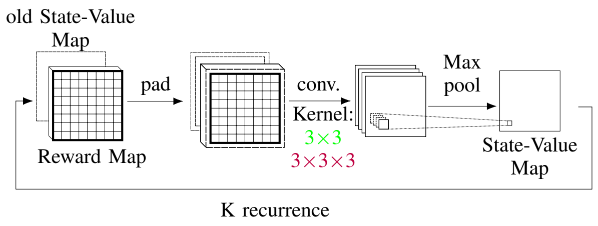


Fig. 5: Value Iteration Module. Elements which only account to 2D grid world planning are shown in green. Elements for the 3D robot locomotion planning are colored purple.

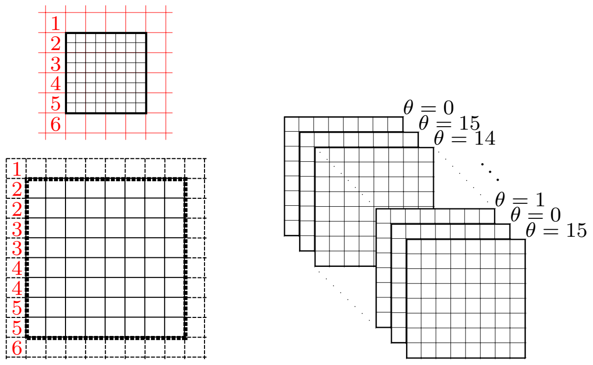


Fig. 6: Left: Padding the map of abstraction level l (bottom) to allow information flow from the map of level $l+1$ (top) to level l. The numbers indicate which values are copied where. Right: Orientation padding during 3D VIs to emphasize that the orientations $\theta = 15$ and $\theta = 0$ are neighbors.

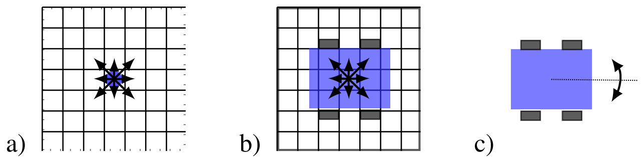


Fig. 7: Possible actions for planning domains. a) Moving to an adjacent neighbor cell in 2D grid worlds, b) drive to an adjacent neighbor state with fixed orientation in 3D robot locomotion planning, and c) turn to the next discrete orientation with fixed orientation in 3D robot locomotion planning.

2) 3D Robot Locomotion: Given a robot that can perform omnidirectional driving and has a certain footprint. Possible actions for the agent are:

- Move to one of the eight adjacent neighbor states with fixed orientation (Fig. 7 b) and
- turn to the next discrete orientation (16 equal orientation steps) with fixed position (Fig. 7 c).

When generating training data and evaluating the network, collision checking is done by checking if any cell which is occupied by the robot footprint is also occupied by an obstacle. Hence, for robots with modular footprints, it is possible to, e.g., take obstacles between their legs.

To enable the network to handle 3D agent states, reward and value maps are extended by one additional dimension for the orientation (Fig. 2). We represent the robot orientation in 16 discrete orientations of equal angular distance. Due to the increased complexity of the agent states, we increase the number of features for *Level-2* to five and for *Level-3* to ten. Furthermore, we increase the number of convolutions within the Reward Module by two additional convolutions for processing the *Level-1* map and one additional convolution for the *Level-2* map. To consider the robot footprint, we transform the reward map at the end of the Reward Module: For each possible robot base pose, we sum over the four cells corresponding to the wheel positions and assign the result to the cell corresponding to the robot base pose.

In the VI Module, the convolution kernel needs to cover all possible actions which results in a 3D kernel. Since the neighborhood relation for the orientation is cyclic, we pad the reward maps and state-value maps on the orientation channel on each end with the values of the opposite end (Fig. 6).

Other than 2D planning, the architecture for 3D planning needs information about the start and goal orientation. The start orientation is fed into our system as an additional parameter. It is only used within the Reactive Policy to select those state-values which belong to neighbor poses of the start pose. The goal orientation is encoded in the goal map in which all cell entries are 0, except for the goal cell which carries the index of the discrete orientation $(1-16)$.

B. Training

Training data is generated by placing obstacles of random number, size and position into a grid world. In addition, multiple goal states are placed randomly. The start state is defined to be in the map center. Subsequently, we use an A^* planner as an expert to generate optimal paths.

Overall, we generated 5,000 environments, each with seven planning tasks, resulting in 35,000 different training scenes. The validation and test sets both consist of 715 additionally generated environments with seven planning tasks each, resulting in 5,005 different scenes for each set.

To increase data efficiency during training, we do not only use the whole expert paths but sub-paths which are generated by randomly placing the start and goal poses on the expert path. Hence, the network is not only trained on the full paths but on many segments of every expert path which significantly increases the amount of training data.

When evaluating our approach against VINs and HVINs, all networks are trained using the RMSprop optimizer as proposed by Geoffrey Hinton in his lecture [17] which was also used in the original VIN publication. However, when using the RMSprop without any further learning rate scheduler, the network converges to sub-optimal local minima. For the 3D task, we therefore combine RMSprop with the cyclic learning rate scheduler proposed in [18]: The learning rate is decreased using a cosine annealing scheme. After several training epochs, we reset the learning rate to a higher value. 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 by 150% while the initial learning rate decreases to 95% of the previous one.

Figure 8 shows the training performance of our approach for both planning domains and two map sizes and compares the 2D grid domain to original VINs and HVINs. The network is designed to output the next action for a given input. To obtain a path for solving a planning problem, we iteratively let the network predict the next action and update the input maps according to the new robot position. A path trajectory is considered successful if it reaches the goal without hitting any obstacles and within no more than twice the optimal number of actions, as determined by the expert A^* planner. The *success* measures if the network was able to plan a path to the goal.

It can be seen that our approach obtains better *success* rates than VINs and HVINs for the 2D task on the validation set. In addition, our approach converges faster and with a higher stability. Especially on 64×64, original VINs show large instabilities in their training behavior. We choose for each network architecture the state with the best performance on the validation set to use this in our experiments.

IV. Experiments

All experiments are 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.

To evaluate the network performance, we consider two measures. The above mentioned *success* rate evaluates the network performance on the desired task: path planning. In addition, to compare against VINs, we evaluate the *accuracy* which describes how often the network chooses the same next action as the expert A^* planner. 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 planner. Hence, there occur cases in which the output of the network is different to the output of the A^* planner but the network still unrolls an optimal path although the *accuracy* measures a mistake.

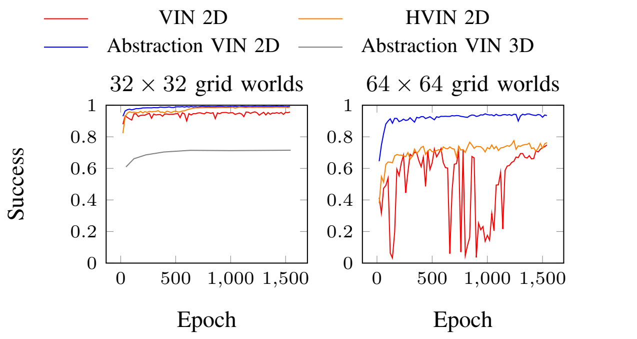


Fig. 8: Training performance of original VINs, HVINs and our approach on the validation set.

TABLE I: Results for 2D grid worlds.

32×32	VIN	HVIN	Abstract VIN
Accuracy	86.37%	**87.21%**	85.65%
Success	95.12%	98.78%	**99.32%**
Path difference	0.230	0.193	**0.144**
Graphics memory	771 MB	753 MB	**569 MB**
64×64	VIN	HVIN	Abstract VIN
Accuracy	75.18%	75.82%	**86.91%**
Success	76.00%	78.22%	**94.53%**
Path difference	0.759	1.191	**0.186**
Graphics memory	1907 MB	1417 MB	**769 MB**

A. Path Planning in 2D Grid Worlds

We compare our approach against VINs and HVINs on the 2D grid world test set. Similar to our approach, we used three hierarchical levels for HVINs, each halving the resolution of the previous level. The lowest resolution level uses the same number of Bellman updates (K) as proposed for original VINs with a similar number of cells. This coarse state-value initialization is then refined twice by two Bellman updates on the map with medium resolution and two consecutive Bellman updates on the map with fine resolution.

In [8], grid world sizes from 8×8 to 28×28 are considered. We compare on a slightly larger map with 32×32 and a significantly larger map with 64×64 cells. The performance of all three networks can be seen in Tab. I.

The results indicate that, on 32×32 grid worlds, our approach outperforms VINs and HVINs in terms of a better *success* rate while using less graphics memory and obtaining better path quality. For 64×64 the difference of our approach

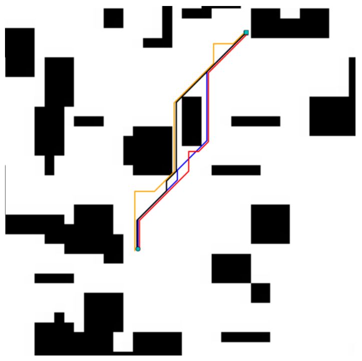


Fig. 9: Result path comparison on a 64 × 64 grid world. VINs (red) and HVINs (orange) react to obstacles when approaching them while our approach (blue) shows better long-term understanding. A optimal path obtained by our A^* planner is depicted in black.

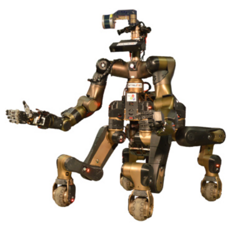


Fig. 10: The Centauro robot.

to VINs and HVINs even increases. Our success rate is significantly higher while memory requirements are only 40.3% of original VINs and 54.3% of HVINs. In particular we obtain paths which are significantly closer to the optimal solution. This can also be seen in Fig. 9 which shows paths for an example task on a 64×64 grid world. While VINs and HVINs react to obstacles when they get close to them, our approach (similar to the optimal path) seems to better include these obstacles in its long-term goal-directed behavior.

B. *Planning 3D Locomotion with Footprint Consideration*

We use the Centauro robot [19] which is a centaur-like platform with a quadruped lower body and an anthropomorphic upper body which is developed to solve a wide range of mobile manipulation tasks in search and rescue environments (Fig. 10). Each leg ends in a 360° steerable actively driven wheel which allows omnidirectional driving. We choose a fixed leg configuration with 0.8 m longitudinal and lateral distance between the wheels. A 3D rotating Velodyne Puck laser scanner at the robot head with spherical field-of-view perceives the environment. Measurements are processed to registered point clouds while the robot is localized using the method of Droeschel et al.[20]. Finally, occupancy maps with a resolution of 0.2 m are generated from these point clouds. Those are used as input to our network. Robot perception and control is implemented in C++. Communication with the network is realized using ROS.

Our approach is evaluated on the test set with 32×32 grid worlds. We achieve a success rate of 70.15% while our paths are on average 2.06% longer than the optimal solution. The required graphics memory is 1093 MB.

We further employ our approach to plan paths for Centauro

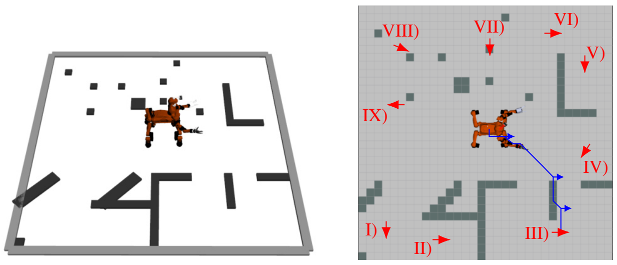


Fig. 11: 3D Locomotion planning experiment. Left: Gazebo arena with Centauro. Right: The corresponding occupancy map with the nine chosen goals and one example result path.

TABLE II: Results of our approach and the A^*-planner for the tasks depicted in Fig. 11.

	Abstract VIN		A^*-planner	
	Path length	Planning Time	Path length	Planning Time
I)	23.41	0.242 sec	23.41	0.153 sec
II)	Not found		24.14	0.775 sec
III)	18.49	0.211 sec	17.90	0.082 sec
IV)	18.80	0.225 sec	18.80	0.274 sec
V)	Not found		27.76	1.719 sec
VI)	15.90	0.187 sec	15.90	0.814 sec
VII)	15.27	0.192 sec	14.44	0.067 sec
VIII)	19.18	0.225 sec	16.25	0.053 sec
IX)	22.13	0.259 sec	22.13	0.665 sec

in a simulated environment with cluttered terrain (Fig. 11). Experiments are performed in the Gazebo simulation environment. Obstacle heights are chosen to be rather small to prevent the laser scanner from handling occlusions. A video with additional footage of the experiments is available online[2]. We place nine different goal poses in the map (Fig. 11) and compare our approach to the expert A^* planner.

The results in Tab. II indicate that our planner obtains optimal or close to optimal paths in most cases. Even challenging tasks which require the robot to take obstacles between its leg (e.g., III and VIII) can be planned. The planner seems to have problems with tasks which require turning actions in narrow sections such as II and V. Moreover, planning times are feasible and have a smaller distribution than for the A^* planner. In some cases the A^* planner is outperformed.

V. Conclusion

In this paper, we present a modification to Value Iteration Networks (VINs) to employ multiple level of abstractions. While the state resolution gets coarser, additional features compensated the information. Since training times and memory requirements limit the state space size, our approach facilitates VINs to solve for larger and more complex queries. Results indicate that we outperform VINs in terms of result quality and memory requirements. We further demonstrate how our approach plans omnidirectional locomotion for a robot in cluttered terrain while considering its footprint. In summary, we increase the applicability of learning-based motion planning approaches to real-world problems.

[2]https://www.ais.uni-bonn.de/videos/ICRA_2019_Schleich

REFERENCES

[1] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[2] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” 1998.

[3] L. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, 566580, 1996.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[5] M. Schwarz, A. Milan, A. S. Periyasamy, and S. Behnke, “RGB-D object detection and semantic segmentation for autonomous manipulation in clutter,” *The International Journal of Robotics Research (IJRR)*, vol. 37, no. 4-5, pp. 437–451, 2018.

[6] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” *The Journal of Machine Learning Research (JMLR)*, vol. 17, no. 1, pp. 1334–1373, 2016.

[7] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, “End to end learning for self-driving cars,” *ArXiv preprint arXiv:1604.07316*, 2016.

[8] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, “Value iteration networks,” in *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[9] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, “Universal planning networks,” *ArXiv preprint arXiv:1804.00645*, 2018.

[10] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, “Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation,” in *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[11] T. Klamt and S. Behnke, “Planning hybrid driving-stepping locomotion on multiple levels of abstraction,” in *IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2018.

[12] R. Bellman, *Dynamic programming*. Courier Corporation, 2013.

[13] R. Bellman, “A markovian decision process,” *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 4 1957, ISSN: 0022-2518.

[14] S. Niu, S. Chen, H. Guo, C. Targonski, M. C. Smith, and J. Kovacevic, “Generalized value iteration networks: Life beyond lattices,” *CoRR*, vol. abs/1706.02416, 2017. arXiv: `1706.02416`. [Online]. Available: `http://arxiv.org/abs/1706.02416`.

[15] P. Karkus, D. Hsu, and W. S. Lee, “Qmdp-net: Deep learning for planning under partial observability,” in *Advances in Neural Information Processing Systems*, 2017, pp. 4694–4704.

[16] T. Klamt and S. Behnke, “Anytime hybrid driving-stepping locomotion planning,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[17] T. Tieleman and G. Hinton, *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude*, Available at `https://www.coursera.org/lecture/neural-networks/rmsprop-divide-the-gradient-by-a-running-average-of-its-recent-magnitude-YQHki`, 2012.

[18] I. Loshchilov and F. Hutter, “SGDR: Stochastic gradient descent with warm restarts,” *ArXiv preprint arXiv:1608.03983*, 2016.

[19] 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,” Accepted for IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2018, `http://www.ais.uni-bonn.de/papers/IROS_2018_Klamt.pdf`.

[20] D. Droeschel, M. Schwarz, and S. Behnke, “Continuous mapping and localization for autonomous navigation in rough terrain using a 3D laser scanner,” *Robotics and Autonomous Systems*, vol. 88, pp. 104 –115, 2017.

Deep Reinforcement Learning to Acquire Navigation Skills for Wheel-Legged Robots in Complex Environments

Xi Chen, Ali Ghadirzadeh, John Folkesson, Mårten Björkman and Patric Jensfelt

***Abstract*— Mobile robot navigation in complex and dynamic environments is a challenging but important problem. Reinforcement learning approaches fail to solve these tasks efficiently due to reward sparsities, temporal complexities and high-dimensionality of sensorimotor spaces which are inherent in such problems. We present a novel approach to train action policies to acquire navigation skills for wheel-legged robots using deep reinforcement learning. The policy maps height-map image observations to motor commands to navigate to a target position while avoiding obstacles. We propose to acquire the multifaceted navigation skill by learning and exploiting a number of manageable navigation behaviors. We also introduce a domain randomization technique to improve the versatility of the training samples. We demonstrate experimentally a significant improvement in terms of data-efficiency, success rate, robustness against irrelevant sensory data, and also the quality of the maneuver skills.**

I. INTRODUCTION

Deep Reinforcement Learning (RL) has enabled training of highly flexible and versatile deep neural networks to obtain action-selection policies for complex problems. The complexity generally arises from (1) complicated dynamics and the way actions at different time-steps affect the long-term outcomes, (2) high-dimensionality of the action space which makes it impossible to enumerate actions using classical approaches, and (3) high-dimensionality and redundancies of the sensory observations. In such cases, deep RL holds the promise of finding solutions, sometimes, demonstrating superior performances compared to hand-crafted solutions or even compared to a human-expert himself performing the task [1], [2]. However, the state-of-the-art methods in deep RL are not generally applicable in other problem domains. The methods suffer from issues such as (1) reward sparsity, i.e., low-probable reward outcome while randomly exploring consequences of actions, (2) temporal credit assignment which refers to the problem of crediting action-decisions made over a period of time given a reward/punishment outcome, (3) data inefficiency, i.e., requiring huge amount of training samples to obtain a policy, and (4) difficulties of learning a task-relevant representation of input data which is critical to acquiring a generalizable action-selection policy.

In this paper, we propose an RL approach to solve a complex mobile robot navigation problem. We train a deep neural network policy to control a mobile robot with high-dimensional action spaces enabling complicated maneuvers, e.g, slimming the body to pass narrow corridors or lifting to cross over obstacles.The trained policy directly processes height-map images to generate appropriate action commands. The contribution of this paper is to introduce a method which:

1) improves policy training by splitting the complicated navigation task into a number of manageable navigation behaviors,
2) proposes a domain randomization technique, guiding policy training to attend important cues of the input observations. We experimentally demonstrate the suitability of the trained policy, e.g., to attend to the obstacles in front of the robot, but not the ones it has already passed.
3) demonstrates an improvement of 20% success rate compared to the state-of-the-art RL methods applied to a challenging mobile robot navigation problem.

The structure of the paper is as follows: In the next section, we outline related work (Sec. II), followed by introducing required background (Sec. III). We introduce our method in Sec. IV, and in Sec. V, we present our experimental results. The conclusions and the future work are presented in Sec. VI.

II. RELATED WORK

In this section, we introduce recent studies that are mostly related to our work. We first introduce a number of the state-of-the-art deep RL methods, followed by an overview of recent deep RL solutions for mobile robot navigation problems. Also, we provide a short overview of domain randomization approaches that are used recently to improve RL policy training.

A. Deep reinforcement learning

Deep RL approaches to train neural network robot policies can be categorized as, (1) end-to-end policy training, (2) concatenating separately trained neural network blocks, and (3) guided policy search. End-to-end policy training methods train deep neural network architectures directly with minimum task-dependent engineering efforts. These methods have been successfully applied to different complex tasks, such as playing Atari games [2], [3], and a number of 3D simulated physics tasks, [4], [5], [6]. However, these approaches are very data-inefficient and may not be directly applicable to real robotic problems. The second approaches, generally train perception and motor control layers of a deep policy network separately. These layers are mostly trained by learning a low-dimensional representation of the data using auto-encoder structures. These approaches have been applied to solve complex visuomotor tasks, e.g., [7], [8]. A major limitation of these approaches is that the structure of the network, data-representations, and training individual blocks require extra engineering efforts. Guided policy search [9] trains a deep policy network, end-to-end, efficiently by converting the policy search problem into supervised learning and trajectory optimization problems. However, the limitation of this approach is that it requires access to the true state of the system during the training phase to solve the trajectory optimization part.

Our proposed solution belongs to the first category of the approaches. We improve sample efficiency by splitting the problem into simpler sub-tasks and also by exploiting domain randomization techniques to enhance the versatility of the training samples.

B. Deep RL solutions to mobile robot navigation

Navigation learning for a mobile robot with high degrees of freedom, such as a wheel-legged robot, requires learning motor controls which stabilize robot motions without falling and also move the robot to a given target while avoiding collisions with obstacles. Previous studies addresses these problems separately. [4], [6], [10], [11] demonstrate stable locomotion skills on flat surfaces with no obstacles. [12] and [13] consider terrain-adaptive motions which enable the robot to cope with more diverse and challenging sets of terrains and obstacles. However, these methods do not take into account a target position. [14] addresses navigation to a target position by training a policy, end-to-end. However, the method is only validated on a robotic platform with low degrees of freedom. [15] and [16] proposed methods to combine the locomotion and motion planning with hierarchical structures with a two-step procedure: (1) training low-level controllers, (2) acquiring a high-level planner given the trained low-level controllers.

We focus on end-to-end training of a deep architecture to learn locomotion skills and also reaching to different target positions which requires long-term planning.Our method is validated on challenging configurations with random start and target positions.

C. Domain Randomization

Recently, domain randomization techniques are exploited to transfer action policies trained in simulation to the real world. These approaches randomize different aspects of the tasks, such as dynamics, [17], or visual sensory observations, [8], [18], [19], [20], We apply domain randomization to improve the versatility of the training samples collected from simple environments with few obstacles. The resulting dataset contains more complex configurations with multiple obstacles and challenging pathways.

III. PRELIMINARIES

In this section, we review related RL algorithms and introduce the notation which is used in the rest of the paper. We assume a Markov Decision Process (MDP) to represent our system, which is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, p(s_0), \gamma)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition probability, $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, $p(s_0) : \mathcal{S} \rightarrow [0, 1]$ is the initial state distribution, and $\gamma \in [0, 1]$ is the discount factor. The actions are sampled based on a parameterized stochastic policy $\pi_\theta : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, which assigns a probability distribution over the actions conditioned on the state value.

For each episode, an initial state is drawn from $p(s_0)$. At every time-step t, an action a_t is sampled from $\pi_\theta(a_t|s_t)$, a reward $r_t = r(s_t, a_t)$ is given by the environment and the next state s_{t+1} is given according to the transition probability $p(s_{t+1}|s_t, a_t)$. For every state-action pair in the trajectory, the return is defined as the sum of discounted rewards, $R_t = \sum_{t'=t} \gamma^{t'-t} r(s_{t'}, a_{t'})]$. The goal is to obtain a policy which maximizes the expected return, $\eta(\pi) = \mathbb{E}[R_t]$, with respect to all possible trajectories following the introduced sampling procedure.

Actor-critic approaches train a policy in three steps, (1) sampling a number of trajectories using the current policy π_θ, (2) estimating a value function representing $V_\pi(s_t) = \mathbb{E}[R_t]$, and (3) updating the policy parameters to increase the likelihood of trajectories with higher returns. A common approach, known as the policy gradient method, updates policy parameters to maximize $\eta(\pi) = \mathbb{E}[\log \pi_\theta(a_t|s_t) A_t]$, where A_t is an estimate of the advantage function, found as $A_t = R_t - V(s_t)$. Intuitively, the policy is updated such that state-action pairs with higher advantages become more probable.

Trust region policy optimization (TRPO) [4] introduced a similar surrogate function,

$$\eta_{\pi_\theta}^{TRPO} = \mathbb{E}[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t - \beta D_{KL}(\pi_{\theta_{old}} || \pi_\theta)],$$

where, D_{KL} represents Kullback Leibler (KL)-divergence. TRPO uses a policy probability ratio, $\psi_\theta = \pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)$, instead of $\log \pi_\theta$; furthermore it penalizes deviations from the old policy by an extra weight parameter β. In a more recent work, proximal policy optimization (PPO), [5] derived an updated version of the TRPO surrogate function by clipping the probability ratio ψ_θ using a function $\mathcal{Z}(.)$ which clips input values to the range $[1 - \epsilon, 1 + \epsilon]$, where ϵ is a hyper-parameter. This is equivalent to the TRPO's KL-divergence penalty term but forces the ratio of the current policy and old policy to be close to 1, instead of penalizing the difference of the policies. The surrogate function is found as the expected minimum value of the clipped and unclipped probability ratios multiplied by the advantage function, i.e.,

$$\eta_{\pi_\theta}^{PPO} = \mathbb{E}[\min(\, \mathcal{Z}(\psi_\theta) A_t \,,\, \psi_\theta A_t \,)]. \tag{1}$$

In this paper, we optimize the surrogate function defined in Eq. 1 to train action policies.

IV. METHOD

In this section, we introduce our method to train an action-selection policy for mobile robot navigation problems using deep reinforcement learning. The policy maps a height-map to a number of actions which move the robot to its target location. We propose to train several secondary policies, each to acquire a certain behavior, and then combine the result of them to train the primary policy. We argue that splitting such a complicated problem into a number of manageable tasks would help the RL agent to overcome difficulties with reward sparsity, credit assignment problem and data inefficiency. Furthermore, we introduce a domain randomization technique to efficiently learn to attend task-relevant aspects of the sensory observations without further interactive training using RL. In the rest of this section, we provide details of our approach regarding (1) training the secondary policies, (2) applying domain randomization to improve perception layers of the policy, (3) training the primary policy, and (4) structure of the network. A flow diagram of how to obtain the primary policy is shown in Fig. 1.

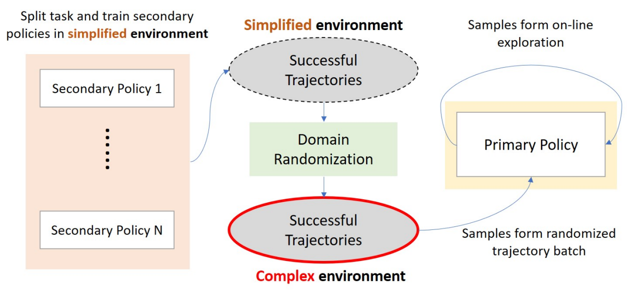


Fig. 1. The flow diagram of our method. First, we split the task and train a number of secondary policies in environments with simple setup to acquire different behaviors. Second, we save the successful trajectories learned by the secondary policies and apply domain randomization to create a batch which contains successful trajectories of the complex environments. Finally we train our primary policy in the complex environments using samples from both on-line exploration and domain randomized trajectory batch.

A. *Policy training to acquire different behaviors*

A behavior is defined as a maneuver strategy to move to a target position while avoiding collisions with an obstacle. We define the following behaviors for our mobile robot navigation problem:

1) moving straight to the target position with no obstacle along the path,
2) moving around an obstacle to reach a target,
3) driving over a high obstacle by lifting the body,
4) driving over a short but wide obstacle by lowering the body and stretching the legs out,
5) squeezing the body to pass through narrow corridors.

A simple setup w.r.t. a given behavior is defined as an environment with obstacles in which random action exploration results in higher task success rate realizing the specific behavior. For example, for the behavior (3), we make obstacles such that the robot has no other options than moving to the target position while lifting the body to avoid collisions. We combine the knowledge obtained by the secondary policies to train the general policy. Dividing a task into a number of behaviors resembles the way a human acquire a multifaceted skill, e.g., playing tennis. In this example, a trainee practices fore-hand and back-hand hits separately to improve each individual skill, and then combines them in a more realistic play condition.

Secondary policies are trained using the same method as the primary policy. We use an actor-critic framework with a network structure which shares parameters between the policy and the value function. Network parameters, denoted by θ, are found such that the following compound loss function is optimized:

$$\mathcal{L}(\theta) = \mathbb{E}[\lambda_1 \eta_{\pi_\theta}(\theta) + \lambda_2 \mathcal{L}_v(\theta)], \tag{2}$$

where, $\mathcal{L}_v$ is the value-function loss defined as $||V_\theta(s_t) - R_t||$, η_{π_θ} is as defined in Eq. 1, and the expectation is found over sampled trajectories as described in Sec. III,

The pseudo-code for training secondary policies is presented in Alg. 1. For every behavior, we initialize a secondary policy as well as a number of simple environments corresponding to that behavior. The first behavior which is learned is to move straight to the target with no obstacles. All other secondary policies are initialized with this trained policy. We train every policy for a number of iterations. A number of trajectories are sampled under the current policy and for every action-state pair in these trajectories, the return, advantage and policy probability ratio are found. Finally, the policy and the value-function are updated with Stochastic Gradient Ascent (SGD) w.r.t. the loss function defined in Eq. 2. The latest trajectories which are found based on the trained policies are stored for later use.

Algorithm 1 Training secondary policies

```
1:  for every behavior $b$ do
2:      initialize policy $\pi_\theta^b$.
3:      prepare a number of simple train environments.
4:      for every environment $E$ do
5:          for every iteration do
6:              sample trajectories given $\pi_\theta^b$ and $E$.
7:              for every $(s_t, a_t)$ pair in every traj. $i$ do
8:                  calculate $R_{t,i}$, $A_{t,i}$, $\psi^b_{\theta_{t,i}}$.
9:              $\theta^b_{old} \leftarrow \theta^b$.
10:             update $\theta$ by SGD, w.r.t. Eq. 2.
11:         store the latest trajectories.
```

B. *Domain Randomization*

We use domain randomization techniques to help the policy to extract task-relevant aspects of the input observations. The policy directly maps an image observation, i.e., the height-map representation of the scene, to the motor commands using a forward pass of the neural network. Given a limited number of training data gathered by actively interacting with the environment and the flexibility of the network, there is a high risk that the policy attends to task-irrelevant components of the observations, which limits its applicability to unseen test environments.

We randomize non-essential aspects of the task, such as the appearance, the positions and the number of obstacles in the scene to improve generalization capabilities of the primary policy. This randomization is applied to the simple environments where the secondary policies are trained without affecting the validity of the stored solutions. In other words, we randomize original environments used by the secondary policies by changing the obstacle configurations such that it would not affect the success of the sequence of motor actions found by the trained secondary policy in the corresponding original environment. In this way, we obtain solutions to very complex navigation problems without running the RL agent. The RL agent is data-inefficient, and is not guaranteed to find a solution.

Every environment used by the secondary policies is randomized to generate a number of new environments. For every new environment, the stored actions corresponding to the original environment are applied sequentially and the action-observation-return tuples are stored. In this case, we will have $n_e \times n_\tau \times n_{e_{rnd}}$ new action-observation trajectories, where n_e is the number of original environments, each correspond to n_τ trajectories, and $n_{e_{rnd}}$ number of randomized environments. These new trajectories are used to train the primary policy as explained in the next section.

C. Training the primary policy

The primary policy is trained with the same method and architecture as the secondary policies but with a different sampling strategy. Trajectories are sampled partially by following the primary policy given a number of new complex environments. The rest of trajectories are directly taken from the batch of domain randomized trajectories without running on the robot. At the beginning of the training phase, samples which are drawn from the primary policy mostly fail because of the low-probability of reward events when making random sequential action-decisions. This reward sparsity is compensated by the samples drawn from the domain randomized batch which only contains successful trials.

The pseudo-code for training primary policy is presented in Alg. 2. The primary policy is initialized either randomly or by any of the secondary policies. A number of training environments with complex obstacle configurations are generated. The agent is trained in every environment based on the samples drawn from the real interactions with the environments and also samples from the domain randomized batch. In the latter case, the advantages of the trajectories are re-calculated with the updated value function. At each iteration, the parameters of the policy and the value function are updated using stochastic gradient ascent given the compound training data.

D. Network Architecture

The network architecture, illustrated in Fig.2, maps the inputs, consisting of a height-map image observation and the robot and target poses, to a distribution over the motor actions. It also outputs the state value by the value-function sub-network. The network consists of three convolutional layers to extract features from the height-map image. The image features are concatenated with the robot configuration and the target position, which are further processed by the fully connected layers to output the action distribution and the state value.

Algorithm 2 Training the primary policy

```
1:  initialize primary policy $\pi_\theta$.
2:  prepare a number of complex environments.
3:  for every environment $E$ do
4:      for every iteration do
5:          sample trajectories given $\pi_\theta$ and $E$.
6:          for every $(s_t, a_t)$ pair in every traj. $i$ do
7:              calculate $R_{t,i}$, $A_{t,i}$, $\psi_{\theta_{t,i}}$.
8:          collect tuples $(s_t, a_t, R_t)$ from the domain randomized batch of traj.
9:          for every $(s_{t,i}, a_{t,i}, R_{t,i})$ tuples do
10:             calculate $A_{t,i}$ using $V_\theta(s_{t,i})$.
11:         concatenate training data.
12:         $\theta_{old} \leftarrow \theta$.
13:         update $\theta$ by SGD, w.r.t. Eq. 2.
```

V. EXPERIMENT

In the experiments, we want to address the following questions:

1) can we improve the training efficiency and the final performance using the batch of domain randomized trajectories?
2) does the primary policy learn to attend the task-relevant components of the input observation?

To answer these questions, we run two experiments on a navigation task with a reconfigurable wheel-legged robot in simulated environments. In the first experiment, we train a baseline model without the domain randomized trajectory batch and compare the result to our primary policy trained with the trajectory batch. We train the baseline policy and the primary policy with the state-of-the-art reinforcement learning algorithm PPO. In the second experiment, we compare the trajectory generated by different obstacle configurations to investigate if the primary policy learns to attend the task-relevant components of the observation.

A. Robot and Environment Setup

1) Robot Model: In the experiments we use a robot with four legs and we assume the robot performs the same action symmetrically to all legs. The robot can move and rotate on the xy plane, the body height and the leg openings are controlled by the three joints on each leg (Fig. 3). The action space of the robot is 5 dimensional.

We use a discretized value to control each action dimension. For every step, the robot can move $+/-0.05m$ along x and y axis, rotate $+/-5 degrees$ around its center axis, can change the body height for $+/-0.02m$, and can stretch legs for $+/-0.02m$.

2) Environment Configuration: The environment is simulated by V-REP ([21]). In each episode, the initial robot orientation and the target location are randomly generated. We use three types of obstacles (Fig. 4), each of them requires different locomotion skills to either drive around or drive over it. The number, shape, and position of the obstacles in the scene are randomly assigned. The input

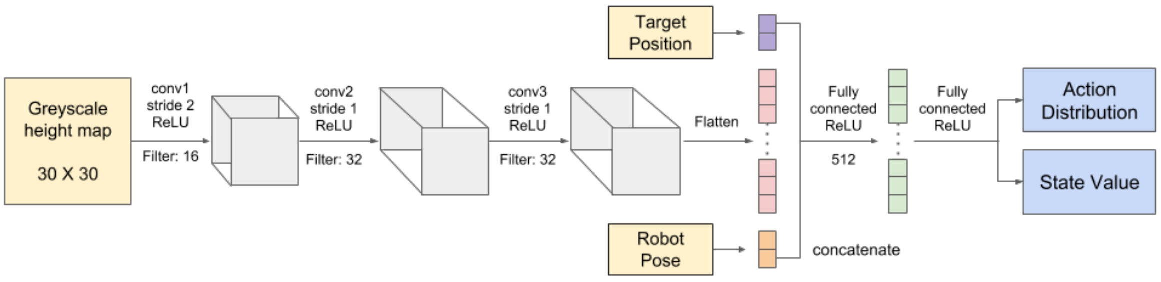


Fig. 2. The network architecture used in this paper.

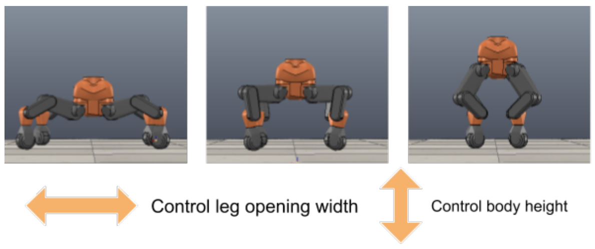


Fig. 3. The robot height and the width of leg opening are controlled through separate channels.

image of the height map has the size of $1.6 \times 1.6m$ centered at the robot with a resolution of $0.05m/pixel$.

The task is successful when the robot reaches the target position. The episode is terminated when the robot drives too far from the target or performs a maximum number of steps.

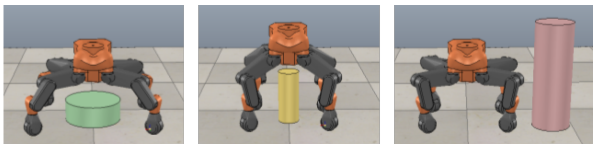


Fig. 4. The three types of obstacle used in the experiments. The green obstacle requires the robot to lower the body and open the legs, the yellow obstacle requires the robot to lift the body, and the robot cannot drive over the red obstacle.

3) Reward: The reward function consists of three components: (1) a fixed time cost for each step $r_{cost} = -0.1$, (2) a progress reward $r_{progress} = d_{t-1} - d_t$, where d_t denotes the distance from the robot to the target at time step t, and (3) a fixed penalty $r_{invalid} = -0.5$ when the robot collides to an obstacle, performs an impossible action or moves too far from the target. The reward for each step is defined as:

$$r = r_{cost} + r_{progress} + r_{invalid} \tag{3}$$

The reward function is designed to encourage the robot to move close to the target as soon as possible and reduce the collision to the obstacles. We do not penalize motor cost or specific motion types, i.e., move backward or sideways.

4) Secondary Policies: Fig. 5 plots the learning curves for all 5 secondary policies. Since we simplified the environment when training the secondary policies, all 5 policies can be learned within a small number of samples.

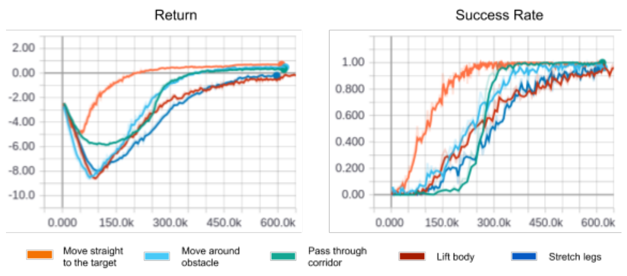


Fig. 5. The learning curves of the 5 secondary policies w.r.t. the average return and success rate. The horizontal axis represents the number of trajectory generated.

5) Domain Randomization: We collect 10^3 trajectories for every secondary policy. For each trajectory, we first mask the areas which are affected by the actions and randomize different obstacle configurations outside the masked areas. Then, we replay the sequence of actions in the trajectory and store the new action-observation-return tuples to the trajectory batch. For each trajectory, we randomize 10^3 new environments. Fig. 6 gives an example of randomizing a new environment from a straight line trajectory.

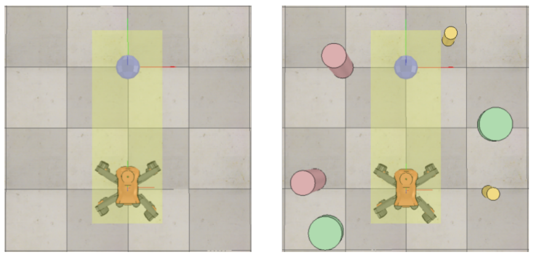


Fig. 6. The process of randomize a new environment from a straight line trajectory. In the left image, the blue point shows the target position and the yellow square masks the essential area which is affected by a straight line trajectory. In the right image, the obstacles are randomly generated in the non-essential area which does not affect the validity of the straight line solution

B. Evaluation

We train the baseline policy with the batch size of $10,000$, the samples in the batch are from on-line exploration. We train our primary policy using the batch size of $12,000$, where $10,000$ samples are from on-line exploration and $2,000$ samples are from th domain randomized trajectory batch.

1) Result of Primary Policy: Fig. 7 plots the learning curves of the primary policy w.r.t. the average return and the success rate. Using samples from the domain randomized trajectories, the overall success rate increases by nearly 20%. While training, we observe a considerable difference in the performance of the case where the robot needs to drive over obstacles. Similar to the problem discussed in [22], the penalty we assign to the collision may hinder exploration, i.e., the robot learns not to move close to the obstacles. This problem is compensated by training with the samples from the domain randomized trajectories which contains successful actions for similar configurations.

The goal of the task is to reach to the target position. However, as a navigation task, the quality of the trajectory is important as well. In Fig. 8, we compare the quality of the trajectory w.r.t. the success rate of generating a collision-free trajectory and the length of the trajectory in the number of steps. For generating a collision-free trajectory, the success rate of our primary policy remains at 80%, while the baseline policy drops to 55%. Moreover, the average length of the collision-free trajectory of our primary policy is also shorter. Using samples from the domain randomized trajectories improves not only the success rate but also the quality of the output motions.

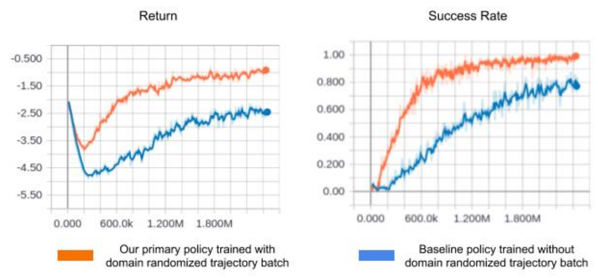


Fig. 7. The average return and the success rate of the two policies. The horizontal axis represents the number of trajectory used in training.

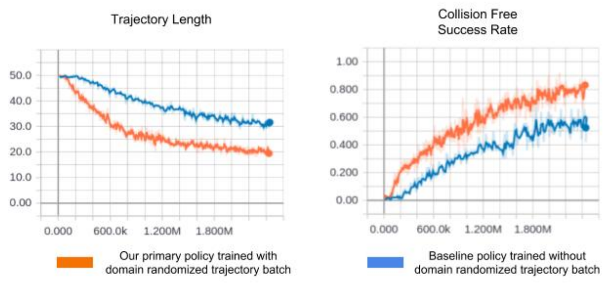


Fig. 8. The trajectory length in number of steps and the successful rate for generating a collision free trajectory. The horizontal axis represents the number of trajectory used in training.

2) Attend task-relevant components of the observation: We compare the trajectories generated using different obstacle configurations to verify if the primary policy learns to attend the task-relevant components of the input observation. We first sample a configuration and generate a baseline trajectory. We then remove one obstacle from the scene and generate another trajectory with the missing obstacle. We repeat the same step for all obstacles in the scene and compare the results to find out how much the missing obstacle affects the policy, or how relevant this missing obstacle is to the task.

Fig. 9 gives an example of the relevance of each obstacle to the task. The obstacles colored by white has low relevance, which means the trajectory does not change after removing them. The obstacles colored by red has higher relevance, which can affect or completely change the choice of trajectories. From Fig. 9 we note that the obstacles which close to the current path or block a better path give more impact to the policy. Our primary policy is able to attend the task-relevant components from the observation.

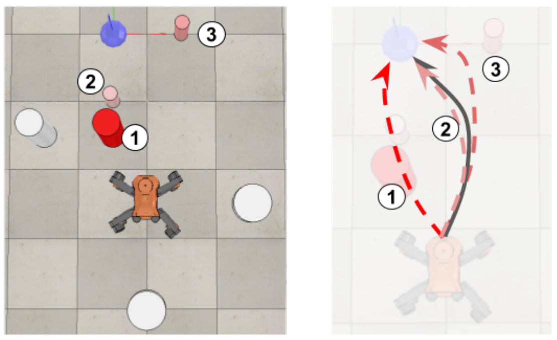


Fig. 9. An example of the relevance of each obstacle to the task. In the left image, the relevance of obstacle is color-coded from white to red, where white denotes low relevance and red denotes high relevance. The number in the image denotes the index of the obstacles and the blue point denotes the target position. In the right image, the gray arrow denotes the trajectory generated using all obstacles in the scene and red arrows are trajectories generated with one missing obstacle. The numbers near the red trajectories indicates which obstacle is removed from the scene.

VI. CONCLUSIONS

In this work, we present a novel approach to acquire navigation skills for the wheel-legged robot by learning and combining a number of manageable secondary policies. Using the trajectory batch created by secondary policies and domain randomization technique, our approach overcomes the difficulties caused by data inefficiency, reward sparsity, temporal credit assignment problem, and improves the final performance on both success rate and motion quality.

In the future, we plan to continue our work with more diverse environment settings, such as introducing obstacles with irregular shape and uneven terrains, which requires more challenging locomotion skills. Also, we intend to investigate the possibility of applying our work to a real-world robotic system.

VII. ACKNOWLEDGMENTS

This work is supported by the European Unions Horizon 2020 research and innovation program under grant agreement No. 644839 (CENTAURO).

REFERENCES

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[3] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[4] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[7] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 512–519.

[8] A. Ghadirzadeh, A. Maki, D. Kragic, and M. Björkman, "Deep predictive policy training using reinforcement learning," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 2351–2358.

[9] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[10] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[11] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," in *Advances in Neural Information Processing Systems*, 2015, pp. 2944–2952.

[12] X. B. Peng, G. Berseth, and M. Van de Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, p. 81, 2016.

[13] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017.

[14] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 31–36.

[15] N. Heess, G. Wayne, Y. Tassa, T. Lillicrap, M. Riedmiller, and D. Silver, "Learning and transfer of modulated locomotor controllers," *arXiv preprint arXiv:1610.05182*, 2016.

[16] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 41, 2017.

[17] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," *arXiv preprint arXiv:1710.06537*, 2017.

[18] J. Tobin, W. Zaremba, and P. Abbeel, "Domain randomization and generative models for robotic grasping," *arXiv preprint arXiv:1710.06425*, 2017.

[19] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 23–30.

[20] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," *arXiv preprint arXiv:1709.07857*, 2017.

[21] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 1321–1326.

[22] M. Andrychowicz, D. Crow, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5055–5065.