



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

CENTAURO

Deliverable D5.3 CENTAURO Driving 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:	KTH – Kungliga Tekniska Hoegskolan
Authors:	T. Klamt, P. Jensfelt, X. Chen, K. Nordberg, D. Droeschel
Work package:	WP5 Navigation
Date of preparation:	2017-12-15
Type:	Report
Version number:	1.1

Document History

Version	Date	Author	Description
0.1	2017-08-12	TK	First draft
0.2	2017-09-22	PJ	Extended introduction, added executive summary and additional work section
0.3	2017-09-23	TK	Added missing parts
0.4	2017-09-24	PJ	Added GCN section (A.2)
0.5	2017-09-26	KN	Added A.1 and A.3
0.6	2017-09-28	PJ & XC	Added future work section
0.7	2017-10-14	PJ	Fixed typos. Moved future work section into an additional work section. Changed from Attachment X to Additional work X to also be able to include work where there is no paper attached.
0.8	2017-10-31	PJ	New version of future work.
0.9	2017-11-01	XC	Additional work 4 added with description of DRL planner approach.
0.9.1	2017-12-14	TK	Add subsection 5.3 to describe experiments with the real robot
0.9.2	2017-12-15	TK	Add subsection 5.2 to describe terrain classification with the real robot
1.0			Submitted version.
1.1		DD	Add missing section 5.1

Executive Summary

This deliverable describes our approach to locomotion planning and navigation, one of the core components in the CENTAURO system. The Centauro robot stands out compared to most other mobile robots in that it can both drive and walk. We have developed a framework that exploits both of these modes. This deliverable describes this system in detail. We also present some ongoing and future work. We include additional work performed in Workpackage 5 in appendices.

Contents

1	Introduction	5
2	Related Work	6
3	Method	8
4	Evaluation	22
5	WP5 on the CENTAURO Robot	28
6	Future Work	31
A	Additional Work in Workpackage 5	32

1 Introduction

This deliverable, "D5.3 CENTAURO Driving Navigation", reports on the design of the navigation planning system used in the CENTAURO project, an unmanned robot which supports two locomotion modalities: driving and walking. Since our locomotion planning approach combines driving and stepping planning in a single approach, we present planning for both locomotion types in this deliverable, even though the title suggest that it would be only about driving. The main part of this deliverable is an extended version of [17].

The rest of the deliverable is structured as follows. Section 2 provides related work locomotion planning. Section 3 describes our method which relies on a multi-level spatial representation. Section 4 presents an evaluation of the suggested method. Finally, Section 6 presents ongoing and future work on navigation. To show the overall progress in Workpackage 5, there are a number of appendices with additional work connected to the workpackage, but not directly related to the specifics of the Driving Navigation.

An overview of the components and functionality in the CENTAURO project related to navigation can be seen in Figure 1. The planning component is at the center of this. It needs input in the form of a 2D height map, terrain class map, the goal pose and the robot state. The 2D height map is calculated from a 3D point-cloud map generated from the 3D rotating laser scanner. The 2D height map and terrain class map are used to calculate the costs for the planner. The 2D height map and terrain class map are used to calculate the costs for the planner.

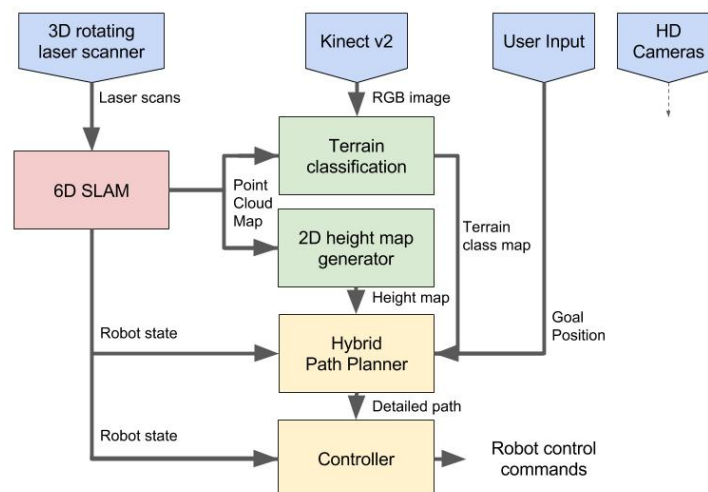


Figure 1: An overview of the components and functionality in WP5.

2 Related Work

Many works address path planning in unstructured terrain. The considered systems provide either purely wheeled/tracked locomotion or are able to traverse terrain by walking. Planning is often done with either grid-based searches, such as A* [11], or sampling-based approaches, such as RRT [21]. Despite the application of similar planning methods, these two locomotion modes differ in many aspects. Since planning on large maps and with many degrees of freedom (DoF) slows down path generation, there are different approaches facing this problem.

2.1 Driving Locomotion Planning

Driving is fast and energy efficient on sufficiently flat terrain, which makes it suitable for traversing longer distances. When supported by three or more wheels, the robot is generally statically stable. Planning of drivable paths in unstructured environments is heavily dependent on the DoF of the platform. Simple robot designs offer longitudinal and rotational movements with a constant robot shape [6], [14]. For search and rescue scenarios, some robots were extended by tracked flippers [5], [25], [3]. These allow the robots to climb stairs and thus increase capabilities but also planning complexity due to additional shape shifting DoFs. Flipper positions are often not considered by the initial navigation path planning and are adjusted to the terrain in a second planning step. Platforms which offer omnidirectional locomotion increase the path planning search space by another dimension [49]. Driving is restricted, however, by terrain characteristics such as height differences and slopes which makes it unsuitable for very rough terrain and for overcoming obstacles.

2.2 Walking Locomotion Planning

Legged locomotion is capable of traversing more difficult terrain since it only requires isolated feasible footholds. The drawback of this locomotion mode is, that motion planning is much more complex. Since legs are lifted from the ground repeatedly, the robot also has to constantly ensure that it remains stable. Due to the high motion complexity of stepping, path planning is often performed in at least two hierarchical levels [16], [42], [30]. A coarse planning algorithm identifies feasible footholds or areas for feasible steps. Detailed motion planning is done in a second step to connect these footholds. Navigation towards the goal is either included in the coarse planning or realized in a higher-level planner.

2.3 Increasing Planning Performance

Planning performance is strongly dependent on the number of DoF and the path length. There exist different approaches to increase planning performance by introducing suboptimality.

One group of approaches accelerates planning by extending the search algorithm itself. A* is often extended with anytime characteristics such as done in Anytime Repairing A* (ARA*) [23]. Search is accelerated by giving the heuristic a weight > 1 . An initial search provides solutions with bounded suboptimality quickly. The result quality is then improved by decreasing the heuristic weight stepwise down to 1, if the given planning time is not exhausted yet. ARA* recycles the search space representations, it generated previously, to accelerate re-planning.

Another group of approaches applies multiresolution characteristics to the planning method. A high resolution environment representation and high resolution planning is only applied in

regions of special interest. Behnke [1] proposed a general concept for A*-based multiresolution planning with a decreasing resolution with increasing distance to the robot. González-Sieira et al. [8] apply high resolution in areas of high environment complexity. Resolution decreases with increasing distance to these areas. Bohlin [2] 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. [18] 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. [7] 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. Similar, Zhang et al. [46] plan in 2D and switch to high-dimensional planning in the robot vicinity and at key points. Similar to multiresolution planning, planning with multiple robot configuration dimensionalities might lead to wrong or bad plans, since a low-dimensional robot representation might estimate 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. [31] 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 count for this work.

3 Method

We present a search based planning approach which combines driving and stepping in a single planner. Steps are generated in a hierarchical manor. The planner utilizes three levels of representation which vary in the planning resolution and dimensionality. Coarser planning comes along with a loss of information which we compensate by adding semantics to the planning scene. Furthermore, we compare different heuristics and improve planning performance by introducing an orientation cost term and applying anytime characteristics.

3.1 Approach

The Centauro robot design does not only allow driving and walking locomotion separately but enables new motions which are neither possible with pure driving nor with pure walking robots. One of these motions is the ability to change the configuration of ground contact points (which we will refer to as the robot footprint) under load without lifting a foot, as Figure 2 demonstrates. This enables motion sequences for stepping with large stability margins. Since we want to benefit from those unique capabilities, we combine driving and stepping locomotion planning in a single planner instead of planning both locomotion types separately.

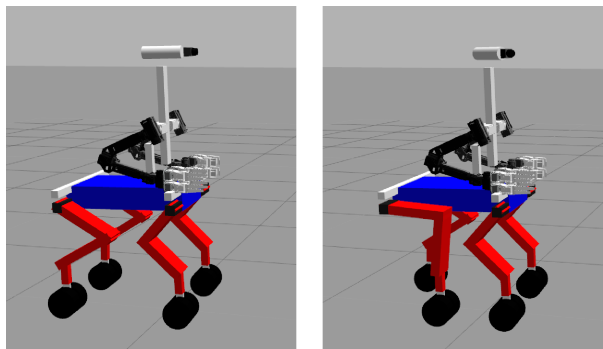


Figure 2: The unique Centauro robot design allows motions which are neither possible with pure driving nor with pure walking robots. One of these motions is to change the robot footprint without lifting a foot, as demonstrated with Momaro.

Hybrid driving-stepping locomotion planning is realized through an A* search. The planner chooses driving mode whenever possible and takes into account the detailed robot footprint. If steps are required, the planner includes those. The required map resolution and planning resolution is high to place steps reliably and manoeuvre through cluttered terrain safely. Moreover, our platform provides omnidirectional driving and individual leg movement which results in a high number of DoF which also have to be considered in the planner. Both, a high resolution and a high number of DoF, have a huge impact on the planning performance. We face this problem with multiple ideas.

To reduce the number of DoF during path search, we split step planning into two hierarchical levels. During planning we consider abstract steps. We define abstract steps to be the direct transition from a pre-stepping pose to an after-stepping pose. An abstract step does not describe the detailed motion sequence to perform such a step and needs to be expanded to be stable and executable by the robot. Only those abstract steps which are part of the result path are expanded to detailed motion sequences. Moreover, we neglect the height of individual feet. This is also computed only for those poses that are part of the result path. An overview over the planner architecture is given in Figure 3.

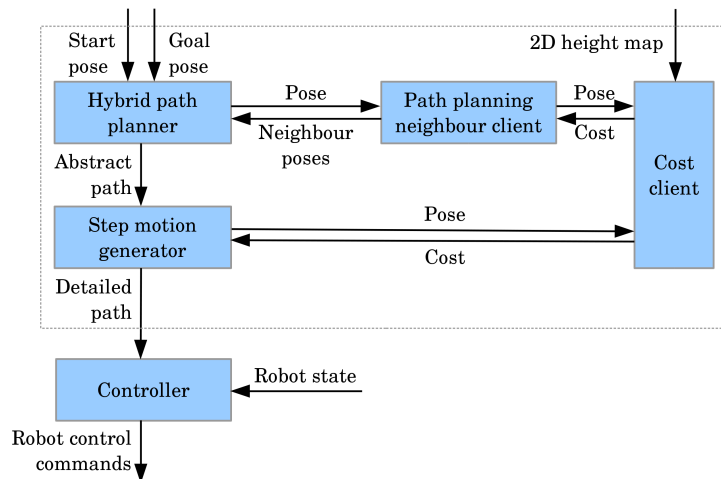


Figure 3: Hybrid locomotion system structure: The hybrid path planner searches an abstract path from start to goal pose. The step motion generator expands this abstract path to a detailed path which can be executed by the robot. A neighbour client provides neighbour states to the planner. Both, this neighbour client and the step motion generator, request pose costs from the cost client which generates costs out of the 2D height map. The resulting path is executed by a controller.

As described before, the required resolution and number of planning dimensions are high to provide save and feasible path. However, this only applies for regions close to the robot position. For regions with larger distance to the robot, planning can be done on coarser resolutions with less dimensions. This is reasonable, since those parts of the plan are reached in the further future and thus are more uncertain. Those plan segments can become more detailed when the robot gets there. Moreover, sensor measurements become less precise with increasing distance to the robot.

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 to 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. 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. Higher levels of representation can be derived from lower levels of representation. The approach is visualized in Figure 4. Level sizes and positions are shown in Figure 5. 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 levels of representation are described in detail in the next subsections.

Basis to the planning is a suitable environment model. To consider the robot footprint, we distinguish between foot costs and body costs and combine those to pose costs as described in CENTAURO Deliverable 5.2. The generation of these costs varies between the different levels of representation and is described for each level individually. The same applies to the robot representation and the set of feasible robot actions at each position.


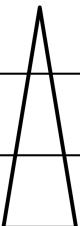
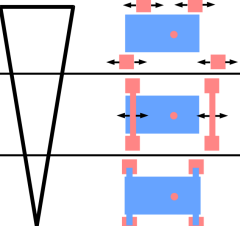



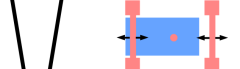





Level	Map Resolution	Map Features	Robot Representation	Action Semantics
1	 <ul style="list-style-type: none"> • 2.5 cm • 64 orient. 	 <ul style="list-style-type: none"> • Height 		 <ul style="list-style-type: none"> • Individual foot actions
2	 <ul style="list-style-type: none"> • 5 cm • 32 orient. 	 <ul style="list-style-type: none"> • Height • Height difference 		 <ul style="list-style-type: none"> • Foot pair actions
3	 <ul style="list-style-type: none"> • 10 cm • 16 orient. 	 <ul style="list-style-type: none"> • Height • Height difference • Terrain class 		 <ul style="list-style-type: none"> • Whole robot actions

Figure 4: 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 state increase.

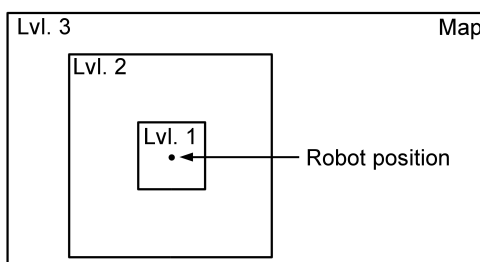


Figure 5: Size and position of the different levels of representation. Level 1 covers the vicinity of the robot. Level 2 is also robot centered and medium sized. Level 3 covers the whole map.

3.2 Level 1 Representation

Representation Level 1 is based on the approach which we presented in our previous work [17]. 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 foot costs. Base costs are derived from the height map itself. The detailed generation of these costs is described in CENTAURO Deliverable D5.2. A height map and the derived foot costs can be seen in Figure 6.

In this level of representation, a robot pose $\vec{r} = (r_b, f_1, \dots, f_4)$ is represented through the robot base configuration $\vec{r}_b = (r_x, r_y, r_\theta)$ with its center position r_x, r_y and orientation r_θ and the individual longitudinal foot positions f_1, \dots, f_4 . At each position, the robot can have 64 different discrete orientations.

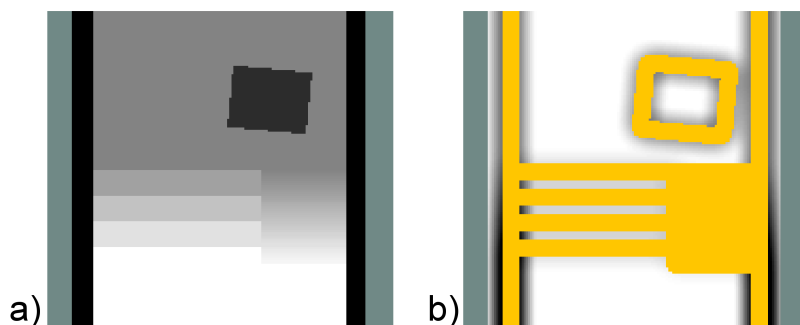


Figure 6: Overview over representation level 1. a) Input to the planner is a height map, showing a corridor with walls at the left and right, a staircase with four steps and two cuboid obstacles. Olive areas are unknown. b) Foot cost map. Yellow areas are not traversable by driving.

Path planning is realized through an A*-search on a pose grid. Feasible driving neighbour poses can be found within a 16-position-neighbourhood and by turning on the spot to the next

discrete orientation, as shown in Figure 7.

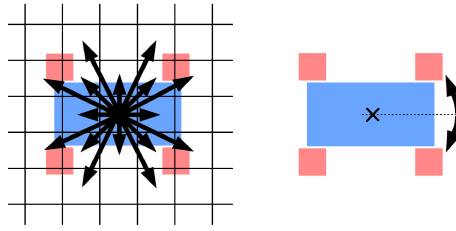


Figure 7: Driving locomotion neighbour states can either be found by straight moves with fixed orientation within a 16-position-neighbourhood (l.) or by orientation changes on a fixed position (r.). Grid and orientation resolution are enlarged for better visualization.

As illustrated in Figure 8, additional stepping manoeuvres are added, if a foot \vec{f}_j is

- close to an obstacle ($\exists c \in \text{map} (C_F(c) = \infty \wedge \|c - \vec{f}_j\| < 0.1 \text{ m})$),
- a feasible foothold c_h with $C_F(c_h) < \infty$ can be found in front of the foot in its sagittal plane that respects a maximum leg length,
- the height difference to the foothold is small ($|H(\vec{f}_j) - H(c_h)| \leq 0.3 \text{ m}$), and
- the distance between the two feet on the “non-stepping” robot side is $> 0.5 \text{ m}$ to guarantee a safe stand while stepping.

A step is represented as an additional possible neighbouring pose for the planner.

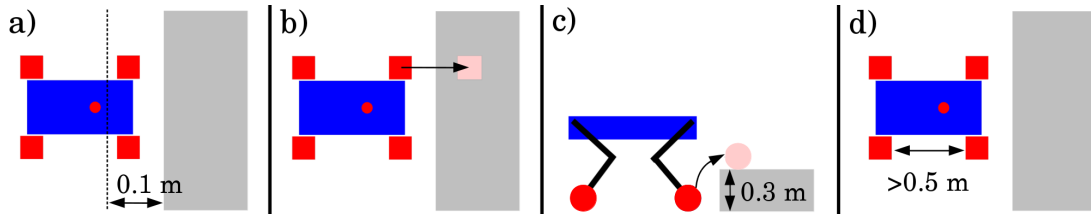


Figure 8: Criteria to add steps to the hybrid planner: a) A foot is close to an obstacle, b) a feasible foothold can be found, c) the height difference to overcome is $\leq 0.3 \text{ m}$ and d) the distance between the feet on the “non-stepping” robot side is $> 0.5 \text{ m}$. Grey areas show an elevated platform.

The step which is considered by the planner at this level is an abstract step which is only expanded to a detailed motion sequence if it is part of the result path. An abstract step is visualized in Figure 9a.

Each step is assigned a cost value

$$C_S = k_7 \cdot L_{\text{step}} + k_8 \cdot (C_F(c_h) - 1) + k_9 \cdot \Delta H_{\text{step}}, \quad (1)$$

where $k_7 = 0.5$, $k_8 = 0.1$ and $k_9 = 2.3$ which includes the step length L_{step} , the foot specific terrain difficulty costs of the foothold to step in c_h , and the maximum height difference ΔH_{step} . If multiple end positions for a step exist, only the cheapest solution is returned to the search algorithm.

Further manoeuvres are required to navigate in cluttered environments. We define the footprint shown in Figure 8a to be the neutral footprint. It provides high robot stability at a small footprint size, and is the preferred configuration for driving.

If both front feet are positioned in front of their neutral position, the robot may perform a longitudinal base shift manoeuvre. The base is shifted forward relative to the feet until one of the front feet reaches its neutral position or a maximum leg length is reached for one of the rear legs (see Figure 9b). Base shifts of length L_{BS} using the average discovered base costs $C_{B,avg}$ and $k_{10} = 0.5$ carry the cost

$$C_{BS} = k_{10} \cdot L_{BS} \cdot C_{B,avg}. \quad (2)$$

If a rear foot is close to an obstacle, the pair of wheels at each front foot may be driven forward while keeping ground contact (Figure 9c) which is a preparation for a rear foot step. If the robot footprint is not neutral, it may drive a single pair of wheels to their neutral position (Figure 9d). A single foot movement of length L_{FM} using the average discovered foot costs $C_{F,avg}$ and $k_{11} = 0.125$ carries the cost

$$C_{FM} = k_{11} \cdot L_{FM} \cdot C_{F,avg}. \quad (3)$$

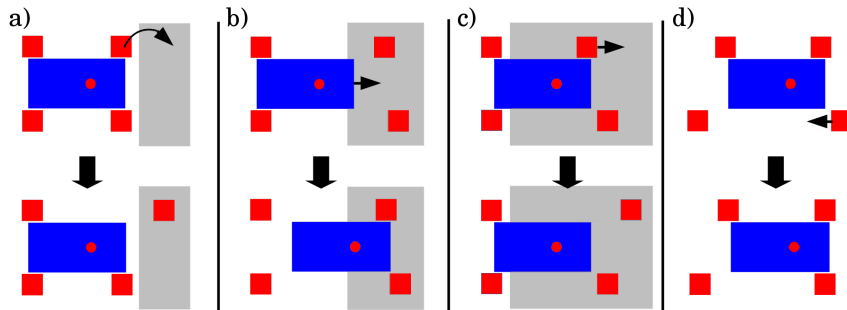


Figure 9: Manoeuvres which extend the driving planner to a hybrid locomotion planner, visualized on a height map: a) Abstract steps, b) longitudinal base shifts, c) driving a pair of wheels at one front foot forward, and d) driving a pair of wheels at any foot back to its neutral position.

Since driving is faster and safer than stepping, we want the planner to consider drivable detours of acceptable length instead of including steps in the plan. We define that, when the robot stands in front of an 0.2 m elevated platform, it should prefer a 1.5 m long detour over a ramp instead of stepping up to this platform (Figure 10). This can be achieved by increasing the costs of stepping-related manoeuvres by a certain factor.

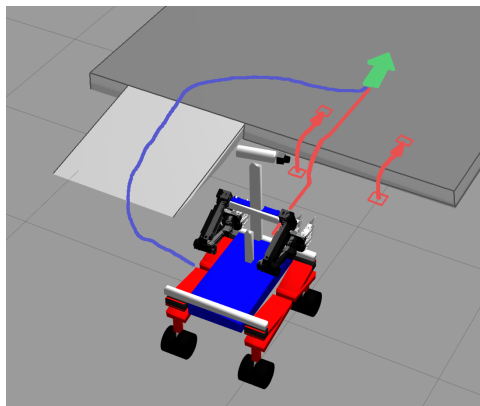


Figure 10: Step-related manoeuvre costs are weighted such that the planner just prefers taking a 1.5 m detour over a ramp (blue path) instead of stepping up (red path) to an elevated platform.

3.3 Level 2 Representation

We use the 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. 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 Figure 11. Each Level 2 height value is computed from the normalized, weighted sum of a 4×4 -region of Level 1 height values (see Figure 11 a). We use a binomial distribution for weighing (see Figure 11 b). A Level 2 map of height differences 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. The Level 1 height difference map is then subsampled to a Level 2 height difference map. A Level 2 height map and height difference map are shown in Figure 12.

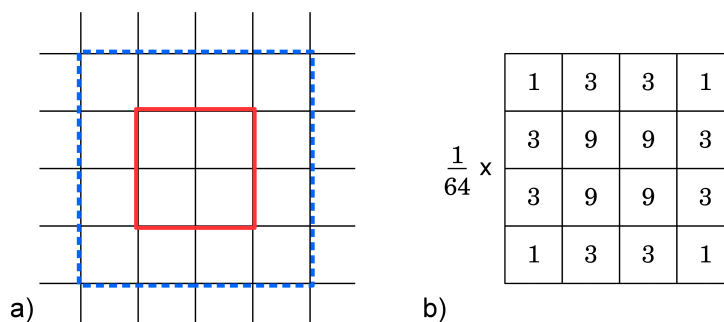


Figure 11: Subsampling method: a) For a Level 2 cell (red square) a 4×4 -window (blue square) of Level 1 cells is considered. b) Normalized binomial distribution to weigh heights and height differences.

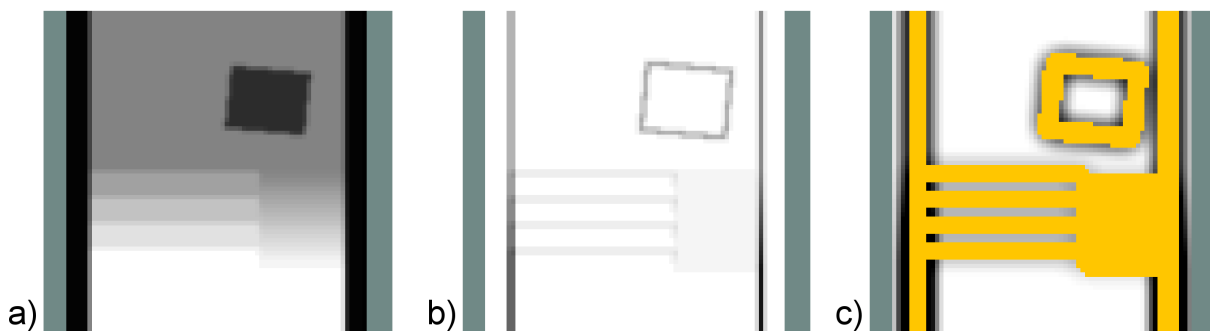


Figure 12: Level 2: a) height map, b) height difference map, c) foot cost map.

We use the generated Level 2 height and height difference input to compute foot and body costs as described for representation Level 1. Body cost computation is similar since it only relies on height information. Foot costs rely on height differences. Since we generate these by accumulating height differences of a larger neighbourhood, Level 2 height difference values are in general lower but spread over a larger area than Level 1 height differences. Consequently we adapt the parametrization to achieve feasible foot costs. A Level 2 height map, height difference map and foot cost map can be seen in Figure 12.

To decrease the robot configuration space dimensionality, we accumulate individual feet to foot pairs. A Level 2 robot pose $\vec{r} = (\vec{r}_b, f_f, f_r)$ is consequently represented by its robot base pose \vec{r}_b and its relative longitudinal foot pair coordinates for the front foot pair f_f and the rear foot pair 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 pair.

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 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 move a foot pair. For this action, the planner needs to distinguish between shifting both feet of this pair, stepping with one foot and shifting the other foot or stepping with both feet, as visualized in Figure 13.

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 moves its front foot pair forward as visualized in Figure 13 a2, the costs for this manoeuvre are the sum of the costs for shifting the front left foot forward and step with the front right foot. Since the map resolution in Level 2 is coarser and height difference values in Level 2 differ from those in Level 1, we reparametrized the cost computation. We do this by performing foot pair manoeuvres in a variety of different scenarios, perform the same manoeuvre in Level 1 and manually change the Level 2 cost parameters until the costs for those manoeuvres in both levels vary by $\leq 5\%$.

In addition to foot pair manoeuvres, the robot is able to shift its base forward (Figure 13 b). If a foot pair is not in its neutral position, it may be shifted towards its neutral configuration (Figure 13 c).

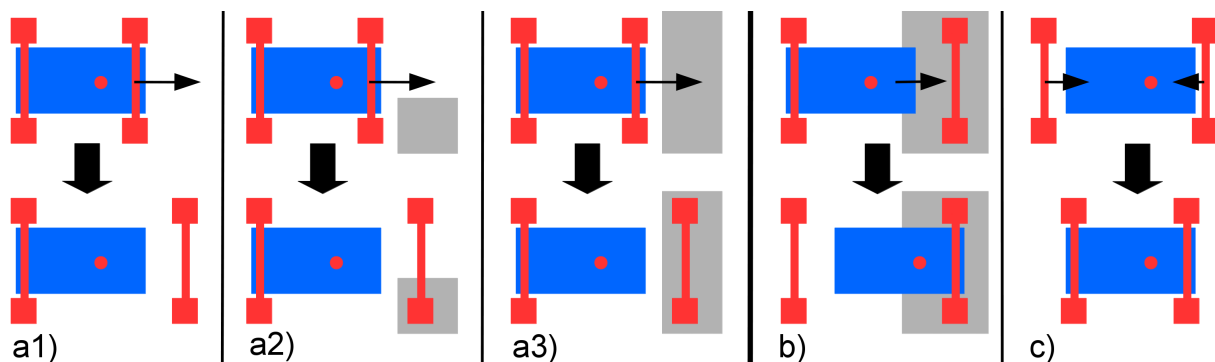


Figure 13: Level 2 stepping-related manoeuvres: When moving a foot pair, it has to be distinguished between shifting both feet (a1), shifting one foot and step with the other (a2) or step with both feet (a3). Moreover, the robot may perform a longitudinal base shift (b) or move each foot pair towards its neutral position (c).

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}_i and \vec{r}_{i+1} , we transform both poses into Level 1 and generate a set S of feasible robot base poses by interpolating between \vec{r}_i and \vec{r}_{i+1} . S is then inflated with a radius of two position steps and one orientation step as visualized in Figure 14. A local planner, which is restricted to S , searches for a Level 1 path between \vec{r}_i and \vec{r}_{i+1} . If

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

we call this path segment "not refineable".

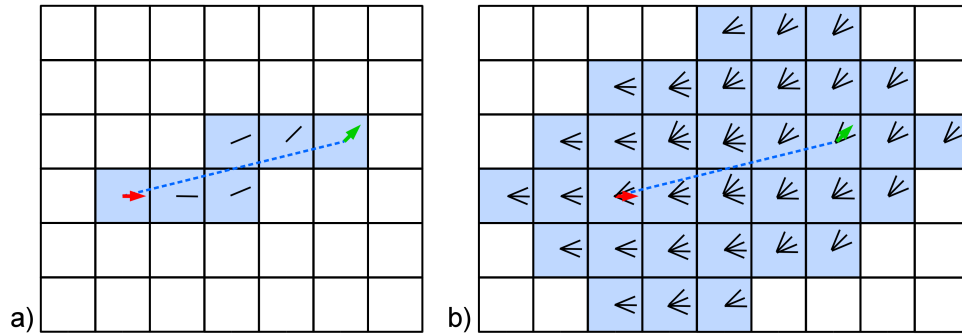


Figure 14: Generating of a set of feasible robot base poses for path refinement: a) For a given start (\vec{r}_i , red arrow) and goal (\vec{r}_{i+1} , green arrow) robot base pose, we generate a set of feasible robot base poses (black lines) by interpolating between the two. b) Inflation by two position steps and one orientation step.

3.4 Level 3 Representation

We apply the subsampling process which was described for Level 2 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 surface: The ground is even and easily traversable by driving,
- Rough surface: The ground is uneven but still traversable by driving,
- Stepping surface: The surface shows 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: The cell cannot be classified.

First, we search for cells of the terrain type "Stepping surface". This is done by searching for cell pairs c_i and c_j that fulfill the following criteria:

- $C_F(c_i) < \infty$: c_i is on a drivable surface,
- $C_F(c_j) < \infty$: c_j is on a drivable surface,
- $\|c_i - c_j\| < 0.45$ 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 , $C_F(c_k) = \infty$ 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, c_i , c_j and T are assigned the terrain class "Stepping surface". Second, we classify the remaining cells by their Level 2 height difference value ΔH :

- "Flat surface" if $\Delta H(c_i) \in [0, 2 * 10^{-4}]$,
- "Rough surface" if $\Delta H(c_i) \in [2 * 10^{-4}, 0.05]$,
- "Wall" if $\Delta H(c_i) \in [0.05, \infty]$,

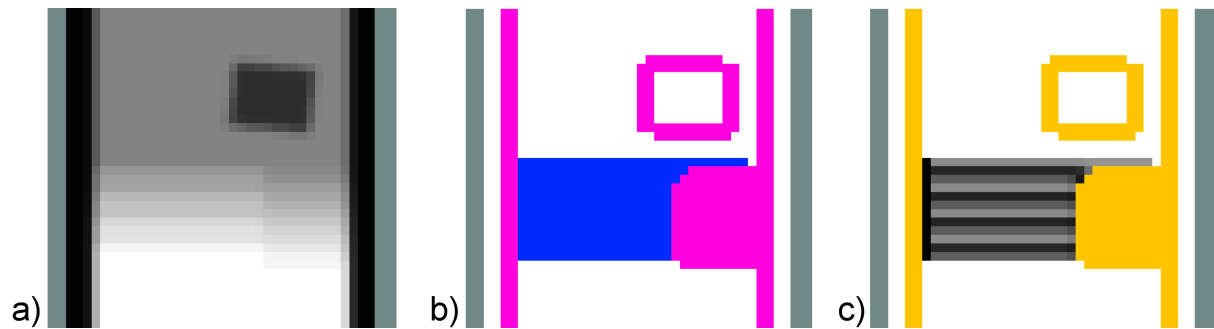


Figure 15: Level 3: a) height map, b) terrain class map (white = flat, blue = stepping, pink = wall), c) foot cost map.

- "Unknown" if $\Delta H(c_i)$ is unknown.

The height difference intervals are tuned manually with respect to a maximum height difference of 4 cm which can be overcome by driving and a maximum 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 easiest occurring terrain class.

Another source for terrain class segmentation can be camera images combined with geometric features as shown in CENTAURO Deliverable D5.2 *Terrain Classification* [4] and [36]. Figure 15 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 \vec{r}_b . Feet have fixed positions relative to the robot base. Hence, the robot is not able to perform individual foot or foot pair movements. 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.

Regarding cost generation, base cost do only depend on height information and are computed as described for Level 1. For foot cost computation, we assign each terrain class a fixed foot cost value. However, stepping cost strongly depend on the height difference to overcome and high steps are much more expensive than low steps. Thus, it is not sufficient to assign all cells of the terrain class "Stepping surface" with a fixed value since the original costs of traversing certain scenarios in Level 1 may differ too much from the estimated costs in Level 3. Instead, we introduce a linear dependency

$$C_F(c_{\text{Step}}) = 3.95 + 33.0 * \Delta H(c_{\text{Step}}) \quad (4)$$

between the height difference ΔH and the foot costs C_F for each cell c_{Step} with the terrain class "Stepping surface". Foot costs for each terrain class are parametrized to match best the cost, the respective path would carry in Level 1. This is done by comparing a set of simple and challenging paths from a start to a goal pose in Level 1 and Level 3 and adjusting the costs for Level 3 until they differ from the Level 1 cost by $\leq 5\%$. A resulting foot cost map can be seen in Figure 15 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 $\vec{r}_s, \dots, \vec{r}_g$ that needs to be refined at once. The first and last robot pose \vec{r}_s and \vec{r}_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}_s and \vec{r}_g . If a Level 3 path needs to be refined to Level 1, Level 2 is taken as an intermediate refinement step.

3.5 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 big 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, shift a wheel pair, ...) 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 planned again in this higher 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 slightly shifted to fit into the next level map cell and discrete orientation. Due to increasing foot restrictions, single feet might be shifted to fit the next level robot representation (e.g., single feet may have to be moved, to align to foot pairs with the same longitudinal coordinate). We check for each transformation if it is feasible and generate costs from the occurring base and foot costs.

3.6 Heuristic

We integrate three different heuristics in our planner. Their performance is compared in Section 4.

A very common heuristic for path planning is the Euclidean distance. Our cost terms are parametrized in a way that perfectly flat terrain carries the cost 1 and thus, driving over perfectly flat terrain induces cost equal to the driving length. Driving cost increase for more challenging terrain. Hence, the Euclidean distance always underestimates path cost and is feasible.

The Euclidean Distance only considers the robot position but does neglect its orientation. We introduce a second heuristic which extends the Euclidean Distance by a cost estimation to overcome the orientation difference from any pose to the goal pose. The cost to overcome an orientation difference is estimated by the cost to turn that angle on place on perfectly flat terrain. This is added to the Euclidean distance.

Both presented heuristics do not consider the terrain, the robot is traversing. Since terrain difficulties (e.g. staircases, obstacles to avoid) have large influence on the path costs, we include those in a heuristic to achieve better cost estimations. We utilize the Level 3 foot cost map to include terrain information in a heuristic which we call Dijkstra heuristic.

For the planner goal pose $\vec{r}_G = (\vec{r}_{bG}, f_{1G}, \dots, f_{4G})$, we determine the Level 3 map cells $c_{f_{1G}}^{\vec{r}_G}, \dots, c_{f_{4G}}^{\vec{r}_G}$ for each of the four foot positions f_{1G}, \dots, f_{4G} . An one-to-any 2D Dijkstra search on Level 3 foot costs is started from each of $c_{f_{1G}}^{\vec{r}_G}, \dots, c_{f_{4G}}^{\vec{r}_G}$. Hence, we get for each Level 3 cell c_i in the map four values g_1, \dots, g_4 which describe the costs to move a single foot from c_i to $c_{f_{1G}}^{\vec{r}_G}, \dots, c_{f_{4G}}^{\vec{r}_G}$.

In addition, we do a similar search for the robot base center. However, the robot base is not directly affected by different terrains, but we know that it cannot move through walls. We use this property to generate an occupancy grid from the Level 3 terrain class map which marks all cells of terrain class "Wall" as untraversable and all other cells of known terrain class

as traversable. A fifth one-to-any 2D Dijkstra search on this occupancy grid is started from the robot base center goal cell $c_{\vec{r}_{bG}}$ and provides for each cell in the map c_i a value g_b which describes the cost of moving the robot base center from c_i to $c_{\vec{r}_{bG}}$.

During path planning, we can use these generated values g_1, \dots, g_4, g_b as a heuristic. For any given robot pose \vec{r}_i , we first determine the Level 3 map cell for each of the four feet $c_{\vec{f}_{1i}}, \dots, c_{\vec{f}_{4i}}$ and the robot base center $c_{\vec{r}_{bi}}$. The accumulated cost $g_i = g_b(c_{\vec{r}_{bi}}) + \sum_{j=1}^4 g_j(c_{\vec{f}_{ji}})$ is a cost estimation for moving the whole robot from \vec{r}_i to \vec{r}_G . Note that the costs for the individual robot component movements (feet and robot base) are lower when those components are considered isolated in comparison to movements of the whole robot where those components underlie geometrical restrictions. But, since a heuristic shall underestimate costs, this is admissible. Further 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.

3.7 Planning Improvement

Due to the fine position and orientation resolution and the high robot state dimensionality in lower representation levels, the search space which is considered for path planning is large. Moreover, we want the planner to consider several detours before taking a step, which further increases planning times. We present methods to accelerate planning and to improve the resulting path quality. Their individual effects are investigated during evaluation.

3.7.1 Robot Orientation Cost

Although our robot is capable of omnidirectional driving, there are multiple reasons to prefer driving forward. Since the sensor setup is not only used for navigation, but also for manipulation, it is designed to provide best measurement results for the area in front of the robot. The required width clearance is minimal when driving in a longitudinal direction, which is helpful in narrow sections such as doors. The same applies to driving backwards. Driving straight backwards requires a smaller clearance than driving diagonal backwards. Finally, our leg design restricts us to perform steps in the longitudinal direction. Thus, when approaching areas where stepping is required, a suitable orientation is helpful. We address this desire of preferring special orientations by multiplying neighbour costs during A* search by the individual factor $k_{\Delta\theta}$, as described in Figure 16.

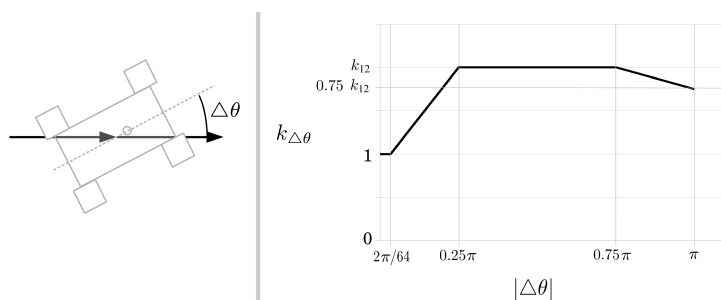


Figure 16: For a difference between the robot orientation and driving direction $\Delta\theta$, the cost factor $k_{\Delta\theta}$ is computed which expresses the desire to drive forward. It is 1 within an orientation step of $2\pi/60$ and increases up to k_{12} . When driving backward, there is a desire to drive straight backwards since the required clearance is smaller.

3.7.2 ARA*

To obtain feasible paths quickly, we extend the A* algorithm to an Anytime Repairing A* (ARA*). When planning with higher weighted heuristics, the planner prefers those driving manoeuvres that bring it as close to the goal as possible. This leads to the undesired effect that resulting paths mainly consist of those driving manoeuvres which go two cells in one direction and one cell in an orthogonal direction, as can be seen in Figure 17 a. To prevent this behaviour, we extend the driving neighbourhood size from 16 to 20, as shown in Figure 17 b.

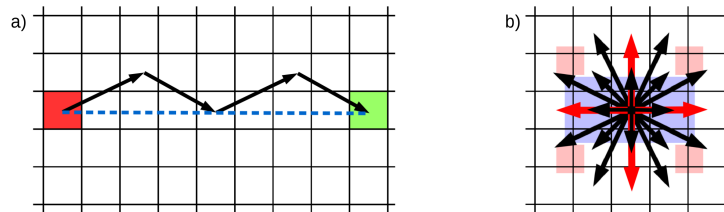


Figure 17: Addressing ARA* preferences of long moves. a) For larger heuristic weights, the ARA* algorithm prefers those driving manoeuvres which bring the robot as close to the goal as possible which leads to undesired paths (black arrows). b) To obtain the desired behaviour (blue line) we extend the driving neighbourhood by the four red manoeuvres.

3.8 Generation of Feasible Motion Sequences

The result of the A* search is an abstract path which lacks executable motion sequences for steps and information about leg length. We expand those abstract path segments which are part of Level 1, which is the area around the robot. Path segments of higher representation levels are expanded, when they are close to the robot and represented in Level 1. The expanded path contains leg length information and stable, detailed motion sequences for steps which can be executed by the controller.

3.8.1 Robot Stabilization

Abstract steps only describe the start and goal poses for a stepping manoeuvre. An executable transition between these poses is a motion sequence which keeps the robot stable at all times. Such a motion sequence is generated for each abstract step in the path. Due to the compliant leg design of our robot, we have no information about the exact position of the feet but have to estimate it from actuator feedback. Hence, we limit stability considerations to static stability. Since actuator speeds are slow, dynamic effects can be neglected. Stability estimation while stepping is done on the support triangle which is spanned by the horizontal position of the remaining three feet with ground contact (Figure 18). If the horizontal robot center of mass (CoM) projection is inside the support triangle, the robot pose is stable. The closer the CoM is to the support triangle centroid (STC), the greater the stability.

Lateral alignment of the CoM and STC is done by base roll motions (Figure 18), which are rotations around the longitudinal axis. These are achieved by changing the leg lengths on one side of the robot. The resulting angle between the wheel axes and ground is compensated by the compliant legs and the soft-foam filled wheels. Roll manoeuvre parametrization is described in Section 3.8.2.

Longitudinal alignment of the CoM and STC is done by driving the remaining pair of wheels on the stepping side (e.g., the rear left foot if the front left foot is stepping) towards the robot

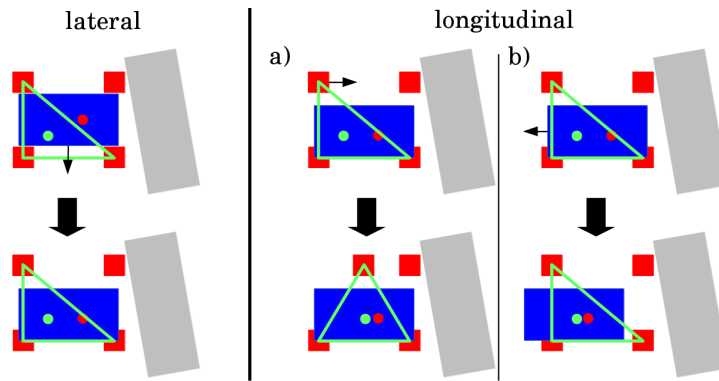


Figure 18: To find a stable position for stepping, the robot CoM (red dot) needs to be aligned with the STC (green dot). Lateral alignment is done by base rolling. Longitudinal alignment is either done by a) driving the remaining pair of wheels on the stepping side towards the center or b) by shifting the robot base.



Figure 19: Our hybrid wheeled-legged mobile manipulation robot Momaro is capable of omnidirectional driving (left) and stepping (right).

center (Figure 18). If this does not suffice because the motion is hindered by obstacles, the remaining longitudinal alignment is done by shifting the robot base. The longitudinal CoM position is also affected by the robot base pitch angle which is described in the next subsection. The presented motions generate a stable robot configuration which allows the desired step to be performed. After stepping, the robot reverses its base roll, foot displacement, and base shift manoeuvres to get back to its nominal configuration.

3.8.2 Leg Lengths

Each robot pose is assigned an individual length for each leg, which describes the vertical position of a foot relative to the robot base. When driving with neutral footprint, a short leg length of 0.27 m is chosen, which provides a low CoM and good controllability through reduced leg compliance (see Figure 19 left). A larger ground clearance is chosen for manoeuvres other than driving to provide great freedom for leg movements (see Figure 19 right). In this case, the base height is determined by the highest foot. A soft constraint is applied that the leg length should be at least 0.45 m. At the same time, a hard constraint from the mechanical system is that none of the legs exceeds its maximum length. The height of each individual foot can be read from the 2D height map. The robot base pitching angle is set to be 70% of the ground slope, as can be seen in Figure 20. This pitching value provides a good compromise between sufficient ground clearance for all four legs and a good CoM position.

As described before, base roll manoeuvres are used to shift the robot CoM laterally. Due to the soft-foam filled wheels, we can estimate the center of rotation $R(y'_{rot}, z'_{rot})$ between the two

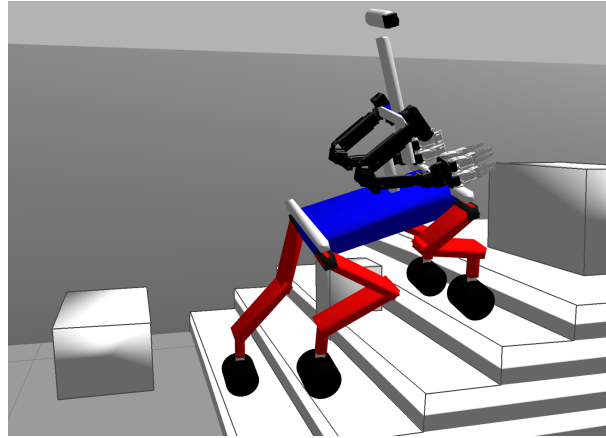


Figure 20: Momaro climbing a flight of stairs. The robot base pitch angle adapts to 70% of the terrain slope.

wheels (Figure 21). In addition, the center of mass position $C(y'_{CoM}, z'_{CoM})$, the angle

$$\alpha = \arctan \left(\frac{y'_{rot} - y'_{CoM}}{z'_{CoM} - z'_{rot}} \right) \quad (5)$$

between \vec{RC} and the vertical axis and the desired lateral center of mass position $y'_{CoM,des}$ are given. Using $\|\vec{RC}\| = \|\vec{RC}_{des}\|$ we compute the desired angle between \vec{RC}_{des} and the vertical axis

$$\alpha_{des} = \arcsin \left(\frac{y'_{rot} - y'_{CoM,des}}{\|\vec{RC}\|} \right) \quad (6)$$

and consequently using the footprint width b we compute the desired leg height difference

$$\Delta h_{leg} = b \cdot \tan(\alpha - \alpha_{des}). \quad (7)$$

This leg height difference is added to both legs on the respective side to induce a base roll manoeuvre. Figure 22 shows how Momaro steps up an elevated platform, using the described motion sequences.

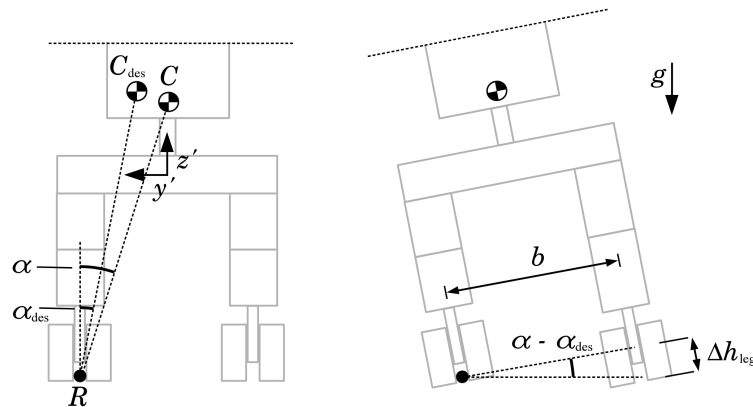


Figure 21: Momaro's lower body in back view. Lateral CoM shifts can be achieved by changing leg length on one side which rolls the robot.

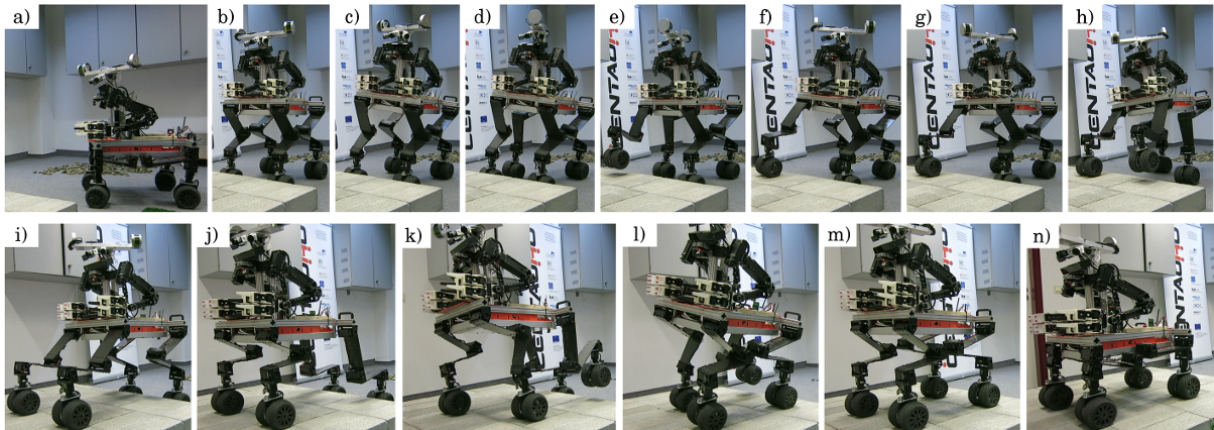


Figure 22: Momaro stepping up an elevated platform. a) It arrives at the platform in low driving position, b) stands up, c) rolls its base to the left to shift its CoM laterally, d) drives its rear right pair of wheels forward to reach a stable stepping configuration. e) It then steps with its front right foot, f) drives its rear right pair of wheels back and g) rolls back its base to reach the configuration it had before the step. The remaining steps follow a similar motion sequence which is shown in less detail. Subsequently, h) Momaro steps with its front left foot. i) It then drives forward and j) shifts its base forward before k) doing a step with the rear left and l) rear right foot. m) When the robot stands on the platform, n) it lowers its base to continue driving.

4 Evaluation

We evaluate the performance of our planner in three experiments which focus on different aspects of our method. All Experiments are performed in the Gazebo simulation environment. The used hardware for planning is one core of a 2.6 GHz Intel i7-6700HQ processor using 16 GB of memory. Video which show several experiment of the real and simulated robot can be seen online^{1 2}.

4.1 Heuristic Comparison

A first experiment compares the Dijkstra heuristic to the plain Euclidean distance and Euclidean distance combined with orientation differences. The robot stands in front of an elevated platform with an irregular edge. A small wall restricts the traversable area. The planning goal is situated between two pillars on this platform. The whole experiment is done in Level 1. Since we use an ARA* algorithm which works with several heuristic weights, we evaluate the influence of these heuristic weights. Figure 23 shows the scenario, the Level 1 foot cost map and the Level 3 foot cost map which is used for the Dijkstra heuristic. Preprocessing the Dijkstra heuristic takes 0.016 s for this map. Figure 24 shows the resulting planning time and result costs for all three heuristics and several heuristic weights.

The results indicate that planning with the Dijkstra heuristic is significantly faster while the path costs are $\leq 5\%$ more expensive compared to the other heuristics. The results further indicate that increasing heuristic weights accelerate planning with the Dijkstra heuristic by up to four orders of magnitude while the path costs increase by $\leq 45\%$ (heuristic weight of 3). For lower heuristic weights, the path costs quickly decrease to an acceptable level of $\leq 105\%$ of the optimal costs (heuristic weight = 1.5).

¹https://www.ais.uni-bonn.de/videos/IROS_2017_Klamt/

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

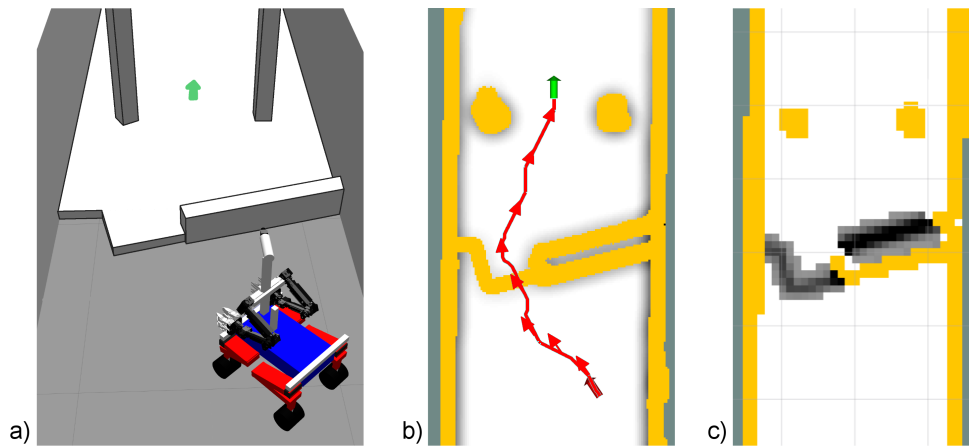


Figure 23: Experiment to compare the different heuristics with each other. a) Scenario: The robot has to step up an elevated platform which is partly blocked by a small wall. The goal pose is between the two columns on this platform. b) Level 1 foot cost map with result path for Dijkstra heuristic and heuristic weight of 1.5. c) Level 3 foot cost map which is used to compute the Dijkstra heuristic.

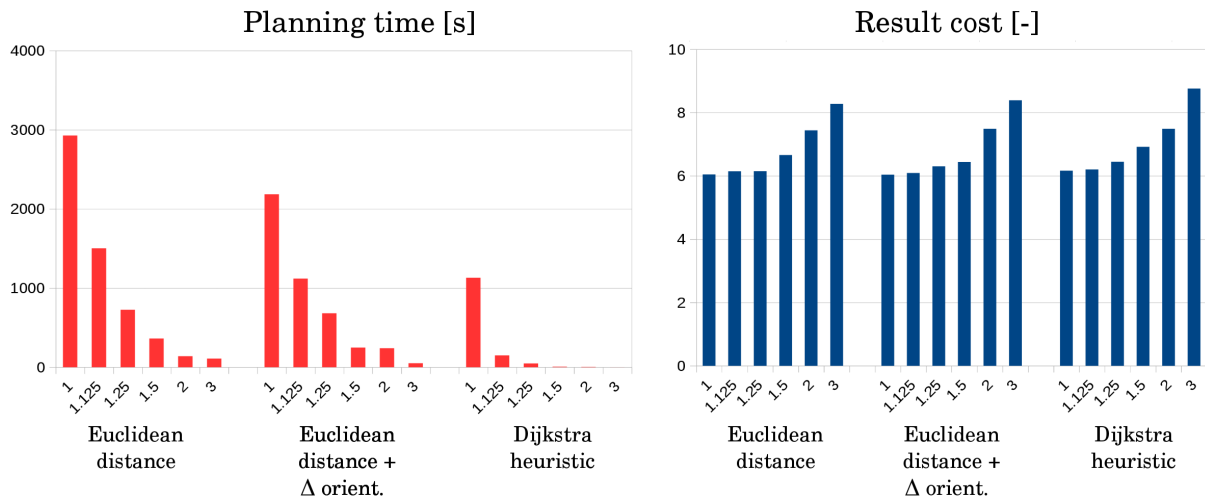


Figure 24: Planning time and path cost for the experiment, shown in Figure 23. The three different heuristics Euclidean distance, Euclidean distance + orientation difference and Dijkstra heuristic are compared for heuristic weights between 1.0 and 3.0.

4.2 Effect of Orientation Cost Term

In a second experiment, we evaluate the effect of the introduced heuristic cost term (see Section 3.7.1). Again, the robot has to climb up an elevated platform which is partly blocked by cluttered obstacles. Figure 25 shows the scenario and two resulting path with and without an orientation cost term. Figure 26 shows the result planning time and path cost and evaluated the effect of the orientation cost term on the orientation difference between robot orientation and driving direction. The used heuristic is the "Euclidean distance + orientation difference" heuristic.

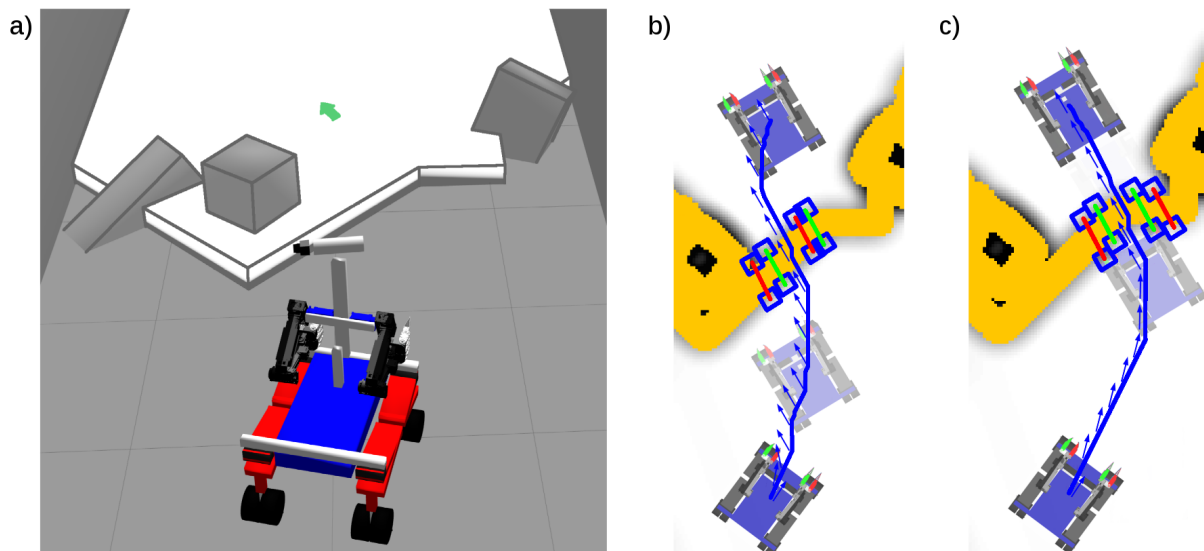


Figure 25: Experiment to evaluate the effect of the orientation cost term. a) Scenario: The robot stands in front of an elevated platform, cluttered with obstacles and has to reach a pose on this platform. b) Resulting path of a plain A* search without orientation cost term and with a heuristic weight of 1.0 on a foot cost map of Level 1. Yellow areas are not traversable by driving. Blue paths show the robot center position, arrows show the orientation. Blue rectangles show used footholds. Red lines represent front foot steps, green lines represent rear foot steps. c) Resulting path with orientation cost term ($k_{12} = 2$).

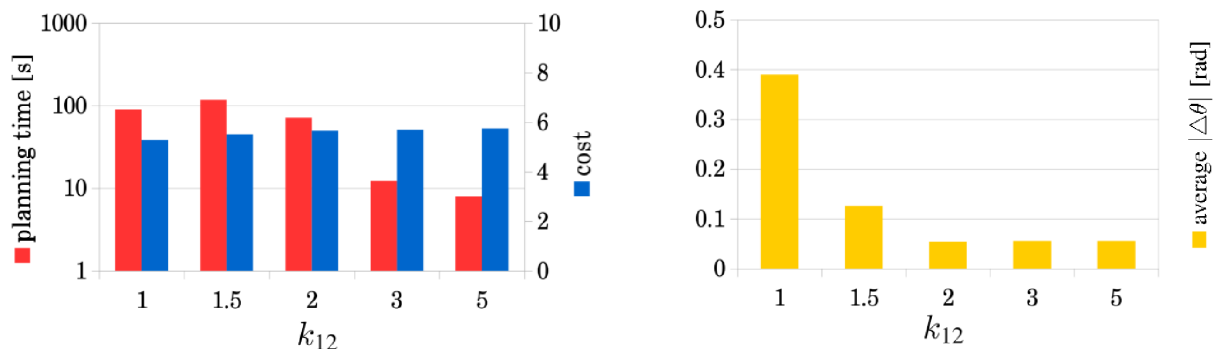


Figure 26: Comparison between the original A* planner ($k_{12} = 1$) and the modification with a robot orientation cost term.

4.3 Effect of Multiple Levels of Representation

In a further experiment, we evaluate the different presented levels of representation. We plan on a height map which contains many different terrain difficulties our robot could face, as shown in Figure 27. In a first run, we generate four plans from robot pose a to e: three for the task solved in each level individually and one using all three levels at the same time. For this, we choose the Level 1 size to be 3×3 m. This size is sufficiently large to plan the next robot manoeuvres in high detail, but still small enough to not slow down the whole search too much due to 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. Level 3 covers the whole map. The heuristic weight is chosen to be 1.25. The Dijkstra heuristic is used and takes 0.027 s for preprocessing.

The resulting path can be seen in Figure 28 and the planner performance is shown in Figure 29. It can be seen that planning on levels of representation > 1 and in the scenario with com-

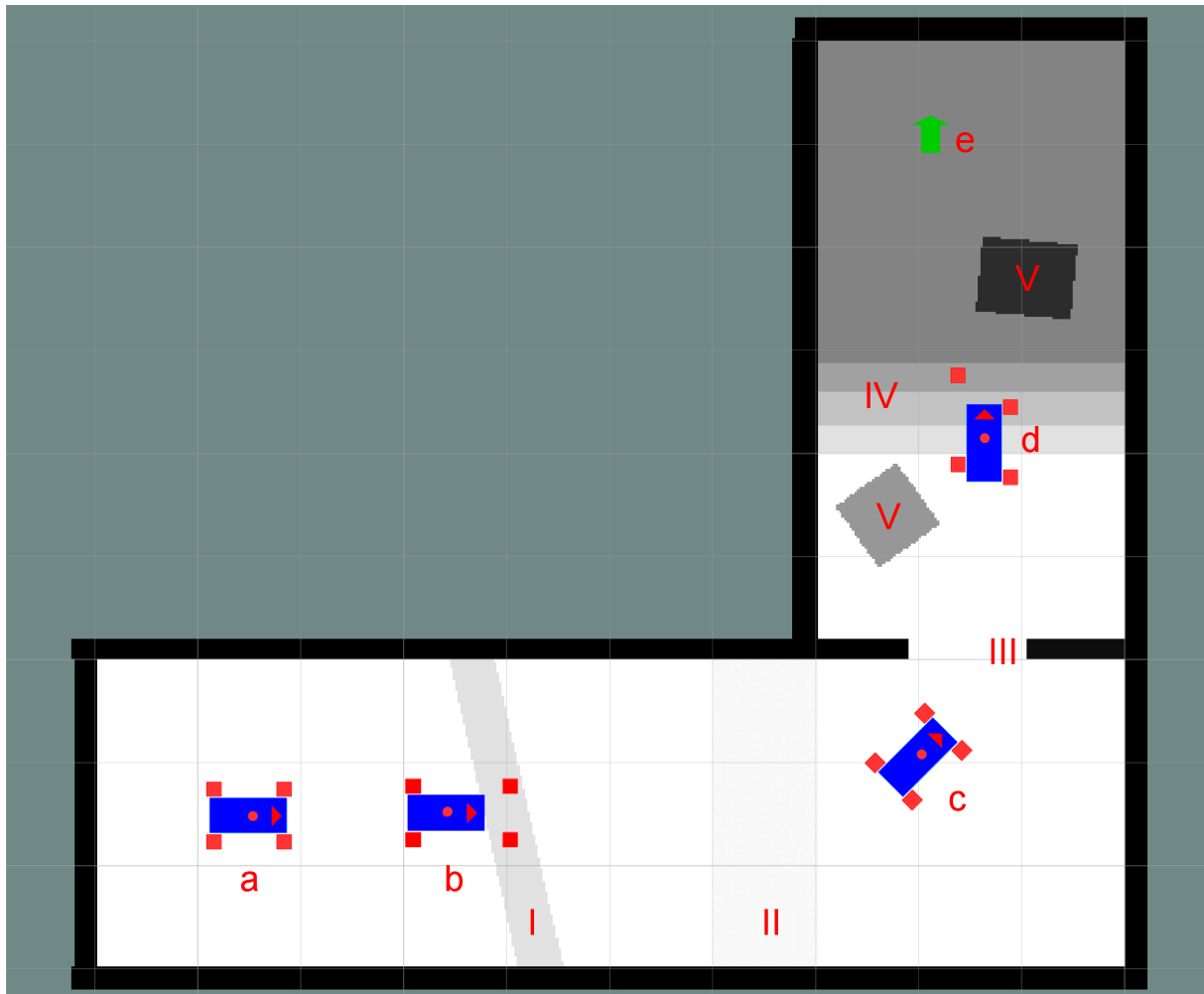


Figure 27: Height map for the second experiment containing a bar obstacle (I), a rough area (II), a door (III), a flight of stairs (VI) and two obstacles (V). a - d are different starting poses for the planner, e is the goal pose.

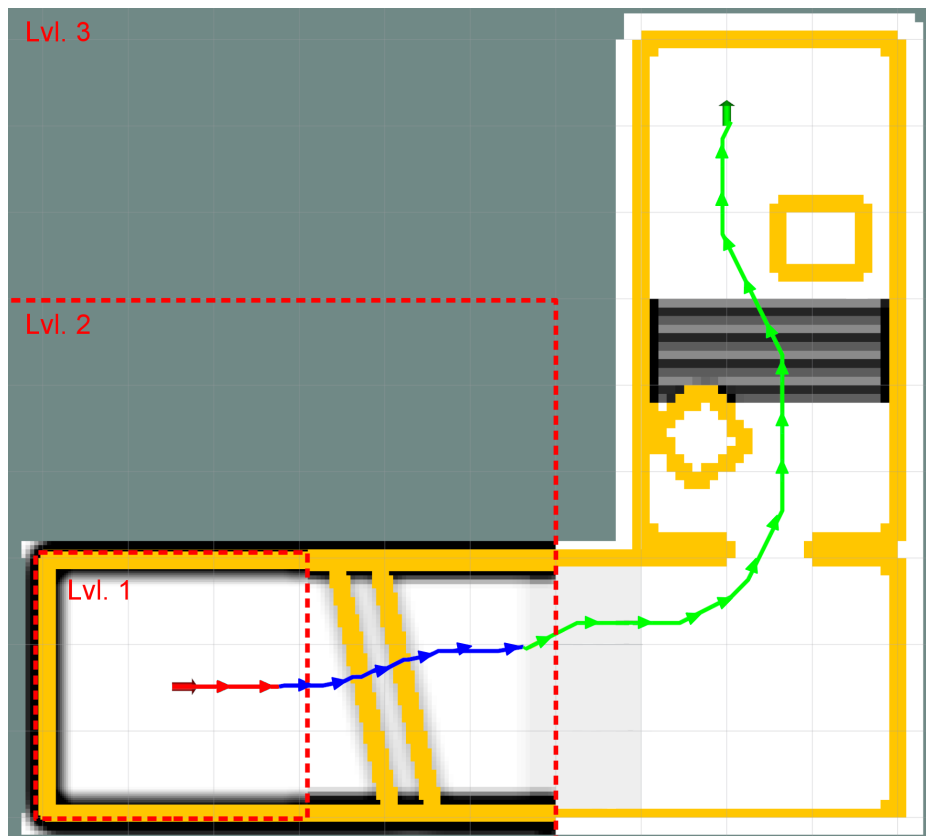


Figure 28: Resulting path for planning in combined levels with a heuristic weight of 1.25. Level 1 path segments = red, Level 2 segments = blue, Level 3 segments = green. Arrows show the robot orientation.

combined levels is faster by at least two orders of magnitude compared to pure Level 1 planning. Regarding the path costs, we distinguish between the path costs in the respective level of representation (estimated cost) and the costs each path carries when refined to Level 1. Comparing the estimated costs to the 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 indicates the quality of the resulting path. It can be seen that the estimated costs always underestimate the refined Level 1 costs but the difference is always $\leq 9\%$. The refined Level 1 paths are $\leq 15\%$ more expensive than the pure Level 1 path.

We finally compare the planner performance when started from different poses in the map as shown in Figure 27. The results in Figure 30 indicate that an important factor for the planner performance is the complexity of the planning within Level 1 but higher heuristic weights lead to feasible performances in any case.

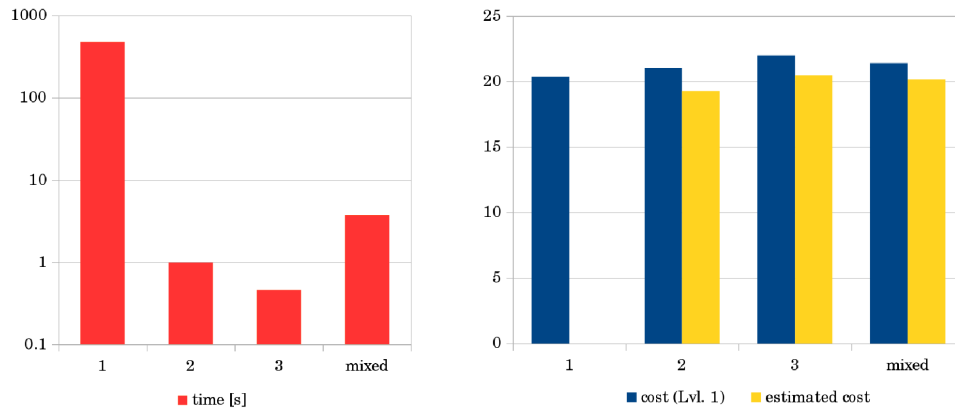


Figure 29: Planning performance for different levels of representation, using the Dijkstra heuristic with a heuristic weight of 1.25.

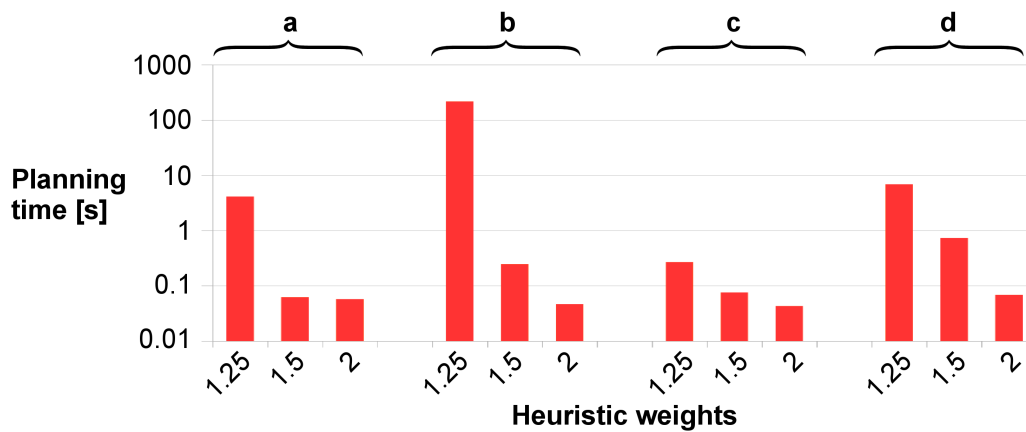


Figure 30: Planning time for different starting poses (see Figure 27) and different heuristic weights, using the Dijkstra heuristic.

5 WP5 on the CENTAURO Robot

As shown in Figure 1, we integrated and tested the communication links between all components of WP5. We established the interfaces of the output robot control commands to Verosim and the real robot. All tasks have been tested with the Momaro model in both simulated and real world environment. The tasks of rough terrain SLAM and semi-autonomous driving/stepping locomotion have been integrated and tested on the CENTAURO model. The task of terrain classification with terrain label "safe, risky and obstacle" has been tested on the data recorded by the CENTAURO robot during the evaluation camp. In this section we will describe the status of each component in WP5 working on the CENTAURO robot. An overview of the current integration status on the CENTAURO robot is shown in Figure 31.

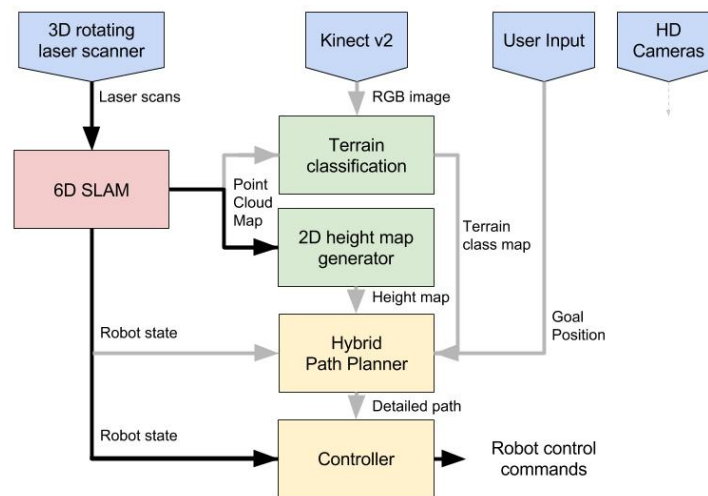


Figure 31: The current integration status of WP5. The black arrow indicates the interface that has been fully tested on the CENTAURO robot and the gray arrow indicates the interface that has been partially tested.

5.1 Rough Terrain SLAM

As for the Momaro robot, we employ our mapping and localization approach based on local multiresolution surfel grid maps [?] on the CENTAURO robot. Similarity between the sensor setups and software systems allowed for easy setup and integration of necessary software components. The laser scanner provides about 300,000 range measurements per second with a maximum range of 100 m. It is rotated at 0.1 rotations per second, resulting in a dense omnidirectional 3D scan per half rotation. Slower rotation is possible if a higher angular resolution is desired. For our current setup, we acquire one full 3D scan every 5 seconds and compensate for sensor motion during acquisition by incorporating measurements of the IMU. Figure 32 shows an example of a generated point cloud and a localized robot.

5.2 Terrain Classification

The terrain classification system generates a terrain class map with label "safe", "risky" and "obstacle" using the RGB image and the registered pointcloud. The RGB image is from the Kinect sensor and the pointcloud is produced by the 6D SLAM component. The output terrain class map is sent to the Hybrid path planner to test the interface.

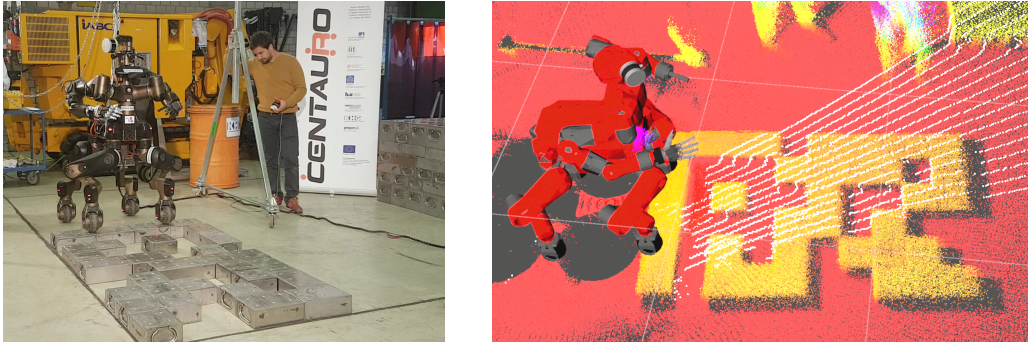


Figure 32: Centauro overcoming a step field: Scenario (left), localized robot and registered point cloud color coded by height (right).

During the evaluation camp at KHG, the robot was controlled semi-autonomously without the high level planning that uses the terrain classification. Therefore the terrain classification component was only tested off-line using the data recorded. Before the evaluation, to adapt to the testing environment at KHG, we manually labeled three images to fine-tune our model. An example classification result of the stepping field is shown in Figure 33. The interface between output terrain class map and the hybrid path planner has been tested using the old data recorded in Bonn.

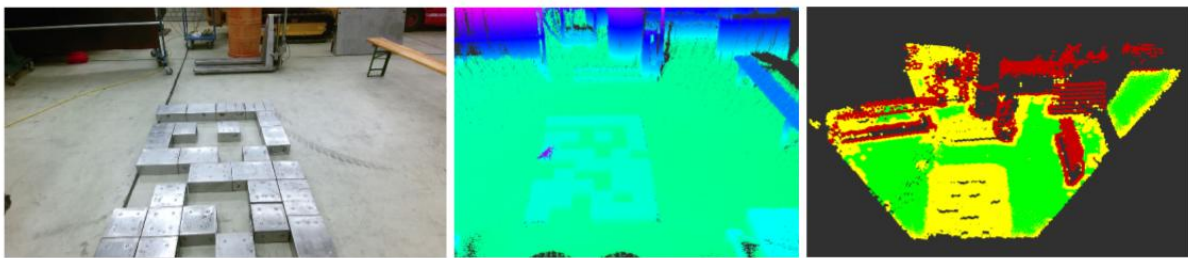


Figure 33: The classification result of KHG stepping test. From left to right is the RGB image, registered pointcloud and the classification result as colored points (green as safe, yellow as risky and red as obstacle).

5.3 Semi-autonomous Driving/stepping Locomotion

During the integration meeting between UBO and IIT in Genoa in November 2017 and the successive evaluation camp in Karlsruhe, we were able to establish the control interface of the Centauro robot. The interface consists of two parts, which are necessary to execute paths that are planned by the hybrid path planner and executed by the successive high-level controller:

- A 3D (v_x, v_y, v_θ) omnidirectional velocity vector. This interface is used to control omnidirectional driving in an arbitrary leg configuration.
- Leg movement actions which move all four feet to given coordinates in Cartesian space in robot coordinates. This interface is used to control steps and all stepping related maneuvers such as base rolls, base shifts or foot shifts.

The 3D omnidirectional velocity vector interface was tested by controlling the robot with a joystick which uses the same interface. We used such control to drive up a ramp, to perform

driving sub-maneuvers while stepping over a gap, to precisely correct the robot position while walking over a step field and to move through a door, as shown in Figure 34.

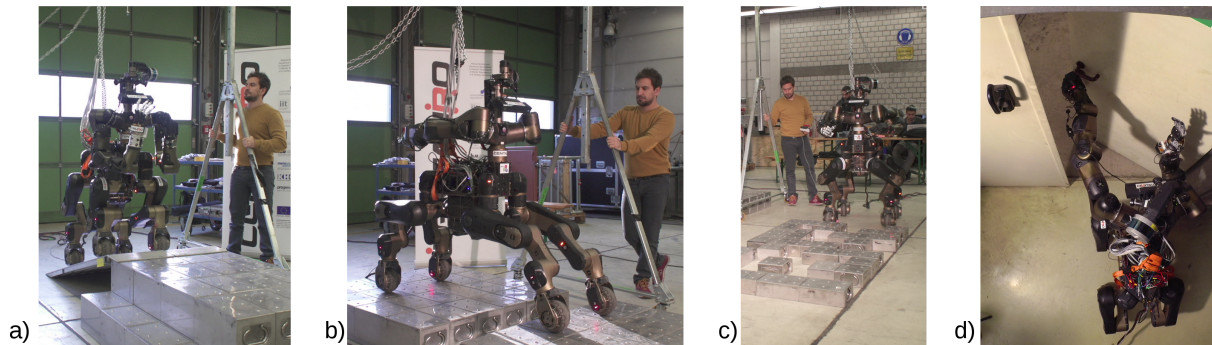


Figure 34: The 3D omnidirectional driving interface was used during several tasks at the evaluation camp in Karlsruhe. a) Driving up a ramp, b) driving submaneuver while stepping over a gap, c) precise robot positioning while navigating a step field and d) opening and moving through a door.

The leg movement action interface was used to perform the step field task in a semi-autonomous way. We developed a user interface to trigger individual robot maneuvers such as steps with a given foot. The action sequences that are necessary to perform such maneuvers are performed autonomously and send to the robot via the described leg movement action interface (e.g. a step maneuver consists of the following action sequence: longitudinal base shift, base roll, foot lift, leg extension, foot lowering, base roll and base shift). Moreover, we used torque measurements from the robot joints to detect ground contact of the stepping foot and stop the foot lowering action respectively as soon as the foot touched the ground. Figure 35 shows the user interface and the Centauro robot while stepping in the step field.

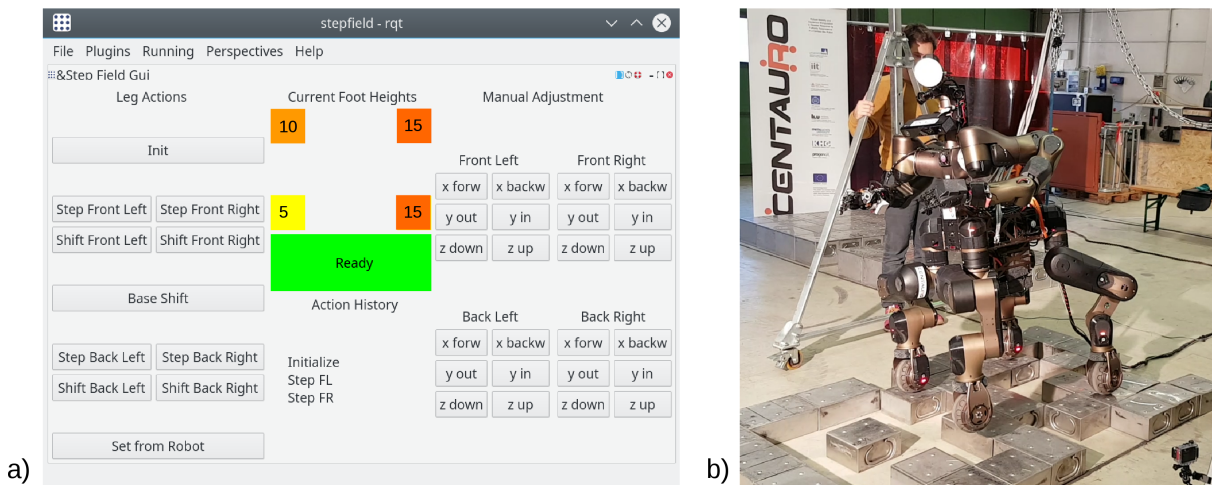


Figure 35: The step field task was solved semi-autonomously by a) a user interface which triggers maneuvers that are b) executed by the robot.

6 Future Work

We plan to extend the current navigation system by introducing new terrain classes, like stairs, which are highly relevant in indoor and urban environments. Furthermore, the existing planning approach will be extended to consider lateral leg movements in both, driving and stepping motions. Up to this point, the method was mainly developed for Momaro, because it was possible to perform experiments with it. Momaro did not allow lateral leg movements due to its design. Since the Centauro robot is available now, the method will be transferred to this robot in both, simulation and real experiments which includes adding additional capabilities of the Centauro robot. Ongoing work with an alternative planning strategy (see Appendix: Additional Work 4) using deep reinforcement-learning could provide an additional suggested path that can be presented to the operator, who can then chose which path to pick. We will integrate the additional planner to the navigation pipeline with Verosim and the real robot after we extend its capability from simulated scenes to real world tasks. After we include the operator in the loop, we plan to take the feedback regarding terrain traversability from the operator to perform an on-line supervised learning for terrain classification.

A Additional Work in Workpackage 5

A.1 Additional Work 1

Computing a Collision-Free Path using Harmonic Potentials

The paper [13] introduces a novel method for finding a safe route through an environment. It is based on the monogenic scale-space in combination with a potential-based representation of the environment, where the latter directly enables the path planning. The paper also demonstrates how the method can be used to handle obstacles. Furthermore, it is also possible to introduce "soft obstacles", such as terrain types that might be more dangerous to traverse than others.

A.1.1 Connection to CENTAURO

The novel method reported above is highly relevant for a robot in a disaster area, especially for the Centauro robot with multiple terrain traversal modes. The proposed approach applies to both the global map and the local navigation map as described in CENTAURO Deliverable D5.1 *CENTAURO Navigation Concept* [15]. Future investigation will show whether the proposed continuous approach will improve performance compared to the current grid-based approach.

Table 1: Absolute Trajectory Error (ATE) for a subset of datasets from the TUM RGB-D datasets using different types of features in the same visual odometry pipeline.

Dataset	GCN	ORB	SIFT	SURF
fr1_floor	0.015m	0.080m	0.073m	0.074m
fr1_desk	0.037m	0.151m	0.144m	0.148m
fr1_360	0.059m	0.278m	0.305m	0.279m
fr3_long_office	0.061m	0.090m	0.076m	0.070m
fr3_large_cabinet	0.073m	0.097m	0.091m	0.143m
fr3_nst	0.020m	0.061m	0.036m	0.030m
fr3_nnf	0.221m	-	-	-

A.2 Additional Work 2

Geometric Correspondence Network for Motion Estimation

In the literature, approaches can roughly be divided into sparse / feature based and dense / direct. In this work [40] we address the problem of identifying and matching key points in sparse / feature based methods. It is well known that the key points have to be selected carefully and that matching them correctly is essential for accurate motion estimation. Current state of the art methods like ORB-SLAM make use of hand crafted features still. We investigate how to train both the detection and matching of key points jointly. To this end we jointly train a CNN on top of an RNN in what we call the Geometric Correspondence Network (GCN). The CNN outputs a 64 channel feature map and the RNN outputs a heat map indicating which parts in the images are suitable as key points. The training is accomplished by extracting high gradient points in one image and then warping them using the known transformation between the frames (given by ground truth pose from SLAM dataset). The training is thus trying to find feature maps from the CNN that provide a good basis for matching at the same time as the RNN learns which points to use.

We trained the network on data from the TUM RGB-D dataset. A requirement for the training is that we have access to ground truth pose data so that we can perform the warping from one image to the next. In a preliminary evaluation we also tested the performance in a visual odometry context. The same pipeline was used with different types of features. Table 1 show the Absolute Trajectory Error (ATE) for a subset of datasets from the TUM RGB-D datasets (not used for training). Notice that the GCN features improves the performance over the tested hand crafted features for all datasets.

A.2.1 Connection to CENTAURO

Estimating the motion of a sensor is central to many tasks, such as local 3D reconstruction and large scale mapping. These are both key areas in CENTAURO. Our GCN method has so far been tested on standard SLAM benchmark datasets and on data from a UAV and have shown very promising results. We plan to investigate how well it works with data from the Centauro robot. As the detection and matching of key points relies mostly on the RGB information, we believe that the method can be used both with stereo data and as well as data from Kinect v2.

Table 2: Benchmark results on the reduced test set in Semantic3D [9]. IoU for categories (1) man-made terrain, (2) natural terrain, (3) high vegetation, (4) low vegetation, (5) buildings, (6) hard scape, (7) scanning artefacts, (8) cars.

	Avg IoU	OA	IoU1	IoU2	IoU3	IoU4	IoU5	IoU6	IoU7	IoU8
TML-PCR[26]	0.384	0.740	0.726	0.730	0.485	0.224	0.707	0.050	0.000	0.150
DeepNet[9]	0.437	0.772	0.838	0.385	0.548	0.085	0.841	0.151	0.223	0.423
TLMC-MSR[10]	0.542	0.862	0.898	0.745	0.537	0.268	0.888	0.189	0.364	0.447
Ours RGB	0.515	0.854	0.759	0.791	0.720	0.335	0.857	0.209	0.123	0.326
Ours D	0.262	0.662	0.281	0.468	0.395	0.179	0.763	0.006	0.001	0.000
Ours N	0.511	0.846	0.815	0.622	0.679	0.164	0.903	0.251	0.186	0.470
Ours RGB+D+N	0.585	0.889	0.856	0.832	0.742	0.324	0.897	0.185	0.251	0.592
Ours D+N	0.543	0.872	0.839	0.736	0.717	0.210	0.909	0.153	0.204	0.574

A.3 Additional Work 3

Deep Projective 3D Semantic Segmentation

The Kinect v2 is a useful sensor for many tasks within robotics. However, the performance is significantly reduced in outdoor environments and for surfaces with high reflectance. In the CENTAURO project, however, additional sensors such as the rotating lidar and the color cameras can be used to complement the Kinect v2 for the workspace perception task. The lidar and the color cameras can be combined to generate a colored point-cloud. In this work we propose also a method of point-cloud segmentation [22], where we render color, depth and other attributes extracted from the point-cloud. The images are then processed by a CNN for image-based semantic segmentation, providing a prediction score for the predefined classes in every pixel. We make the final class selection from the aggregated prediction scores, using all images where the particular points are visible. An overview of the method is illustrated in Figure 36.

The method was evaluated on the Semantic 3D dataset, improving over previous methods, see Table 2.

A.3.1 Connection to CENTAURO

The novel segmentation method can be seen as an alternative to the sequential pipeline described in *Deliverable D5.2 CENTAURO Terrain Classification* [4].

A.4 Additional Work 4

Reinforcement learning for motion planning

A.4.1 Introduction

Over the past decade, legged robot such as the Big dog and little dog have shown its potential in unstructured and challenging terrain. The more recent 'wheel-on-leg' design such as Momaro and Mammoth [33] combines the advantages of the flexibility legs on rough terrain with the efficiency of traditional wheeled robot on flat terrain. The reconfigurable mobile robot is able to dynamically alter its geometric structure to extend the mobility of the robot to adapt to a variety

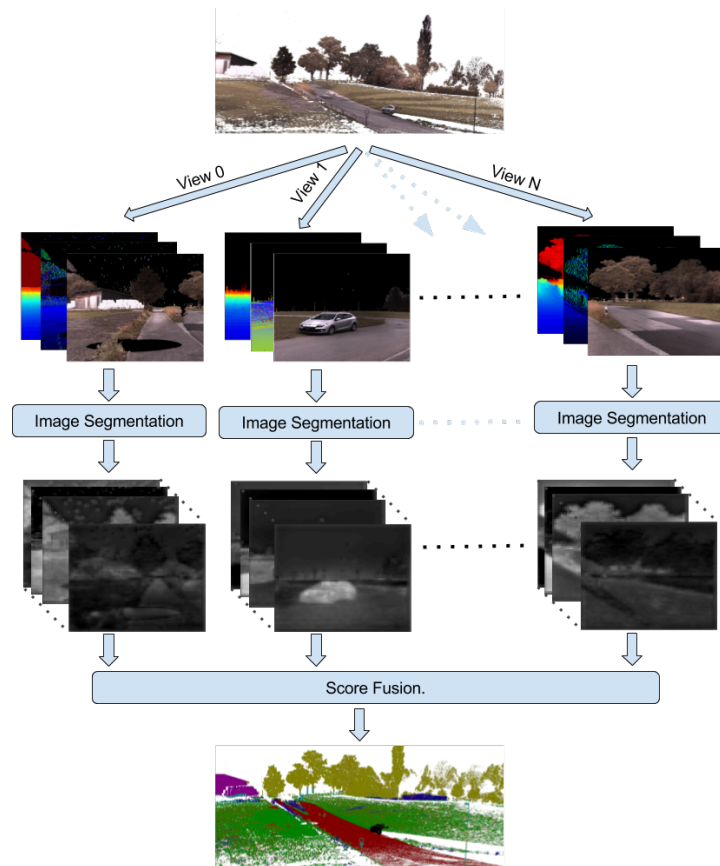


Figure 36: Illustration of the proposed method for point-cloud segmentation. [22] The input point-cloud is rendered from multiple views generating dense image representations. These images are processed by a CNN for semantic segmentation. In the end the semantic score maps are fused and mapped back to the 3D points.

of environments. However, motion planning for robots with many DoFs in cluttered and rough terrain is challenging as the set of feasible configurations that allow collision-free and stable motion lie in a small subspace of the whole configuration.

The high dimensional whole body planning problem is often decoupled into a set of lower dimensional sub-problems to reduce the search space and computation time for each component. [24] summarize the decoupled planning as *motion-before-contact* approach and *contact-before-motion* approach.

The *motion-before-contact* approaches, such as [43], [19], [44] and [45], address the efficiency issue by first planning a rough trajectory for the robot body center without considering the configuration of its leg. Footsteps are then planned to roughly follow this trajectory while considering appropriate constraints. The *contact-before-motion* approach, such as [41], translates the whole body motion planning to a lower dimension footstep trajectory planning. It first selects a set of feasible foot holds, then finds the transitions between all the possible foot holds and finally computes the path that links the initial and the target positions. Compared to the *motion-before-contact* approach, the *contact-before-motion* approach generates more versatile movements but it is more computationally expensive. Both the *motion-before-contact* approach and the *contact-before-motion* approach attempt to construct an initial trajectory that provides a restricted search space to allow a simpler planner to be used to generate the whole body plan. This restriction step, however, may eliminate good solutions before the second planner starts to compute the whole body trajectory. Moreover, the initial trajectory may not be feasible for the whole body plan, as it is generated in a reduced dimensional space.

The Rapidly-Random Exploring Tree (RRT) algorithm [20] offers an efficient solution of single-query motion planning problem by growing a tree in the configuration space towards newly sampled configurations. While it is suitable for high DOF systems, it requires the knowledge of the robot model and the environment to define the validation of the sampled configurations.

Our goal is to develop a target-driven end-to-end whole body planner for a reconfigurable robot with many DoFs. As a first step we consider the design of a local planner, which is able to handle cluttered environments containing narrow passages. We use an on-line deep reinforcement learning (DRL) procedure which does not require a human-guided training phase. We take a local height map, current robot pose and the target position as input and directly map it to the action commands. The aim is a local planner that this way is able to generalize to new scenes with different target positions without additional training.

In the future work we plan to combine it with an RRT planner to improve the local connection efficiency of RRT and generate global plans over long distances. The RRT planner could take the abstract environment information, such as the building blue print, and the global target as input to sample a configuration in a lower dimensional space. The sampled configuration could then be used as the sub-target to the DRL planner. The DRL planner then processes the local observation and the sampled configuration to generate a whole body path to connect the current configuration to the sampled configuration.

A.4.2 Related Work

Previous work already showed the performance gain by deep learning. [27] presents an approach that learns the heading policy from human demonstration for a mobile robot driving in an off-road environment. [35] follows a similar process and applies it to a micro aerial vehicle (MAV). The MAV takes image data as input and is able to autonomously fly through a forest environment while avoiding collision with trees. [38] presents a laser-based end-to-end motion

planning approach based on deep auto-encoders learned from a dataset of sensor input and motor commands collected while being tele-operated by a human. [32] collects trajectories using an existing global path planning approach and map the laser data and the target position to the motion commands. Such learning processes are highly dependent on the demonstration. A time consuming data collection procedure is inevitable.

A growing amount of success has been reported for DRL in navigation and motion planning. [47] provides an approach for robot navigation based on depth images. The model is trained using a deep Q-network (DQN), where successor features are used to transfer the strategy to unknown environment efficiently. [12] and [28] have shown impressive results of learning locomotion skills in both 2D and 3D. The models are evaluated using simulated characters with many DoFs. However, the models are not applicable as a local motion planner as the target position is not taken into account. [39] presents a target-driven mapless motion planner using laser to generate steering commands. [48] trains an image-based planner where the robot learned to navigate to the reference image place based on the current view. All of these works are applied to the robots with few DoFs. [29] adopts a two-level hierarchical control framework: a high-level controller to learn the desired steps and a low-level controller to perform the steps. They evaluate the framework using a variety of environment-aware locomotion tasks including navigation and obstacle avoidance. However the navigation and obstacle avoidance task does not contain narrow passages.

A.4.3 Learning the Local Planner

Proximal Policy Optimization Algorithms In order to learn effectively in a challenging setting like ours with many DoFs and a complex environment, it is necessary to have a reliable and scalable reinforcement learning algorithm. The Proximal Policy Optimization (PPO) [12], [37] has shown impressive performance on learning complex locomotion. We use the distributed version of PPO (DPPO). With DPPO the data collection is distributed to multiple workers which empirically has been shown to improve effectiveness. The PPO algorithm is based on the actor-critic paradigm which consist of two distinct components: a policy network "actor" to maintain and update an action-selection policy; and a value network "critic" to estimate the expected return associated with the actor's policy. The trajectory is generated by calling the actor iteratively until timeout or the target configuration is reached.

Environment and Robot Model

Robot Action Space As described in deliverable D2.3, each leg of CENTAURO robot consists of six DoFs based on the spider like configuration. The design of the leg is shown in Figure 37. To simplify the problem we assume all four legs perform the same action symmetrically. We use the *hip_pitch* and *knee_pitch* joints to adjust the robot posture and leave the *hip_yaw* joint as fixed. The *ankle_pitch* joint is computed using inverse kinematics and the *ankle_yaw* joint and the *wheel* axis is adjusted by a controller.

We transfer the action space from the *hip_pitch* and *knee_pitch* joints to leg opening width and body height. Together with the body translation along x and y axis and the body heading angle *yaw*, we consider the lower body action space to be 5 dimensional.

We apply discretized action commands where each dimension of the action contains three modes. Mode 1 performs an action along the positive direction with a fixed unit, mode 2 performs an action along the negative direction and mode 3 performs no action.

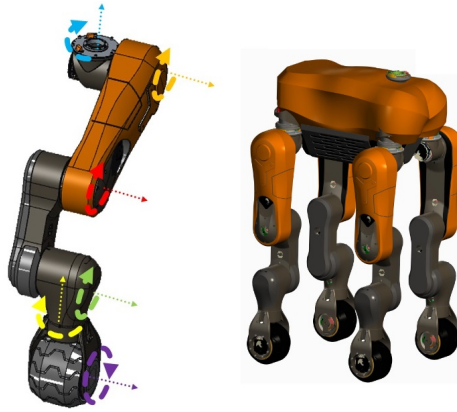


Figure 37: The lower body design of CENTAURO Robot. Blue encodes the *hip_yaw* joint, orange encodes the *hip_pitch* joint, red encodes the *knee_pitch* joint, green encodes the *ankle_pitch* joint, yellow encodes the *ankle_yaw* joint and purple encodes the *wheel* axis.

Training Environment The training procedure of our planner is implemented in virtual environments simulated by V-REP ([34]). The training environment consists of flat ground and ten cylinder obstacles. The obstacles have three different heights which require three types of locomotion skills to handle. The obstacle with low height is easy to drive over, only making sure that the wheel itself does not collide with the obstacle. The obstacle with middle height may collide with both wheels and legs, the planner has to lift the robot body and let it pass between the narrow opening of the legs. The highest obstacle cannot be driven over, the planner should generate a path around it to avoid any collision with the robot body. In the training environment, we have three tall obstacles, three middle obstacles and four low obstacles. The obstacles and one example training scene is shown in Figure 38.

In every episode, the obstacle number, the obstacle position, the robot's initial pose and the target position are randomly generated in the area of 3×3 meter square. The robot initial pose and the target position are guaranteed to be collision-free. The episode terminates when time out or the target is reached. When the robot collide with an obstacle, we restore the robot back to its previous collision-free pose, give negative reward and re-sample another action from the action distribution. For training to be successful the robot must occasionally avoid all obstacles and reach to the target. When there is narrow passage which allows only a small set of poses to pass through, the probability of this happening is very small. In order to find path in narrow passage, we do not terminate the episode when colliding with obstacles. Instead, we let the robot search near the obstacle to look for any feasible path. Then we gradually improve the initial path and keep searching for new paths.

Repeat Failed Episode It is difficult to control the complexity of the episode when all components are randomly generated. The policy may converge to the local minima if the episodes used to train the model contain bias. In our case, we notice that the path easily converged to a straight line, since the obstacles are sparse and more than half of the randomly generated episodes can be solved by moving straight to the target. We apply a repeat buffer to save the episode which failed to reach to the target. When generating new episodes, we assign higher probability to repeat a failed episode than to generate a random one. The failed episode contains steps which are against to the current policy. The difficulty of the episode in the repeat buffer is increased as the performance of the model increases. We observed that the model trained with a repeat buffer improves faster and generalizes better to new episodes. To encourage exploration,

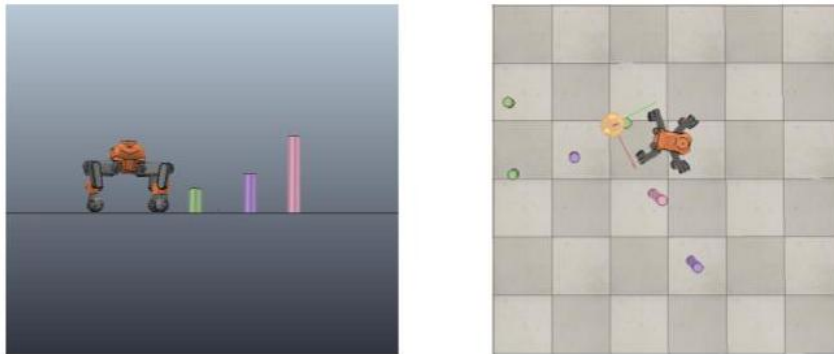


Figure 38: The left image shows the obstacles with three different height and the right image shows an example of the training scene. The yellow point on the right image represents the target position.

we add a random shift to the robot pose when restoring a failed episode.

Rewards We construct a reward function to guide the robot to reach to the target and avoid collision. The reward consists of three components: a time penalty for each step r_{step} , a motion cost r_{cost} and a collision penalty $r_{collide}$.

$$reward = r_{step} \times w_{obs_dist} + r_{cost} + r_{collide} \quad (8)$$

where w_{obs_dist} is a scale factor computed using the minimum distance between the obstacle and the robot body. w_{obs_dist} equals to 1 when the distance between the robot and the obstacle is greater than a threshold and w_{obs_dist} is greater than 1 when the distance is smaller. The penalty is larger when the robot moves close to obstacles. r_{cost} represents the cost of the action performed and $r_{collide}$ is the penalty when robot collides with an obstacle. When the target is reached, the agent will get a positive reward r_{target} and the episode is terminated.

Observation The agent receives three types of observations: (1) a height map with a radius of 1 meter around the robot body center, (2) the current robot leg open width and body height and (3) the distance and angle of the target to the robot. All observations are with respect to the robot frame.

Policy parameterization Similar to [12] we separate the policy network into two subnetworks. One subnetwork extracts features from the local height map and the other subnetwork processes the current robot pose and the target information. We concatenate the output of the two subnetworks and map it to the action and value using a fully connected network. The structure of the network is shown in Figure 39. In our experimental setup, the height map has resolution of $0.1m$ and each obstacle is represented by a single pixel. We found that the fully-connected layer performs better than convolutional-layer to process the height map.

A.4.4 Preliminary Result

Learning curves For every iteration, we plot the average value of the episode length, discounted return and the success rate to trace the learning progress. For every fifth iteration, we evaluate the model in the environment used for regular planning test as described below. The

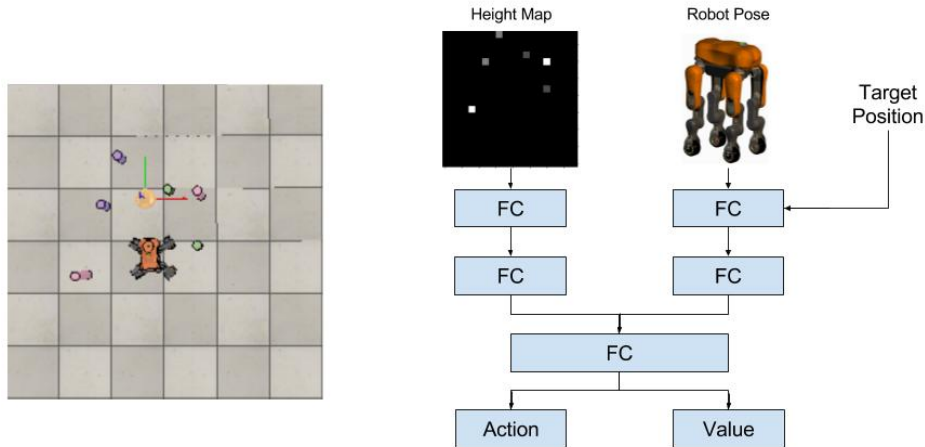


Figure 39: The left image shows an example of the training scene. The right image shows the network structure. The height map is generated based on the configuration of the left image.

learning curves are shown in Figure 40, 41, 42 and 43. We use DPPO with eight workers to collect the training samples and we use the batch size of 2048×8 to update the model.

From Figure 42 we can see that the success rate starts to increase after a few iterations and finally stabilized at around 80%. When we train the model, we assign a probability of 80% to restore an episode form the failed episode buffer. As mentioned above, the difficulties of the training samples evolve with the performance of the model, which makes it harder to visualize the improvement from the training data. However, from Figure 43 we can see that the success rate on the test environment increased continually.

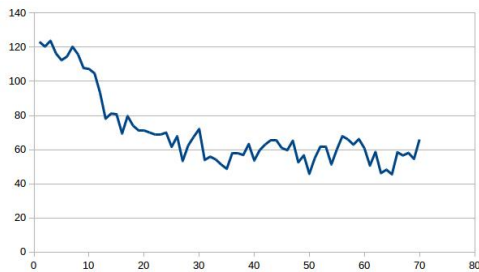


Figure 40: Learning curve of the average episode length for every iteration.

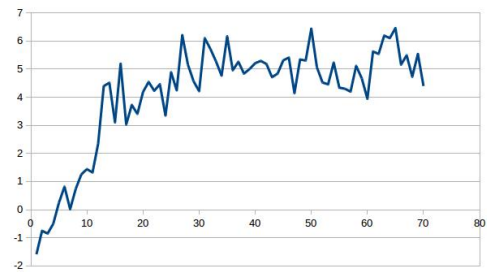


Figure 41: Learning curve of the average episode return for every iteration.

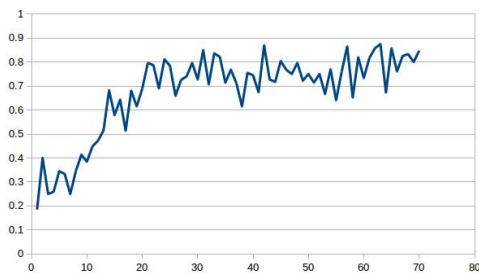


Figure 42: Learning curve of the success rate on training samples for every iteration.

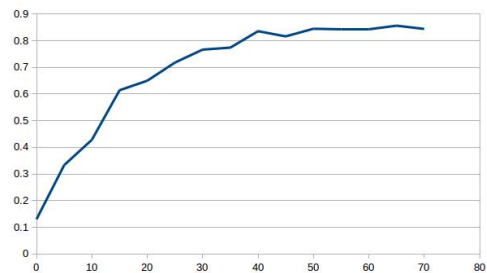


Figure 43: Learning curve of the success rate on testing samples for every iteration.

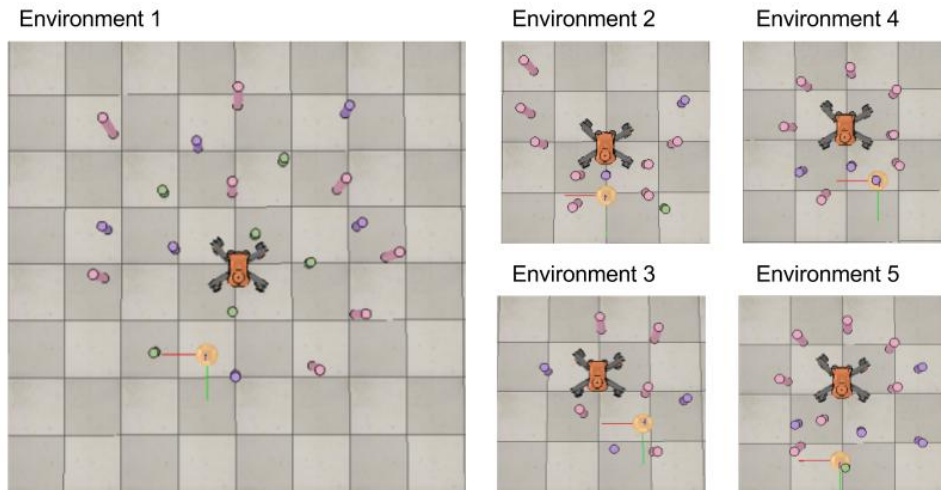


Figure 44: The five environment configurations used for testing. The environment 1 is used for regular planning test and the environment 2, 3, 4, 5 are used for narrow passage test. The target position of the environment 4 and 5 are above the obstacles.

Testing Results To evaluate the current performance of our model we designed two testing scenarios: a narrow passage test and a regular planning test. We prepared four environments for the narrow passage test and one environment for the regular planning test. The five environments are shown in Figure 44.

In the narrow passage test, the target position is $0.5m$ from the robot and space to reach to the goal is narrow. The robot has to lift or shrink its body to pass the obstacles. The narrow passage test is used to investigate whether the planner learned how to adjust the robot body posture based on the observation. In every iteration, the robot orientation is randomly initialized. The regular planning test is used to evaluate the overall planning performance. The testing environment is similar to the training environment. It contains a flat ground of size $4 \times 4m^2$ and 20 obstacles with fixed positions. In every iteration the robot position is randomly generated and the target position is $1m$ away from the robot. To be considered successful, the planner should navigate the robot to the target position within 100 steps.

The regular planning test environment has some overlaps with our training data, however, the narrow passage test environments are not in the training data. As we use only three tall obstacles to train the model, there is no case like Environment 4, with seven tall obstacles at the same time.

We tested 100 times for every environment and the average episode length and the success rate is shown in Figure 45 and Figure 46. The robot accomplish most of the regular planning test in Environment 1 with different initial and target settings. The results of the narrow passage test in Environment 2 and 3 are successful, however, the Environment 4 and 5 are still challenging as the robot needs to pass more than one obstacle to reach to the goal. From the result we show that the model can successfully adapt to similar environment without additional training and has the potential to generalize to unseen environments. However, the length of the episode is still far from the optimal. Further improvements are needed to reduce the number of actions that may cause collisions to the obstacles, and to increase the capability of generalization to new environment.

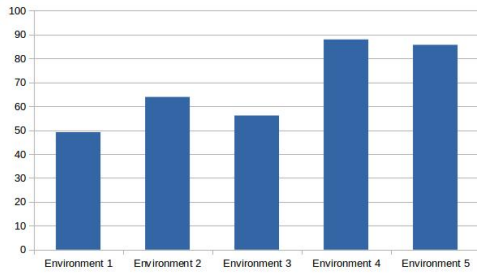


Figure 45: The average episode length in testing environments

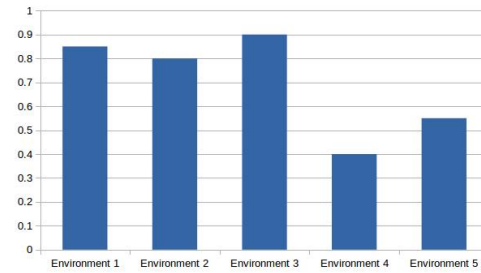


Figure 46: The success rate in testing environments

A.4.5 Connection to CENTAURO

The method mentioned above focus on the task of driving navigation with reconfigurable leg postures. The method can be used as an alternative planner to suggest additional path to the operator from the learning perspective. Future development will improve the path quality and extend its capability from the simulated environment to the real world task.

References

- [1] Sven Behnke. Local multiresolution path planning. In *Robot Soccer World Cup*, pages 332–343. Springer, 2003.
- [2] Robert Bohlin. Path planning in practice; lazy evaluation on a multi-resolution grid. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 1, pages 49–54. IEEE, 2001.
- [3] Michael Brunner, Bernd Brüggemann, and Dirk 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.
- [4] X. Chen, F. Schilling, T. Klamt, and P. Jensfelt. Deliverable D5.2 CENTAURO Terrain Classification.
- [5] Francis Colas, Srivatsa Mahesh, François Pomerleau, Ming Liu, and Roland Siegwart. 3D path planning and execution for search and rescue ground robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.
- [6] Brian P Gerkey and Kurt Konolige. Planning and control in unstructured terrain. In *ICRA Workshop on Path Planning on Costmaps*, 2008.
- [7] Kalin Gochev, Benjamin Cohen, Jonathan Butzke, Alla Safonova, and Maxim Likhachev. Path planning with adaptive dimensionality. In *Fourth annual symposium on combinatorial search*, 2011.
- [8] Adrián González-Sieira, Manuel Mucientes, and Alberto Bugarín. An adaptive multi-resolution state lattice approach for motion planning with uncertainty. In *Robot 2015: Second Iberian Robotics Conference*, pages 257–268. Springer, 2016.
- [9] Timo Hackel, Nikolay Savinov, Lubor Ladicky, Jan D Wegner, Konrad Schindler, and Marc Pollefeys. Semantic3d. net: A new large-scale point cloud classification benchmark. *arXiv preprint arXiv:1704.03847*, 2017.
- [10] Timo Hackel, Jan D Wegner, and Konrad Schindler. Fast semantic segmentation of 3d point clouds with strongly varying density. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Prague, Czech Republic*, 3:177–184, 2016.
- [11] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.
- [12] Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.
- [13] Karl Holmquist, Deniz Senel, and Michael Felsberg. Computing a collision-free path using harmonic potentials. In *submitted to ICRA 2018*, 2018.
- [14] Thomas M Howard and Alonzo Kelly. Optimal rough terrain trajectory generation for wheeled mobile robots. *The International Journal of Robotics Research*, 26(2):141–166, 2007.

- [15] P. Jensfelt, J. Folkesson, G. Meneghetti, M. Felsberg, and S. Behnke. Deliverable D5.1 CENTAURO Navigation Concept.
- [16] Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, Michael Mistry, and Stefan Schaal. Learning, planning, and control for quadruped locomotion over challenging terrain. *The International Journal of Robotics Research*, 30(2):236–258, 2011.
- [17] Tobias Klamt and Sven Behnke. Anytime hybrid driving-stepping locomotion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.
- [18] C Kohrt, Anthony G Pipe, Janice Kiely, R Stamp, and G Schiedermeier. A cell based voronoi roadmap for motion planning of articulated robots using movement primitives. In *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*, pages 1542–1549. IEEE, 2012.
- [19] J. Zico Kolter, Mike P. Rodgers, and Andrew Y. Ng. A control architecture for quadruped locomotion over rough terrain. In *Proceedings - IEEE International Conference on Robotics and Automation*, 2008.
- [20] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.
- [21] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.
- [22] Felix Järemo Lawin, Martin Danelljan, Patrik Tosteberg, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. Deep projective 3d semantic segmentation, 2017.
- [23] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *NIPS*, 2003.
- [24] Carlos Mastalli, Ioannis Havoutis, Alexander W. Winkler, Darwin G. Caldwell, and Claudio Semini. On-line and on-board planning and perception for quadrupedal locomotion. In *IEEE Conference on Technologies for Practical Robot Applications, TePRA*, 2015.
- [25] Matteo Menna, Mario Gianni, Federico Ferri, and Fiora Pirri. Real-time autonomous 3D navigation for tracked vehicles in rescue environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014.
- [26] Javier A Montoya-Zegarra, Jan D Wegner, L’ubor Ladický, and Konrad Schindler. Mind the gap: modeling local and global context in (road) networks. In *German Conference on Pattern Recognition*, pages 212–223. Springer, 2014.
- [27] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L Cun. Off-road obstacle avoidance through end-to-end learning. In *Advances in neural information processing systems*, pages 739–746, 2006.
- [28] Xue Bin Peng, Glen Berseth, and Michiel Van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35(4):81, 2016.
- [29] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):41, 2017.

- [30] Nicolas Perrin, Christian Ott, Johannes Engelsberger, Olivier Stasse, Florent Lamiroux, and Darwin G Caldwell. Continuous legged locomotion planning. *IEEE Transactions on Robotics*, 33(1):234–239, 2016.
- [31] Janko Petereit, Thomas Emter, and Christian W Frey. Mobile robot motion planning in multi-resolution lattices with hybrid dimensionality. *IFAC Proceedings Volumes*, 46(10):158–163, 2013.
- [32] Mark Pfeiffer, Michael Schaeuble, Juan Nieto, Roland Siegwart, and Cesar Cadena. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1527–1533. IEEE, 2017.
- [33] William Reid, Robert Fitch, Ali Haydar Göktoan, and Salah Sukkarieh. Motion Planning for Reconfigurable Mobile Robots Using Hierarchical Fast Marching Trees.
- [34] Eric Rohmer, Surya PN Singh, and Marc Freese. V-rep: A versatile and scalable robot simulation framework. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1321–1326. IEEE, 2013.
- [35] Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadepta Dey, J Andrew Bagnell, and Martial Hebert. Learning monocular reactive uav control in cluttered natural environments. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1765–1772. IEEE, 2013.
- [36] Fabian Schilling, Xi Chen, John Folkesson, and Patric Jensfeld. Geometric and visual terrain classification for autonomous mobile navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.
- [37] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [38] James Sergeant, Niko Sünderhauf, Michael Milford, and Ben Upcroft. Multimodal deep autoencoders for control of a mobile robot. In *Australasian Conference for Robotics and Automation (ARAS)*, 2015.
- [39] Lei Tai, Giuseppe Paolo, and Ming Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. *arXiv preprint arXiv:1703.00420*, 2017.
- [40] Jiexiong Tang, John Folkesson, and Patric Jensfeld. Geometric correspondence network for motion estimation. In *submitted to ICRA 2018*, 2018.
- [41] Paul Vernaza, Maxim Likhachev, Subhrajit Bhattacharya, Sachin Chitta, Aleksandr Kushleyev, and Daniel D. Lee. Search-based planning for a legged robot over rough terrain. In *Proceedings - IEEE International Conference on Robotics and Automation*, 2009.
- [42] Martin Wermelinger, Péter Fankhauser, Remo Diethelm, Philipp Krüsi, Roland Siegwart, and Marco Hutter. Navigation planning for legged robots in challenging terrain. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

- [43] Martin Wermelinger, Peter Fankhauser, Remo Diethelm, Philipp Krüsi, Roland Siegwart, and Marco Hutter. Navigation Planning for Legged Robots in Challenging Terrain.
- [44] Alexander W Winkler, Carlos Mastalli, Ioannis Havoutis, Michele Focchi, Darwin G Caldwell, and Claudio Semini. Planning and Execution of Dynamic Whole-Body Locomotion for a Hydraulic Quadruped on Challenging Terrain.
- [45] David Wooden, Matthew Malchano, Kevin Blankespoor, Andrew Howardy, Alfred A. Rizzi, and Marc Raibert. Autonomous navigation for BigDog. In *Proceedings - IEEE International Conference on Robotics and Automation*, 2010.
- [46] Haojie Zhang, Jonathan Butzke, and Maxim Likhachev. Combining global and local planning with guarantees on completeness. In *IEEE/RSJ International Conference on Robotics and Automation (ICRA)*, 2012.
- [47] Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard. Deep reinforcement learning with successor features for navigation across similar environments. *arXiv preprint arXiv:1612.05533*, 2016.
- [48] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3357–3364. IEEE, 2017.
- [49] Zahra Ziaei, Reza Oftadeh, and Jouni Mattila. Global path planning with obstacle avoidance for omnidirectional mobile robot using overhead camera. In *IEEE International Conference on Mechatronics and Automation*, 2014.

Anytime Hybrid Driving-Stepping Locomotion Planning

Tobias Klamt and Sven Behnke

Abstract— Hybrid driving-stepping locomotion is an effective approach for navigating in a variety of environments. Long, sufficiently even distances can be quickly covered by driving while obstacles can be overcome by stepping. Our quadruped robot Momaro, with steerable pairs of wheels located at the end of each of its compliant legs, allows such locomotion. Planning respective paths attracted only little attention so far.

We propose a navigation planning method which generates hybrid locomotion paths. The planner chooses driving mode whenever possible and takes into account the detailed robot footprint. If steps are required, the planner includes those. To accelerate planning, steps are planned first as abstract manoeuvres and are expanded afterwards into detailed motion sequences. Our method ensures at all times that the robot stays stable. Experiments show that the proposed planner is capable of providing paths in feasible time, even for challenging terrain.

I. INTRODUCTION

Hybrid driving-stepping locomotion enables robots to traverse a wide variety of terrain types. Application domains, such as search and rescue and delivery services, pose considerable navigation challenges for robots due to non-flat grounds. Sufficiently flat terrain can be traversed by driving, which is fast, efficient and safe, regarding the robot stability. However, driving traversability is limited to moderate slopes and height differences and obstacle-free paths. Stepping locomotion requires only adequate footholds and, hence, enables mobility in cases where driving is unfeasible. But stepping is also slower and decreases the robot stability.

Most mobile ground robots use either driving locomotion or stepping locomotion, and there exist path planning methods for both such locomotion modes independently [1]–[7]. Our mobile manipulation robot Momaro [8] (see Fig. 1), however, supports both locomotion types due to its four legs ending in steerable pairs of wheels. This unique design allows omnidirectional driving on sufficiently flat terrain and stepping to overcome obstacles. In contrast to purely walking robots, Momaro is able to change its configuration of ground contact points (which we will refer to as its footprint) under load without lifting a foot. This enables motion sequences for stepping that have large stability margins. Multiple platforms that are capable of driving-stepping locomotion have been developed [9]–[13], but planning which combines the advantages of both locomotion types was addressed for none of these.

All authors are with Rheinische Friedrich-Wilhelms-Universität Bonn, Computer Science Institute VI, Autonomous Intelligent Systems, Friedrich-Ebert-Allee 144, 53113 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. Our hybrid wheeled-legged mobile manipulation robot Momaro is capable of omnidirectional driving (left) and stepping (right).

In our previous work with Momaro [14], we demonstrated semi-autonomous driving, 2D (x, y) path planning and execution in a Mars-like environment accompanied by manipulation tasks. In this work, we extend the driving path planning method to incorporate the robot orientation θ and its detailed footprint in order to increase driving flexibility. We improve the path quality by introducing an orientation cost term.

In addition, we demonstrated stepping over a wooden bar obstacle, climbing stairs, and egressing a car with Momaro at the DARPA Robotics Challenge (DRC) [15]. All of the DRC tasks were performed via teleoperation based on pre-defined motion sequences. Teleoperation depends on a good data connection between the operator station and robot and generates a high cognitive load for the operators. Autonomous locomotion is desirable to relieve the operators and to increase speed and safety.

We extend the locomotion planner to generate stepping motions. Driving in difficult terrain and stepping require a high planning resolution which increases planning times. To keep the search space feasible, we first generate abstract steps that we later expand to detailed motion sequences.

To summarize, the main contributions in this paper are:

- a three-dimensional (x, y, θ) driving path planning method allowing driving in constrained uneven environments by consideration of the detailed robot footprint,
- the introduction of orientation costs, favoring a preferred driving direction to align the robot with the path,
- a hierarchical step planner, which generates detailed manoeuvres to perform individual steps under the constraint to always keep the robot statically stable, and
- application of anytime planning to quickly find paths with bounded suboptimality.

We demonstrate our approach in simulation and with the real robot and systematically evaluate the effect of our acceleration methods. The results indicate that our planner provides paths in feasible time even for challenging tasks.

II. RELATED WORK

Path planning in unstructured terrain has been addressed by many works. The considered systems provide either purely wheeled/tracked locomotion or are able to traverse terrain by walking. Planning is often done with either grid-based searches, such as A* [16], or sampling-based approaches, such as RRT [17]. Despite the application of similar planning methods, these two locomotion modes differ in many aspects.

Driving is fast and energy efficient on sufficiently flat terrain, which makes it suitable for traversing longer distances. When supported by three or more wheels, the robot is generally statically stable. Planning of drivable paths in unstructured environments is heavily dependent on the degrees of freedom (DoF) of the platform. Simple robot designs offer longitudinal and rotational movements with a constant robot shape [18], [19]. For search and rescue scenarios, some robots were extended by tracked flippers [1], [2], [3]. These allow the robots to climb stairs and thus increase capabilities but also planning complexity due to additional shape shifting DoFs. Flipper positions are often not considered by the initial navigation path planning and are adjusted to the terrain in a second planning step. Platforms which offer omnidirectional locomotion increase the path planning search space by another dimension [4]. Driving is restricted, however, by terrain characteristics such as height differences and slopes which makes it not suitable for very rough terrain and for overcoming obstacles.

Legged locomotion is capable of traversing more difficult terrain since it only requires isolated feasible footholds. The drawback of this locomotion mode is that motion planning is much more complex. Since legs are lifted from the ground repeatedly, the robot also has to constantly ensure that it remains stable. Due to the high motion complexity of stepping, path planning is often performed in at least two hierarchical levels [5], [6], [7]. A coarse planning algorithm identifies feasible footholds or areas for feasible steps. Detailed motion planning is done in a second step to connect these footholds. Navigation towards the goal is either included in the coarse planning or realized in a higher-level planner.

Since both locomotion modes have complementary advantages, it is promising to combine those. Halme et al. [9] and Takahaashi et al. [10] developed quadruped robots with legs ending in wheels. Control mechanisms to overcome obstacles are presented, but locomotion planning is not addressed. The hybrid locomotion robots HUBO [11] and CHIMP [12] were used by the winning and the third best teams at the DARPA Robotics Challenge. HUBO provides legged and wheeled locomotion, but needs to shift its shape to switch between those. Thus, hybrid locomotion, which combines advantages of both locomotion types, is not possible. CHIMP provides bipedal and quadruped hybrid locomotion. However, hybrid locomotion planning is not presented. Finally, a bipedal robot, capable of driving and walking, and a respective planning algorithm is presented by Hashimoto et al. [13]. Depending on the terrain, it either chooses walking or driving

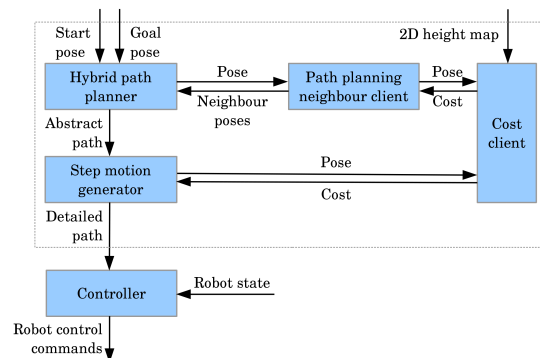


Fig. 2. Hybrid locomotion system structure: The hybrid path planner searches an abstract path from start to goal pose. The step motion generator expands this abstract path to a detailed path which can be executed by the robot. A neighbour client provides neighbour states to the planner. Both, this neighbour client and the step motion generator, request pose costs from the cost client which generates costs out of the 2D height map. The resulting path is executed by a controller.

mode. A combination of both, which might bring further advantages, is not considered. Recently, Boston Dynamics introduced its biped platform Handle¹ with legs ending in wheels. It demonstrated manoeuvres which require very good dynamic control but path planning was not presented.

Our approach combines both locomotion modes in a single planning algorithm and thus has many benefits of both.

III. HARDWARE

We use our quadruped robot platform Momaro [8] (see Fig. 1) with articulated legs ending in directly-driven 360° steerable pairs of wheels. Those offer omnidirectional driving and the possibility to change the robot footprint under load which neither can be done by pure driving nor by pure walking robots and enables novel movement strategies.

Each leg consists of three pitch joints which allow leg movements in the sagittal plane. Lateral leg movements are possible only passively. Legs show compliant behaviour due to their elastic carbon composite links, which work as a passive suspension system on rough terrain. Moreover, soft foam-filled wheels compensate small terrain irregularities.

A continuously rotating Velodyne Puck 3D laser scanner with spherical field-of-view at the robot head and an IMU provide measurements for terrain perception.

IV. ENVIRONMENT REPRESENTATION

An overview of the planning system structure is given in Fig. 2. Range measurements from the 3D laser scanner are used for mapping and localization by utilizing a multiresolution surfel map [20]. Input to the planner are a 2D height map and the start and goal robot poses. A robot pose $\vec{r} = (r_x, r_y, r_\theta, \vec{f}_1, \dots, \vec{f}_4)$ includes the robot base position r_x, r_y and orientation r_θ and the foot positions $\vec{f}_j = (f_{j,x}, f_{j,y})$ in map coordinates. The grid resolution is set to 2.5 cm with 64 possible orientations at each position.

¹<https://youtu.be/-7xvqQeoA8c>

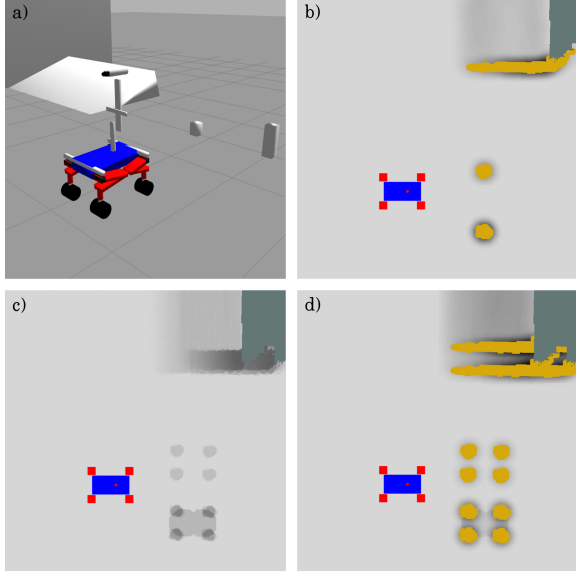


Fig. 3. Driving cost computation: a) Simulated scenario in which the robot stands in front of a ramp, a small and a tall pole. b) Foot costs. Yellow areas are not traversable by driving, olive areas are unknown. c) Body costs. The robot can take the small pole between its legs while the tall pole generates costs for lifting the robot body. Costs are shown for the current robot orientation. d) Pose costs combine body costs and foot costs at their respective positions.

The cost client computes pose cost values from the height map for a robot pose as follows: From the height map, local unsigned height differences ΔH are computed to generate the foot specific cost

$$C_F(c_j) = 1 + k_1 \cdot \sum_{c_i \in \text{map}} \Delta H(c_i) \cdot w(c_i) \quad (1)$$

where $k_1 = 100$ and

$$w(c_i) = \begin{cases} \infty & \text{if } \|c_i - c_j\| < r_F \wedge \Delta H(c_i) > 0.05, \\ 1 - \frac{\|c_i - c_j\|}{r_N} & \text{if } \|c_i - c_j\| < r_N, \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

for a map cell c_j in which the foot \vec{f}_j is located. Foot costs are assigned an infinite value if untraversable height differences > 0.05 m occur in a surrounding of the size of a foot ($r_F = 0.12$ m). In a neighbourhood of greater size ($r_N = 0.3$ m), height differences are accumulated weighted by their distance to c_j . Foot costs are defined to be 1 in flat surroundings and increase if challenging terrain occurs. C_F includes traversability information and describes the surrounding of each foot position (see Fig. 3).

The robot shape allows obstacles to pass between the robot legs. However, if obstacles are too high they might collide with the robot base. The base cost

$$C_B(\vec{r}) = 1 + k_2 \cdot \max(H_{\max, \text{uB}} - H_B, 0) + k_3 \cdot \Delta H_{\max, \text{F}}, \quad (3)$$

where $k_2 = 1$ and $k_3 = 0.5$ compares the maximum terrain height under the robot body $H_{\max, \text{uB}}$ with the body height H_B and assigns additional costs if the space is not sufficient. In addition, the height difference between the lowest and

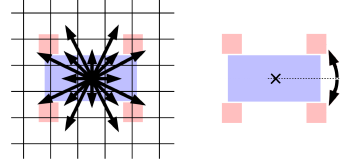


Fig. 4. Driving locomotion neighbour states can either be found by straight moves with fixed orientation within a 16-neighbourhood (l.) or by orientation changes on a fixed position (r.). Grid and orientation resolution are enlarged for better visualization.

highest foot $\Delta H_{\max, \text{F}}$ generates costs since this is a measure for the terrain slope under the robot. Again, the basic cost is 1 which increases for challenging terrain. The robot base is estimated by two circles of 0.25 m radius to avoid expensive detailed collision checking.

All cost values are combined into the pose cost

$$C(\vec{r}) = k_4 \cdot \max_j(C_F(\vec{f}_j)) + k_5 \cdot \sum_{j=1}^4 C_F(\vec{f}_j) + k_6 \cdot C_B(\vec{r}), \quad (4)$$

where $k_4 = 0.1$, $k_5 = 0.1$ and $k_6 = 0.5$. Pose costs are defined to be 1 on flat terrain where both, foot and body costs, induce 50% of the pose costs. We want to consider the terrain under all four wheels but want to prefer a pose with four slightly challenging contact points over a pose with three non-challenging and one very difficult contact point. Hence, it is neither sufficient to sum up all individual foot costs nor to just take the maximum. A weighted sum of both, however, achieves the desired functionality.

V. PATH PLANNING

Path planning is done in a hybrid planner, which prefers the driving mode and considers steps only if necessary. It is realized through an A*-search on a pose grid. The used heuristic combines the Euclidean distance with orientation differences. For each pose, the path planning neighbour client provides feasible neighbouring poses (see Fig. 2). Driving neighbours can be found within a 16-neighbourhood and by turning on the spot to the next discrete orientation (Fig. 4).

As illustrated in Fig. 5, additional stepping manoeuvres are added, if a foot \vec{f}_j is

- close to an obstacle $\left(\exists c \in \text{map} \left(C_F(c) = \infty \wedge \|c - \vec{f}_j\| < 0.1 \text{ m} \right) \right)$,
- a feasible foothold c_h with $C_F(c_h) < \infty$ can be found in front of the foot in its sagittal plane that respects a maximum leg length,
- the height difference to the foothold is small $(|H(\vec{f}_j) - H(c_h)| \leq 0.3 \text{ m})$, and
- the distance between the two feet on the “non-stepping” robot side is > 0.5 m to guarantee a safe stand while stepping.

A step is represented as an additional possible neighbouring pose for the planner.

The step which is considered by the planner at this level is an abstract step. We define an abstract step to be the direct transition from a pre-step pose to an after-step pose. It does

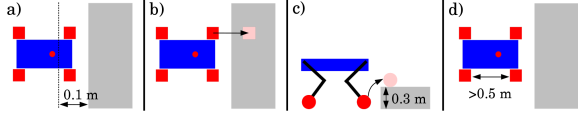


Fig. 5. Criteria to add steps to the hybrid planner: a) A foot is close to an obstacle, b) a feasible foothold can be found, c) the height difference to overcome is ≤ 0.3 m and d) the distance between the feet on the “non-stepping” robot side is > 0.5 m. Grey areas show an elevated platform.

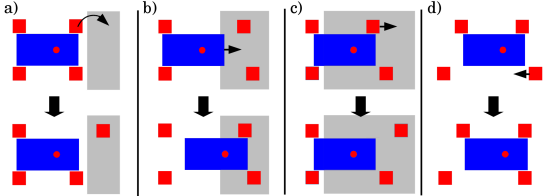


Fig. 6. Manoeuvres which extend the driving planner to a hybrid locomotion planner, visualized on a height map: a) Abstract steps, b) longitudinal base shifts, c) driving a pair of wheels at one front foot forward, and d) driving a pair of wheels at any foot back to its neutral position.

not describe the motion sequence to perform the step and needs to be expanded to be stable and executable by the robot. An abstract step is visualized in Fig. 6a.

Each step is assigned a cost value

$$C_S = k_7 \cdot L_{\text{step}} + k_8 \cdot (C_F(c_h) - 1) + k_9 \cdot \Delta H_{\text{step}}, \quad (5)$$

where $k_7 = 0.5$, $k_8 = 0.1$ and $k_9 = 2.3$ which includes the step length L_{step} , the foot specific terrain difficulty costs of the foothold to step in c_h , and the maximum height difference ΔH_{step} . If multiple end positions for a step exist, only the cheapest solution is returned to the search algorithm.

Further manoeuvres are required to navigate in cluttered environments. We define the footprint shown in Fig. 5a to be the neutral footprint. It provides high robot stability at a small footprint size, and is the preferred configuration for driving.

If both front feet are positioned in front of their neutral position, the robot may perform a longitudinal base shift manoeuvre. The base is shifted forward relatively to the feet until one of the front feet reaches its neutral position or a maximum leg length is reached for one of the rear legs (see Fig. 6b). Base shifts of length L_{BS} using the average discovered base costs $C_{\text{B,avg}}$ and $k_{10} = 0.5$ carry the cost

$$C_{\text{BS}} = k_{10} \cdot L_{\text{BS}} \cdot C_{\text{B,avg}}. \quad (6)$$

If a rear foot is close to an obstacle, the pair of wheels at each front foot may be driven forward while keeping ground contact (Fig. 6c) which is a preparation for a rear foot step. If the robot footprint is not neutral, it may drive a single pair of wheels to their neutral position (Fig. 6d). A single foot movement of length L_{FM} using the average discovered foot costs $C_{\text{F,avg}}$ and $k_{11} = 0.125$ carries the cost

$$C_{\text{FM}} = k_{11} \cdot L_{\text{FM}} \cdot C_{\text{F,avg}}. \quad (7)$$

Since driving is faster and safer than stepping, we want the planner to consider drivable detours of acceptable length

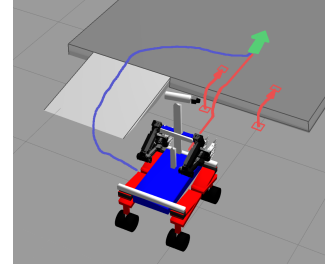


Fig. 7. Step-related manoeuvre costs are weighted such that the planner just prefers taking a 1.5 m detour over a ramp (blue path) instead of stepping up (red path) to an elevated platform.

instead of including steps in the plan. We define that, when the robot stands in front of an 0.2 m elevated platform, it should prefer a 1.5 m long detour over a ramp instead of stepping up to this platform (Fig. 7). This can be achieved by increasing the costs of stepping-related manoeuvres by a certain factor.

VI. STEP MOTION GENERATION

The result of the A* search is a cost-optimal abstract path which lacks executable motion sequences for steps and information about foot heights. The resulting path is expanded during step motion generation. It finds stable robot positions for steps and adds leg height information to the path. Again, costs are obtained from the cost client.

A. Robot Stabilization

Abstract steps only describe the start and goal poses for a stepping manoeuvre. An executable transition between these poses is a motion sequence which keeps the robot stable at all times. Such a motion sequence is generated for each abstract step in the path. Due to the compliant leg design of our robot, we have no information about the exact position of the feet but have to estimate it from actuator feedback. Hence, we limit stability considerations to static stability. Since actuator speeds are slow, dynamic effects can be neglected. Stability estimation while stepping is done on the support triangle which is spanned by the horizontal position of the remaining three feet with ground contact (Fig. 8). If the horizontal robot center of mass (CoM) projection is inside the support triangle, the robot pose is stable. The closer the CoM is to the support triangle centroid (STC), the greater the stability.

Lateral alignment of the CoM and STC is done by base roll motions (Fig. 8), which are rotations around the longitudinal axis. These are achieved by changing the leg lengths on one side of the robot. The resulting angle between the wheel axes and ground is compensated by the compliant legs and the soft-foam filled wheels. Roll manoeuvre parametrization is described in Sec. VI-B.

Longitudinal alignment of the CoM and STC is done by driving the remaining pair of wheels on the stepping side (e.g., the rear left foot if the front left foot is stepping) towards the robot center (Fig. 8). If this does not suffice because the motion is hindered by obstacles, the remaining longitudinal alignment is done by shifting the

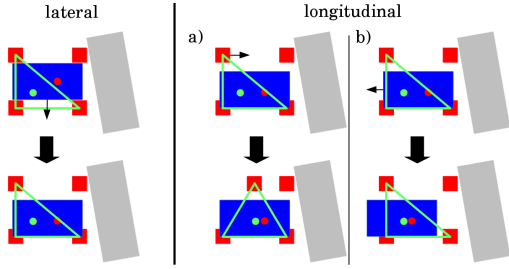


Fig. 8. To find a stable position for stepping, the robot CoM (red dot) needs to be aligned with the STC (green dot). Lateral alignment is done by base rolling. Longitudinal alignment is either done by rolling the remaining foot on the stepping side towards the center (a) or by shifting the robot base (b).

robot base. The longitudinal CoM position is also affected by the robot base pitch angle which is described in the next subsection. The presented motions generate a stable robot configuration which allows the desired step to be performed. After stepping, the robot reverses its base roll, foot displacement, and base shift manoeuvres to get back to its nominal configuration.

B. Leg Heights

Each robot pose is assigned an individual height for each leg, which describes the vertical position of a foot relative to the robot base. When driving with neutral footprint, a low leg height of 0.27 m is chosen, which provides a low CoM and good controllability through reduced leg compliance (see Fig. 1 left). A larger ground clearance is chosen for manoeuvres other than driving to provide great freedom for leg movements (see Fig. 1 right). In this case, the base height is determined by the highest foot. A soft constraint is applied that the leg height should be at least 0.45 m. At the same time, a hard constraint from the mechanical system is that none of the legs exceeds its maximum leg length. The height of each individual foot can be read from the 2D height map. The robot base pitching angle is set to be 70% of the ground slope, as can be seen in Fig. 19. This pitching value provides a good compromise between sufficient ground clearance for all four legs and a good CoM position.

As described before, base roll manoeuvres are used to shift the robot CoM laterally. Due to the soft-foam filled wheels, we can estimate the center of rotation $R(y'_{rot}, z'_{rot})$ between the two wheels (Fig. 9). In addition, the center of mass position $C(y'_{CoM}, z'_{CoM})$, the angle

$$\alpha = \arctan\left(\frac{y'_{rot} - y'_{CoM}}{z'_{CoM} - z'_{rot}}\right) \quad (8)$$

between \vec{RC} and the vertical axis and the desired lateral center of mass position $y'_{CoM,des}$ are given. Using $\|\vec{RC}\| = \|\vec{RC}_{des}\|$ we compute the desired angle between \vec{RC}_{des} and the vertical axis

$$\alpha_{des} = \arcsin\left(\frac{y'_{rot} - y'_{CoM,des}}{\|\vec{RC}\|}\right) \quad (9)$$

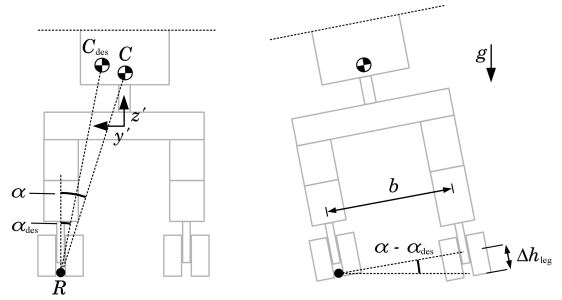


Fig. 9. Momaro's lower body in back view. Lateral CoM shifts can be achieved by changing leg length on one side which rolls the robot.

and consequently using the footprint width b we compute the desired leg height difference

$$\Delta h_{leg} = b \cdot \tan(\alpha - \alpha_{des}). \quad (10)$$

This leg height difference is added to both legs on the respective side to induce a base roll manoeuvre. Fig. 10 shows how Momaro steps up an elevated platform, using the described motion sequences.

VII. PATH PLANNING EXTENSIONS

Due to the fine position and orientation resolution, the search space which is considered for path planning is large. Moreover, we want the planner to consider several detours before taking a step, which further increases planning times. We present methods to accelerate planning and to improve the resulting path quality. Their individual effects are investigated during evaluation.

A. Robot Orientation Cost

Although our robot is capable of omnidirectional driving, there are multiple reasons to prefer driving forward. Since the sensor setup is not only used for navigation, but also for manipulation, it is designed to provide best measurement results for the area in front of the robot. The required width clearance is minimal when driving in a longitudinal direction, which is helpful in narrow sections such as doors. The same applies to driving backwards. Driving straight backwards requires a smaller clearance than driving diagonal backwards. Finally, our leg design restricts us to perform steps in the longitudinal direction. Thus, when approaching areas where stepping is required, a suitable orientation is helpful. We address this desire of preferring special orientations by multiplying neighbour costs during A* search by the individual factor $k_{\Delta\theta}$, as described in Fig. 11.

B. ARA*

To obtain feasible paths quickly, we extend the A* algorithm to an Anytime Repairing A* (ARA*) [21]. Its initial search provides solutions with bounded suboptimality by giving the heuristic a weight > 1 . The result quality is then improved by decreasing the heuristic weight stepwise down to 1, if the given planning time is not exhausted yet. ARA* recycles the representations it generated previously to accelerate replanning.

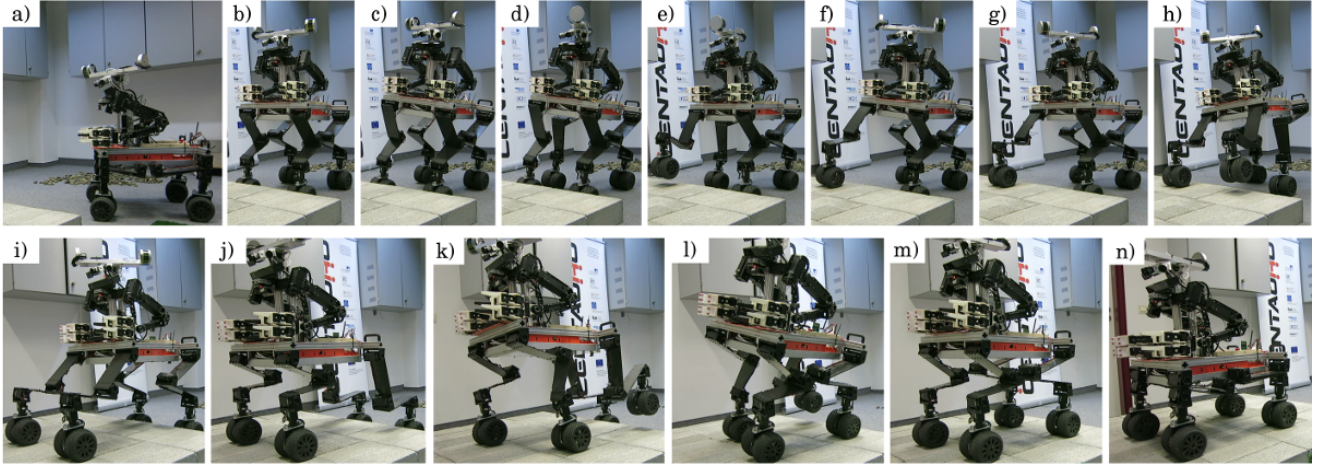


Fig. 10. Momaro stepping up an elevated platform. a) It arrives at the platform in low driving position, b) stands up, c) rolls its base to the left to shift its CoM laterally, d) drives its rear right pair of wheels forward to reach a stable stepping configuration. e) It then steps with its front right foot, f) drives its rear right pair of wheels back and g) rolls back its base to reach the configuration it had before the step. The remaining steps follow a similar motion sequence which is shown in less details. Subsequently, h) Momaro steps with its front left foot. i) It then drives forward and j) shifts its base forward before k) doing a step with the rear left and l) rear right foot. m) When the robot stands on the platform, n) it lowers its base to continue driving.

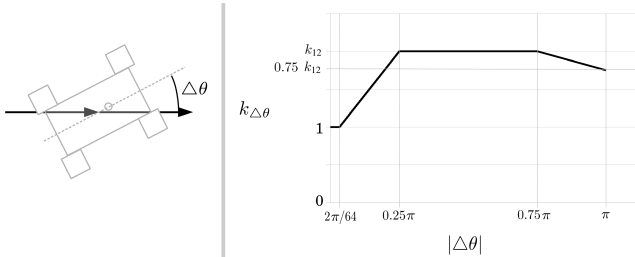


Fig. 11. For a difference between robot orientation and driving direction $\Delta\theta$ the cost factor $k_{\Delta\theta}$ is computed which expresses the desire to drive forward. It is 1 within an orientation step of $2\pi/60$ and increases up to k_{12} . When driving backward, there is a desire to drive straight backwards since the required clearance is smaller.

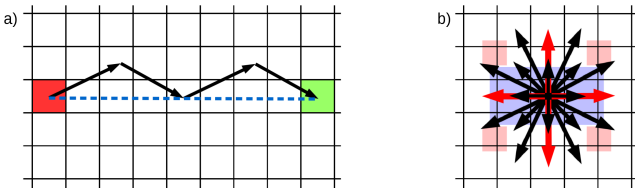


Fig. 12. Addressing ARA* preferences of long moves. a) For larger heuristic weights, the ARA* algorithm prefers those driving manoeuvres which bring the robot as close to the goal as possible which leads to undesired paths (black arrows). b) To obtain the desired behaviour (blue line) we extend the driving neighbourhood by the four red manoeuvres.

When planning with higher weighted heuristics, the planner prefers those driving manoeuvres that bring it as close to the goal as possible. This leads to the undesired effect that resulting paths mainly consist of those driving manoeuvres which go two cells in one direction and one cell in an orthogonal direction, as can be seen in Fig. 12 a. To prevent this behaviour, we extend the driving neighbourhood size from 16 to 20, as shown in Fig. 12 b.

VIII. PLAN EXECUTION

We utilize the control framework described by Schwarz et al. [15]. Input for omnidirectional driving is a velocity command $\vec{w} = (v_{x'}, v_{y'}, \omega)$ with horizontal linear velocities $v_{x'}$ and $v_{y'}$ in robot coordinates and a rotational velocity ω around the vertical robot axis. We obtain \vec{w} by computing a B-spline through the next five driving poses and aim towards a pose $\vec{p} = (p_x, p_y, p_\theta)$ on this B-spline in front of the robot.

For manoeuvres which require leg movement, the input to the control framework are 2D (x', z') foot poses which can be directly derived from the resulting path.

IX. EXPERIMENTS

We evaluate our path planning method and the presented extensions in the Gazebo simulation environment². Experiments are done on one core of a 2.6 GHz Intel i7-6700HQ processor using 16 GB of memory. A video of the experiments is available online³.

In a first scenario, the robot stands in a corridor in front of an elevated platform and some cluttered obstacles. It needs to find a way to a goal pose on this platform as can be seen in Fig. 13. We compare the performance of the planner for different values of k_{12} in Fig. 14. The parameter k_{12} is defined in Fig. 11. All shown path costs are the costs the path would have in the plain A* planner to keep them comparable. It can be seen that an increasing robot orientation cost factor decreases the difference between robot orientation and driving direction. This can also be observed in Fig. 16. Moreover, planning is accelerated for higher values (≥ 2) of k_{12} , while path costs increase only slightly.

In addition, we evaluate the ARA* approach in the same scenario. We choose exponentially decaying heuristic

²<http://www.gazebo.org>

³https://www.ais.uni-bonn.de/videos/IROS_2017_Klamt/

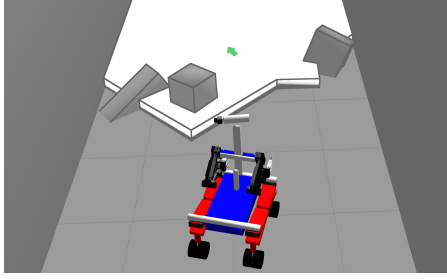


Fig. 13. Gazebo scenario to compare planner variants. Momaro stands in front of an elevated platform, cluttered with obstacles and has to reach a pose on this platform.

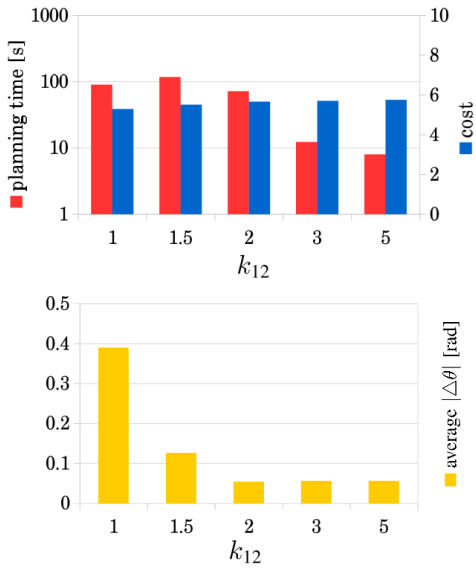


Fig. 14. Comparison between the original A* planner ($k_{12} = 1$) and the modification with robot orientation cost factor.

weights, starting at 3.0 while $k_{12} = 2$. The performance results are shown in Fig. 15 and a path can be seen in Fig. 16. ARA* provides its first result in 32ms, and this is 31% more costly than the optimal solution. A solution with only 2% higher costs is found in ~10s which is sufficiently fast compared to the required execution times. Planning paths using an optimal heuristic weight takes infeasibly long ($> 100s$). Searching optimal results takes longer than in the plain A* variant because higher heuristic weights are considered first and the neighbourhood size changed from 16 to 20. It can be seen that the effect of the robot orientation cost factor increases with decreasing heuristic weights.

To demonstrate the capabilities of our planner, we present a second experiment in which Momaro has to climb a staircase which is blocked by obstacles (Fig. 17). Tracked vehicles would have great difficulties to overcome this. Our robot climbs the stairs and then drives sideways while taking the obstacle between its legs. Our planner finds a first path with heuristic weight of 3.0 in 1.02 s. Fig. 19 shows Momaro on the staircase and visualizes how the robot base adapts its pitch angle to the terrain slope.

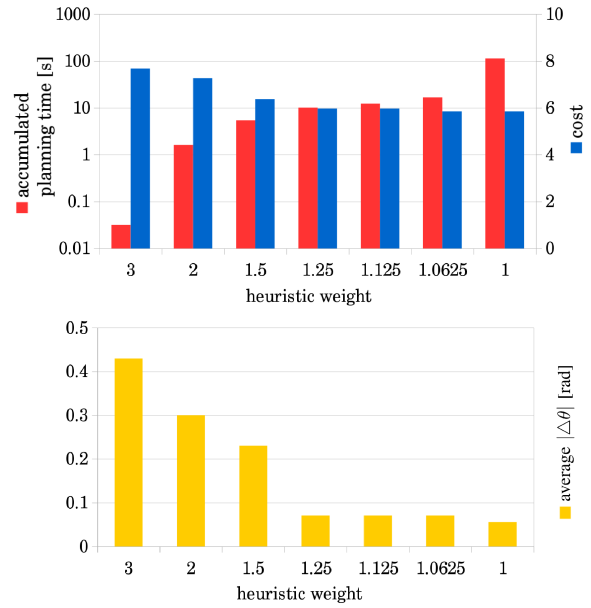


Fig. 15. Performance and result quality of the ARA* algorithm where $k_{12} = 2$.

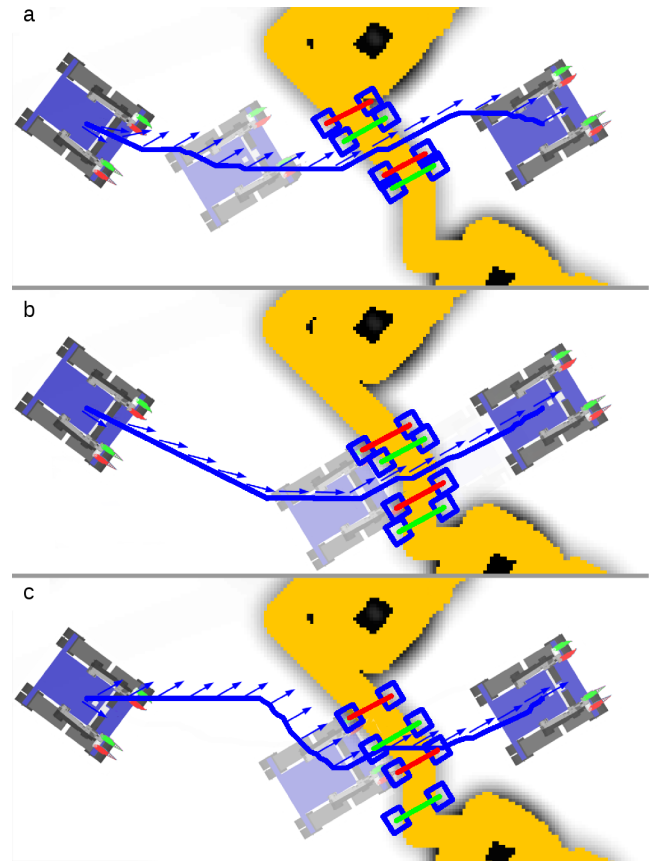


Fig. 16. Resulting paths of our planner in different settings on a foot cost map. Yellow areas are not traversable by driving. Blue paths show the robot center position, arrows show the orientation. Blue rectangles show used footholds. Red lines represent front foot steps, green lines represent rear foot steps. a) Result of the plain A* algorithm, b) orientation differences are considered ($k_{12} = 2$), c) first result of the ARA* algorithm using a heuristic weight of 3.0.

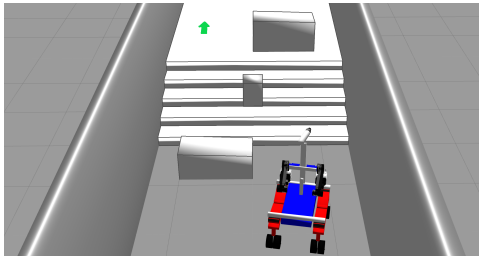


Fig. 17. Challenging scenario to demonstrate the planner capabilities. A staircase with obstacles on it requires a combination of stepping and driving sideways.

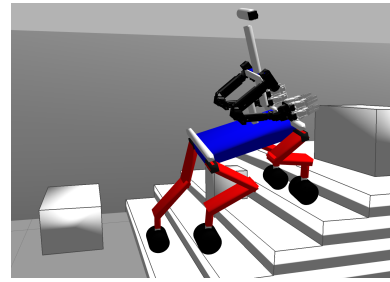


Fig. 19. Momaro climbing stairs. The robot base pitch angle adapts to 70% of the terrain slope.

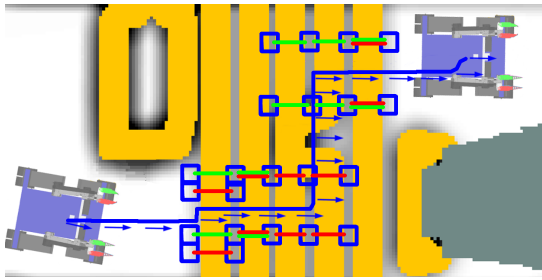


Fig. 18. Planner output for the staircase scenario on a foot cost map using a heuristic weight of 3. The blue path shows the robot center position. Arrows show the robot orientation. Blue squares show used footholds. Red lines represent front foot steps; green lines represent rear foot steps.

X. CONCLUSION

In this paper, we presented a hybrid locomotion planning approach which combines driving and stepping in a single planner. It provides paths with bounded suboptimality in feasible time and is capable of path planning in challenging environments. Due to the high dimensionality of the search space and the desire to consider detours instead of stepping, finding optimal solutions may take considerable time. We address this by using an anytime approach with larger heuristic weights. The planned paths are executed by our mobile manipulation robot Momaro. Experiments demonstrated that our method generates paths for challenging terrain, which could not be traversed by driving or stepping alone.

REFERENCES

- [1] F. Colas, S. Mahesh, F. Pomerleau, M. Liu, and R. Siegwart, "3D path planning and execution for search and rescue ground robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.
- [2] M. Menna, M. Gianni, F. Ferri, and F. Pirri, "Real-time autonomous 3D navigation for tracked vehicles in rescue environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014.
- [3] 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.
- [4] 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*, 2014.
- [5] M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, and S. Schaal, "Learning, planning, and control for quadruped locomotion over challenging terrain," *The International Journal of Robotics Research*, vol. 30, no. 2, pp. 236–258, 2011.
- [6] 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.
- [7] N. Perrin, C. Ott, J. Engelsberger, O. Stasse, F. Lamiroux, and D. G. Caldwell, "Continuous legged locomotion planning," *IEEE Transactions on Robotics*, vol. 33, no. 1, pp. 234–239, 2016.
- [8] M. Schwarz, T. Rodehutsors, 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.
- [9] A. Halme, I. Leppänen, J. Suomela, S. Ylönen, and I. Kettunen, "Workpartner: Interactive human-like service robot for outdoor applications," *The international journal of robotics Research*, vol. 22, no. 7-8, pp. 627–640, 2003.
- [10] M. Takahaashi, K. Yoneda, and S. Hirose, "Rough terrain locomotion of a leg-wheel hybrid quadruped robot," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2006.
- [11] H. Bae, I. Lee, T. Jung, and J.-H. Oh, "Walking-wheeling dual mode strategy for humanoid robot, DRC-HUBO+," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.
- [12] A. Stentz, H. Herman, A. Kelly, E. Meyhofer, G. C. Haynes, D. Stager, B. Zajac, J. A. Bagnell, J. Brindza, C. Dellin, *et al.*, "Chimp, the CMU highly intelligent mobile platform," *Journal of Field Robotics*, vol. 32, no. 2, pp. 209–228, 2015.
- [13] K. Hashimoto, T. Hosobata, Y. Sugahara, Y. Mikuriya, H. Sunazuka, M. Kawase, H.-o. Lim, and A. Takamishi, "Realization by biped leg-wheeled robot of biped walking and wheel-driven locomotion," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.
- [14] M. Schwarz, M. Beul, D. Droeschel, S. Schüller, A. S. Periyasamy, C. Lenz, M. Schreiber, and S. Behnke, "Supervised autonomy for exploration and mobile manipulation in rough terrain with a centaur-like robot," *Frontiers in Robotics and AI*, vol. 3, 2016.
- [15] M. Schwarz, T. Rodehutsors, D. Droeschel, M. Beul, M. Schreiber, N. Araslanov, I. Ivanov, C. Lenz, J. Razlaw, S. Schüller, D. Schwarz, A. Topalidou-Kyniazopoulou, and S. Behnke, "NimbRo Rescue: Solving disaster-response tasks through mobile manipulation robot Momaro," *Journal of Field Robotics*, vol. 34, no. 2, pp. 400–425, 2016.
- [16] 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.
- [17] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.
- [18] B. P. Gerkey and K. Konolige, "Planning and control in unstructured terrain," in *ICRA Workshop on Path Planning on Costmaps*, 2008.
- [19] 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.
- [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.
- [21] M. Likhachev, G. J. Gordon, and S. Thrun, "Ara*: Anytime A* with provable bounds on sub-optimality," in *NIPS*, 2003.

Computing a Collision-Free Path using Harmonic Potentials

Karl Holmquist¹ and Deniz Şenel² and Michael Felsberg¹

Abstract— Mobile robots require various functionalities for freely moving in environments containing both static and dynamic obstacles. One of the most relevant capabilities in terms of navigating a mobile robot in such an environment is to find the safest path. This paper addresses the problem of finding a collision-free path for a mobile robot in two steps. First, the tessellation of the environment map represents it as a graph. Second, the local features of the map obtained from its monogenic scale space, creates a potential field that provides the distinction of obstacles and safe areas.

I. INTRODUCTION

Many mobile robot applications require planning of a safe path that leads a robot to a goal position from a starting point. A common duty of a mobile robot is to navigate through an indoor environment [1]. The task can be vacuum cleaning of an apartment, delivery of items in hospitals, schools, libraries, supermarkets, surveillance, etc. In all of these applications, the challenge is to design the algorithm that generates an obstacle-free path from a robot's start position to a target position. Such planning includes intermediate steps as building the map of the environment [2], finding an obstacle free path to the target position, and optionally optimization of the path.

This paper proposes a method to compute obstacle-free paths using Riesz potentials. First, tessellation of the map of an environment is performed so that the environment is split such that there are no concave objects. Next, in order to preserve the complete information about the environment after splitting it into rectangles, we create a graph representation of the map using the adjacency of the rectangles. Existence of edges is determined according to existence of an opening that connects two rectangles such as a door. An example tessellation and graph representation of an indoor environment can be seen in Fig. 2. Potential field is generated for all nodes of the graph using the monogenic scale space. Lastly, using the potential values, a collision-free path is generated towards the goal position. All steps of the algorithm can be seen from Fig. 1.

There exist various methods in the literature for mobile robot path planning purposes and each method has its own strengths and weaknesses. Potential field methods have been very popular and widely used for path planning applications [3], [4], [5]. There are also sampling based methods that use a grid map of the environment [6], [7], [8]. Another

popular technique is the combination of potential values and grid map of the environment as in [9], [10]. Our method does not generate a set of paths to follow in the map as in [9], instead it calculates the complete field for a region allowing fast calculation of path from any position. This allows for applying a motion model or use of a momentum based gradient descent approach to find a smooth trajectory for the robot to follow. An explicit creation of grids of the environment is also not necessary. Since we perform tessellation by finding the largest possible rectangles, we can save computational resources when compared to creating grids of the environment.

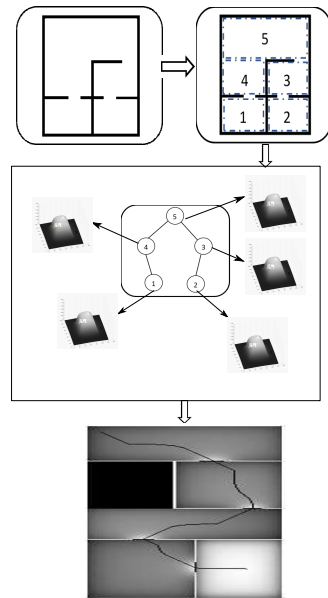


Fig. 1: Flow chart of the proposed algorithm.

Another approach which is commonly used in diverse areas is the A^* -algorithm [11]. This algorithm explores a state-space, by using a heuristic over which direction is the most beneficial to explore first while trying to find the most cost-efficient path to the terminal state. However, the shortest path is often dangerous, since it will pass close to obstacles. A comparison between our proposed method and a A^* implementation has been done and the effect of augmenting the heuristic of the A^* -algorithm with the potential field generated by using the monogenic scale space is also presented.

The safest path is defined as the path which keep the maximal distance to obstacles. While if multiple solutions

¹Karl Holmquist and Michael Felsberg are with the Department of Electrical Engineering, Computer Vision Laboratory, Linköping University, Sweden karl.holmquist@liu.se

²Deniz Şenel is a visiting researcher with the Department of Electrical Engineering, Computer Vision Laboratory, Linköping University, Sweden deniz.sevis@boun.edu.tr

exist, the shortest path in an euclidean sense is chosen. This will allow robot to navigate even when the positioning system has a low precision both in the ego localization and in the map.

The paper is organized as follows: Next section describes some of the theoretical background. The proposed method is explained in detail in Section III. Simulation results are presented in Section IV. Finally, in section V, the paper is concluded with some discussion.

II. MATHEMATICAL BACKGROUND

Scale-space theory is a framework that is used for multi-scale representation of image structures which is parameterized by the power of the kernel function. α -scale spaces are generated by the α th power of the negative Laplace operator and as introduced in [12] a special case is the Poisson scale space. It has multiple advantages over Gaussian scale spaces as stated in [13]. Further, its analytic formulation is known [14] and uncertainty of the Poisson kernel is close to the optimum [15]. Based on the Poisson scale space, the monogenic scale space, which is a vector valued scale space, can be implemented. The monogenic signal is an image analysis technique and 2-dimensional generalization of the analytic signal that is introduced in [16] and provides local amplitude and local phase features of the signal as well as the local orientation [13]. It is widely used in image processing and computer vision applications, e.g. for optical flow calculation.

The monogenic scale space is a vector valued extension of the Poisson scale space. The Poisson scale space is directly related to potential theory as the generator for the scale space is defined as $-\nabla \cdot h_2$ rather than as in the Gaussian scale space $\Delta = \nabla \cdot \nabla$. The h_2 term is the convolution kernel of the Riesz transform. The Riesz transform is a generalization of the Hilbert transform and is used for extending the Poisson scale space to the monogenic scale space [13]. The monogenic scale space separates the image into two conjugate spaces, the signal at scale s (the Poisson scale space) and the flux of the signal corresponding to the direction and magnitude of the flow of the potential field at scale s [12].

The monogenic signal is, if computed using the Discrete Fourier Transform (DFT), subject to periodic extensions of the original signal since the DFT considers discrete frequencies. In order to compensate for this [13] describes the monogenic scale space on a bounded domain. This description of the monogenic scale space adds Neumann boundary conditions to the signal, which enforce that the flow perpendicular to the border of the domain, $\partial\Omega$ is zero.

$$\frac{\partial u}{\partial s} \Big|_{\partial\Omega} = 0. \quad (1)$$

Given a rectangle $[0, w] \times [0, h]$ where w and h correspond to width and height, respectively, containing the signal $f(x, y)$, the Monogenic scale space $u(x, y, s)$ is calculated as:

$$u^{(w,h)}(x, y, s) = \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \left(\int_0^w \int_0^h f(x', y') l_{mn}(x', y') dx' dy' \right) \times l_{mn}(x, y) e^{-\sqrt{\left(\frac{n\pi}{h}\right)^2 + \left(\frac{m\pi}{w}\right)^2} s} \quad (2)$$

With l_{mn} is given by:

$$l_{mn}(x, y) = \frac{2 \cos\left(\frac{m\pi x}{w}\right) \cos\left(\frac{n\pi y}{h}\right)}{\sqrt{wh}}. \quad (3)$$

This transform can be implemented efficiently by using Discrete Cosine Transform (DCT) and Discrete Sine Transform (DST).

III. PATH PLANNING METHOD

In this work, the goal is to find an obstacle-free path such that a mobile robot can travel from a starting position to a goal position in an indoor environment using the potential field generated using the monogenic scale space representation of the map of the environment. The map is represented by a binary image indicating the free space and obstacles, such as walls. Since it is represented as an image it will inherently be quantized into a grid representation. The map may have been created earlier by a robot surveying the area or it might be based on an architectural floor map of the area. In this study, simulations are performed on an artificial map. However, the proposed method can easily be adapted to a real map of an environment. An example map of an indoor environment with boundary and walls represented using black solid lines are illustrated in Fig. 2a.

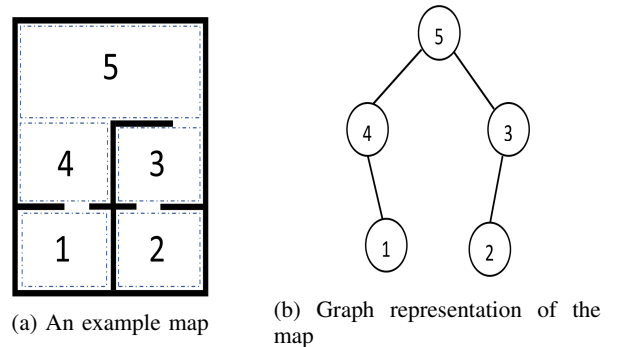


Fig. 2: Tessellation of an example map with maximal rectangles and its graph representation

A. Tessellation of the Environment

In this step, tessellation of the environment with rectangles is performed using the map in order to be able to represent it as a graph. The tessellation of the environment is done to create a representation such that its areas do not contain any concave objects. Those cause problems in the potential field representation as they produce local minima.

The tessellation of the environment is done iteratively starting from a known starting position or an arbitrary position on the map. Starting from this position, a rectangle is fitted to the obstacle-free area by first finding the maximum

length of the required rectangle and then expanding its height until the maximum height is found. From this initial rectangle, the openings in which the border of the rectangle does not correspond to a wall is seen as openings. These openings serve as new points from where new rectangles should be created. The area inside of the found rectangles are excluded from the map in order to avoid new rectangles to overlap with already found ones.

Tessellation of the environment using rectangles can be seen in Fig. 2a where the rectangles are denoted by dashed-dotted lines. The tessellation can also be done in an obstacle free map and later applied to the same map with obstacles, c.f. Fig. 3a and Fig. 5a

B. Graph Representation of the Environment

Since the tessellation is splitting the environment into small parts, the complete information of the map is lost unless the connections between the rectangles are maintained. The connections between rectangles in the map are represented by a set of nodes and their edges which form an undirected graph, possibly containing cycles. This graph is built during the tessellation using the information from the connections between rectangles.

Using the tessellation performed in the previous step, the environment can be represented as a graph while each rectangle corresponds to a node of a graph. Existence of an edge can be determined according to the separation of the rectangles. For example, if there is a wall between two consecutive rectangles of a map, then we classify those nodes as disconnected. If there is an opening between two rectangles like a door on a wall, then two nodes are said to be adjacent and there exists an edge between them.

In Fig. 2, an artificially created map and its graph representation are given to demonstrate the process. The map is tessellated using large rectangular regions since they are convex. There is also an simple analytical expression of the monogenic scale space on a rectangular region which allows an efficient implementation, see section II. These rectangular regions are considered as nodes of a graph. There exist an edge between two nodes if the regions are connected, namely there is a passage available that the robot can pass through like a door. Otherwise, they are disconnected like rooms separated via a wall. For example, according to map given in Fig. 2a, rooms 1 and 2 are separated by a wall hence the nodes 1 and 2 are disconnected also in Fig. 2b. However, rooms 1 and 4 are connected via a door. Therefore, there exists an edge between nodes 1 and 4 in Fig. 2b.

C. Finding and Calculating the Path

The goal is to find the path that leads a robot to the goal position, namely starting from a node of the environment graph we try to find the path that ends at the terminal node of the graph. After deciding the collection of edges leading to the terminal node, we calculate the exact path that the robot must follow to reach its goal position by following the increment of the potential values.

1) *Finding a Path in the Graph:* Since the optimality of this path is of little concern when navigating a hazardous environment, it is sufficient to find a path connecting the two nodes in the graph. As such, the path is found by a simple depth first search while avoiding revisits to previously visited nodes. In the case of Fig. 2, assuming that the robot is at room one in Fig. 2a which corresponds to node 1 in Fig. 2b and the goal is in room 5. Then, the path would be the one with the starting node 1 and end at node 5 through node 4, namely $1 \rightarrow 4 \rightarrow 5$.

2) *Calculating the Path Inside Each Node:* The second part of the path planning problem is to find a safe path in each node, entering and exiting at the openings which leads to the correct adjacent node. This is done by interpreting the area as a 2D-signal and calculating the monogenic scale space for a certain scale. The monogenic signal, as a generalization of the analytical signal provides features like local amplitude, local phase, and local orientation which are point-wise orthogonal [13]. It is shown that, α -scale space corresponds to Poisson scale space for the case where $\alpha = 0.5$ in [12]. Poisson scale-space must be considered as a potential problem instead of a heat problem since isotopes within the Poisson scale-space corresponds to equipotential curves of Riesz potentials and the isophotes within the conjugate Poisson scale-space correspond to flow lines [12].

Using the gray-scale image representation of the environment map, a potential field is generated by calculating the monogenic signal of the image at scale s . The monogenic signal at a certain scale corresponds to a low-pass filtered version of the original signal. However, since it uses a Poisson kernel rather than a Gaussian kernel it corresponds to the potential field rather than the heat dissipation which is the case for Gaussian kernels. This field can directly be used for finding a path through the environment by following the direction of increasing potential since the goal is defined as a source with the highest potential and the obstacles are the sinks with lower potentials depending on the size of the obstacles.

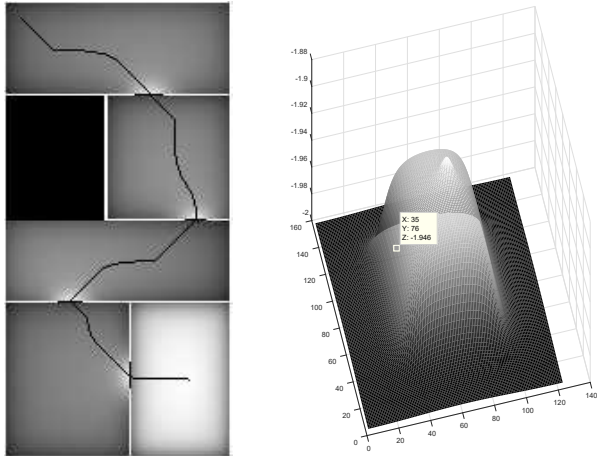
By creating a potential source in the exit and sinks at the position of the obstacles, the monogenic signal can be calculated for a fixed scale. The scale should be chosen so that the source may affect the intensity of the whole image but before the entire image becomes homogeneous.

IV. SIMULATIONS AND RESULTS

First, in this section we apply the proposed method to a scenario where the mobile robot navigates through rooms to reach its goal position. For this purpose, artificially created maps are used. Two different scenarios are considered: obstacle-free rooms and rooms with obstacles. Secondly, the first map is used to compare a direct approach of finding a path using the A^* -algorithm [11]. As well as applying a modified A^* -algorithm on the generated Riesz potential field.

In the first example given in Fig. 3, the starting position of the robot is the upper left corner of the top room while the terminal position is in the middle of the bottom right room

on the map in Fig. 3a. Unvisited rooms are left dark since no potential field has been calculated in these. The calculated path can be seen in Fig. 3a, indicated by a black line. As an example, the potential map of the last room in the path is visualized as a surface plot in Fig. 3b. The marked position in Fig. 3b is the starting position in that room. The original map before creating the potential field can be seen in Fig. 4. Here white lines correspond to walls.



(a) Environment map with calculated obstacle-free path (b) Potential values within the last room.

Fig. 3: First scenario: obstacle-free rooms.

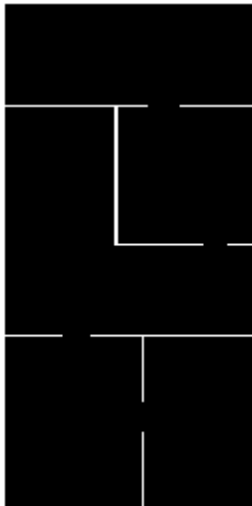
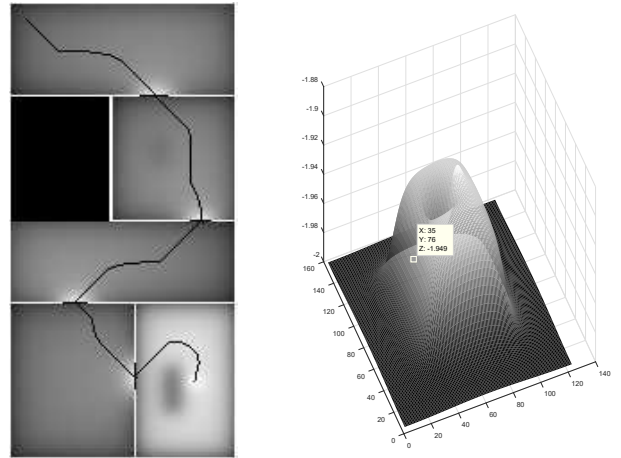


Fig. 4: Map of obstacle free environment.

In the second example, there are obstacles placed in some rooms which can be seen as darker areas in the potential maps of the room in Fig. 5a. Corresponding potential values of the last room is given in Fig. 5b. Again, the starting position is the left corner of the top room and goal is the middle of the bottom room in Fig. 5. As can be seen, the robot can reach the goal position without colliding with walls or any of the obstacles as indicated by the black path in Fig. 5a. The marked position in Fig. 5b is the starting position

in that room. The original map before creating the potential field can be seen in Fig. 6. Here white lines and rectangles are walls and obstacles, respectively.



(a) Environment map with calculated obstacle-free path (b) Potential values within the last room.

Fig. 5: Second scenario: rooms with obstacles inside.

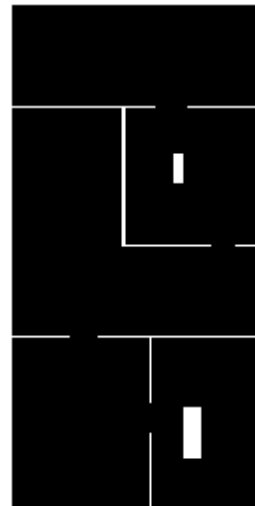


Fig. 6: Map of environment with two obstacles inside of rooms.

In the last example, we demonstrate a possible path that the robot follows to reach its goal in a more complex but obstacle free environment using the potential values of the tessellated map in Fig. 7. The calculated path is indicated as a black line. The path is calculated according to the nodes that lead to the final node.

We also performed a comparative study using one of the most popular methods for path planning which is the A^* -algorithm. The application of the A^* -algorithm with a maximum step length of one in each direction is shown in Fig. 8a. As the only thing affecting the choice of path is the number of steps needed, the path will be optimized in terms of distance. However, this tends to be dangerous in most

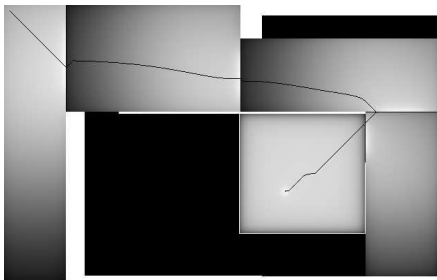


Fig. 7: Example scenario of a more complex environment.

environments and problematic with real time experiments since the path actually touches the walls of the rooms.

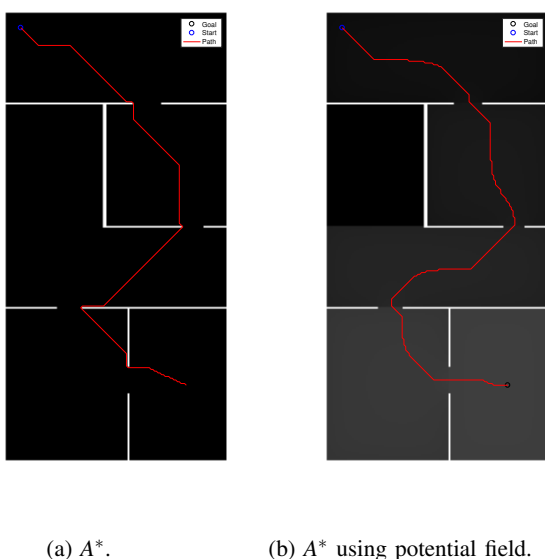


Fig. 8: Comparison of performing the A^* -algorithm using left: a distance based heuristic and right: using the potential field generated using the proposed method on map in Fig. 4.

In comparison, when applying the A^* -algorithm to the potential field calculated using the monogenic scale space, see Fig. 8b. The found path appears much safer, passing through the middle of the doorways and clearly avoiding the walls. This clearly illustrates the need for having a more complex error function to optimize, rather than a naive, euclidean distance measure. It also shows that the A^* -algorithm can be implemented as the path selecting step after generating the potential field, with a good result. Though it should be noted that since A^* searches over all nodes, it will be slower than directly following the direction of highest potential.

Table I shows the run-time as well as the number of steps necessary for the algorithm to get from the starting position to the goal position in Fig. 8 and Fig. 3a. As can be seen in the table, the time for running the A^* -algorithm is highly dependent on the heuristic used when choosing path to explore. But when run on the map in Fig 4 it can

be seen that all found path are of approximately the same length, while only using A^* leads to a path much close to the obstacles than the other two. One thing to note is that the monogenic scale space using steepest ascent and using A^* to find the path results in similar paths, even though the A^* -algorithm is a bit slower.

TABLE I: Comparison in run-time and length of found path.

	Number of steps	Typical time
A^* -algorithm	306	0.36s
Monogenic scale space	307	0.12s
Monogenic scale space + A^*	305	0.12s + 0.04s

V. CONCLUSION

In this paper we propose a method to find a solution to the problem of planning a collision-free path which requires solving intermediate steps: tessellation of the environment map, representing it as a graph to preserve the complete information about the environment using the adjacency of the rectangles, and finding the monogenic scale space of each node to calculate the potential field of the environment.

Tessellation allows a sparse generation of the potential field limiting the calculations to the nodes that are actually visited. It can also be done over all nodes in order to choose an optimal path. The tessellation allows to compute scale space easily on a rectangular domain. The use of rectangular domains also allows the monogenic scale space to avoid including local minima in each region. Even if it can be performed for the whole map, there is a risk of the potential field containing local minima. Since the potential value depends directly on the distance to the source and sinks, multiple obstacles may block the way.

The monogenic scale space is calculated by use of Fourier transforms. When the inverse discrete Fourier transform is applied afterwards the coarseness of the generated potential field can be controlled depending on required precision in the path planning. The usage of the monogenic scale space to create the potential field also allows other approaches for finding a path, such as fast marching methods. Although, care needs to be taken of the magnitude of potential sinks and sources and the scale to avoid creating flat patches in the potential field.

We also present a comparative study with a popular approach used in path planning, the A^* -algorithm. Simulation results reveal that, the proposed method clearly outperforms the A^* -algorithm in terms of both required simulation steps and time. Moreover, use of potential field method provides a safer path than the A^* -algorithm finds on its own.

We are planning to extend our work by applying the proposed method to an experimental study with one or more mobile robots in an environment including both static and dynamic obstacles possibly in a partly unknown environment. This method can easily be extended to higher dimensional maps.

REFERENCES

- [1] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, pp. 21–71, 1999.
- [2] J. J. Leonard and H. F. Durrant-Whyte, "Simultaneous map building and localization for an autonomous mobile robot," in *Proc. IEEE IROS '91: International Workshop on Intelligence for Mechanical Systems*, Osaka, Japan, Nov. 1991, pp. 1442–1447.
- [3] Y. Wang and G. S. Chirikjian, "A new potential field method for robot path planning," in *Proc. IEEE ICRA '2000*, San Francisco, USA, Apr. 2000, pp. 977–982.
- [4] M. G. Park and M. C. Lee, "Artificial potential field based path planning for mobile robots using a virtual obstacle concept," in *Proc. IEEE ASME International Conference on Advanced Intelligent Mechatronics*, 2003, pp. 735–740.
- [5] Y. K. Hwang and N. Ahuja, "A potential field approach to path planning," *IEEE Trans. Robot. Automat.*, vol. 8, pp. 23–32, feb. 1992.
- [6] D. Ferguson and A. Stentz, "Field d*: An interpolation-based path planner and replanner," *Robotics Research*, pp. 239–253, 2007.
- [7] S. Thrun and A. Bcken, "Integrating grid-based and topological maps for mobile robot navigation," in *Proc. National Conference on Artificial Intelligence*, Oregon, Aug. 1996.
- [8] F. Duchoň, A. Babinec, M. Kajan, P. Beňo, M. Florek, T. Fico, and L. Jurišica, "Path planning with modified a star algorithm for a mobile robot," *Procedia Engineering*, vol. 96, pp. 59–69, 2014.
- [9] S. Garrido, L. Moreno, D. Blanco, and P. Jurewicz, "Path planning for mobile robot navigation using voronoi diagram and fast marching," *International Journal of Robotics and Automation*, vol. 2, pp. 42–64, 2011.
- [10] M. Soullignac, "Feasible and optimal path planning in strong current fields," *IEEE Trans. Robot.*, vol. 27, pp. 89–98, feb. 2011.
- [11] R. Dechter and J. Pearl, "Generalized best-first search strategies and the optimality of a*," *J. ACM*, vol. 32, no. 3, pp. 505–536, July 1985. [Online]. Available: <http://doi.acm.org/10.1145/3828.3830>
- [12] R. Duits, M. Felsberg, L. Florack, and B. Platel, " α scale spaces on a bounded domain," in *Proc. International Conference on Scale-Space Theories in Computer Vision*, Greece, Sept. 2003, pp. 494–510.
- [13] M. Felsberg, "The monogenic scale space on a rectangular domain and its features," *International Journal of Computer Vision*, vol. 64, pp. 187–201, 2005.
- [14] M. Felsberg and G. Sommer, "Scale adaptive filtering derived from the laplace equation," in *Proc. DAGM Symposium Mustererkennung, München*, 2001, pp. 124–131.
- [15] M. Felsberg, "Low-level image processing with the structure multivector," Ph.D. dissertation, Christian-Albrechts-University of Kiel, Germany, 2002. [Online]. Available: <http://www.informatik.unikiel.de/reports/2002/0203.html>
- [16] M. Felsberg and G. Sommer, "The monogenic signal," *IEEE Trans. Signal Processing*, vol. 49, pp. 3136–3144, Dec. 2001.

Geometric Correspondence Network for Motion Estimation

Jiexiong Tang¹, John Folkesson¹ and Patric Jensfelt¹

Abstract—In this paper, we propose a learning scheme for generating geometric correspondences to be used for visual odometry. A recurrent neural network (RNN) on top of a convolutional neural network (CNN) are jointly optimized to both detect the location of keypoints and generate a descriptor in one unified structure. The network is trained by warping points in one frame to the next with a rigid body transform. Essentially, learning from warping. The overall training is focused on movements of the camera rather than movements within the image, which leads to better consistency in the matching and ultimately better the motion estimation which we demonstrate in the experiments. Experimental results show that the proposed method give significantly less absolute tracking error (ATE) than using hand crafted keypoint detectors and descriptors. Furthermore, as a demonstration of the promise of our method we use a naive SLAM implementation based on these keypoints and get a performance on par with ORBSLAM.

I. INTRODUCTION

Motion estimation is a fundamental part of mobile robotic systems. The advantages of being able to estimate motion solely using an RGB-D sensor are significant. The sensors are relatively low weight, low power and low cost compared to lidar for example. At the same time they can give accurate estimates under the right conditions. Estimates based on a single sensor are easier to integrated into a system than ones based on specific suites of sensors as testing and validation can be done with the sensor detached from the system.

In this paper we focus on visual odometry (VO), i.e. frame to frame motion estimation using information from a vision sensor. In particular, we investigate the use of a deep neural network to learn how to both detect keypoints and how to match them in an end-to-end fashion using an RGB-D sensor. We only optimize for geometric correspondence rather than semantic correspondence. A convolutional neural network (CNN) is used together with a recurrent neural network (RNN) and these are jointly optimized to perform both keypoint detection and feature descriptor generation in one unified structure which we call the Geometric Correspondence Network (GCN). To train the network we extract high gradient points in one image and warp these using the camera motion to the next image. It is important to note that we do not try to learn to reproduce these high gradient points or any standard descriptor based on them, but rather the network learns to find features and descriptors that can

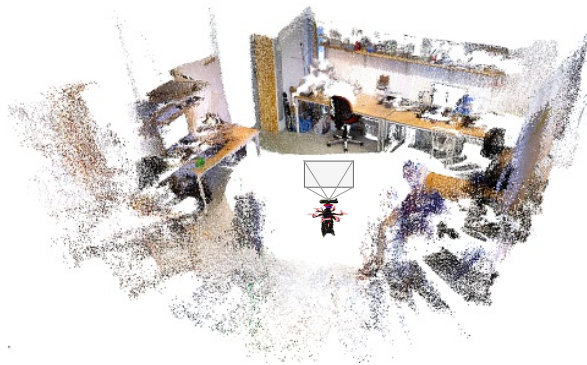


Fig. 1: RGB-D frames accumulated using frame to frame camera motion estimation (i.e., using no keyframes and loop closures) with our proposed method, GCN, for keypoint generation. Data is collected by RGB-D sensor on a hexacopter flying around the room.

be matched well. In fact, the points in the warped image may not even be as detectable based on the gradient as they were in the unwarped image. We make use of a benchmark dataset for SLAM [1] for which ground truth positions are available to perform the warping to generate the training data. We compare GCN against a number of other visual features for visual odometry using the same pipeline¹. To further evaluate the method we implement a simple SLAM backend² to allow for loop closure and compare our results to other SLAM methods.

Decades of work has gone into engineering features for recognition and classification of object, place recognition, loop closure detection, stereo image matching, etc. Recently, deep neural networks have been shown to outperform hand crafted features for all these tasks. The problem of finding correspondences for frame to frame keypoint matching for a visual odometry is a little investigated area in comparison. With consideration to that, achieving results that match popular methods is encouraging.

In summary, the main contribution of our work is a framework (GCN) for simultaneously learning both feature locations (for motion estimating) and descriptors (for matching) optimized for the task of camera motion estimation.

¹ The authors are all with the Centre for Autonomous Systems at KTH Royal Institute of Technology, Stockholm, SE-10044, Sweden jiexiong@kth.se

*This work was partially supported by the Wallenberg Autonomous Systems and Software Program (WASP). This project has received funding from the European Unions Horizon 2020 research and innovation program under grant agreement No. 644839 (CENTAURO).

¹OpenCV's PnP with RANSAC

²Pose graph plus optimization

II. RELATED WORK

This paper is related to many different well researched domains. We will focus on the relation to existing VO / SLAM solutions and deep learning used in related applications.

Existing methods to VO and SLAM can be divided into sparse, or feature based, methods and dense, or direct, methods. In the sparse methods, keypoints are first extracted and then they are matched based on geometric constraints and local descriptors. The motion is estimated based on the correspondences between points in two frames. ORBSLAM [2], [3] represents the state-of-the-art for SLAM today in this category. ORB [4] feature are extracted in new frames and are then matched to recent frames to perform camera motion estimation and to distant frames using a binary bag of words representation [5] to detect loop closures. ORBSLAM is available in implementations for monocular, stereo and RGB-D sensor setups. In dense methods, information from all the pixels in the image are used, thus eliminating the need for feature detection. Optimizing is instead done directly using the intensities in the images. In [6], Comport et. al. present a visual odometry system able to estimate the motion of the sensor with a vertical error of only 20cm over a distance of 200m in a urban driving scenario. Kerl et. al. present the Direct Visual Odometry (DVO) method in [7]. They build a pose graph and use g2o [8] for optimization to reduce the error. As an alternative to matching frame to frame, KinectFusion [9], Kintinous [10] and ElasticFusion [11] build a volumetric representation to which new frames are matched. The methods SVO [12] and LSD-SLAM [13] fall somewhere in between dense and sparse, combining the best traits from the feature based methods with those from the dense methods and having. The work in our paper is most closely related to feature based methods such as ORBSLAM. However, whereas in ORBSLAM they use an off-the-shelf feature (ORB) and design a complete pipeline around it, we focus on the design of the feature itself and thus complement rather than compete.

Deep learning based methods are already defining the state-of-the-art in many applications in computer vision. As a key component for 3D scene reconstruction, stereo matching using CNNs have been shown to perform very well [14], [15], [16]. In recent work, [17], the efficiency of deep stereo matching is improved by exploiting an inner product layer between two feature representations. Deep learning has also been applied to predict optical flow. DeepFlow [18] used DeepMatch [19] to compute optical flow using multi-layer architectures that match sub-patches on different scales. To reduce the computational cost, a faster and edge preserving method have been proposed in [20]. Combining DeepMatch with contour detection, EpicFlow[21] interpolates matches and obtain a dense estimation. In FlowNet [22], [23], a CNN is trained by a large synthetic dataset[24] to directly predict per-pixel flow. This line of work is closely related to our work as optical flow can be used to estimate the camera motion. However, we differ in that we focus on detection and match of the points that are most suited to estimate the

motion of the camera.

In another stream of work, a network is trained to estimate the depth from monocular images [25], [26], in essence creating a virtual RGB-D sensor for a standard monocular camera. In CNN-SLAM [27] this approach is used in a mapping framework, allowing them to recover the scale using only monocular information. Place recognition is another area for which deep nets have been applied with great success [28], [29], [30]. Similar to us [29] makes use of siamese networks. They are fine-tuned to match aerial and ground-level images. Addressing the same problem of matching aerial and ground level images is [31]. Like in our work, the aim is to learn to both detect and describe keypoints. A fully-convolutional recursive network outputs keypoint responses at different scales. Patches are extracted around these keypoints and descriptors are generated.

In [32], the camera motion is estimated using features from a siamese network from color images in a classification manner. The translations and rotations are discretized within pre-defined ranges to a small number of bins (10 bins in the default setting). The network is trained to classify the ego motion with classes defined as these bins. The discretization limits the accuracy of the motion estimation and the pre-defined range limits allowed motion of the camera.

In [33], the Universal Correspondence Network (UCN) is presented as a way to extract dense features for applications where one needs geometric and semantic correspondences. In contrast to patch similarity based methods, UCN directly optimizes features for visual correspondence. This is the work that is closest to our work. But the aim is a universal approach rather than an approach targeting a specific application like ours. In an evaluation of camera motion estimation they show that UCN performs better than [32] despite being more general. However, it is only on par with sparse features that have been designed for the task of motion estimation. Though general as an visual correspondence framework and outperforming patch similarity based methods in several settings, it does not beat traditional methods in the context of camera motion estimation. In contrast to UCN, we focus specifically on camera motion estimation and show that this allows us to significantly outperform hand crafted features, and by more than UCN for this task.

In summary, our work is closely related to VO and SLAM work but in contrast to most of this work we do not focus on the estimation problem but rather the learning of a keypoints to support the camera motion estimation. We use a deep learning approach but unlike related work in this area we train our network for the task of camera motion estimation rather than addressing the general problem of generating correspondences such as UCN [33].

III. MOTION ESTIMATION

In this section, we briefly review the feature based Bundle Adjustment (BA) algorithm. Let the points in the reference image be denoted as $x_i = [u_i, v_i]^T, i \in [1, N]$ and their correspondences in current image $y_i = [u'_i, v'_i]^T, i \in [1, N]$. To align these two images and estimate the relative transform

between them, we first project y_i to 3D coordinates with the warping function $\pi(\cdot)$:

$$\pi(\mathbf{x}_i, d_i) = \left[\frac{d_i(u_i - c_x)}{f_x}, \frac{d_i(v_i - c_y)}{f_y}, d_i \right]^T \quad (1)$$

where d_i is the depth measurement of the corresponding point and c_x, c_y, f_x, f_y are the camera intrinsics. After the projection, we warp the 3D points with a rigid body transformation and project them back to image plane. Geometrical errors caused by mis-alignment can be calculated as follows:

$$E_{geo}(\boldsymbol{\xi}) = \sum_i r_i(\boldsymbol{\xi}) = \sum_i (\mathbf{x}_i - \pi^{-1}(\mathbf{R} \cdot \pi(\mathbf{y}_i) + \mathbf{t}))^2 \quad (2)$$

where $\mathbf{R} \in \mathbb{R}^{3 \times 3}$ is rotation matrix and $\mathbf{t} \in \mathbb{R}^3$ is the translation vector. They together compose a rigid body transform $\exp(\hat{\boldsymbol{\xi}}) \in \mathbb{SE}(3)$, which is defined by $\boldsymbol{\xi} = [\boldsymbol{\omega}^T, \mathbf{t}^T]^T \in \mathfrak{se}(3)$. $\boldsymbol{\xi}$ is a member of the Lie algebra and it is mapped to the Lie group $\mathbb{SE}(3)$ through the matrix exponential $\exp(\cdot)$:

$$\exp(\hat{\boldsymbol{\xi}}) = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \quad (3)$$

$$\hat{\boldsymbol{\xi}} = \begin{bmatrix} [\boldsymbol{\omega}]_{\times} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$$

where $[\boldsymbol{\omega}]_{\times}$ is the skew-symmetric matrix of $\boldsymbol{\omega}$.

The estimated relative pose can be obtained by optimizing the residual error in Eq.2. The Gaussian-Newton (GN) method is used to solve this non-linear least squares problem. GN calculates $\boldsymbol{\xi}$ iteratively as follows:

$$\boldsymbol{\xi}^{(n+1)} = \boldsymbol{\xi}^{(n)} - (\mathbf{J}_r^T \mathbf{J}_r)^{-1} \mathbf{J}_r^T \mathbf{r}(\boldsymbol{\xi}^{(n)}) \quad (4)$$

where \mathbf{J}_r is the Jacobian matrix with respect to the residual measurements.

IV. PROPOSED METHOD

The overall network structure of GCN can be seen in Fig.2. It is a deep CNN with a shallow recurrent network on top to perform both dense feature extraction and keypoint detection. It is well known that good points to use for estimating the rigid body transformations are those with high gradient in both horizontal or vertical directions in the image. In contrast to [33], we therefore train with input from keypoint detectors rather than random points. However, it is not guaranteed that the same keypoints will appear in two images, despite being consecutive. To tackle this problem during training, keypoints in first image are extracted and warped to the next frame, using the known frame to frame transformation. These then serve as the ground truth correspondences. By doing so, we optimize the network for the case where correspondences of points can be found by warping them with a rigid body transform not by their appearance in the second frame. This method will not match points that have similar appearances in the two images but are not actually geometric points such as lines formed by crossing edges at different depths. That does not mean such poor features will not be found by the trained net but at least we are not using them for training.

In the remainder of this section, we first introduce deep metric learning for training dense feature descriptors. Then,

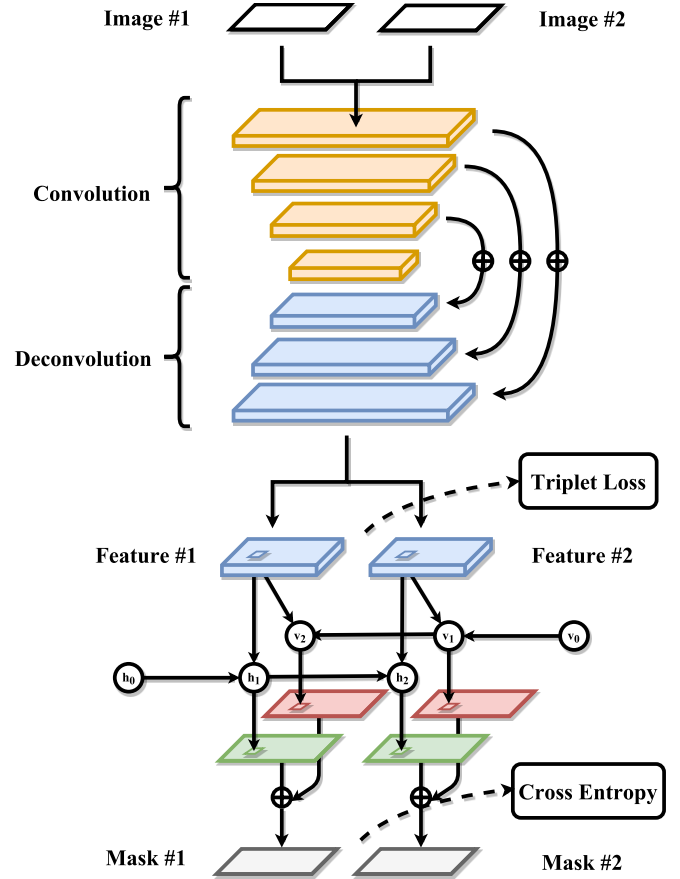


Fig. 2: The overall learning scheme and data flow of GCN is shown with colors represent different blocks. The upper part is the CNN that extracts features in a dense way. It consists of a fully convolutional part for multi-scale feature representation and a deconvolutional part for detail refinement. The lower part is a shallow recurrent structure to predict the location of keypoints in both images. Note that the actual networks have more hidden layers than drawn.

we present the method for jointly training a detector using temporal information of resultant deep features. Finally, we summarize the overall multi-task learning scheme.

A. Dense Feature Extraction

Pyramid Network Backbone. As shown in Fig. 2, the proposed network structure for extracting dense feature can be divided into two parts: omne convolutional and one deconvolutional part. For the convolutional part we used ResNet50 [34], which is a 50 layer CNN with a bottleneck structure and batch normalization. Its weights are pre-trained with the ImageNet [35] object classification dataset with 1.2 million labeled images. When the image is fed into ResNet50, its size is gradually reduced by pooling operation and convolution with stride larger than one. To recover the features to its original size, the deconvolutional part is used for upsampling the features. Inspired by the structure in [36], multiple shortcut paths are made between intermediate

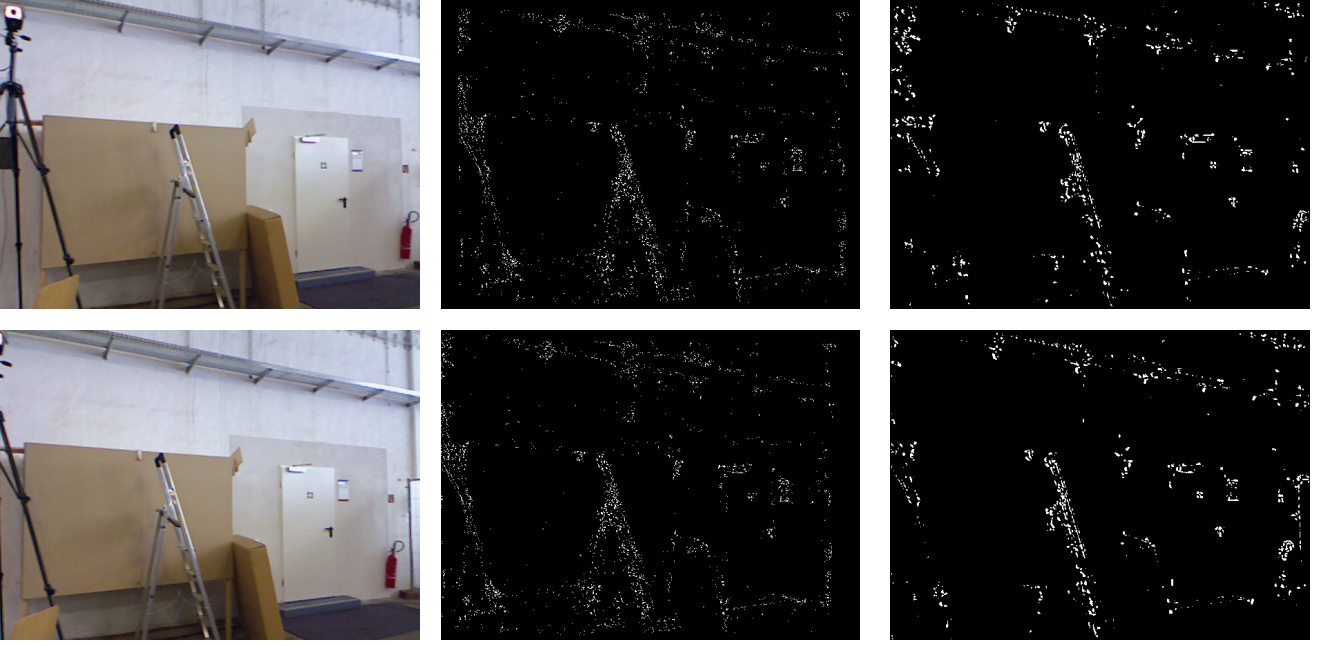


Fig. 3: An example of keypoint prediction using GCN for the TUM datasets [1]. The two color images are the input for the GCN. In each row, the three columns are raw RGB image, ground truth and prediction, respectively. For the ground truth, points with no depth or warped points that are outside of image boundary are discarded. We notice that the regions with no texture are not generating keypoints and that some regions where the keypoint detector used to generate the ground truth has generated keypoints, do not generate any keypoints either.

layers of the convolutional and the deconvolutional parts to obtain more fine-grained feature. A shortcut is a direct concatenation of feature maps by summing over channels.

Deep Metric Learning. Deep metric learning (DML) is applied to fine-tune the CNN to a dense feature extractor. DML maps the input samples to a feature space where similar samples are closer and dissimilar samples are farther with the l_2 distance as metric. Training with DML allows the features to be optimized for nearest neighbour matching.

We use triplet loss [37] to perform the DML. It has an objective function that penalizes three samples at a time: one anchor point and its paired positive/negative samples. In our proposed method, we calculate triplet loss on every point candidates as follows:

$$L_{metric} = \sum_i \max(0, s_{\mathbf{x}_i, \mathbf{y}_i^*}^2 - s_{\mathbf{x}_i, \mathbf{z}_i^*}^2 + m) \quad (5)$$

$$s_{\mathbf{x}, \mathbf{y}} = \|\mathbf{f}_1(\mathbf{x}) - \mathbf{f}_2(\mathbf{y})\|_2$$

where $s_{\mathbf{x}, \mathbf{y}}$ is the l_2 distance between two feature vectors at location x and y from output feature f_1 and feature f_2 (shown in Fig.2). m is a margin determining how far dissimilar points should be pushed away in feature space. \mathbf{x}_i is called the anchor point which is the keypoint in feature #1. Points \mathbf{y}_i^* and \mathbf{z}_i^* are its positive and negative matched points in feature #2. The location of positive matched point y_i^* is obtained by warping x_i with ground truth $\mathbf{R}^*, \mathbf{t}^*$:

$$\mathbf{y}_i^* = \pi^{-1}(\mathbf{R}^* \cdot \pi(\mathbf{y}_i) + \mathbf{t}^*) \quad (6)$$

The location of the negative sample z_i^* is found by hardest

negative sample mining:

$$\mathbf{z}_i^* = \underset{\mathbf{z}_i | \mathbf{z}_i \neq \mathbf{y}_i^*}{\operatorname{argmin}} s_{\mathbf{x}_i, \mathbf{z}_i} \quad (7)$$

That is, pick the closest point in feature space that is not a positive sample. Any point other than the true matched can be used as the negative pair for the anchor, but the hard negative sample will contribute the most to the loss function, and thus the gradient, and thereby accelerates the DML.

B. RECURRENT MASK PREDICTION

Recurrent Structure. We treat the keypoint detection as a binary classification problem which uses the dense features as input. To predict the keypoint locations simultaneously for the two input images, we want the network to exploit temporal information as well as spatial information. We use a shallow recurrent convolutional network (RCNN) to achieve this goal. The proposed RCNN can be formulated as follows:

$$\begin{aligned} \mathbf{h}_1 &= \operatorname{ReLU}(\mathbf{W}_{i2h} \circ [\mathbf{f}_1, \mathbf{h}_0]), \mathbf{h}_2 = \operatorname{ReLU}(\mathbf{W}_{i2h} \circ [\mathbf{f}_2, \mathbf{h}_1]) \\ \mathbf{v}_1 &= \operatorname{ReLU}(\mathbf{W}_{i2h} \circ [\mathbf{f}_1, \mathbf{v}_0]), \mathbf{v}_2 = \operatorname{ReLU}(\mathbf{W}_{i2h} \circ [\mathbf{f}_2, \mathbf{v}_1]) \\ \mathbf{o}_1 &= \operatorname{softmax}(\mathbf{W}_{h2o} \circ \mathbf{h}_1 + \mathbf{W}_{h2o} \circ \mathbf{v}_2), \\ \mathbf{o}_2 &= \operatorname{softmax}(\mathbf{W}_{h2o} \circ \mathbf{h}_2 + \mathbf{W}_{h2o} \circ \mathbf{v}_1), \end{aligned} \quad (8)$$

where \circ is convolution operation, \mathbf{h} and \mathbf{v} are two hidden states used to store informations and \mathbf{o} is the 2 channels output at given location where a softmax function will be applied to calculate the probability of being a keypoint. \mathbf{W}_{i2h} and \mathbf{W}_{h2o} are hidden weights to be learned and

represent weights of input-to-hidden path and hidden-to-output path, respectively.

Equation (8) above shows the data flow of the proposed recurrent structure. By designing the data flow, we can directly control how the detection is performed. Firstly, two dense features are input with their original order. Then, the two features are switched and fed into the RCNN again. Finally, the output of the previous two are concatenated as the final output of the RCNN. This is inspired by [3] where alignment of images is performed twice to utilize as much information as possible from them.

Mask Classification The training data for the keypoint classification is a mask with label 0 or 1 to indicate whether a pixel is classified as a keypoint or not (middle column in Fig. 3). The masks are generated by giving keypoints in first images a value 1 and warping them to the next image. Then, a weighted cross entropy is calculated over every pixel as follows:

$$L_{mask} = L_{ce}(\mathbf{o}_1, \mathbf{x}) + L_{ce}(\mathbf{o}_2, \mathbf{y}^*)$$

$$L_{ce}(\mathbf{o}, \mathbf{x}) = - \sum_i (\alpha_1 c_{\mathbf{x}_i} \log(\text{softmax}(\mathbf{o}(\mathbf{x}_i))) + \alpha_2 (1 - c_{\mathbf{x}_i}) \log(1 - \text{softmax}(\mathbf{o}(\mathbf{x}_i))) \quad (9)$$

where c is the label of a given 2D point and \mathbf{o} is the predicted possible keypoint. As in the previous subsection, \mathbf{x}_i and \mathbf{y}_i^* are keypoints and their correspondences obtained by warping, respectively. α is a parameter to balance between different classes. This weighting is crucial for the training convergence as there is an obvious imbalance among these two classes: the number of keypoints (thousands) versus the total number of pixel (hundreds of thousands).

C. MULTI-TASK TRAINING

Algorithm 1: Multi-task Training

Input : Training set $X = \{(I, D, \xi)\}$, Weights of CNN \mathbf{W} , learning rate r , iteration number N_i , batch size N_t .

Output: Weights of CNN \mathbf{W} .

```

1 Initialization: Initialize  $\mathbf{W}$  with pre-trained network.
2 for  $n = 1; n \leq N_i; n++$  do
3   Random query a pair of images in dataset and
   calculate their relative pose with their poses  $\xi$ ;
4   Extract keypoint in first image and warped them to
   the second image with relative pose, generate two
   binary masks indicate the keypoint location;
5   Applying following loss with Eq.5 and 9:
6    $L_{mul} = L_{metric} + L_{mask}$ 
7   for  $k = 1; k \leq N_l; k++$  do
8     Backward propagate loss for each layer
9   end
10  Update  $\mathbf{W}$  with learning rate  $r$ 
11 end
12 return  $\mathbf{W}$ 

```

The final loss for the multi-task training is a combination of triplet loss in Eq. 5 and cross entropy in Eq. 9. These

are weighted equally. The adaptive gradient decent method, ADAM [38], is used for optimizing this combined loss. The weights are randomly initialized except for Resnet50. As commonly done in fine tuning, the learning rate for Resnet50 is set smaller than the random initialized weights (10^{-4} and 10^{-3} in our case). The margin for the triplet loss is set to 1, and weights for $[\alpha_1, \alpha_2]$ handling imbalanced classes in the weighted cross entropy is set as [0.1, 1.0].

V. EXPERIMENTS

In this section, we evaluate the effectiveness of our framework with the TUM RGB-D benchmark [1]. This benchmark provides synchronized ground truth trajectory for an RGB-D camera in various scenes. The Absolute Trajectory Error (ATE) is used as the metric for the tests. We first compared GCN with ORB, SIFT and SURF as keypoints for frame to frame motion estimation. The latter ones are commonly used features in visual tracking. Then, we incorporate GCN in a SLAM system, to evaluate against two state-of-the-art SLAM systems: ORB-SLAM2 and Elastic Fusion. The first experiment will demonstrate the quality of the feature for motion estimation compared to other features using the same frame to frame motion estimation pipeline. The second test will provide a comparison against existing, and end-to-end solutions for RGB-D based motion estimation to allow for a quantitative assessment of the performance on this task.

A. Implementation

Training and Testing Settings To properly validate the generalization ability, we used sequences from different categories in the TUM benchmark for training, *fr2*, and testing, *fr1/3*. This ensures that there is no overlap between the training and testing sequences. The list of the selected datasets are shown in Tab.I. They cover typical indoor scenes with semantic objects, e.g., desk, computer and chairs, etc. We subsample the sequences to use only every fourth image, to provide harder samples (images further apart) for training.

TABLE I: TUM DATASETS USED FOR TRAINING

Dataset	# Frame Pairs
fr2_xyz	3611
fr2_rpy	3217
fr2_desk	2219
fr2_360_hemisphere	2643
fr2_360_kidnap	1409
fr2_large_with_loop	1223
fr2_pioneer_360	826
fr2_pioneer_slam	2168
Total	17316

To generate the training data, we use high gradient points ranked by the Harris algorithm. To ensure a distribution of keypoints over the entire image and to allow highly textured regions to generate many keypoints we run the algorithm twice. First on the whole image and then on each sub-image when dividing the images into a 4×4 grid. Due to naturally occurring noise³, misalignments exist in the image planes

³e.g. noise in depth measurements, errors in the ground truth and imperfect camera calibration

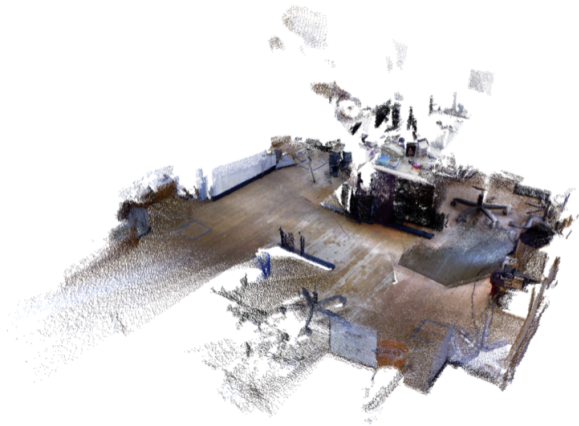


Fig. 4: Raw pointcloud reconstruction using fr1-floor and fr3-long. More can be found at: www.cas.kth.se/gcn.

TABLE II: ATE USING FRAME TO FRAME TRACKING

Dataset	GCN	ORB	SIFT	SURF
fr1_floor	0.015m	0.080m	0.073m	0.074m
fr1_desk	0.037m	0.151m	0.144m	0.148m
fr1_360	0.059m	0.278m	0.305m	0.279m
fr3_long_office	0.061m	0.090m	0.076m	0.070m
fr3_large_cabinet	0.073m	0.097m	0.091m	0.143m
fr3_nst	0.020m	0.061m	0.036m	0.030m
fr3_nnf	0.221m	-	-	-

after warping keypoints using the ground truth trajectory. To compensate for this, the KNN matching criteria is relaxed to prevent divergence in the training. For negative sample mining, only hardest samples with coordinate distances to target correspondence more than 5 pixels are used.

SLAM system After finding geometric correspondences with GCN, a lightweight keyframe based mapping system is used to compare with state-of-the-art SLAM systems. This pre-integrated system includes basic BA, pose graph optimization and vocabulary based loop closure detection. The poses of keyframes are continuously optimized during tracking as follows:

$$E_{graph} = \sum_{\langle i,j \rangle \in C} (\xi_{i,j} \boxplus \xi_j^{-1} \boxplus \xi_i)^T \Omega_{i,j} (\xi_{i,j} \boxplus \xi_j^{-1} \boxplus \xi_i) \quad (10)$$

where C is the edge set of relative poses collected in the map. \boxplus is the generalized sum operation on the Lie group manifold. Ω is information matrix proportional to the uncertainty of parameters. The loop closure detection is performed by the bag-of-words (BoW) library [5] for querying image with similarity. It is an appearance based retrieval method that converts features to a BoW vector and match using a hierarchical tree structure.

Development Environment The test platform is equipped with a GTX 1080 graphic card and i7-4790 CPU. The overall training is implemented with deep learning framework Pytorch. The BA and pose graph optimization is implemented with g2o [8] and the OpenCV libraries, where a GPU version of KNN is used for feature matching.

TABLE III: ATE USING CLOSE LOOP SYSTEM

Dataset	Frames#	Ours	ORB-SLAM2	Elastic Fusion
fr1_floor	1227	0.038m	0.036m	-
fr1_desk	573	0.029m	0.016m	0.020m
fr1_360	744	0.069m	0.213m	0.108m
fr3_long_office	2488	0.040m	0.010	0.017m
fr3_large_cabinet	984	0.097m	-	0.099m
fr3_nst	1639	0.020m	0.019m	0.017m
fr3_nnf	455	0.064m	-	-

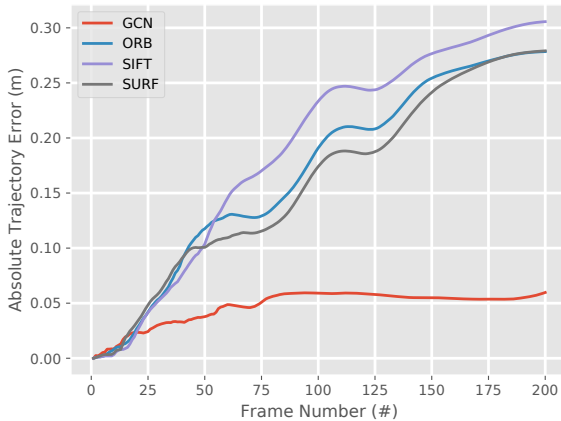
B. Evaluation of Motion Estimation without Loop Closing

We compare performance of GCN, ORB[4], SIFT[39] and SURF[40] on frame-by-frame tracking. During the test, features of different methods are extracted separately and fed into the same BA frontend for motion estimation. This frontend performs KNN matching for the resultant features and PnP with RANSAC to estimate camera motion. Reciprocal verification is performed during KNN matching. Feature number and parameters used in PnP with RANSAC are set the same for fair comparison. Thus, the only variable in this sequential motion estimate setting is the feature used.

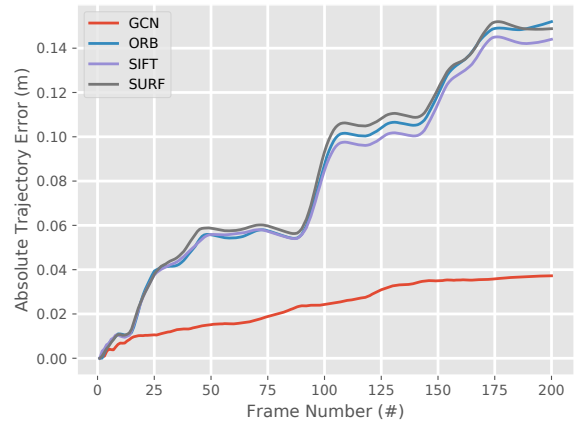
Tab. II shows the ATE for frame-to-frame tracking using the first 200 frames from each sequence. For all four methods, the number of keypoints to extract are set to 2000. We use 200 frames from each sequence to show how ATE accumulated in different scenes. Compared with other methods, GCN achieves the best performance on all sequences. Especially in scene fr3_nnf, a wall with weak texture, GCN is still able to produce trackable features while others lose track. Fig. 5 shows how ATE is changing over time for the different methods. The results show how drift is accumulated and gradually effecting the overall trajectory. It can be seen that our method rises to a stable low error while the other methods often start out similar but then it rises for a longer time before reaching a stable error.

C. Evaluation of Closed Loop System

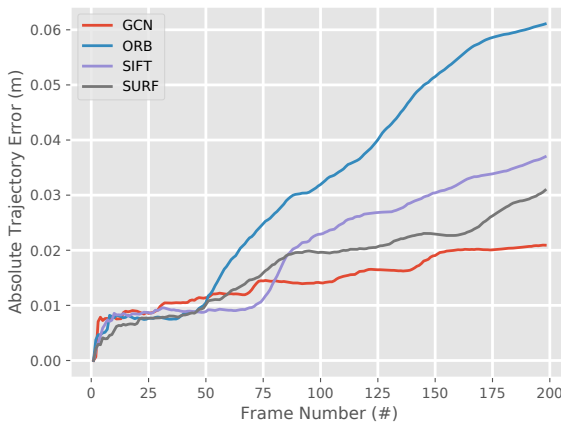
We compare our lightweight SLAM system that uses GCN with open source SLAM frameworks: ORB-SLAM2 and



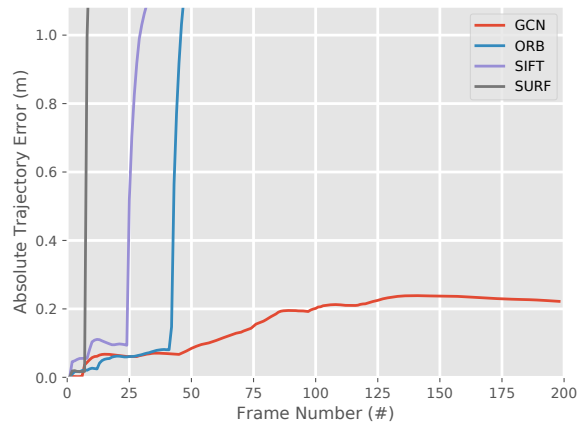
(a) fr1-360



(b) fr1-desk



(c) fr3-nst



(d) fr3-nnf

Fig. 5: Figure show the changing of ATE using different scenes during same amount of time.

Elastic Fusion, two representatives state-of-the-art methods in sparse and dense geometric reconstruction, respectively. Note that these SLAM systems have been developed more comprehensively than our naive SLAM system.

ATE using all frames from the datasets are shown in Tab.III. Our method achieves competitive results compared with the other two methods. We see that even with the longer sequences our method does not lose track. In sequence fr1_floor, Elastic Fusion lost track at places where there are missing frames, causing large displacements. Also, this scene is challenging for a fusion based method as most of existing structures are flat. Since there are rich textures on the floor, both our method and ORB-SLAM2 works well and obtained similar results. Then, in fr1_360, the ORB feature didn't work as well as GCN, it results in an observable deterioration in the ATE of ORB-SLAM2. Similarly, in fr3_large_cabinet, ORB-SLAM2 lost track when the camera comes to the back of the cabinet with a rotation. The number of keypoints dropped sharply when camera looked at the white back of the cabinet. In fr3_nnf, as shown in Section V-B, the classical

features cannot cope with this weak textured wall. Thus, both ORB-SLAM2 and Elastic Fusion failed to start the tracking at beginning.

VI. SUMMARY AND CONCLUSIONS

In this paper, we presented a unified framework for learning both detection of keypoints and descriptors for these. The framework is optimized for the task of camera motion estimation. We showed how we can generate training data by using data from SLAM benchmark datasets for which the ground truth of the camera is provided. We demonstrated how the resulting features outperform currently used features for motion estimation without loop closure and that we, even with a minimalistic implementation of loop closure optimization, achieve results that are on par with state-of-the-art methods such as ORB-SLAM2.

Our target application is a fully autonomous flying robot. In the future, we plan to improve the efficiency of GCN in terms of inference and matching. For the inference, network depth and storage consumption can be greatly reduced as

shown in Deep Compression[41] and SqueezeNet[42]. Alternative a shallower backbone network can be used at the cost of a minor loss in accuracy. For matching, feature like ORB can be matched very efficiently since the descriptor is binarized. Work in DeepBit [43] shows that a CNN is also capable of producing representative binary features. With these possible improvements, GCN has potential to be well exploited in our future real-time application.

REFERENCES

- [1] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *2012 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2012.
- [2] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Trans. on Robotics*, vol. 31, no. 5, pp. 1147–1163, Oct 2015.
- [3] R. Mur-Artal and J. D. Tardos, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Trans. on Robotics*, vol. PP, no. 99, pp. 1–8, 2017.
- [4] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Intl. Conf. on Computer Vision*, 2011.
- [5] D. Galvez-Lpez and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Trans. on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.
- [6] A. I. Comport, E. Malis, and P. Rives, "Accurate quadrifocal tracking for robust 3d visual odometry," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2007.
- [7] C. Kerl, J. Sturm, and D. Cremers, "Dense visual slam for rgb-d cameras," in *2013 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2013.
- [8] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2011.
- [9] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE Intl. Symposium on Mixed and Augmented Reality*, 2011.
- [10] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. McDonald, "Real-time large-scale dense rgb-d slam with volumetric fusion," *Intl. Journal of Robotics Research*, vol. 34, no. 4-5, pp. 598–626, 2015.
- [11] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "Elasticfusion: Real-time dense slam and light source estimation," *Intl. Journal of Robotics Research*, vol. 35, no. 14, pp. 1697–1716, 2016.
- [12] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, "Svo: Semidirect visual odometry for monocular and multicamera systems," *IEEE Trans. on Robotics*, vol. 33, no. 2, pp. 249–265, 2017.
- [13] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European Conf. on Computer Vision (ECCV)*, 2014.
- [14] S. Zagoruyko and N. Komodakis, "Learning to compare image patches via convolutional neural networks," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [15] J. Žbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 2287–2318, 2016.
- [16] J. Žbontar and Y. LeCun, "Computing the stereo matching cost with a convolutional neural network," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [17] W. Luo, A. G. Schwing, and R. Urtasun, "Efficient deep learning for stereo matching," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [18] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid, "Deepflow: Large displacement optical flow with deep matching," in *IEEE Intl. Conf. on Computer Vision (ICCV)*, 2013.
- [19] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid, "Deep-matching: Hierarchical deformable dense matching," *Intl. Journal of Computer Vision*, 2016.
- [20] L. Bao, Q. Yang, and H. Jin, "Fast edge-preserving patchmatch for large displacement optical flow," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.
- [21] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid, "Epicflow: Edge-preserving interpolation of correspondences for optical flow," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [22] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *IEEE Intl. Conf. on Computer Vision (ICCV)*, 2015.
- [23] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," *CoRR*, 2016.
- [24] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [25] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," in *Intl. Conf. on 3D Vision (3DV)*, 2016.
- [26] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *IEEE Intl. Conf. on Computer Vision (ICCV)*, 2015.
- [27] K. Tateno, F. Tombari, and I. L. and Nassir Navab, "Real-time dense monocular slam with learned depth prediction," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
- [28] N. Sünderhauf, S. Shirazi, A. Jacobson, F. Dayoub, E. Pepperell, B. Uroft, and M. Milford, "Place recognition with convnet landmarks: Viewpoint-robust, condition-robust, training-free," in *Robotics: Science and Systems*, July 2015.
- [29] T.-Y. Lin, Y. Cui, S. Belongie, and J. Hays, "Learning deep representations for ground-to-aerial geolocalization," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [30] D.-K. Kim and M. R. Walter, "Satellite image-based localization via learned embeddings," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Singapore, 2017.
- [31] H. Altwaijry, A. Veit, and S. Belongie, "Learning to detect and match keypoints with deep architectures," in *British Machine Vision (BMVC)*, 2016.
- [32] P. Agrawal, J. Carreira, and J. Malik, "Learning to see by moving," in *IEEE Intl. Conf. on Computer Vision (ICCV)*, 2015.
- [33] C. B. Choy, J. Gwak, S. Savarese, and M. Chandraker, "Universal correspondence network," in *Advances in Neural Information Processing Systems (NIPS)*, 2016.
- [34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [35] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *Intl. Journal of Computer Vision*, 2015.
- [36] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [37] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2015.
- [38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.
- [39] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Intl. Journal of Computer Vision*, 2004.
- [40] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European Conf. on Computer Vision (ECCV)*, A. Leonardis, H. Bischof, and A. Pinz, Eds., 2006.
- [41] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," *CoRR*, 2015.
- [42] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, 2016.
- [43] K. Lin, J. Lu, C.-S. Chen, and J. Zhou, "Learning compact binary descriptors with unsupervised deep neural networks," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Deep Projective 3D Semantic Segmentation

Felix Järemo Lawin, Martin Danelljan, Patrik Tosteberg,
Goutam Bhat, Fahad Shahbaz Khan, Michael Felsberg

Computer Vision Lab, Dept. of Electrical Engineering, Linköping University

Abstract. Semantic segmentation of 3D point clouds is a challenging problem with numerous real-world applications. While deep learning has revolutionized the field of image semantic segmentation, its impact on point cloud data has been limited so far. Recent attempts, based on 3D deep learning approaches (3D-CNNs), have achieved below-expected results. Such methods require voxelizations of the underlying point cloud data, leading to decreased spatial resolution and increased memory consumption. Additionally, 3D-CNNs greatly suffer from the limited availability of annotated datasets.

In this paper, we propose an alternative framework that avoids the limitations of 3D-CNNs. Instead of directly solving the problem in 3D, we first project the point cloud onto a set of synthetic 2D-images. These images are then used as input to a 2D-CNN, designed for semantic segmentation. Finally, the obtained prediction scores are re-projected to the point cloud to obtain the segmentation results. We further investigate the impact of multiple modalities, such as color, depth and surface normals, in a multi-stream network architecture. Experiments are performed on the recent Semantic3D dataset. Our approach sets a new state-of-the-art by achieving a relative gain of 7.9%, compared to the previous best approach.

Keywords: Point clouds, semantic segmentation, deep learning, scanning artifacts, hard scape

1 Introduction

The rapid development of 3D acquisition sensors, such as LIDARs and RGB-D cameras, has lead to an increased demand for automatic analysis of 3D point clouds. In particular, the ability to automatically categorize each point into a set of semantic labels, known as semantic point cloud segmentation, has numerous applications such as scene understanding and robotics. While the problem of semantic segmentation of 2D-images has gained a considerable amount of attention in recent years, semantic segmentation of point clouds has received little interest despite its significance. In this paper, we propose a framework for semantic segmentation of point clouds that greatly benefits from the recent developments in semantic image segmentation.

With the advent of deep learning, many tasks within computer vision have seen a rapid progress, including semantic segmentation of images. The key factors for this development are the introductions of large labeled datasets [2] and GPU implementations of Convolutional Neural Networks (CNNs). However, CNNs have not yet been successfully applied for semantic segmentation of 3D point clouds due to several challenges.

In contrast to the regular grid-structure of image data, point clouds are in general sparse and unstructured. A common strategy is to resort to voxelization in order to directly apply CNNs in 3D. This introduces a radical increase in memory consumption and leads to a decrease in resolution. Additionally, labeled 3D data, which is crucial for training CNNs, is scarce due to difficulties in data annotation.

In this work, we investigate an alternative approach that avoids the aforementioned difficulties induced by 3D CNNs. As our first contribution, we propose a framework for 3D semantic segmentation that exploits the advantages of deep image segmentation approaches. The point cloud is first projected onto a set of synthetic images, which are then used as input to the deep network. The resulting pixel-wise segmentation scores are re-projected into the point cloud. The semantic label for each point is then obtained by fusing scores over the different views. As our second contribution, we investigate the impact of different input modalities, such as color, depth and surface normals, extracted from the point cloud. These modalities are fused in a multi-stream network architecture to obtain the final prediction scores.

Compared to semantic segmentation methods based on 3D CNNs [17], our approach has two major advantages. Firstly, our method benefits from the abundance of the already existing data sets for image segmentation and classification, such as ImageNet [2] and ADE20K [28]. This significantly reduces, or even eliminates the need of 3D data for training purposes. Secondly, by avoiding the large memory complexity induced by voxelization, our method achieves a higher spatial resolution which enables better segmentation quality.

We perform qualitative and quantitative experiments on the recently introduced Semantic3D dataset [6]. We show that different modalities contain complementary information and their fusion significantly improves the final segmentation performance. Further, our approach sets a new state-of-the-art performance on the Semantic3D dataset, outperforming both classical machine learning methods and 3D-CNN based approaches. Figure 4 shows an example segmentation result using our method.

2 Related Work

The task of semantic point cloud segmentation has received an increasing amount of attention due to the rapid development of sensors capable of capturing high-quality 3D data. RGB-D cameras, such as the Microsoft Kinect, have become popular for robotics and computer vision tasks. While RGB-D cameras are more suitable for indoors environments, terrestrial laser scanners capture large-scale point clouds for both indoors and outdoors applications. Both RGB-D cameras and modern laser scanners are capable of capturing color in association with the 3D information using calibrated RGB cameras. Besides visualization, this additional information is highly useful for automated analysis and processing of point clouds. While color is not a necessity for our approach, it alleviates the task of semantic segmentation and enables the use of large-scale image datasets.

Most previous works [1,7,11,16,13] in 3D semantic segmentation apply a combination of (i) hand-crafted features, (ii) discriminative classifiers and (iii) spatial smoothness models. In this setting, the construction of discriminative 3D-features (i) is ar-

guably the most important task. Popular alternatives include features based on the 3D structure tensor [7,26,11,1], histogram-based descriptors [7,16,11] such as Spin Images [10] and SHOT [21], and simple color features [26,16,11]. The classifiers (ii) are often based on maximum margin methods [13,1] or employ random forests [7,11,16]. To utilize spatial correlation between semantic labels (iii), many methods apply graphical models, such as the Conditional Random Field (CRF) [26,1,13].

Recently, deep convolutional neural networks (CNNs) have been successfully applied for semantic segmentation of 2D images [15]. Their main strength is the ability to learn high-level discriminative features, which eliminates the need of hand-designed representations. The rapid progress of deep CNNs for a variety of computer vision problems is generally attributed to the introduction of large-scale datasets, such as ImageNet [2], and improved performance for GPU computing.

Despite its success for image data, the application of CNNs to 3D point cloud data [9,20,27] have been severely hindered due to several important factors. Firstly, a point cloud does not have the neighborhood structure of an image. The data is instead sparse and scattered. As a consequence, CNN-based methods resort to voxelization strategies of the underlying point cloud data to enable 3D-convolutions to be performed (3D-CNNs). Secondly, voxelization have several disadvantages, including loss of spatial resolution and large memory requirements. 3D-CNNs are therefore restricted to small volumetric models or processing data in many smaller chunks, which limits the use of context. Thirdly, annotated 3D data is extremely limited, especially for the 3D semantic segmentation task. This greatly limits the power of CNNs for semantic segmentation of generic 3D point clouds.

In contrast, our approach avoids these short comings by projecting the point cloud into dense 2D image representations, thus removing the need for voxelizations. The 2D images can then be efficiently processed using 2D convolutions. Also, performing segmentation in image space allows us to leverage well developed 2D segmentation techniques as well as large amount of annotated data.

3 Method

In this section we present our method for point cloud segmentation. The input is an unstructured point cloud and the objective is to assign a semantic label to each point. In our method we render the point cloud from different views by projecting the points into synthetic images. We render color, depth and other attributes extracted from the point cloud. The images are then processed by a CNN for image-based semantic segmentation, providing a prediction scores for the predefined classes in every pixel. We make the final class selection from the aggregated prediction scores, using all images where the particular points are visible. An overview of the method is illustrated in Figure 1. A more detailed description is provided in the following sections.

3.1 Render views

The objective of the point cloud rendering is to produce structured 2D-images that are used as input to a CNN-based semantic segmentation algorithm. A variety of information stemming from the point cloud can be projected onto the synthetic images. In

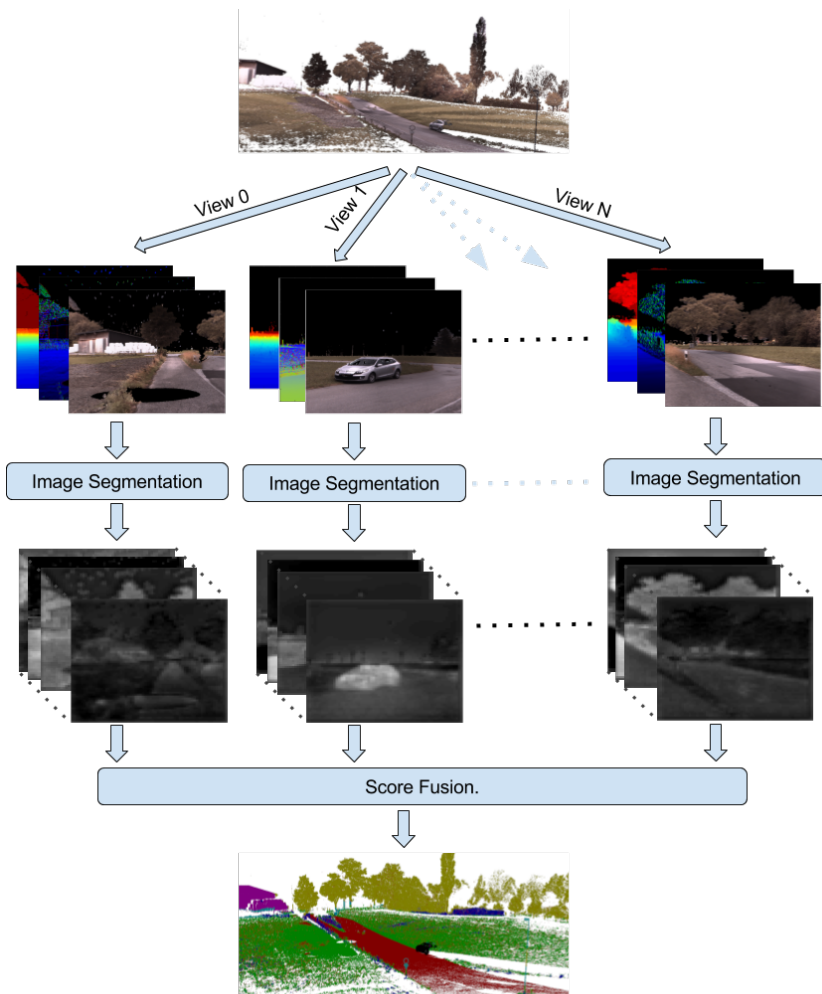


Fig. 1: An overview of the proposed method. The input point cloud is projected into multiple virtual camera views, generating 2D color, depth and surface normal images. The images for each view are processed by a multi-stream CNN for semantic segmentation. The output prediction scores from all views are fused into a single prediction for each point, resulting in a 3D semantic segmentation of the point cloud.

this work we particularly investigate the use of depth, color, and normals. However, the approach can be trivially extended to other features such as HHA [5] and other local information extracted from the point cloud. In order to map the semantic information back to the 3D points, we also need to keep track of the visibility of the projected points.

Our choice of rendering technique is a variant of point splatting [24,29], where the points are projected with a spread function into the image plane. While other rendering

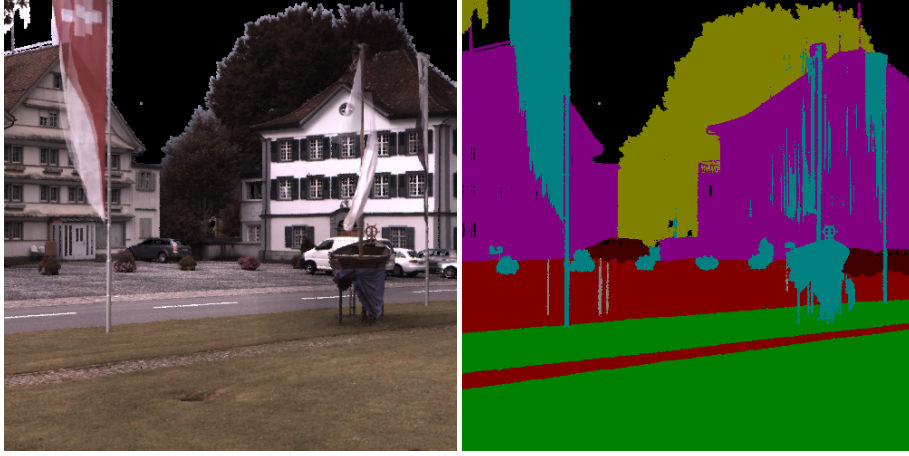


Fig. 2: Example of rendering output. Left: color image. Right: label image.

techniques, such as surface reconstruction as in [12], require demanding preprocessing steps of the point cloud in 3D space, splatting could be completely processed in image space. This further enables efficient and easily parallelizable implementations, which is essential for large-scale or dense point clouds.

Splatting-based rendering is performed by first projecting each 3D-point \mathbf{x}_i of the point cloud into the image coordinates \mathbf{y}_i of a virtual camera. The projected points are stored along with their corresponding depth values z_i and feature vectors \mathbf{c}_i . The latter can include, e.g., the RGB-color and normal vector of the point \mathbf{x}_i . The projection of a 3D-point is distributed by a Gaussian point spread function in the image plane,

$$w_{i,j} = G(y_i - p_j, \sigma^2). \quad (1)$$

Here, $w_{i,j}$ is the contributed weight of point x_i to pixel j in the projected image. It is obtained by evaluating an isotropic Gaussian kernel G with scale σ^2 at the pixel location p_j . In order to reduce computational complexity, the kernel is truncated at a distance r . However, point spread functions, which originate from different surfaces, may still intersect in the image plane. Thus, the visibility of the projected points needs to be determined to avoid contributions of occluded surfaces. Moreover, the sensor data may contain significant foreground noise, such as scanning artifacts, which complicates this task. The challenge is to exclude the contribution from the noise and the occluded surfaces in the rendering process.

In traditional splatting [29], the resulting pixel value is obtained from the weighted average of the point spread functions in an accumulated fashion, using the weights $w_{i,j}$. If the depth of a new point significantly differs from the current weighted average, the pixel depth is either re-initialized with the new value if the point is closer than a specific threshold, or discarded if it is further away [29]. However, this implies that the resulting pixel value depends on both the threshold value and the order in which the

points are projected. Furthermore, noise in the foreground will have significant impact on the resulting images, as it is always rendered.

Similar to the method proposed in [19], we perform mean-shift clustering [24] of the projected points in each pixel with respect to the depth z_i weighted with $w_{i,j}$ using a Gaussian kernel density estimator $G(d, s^2)$, where s^2 denotes the kernel width. Starting from the depth value $d_i^0 = z_i$ for each point $i \in I_j$ that contributes to the current pixel j , $I_j = \{i : \|p_j - y_i\| < r\}$, the following expression is iterated until convergence

$$d_i^{m+1} = \frac{\sum_{i \in I_j} w_{i,j} G(d_i^m - z_i, s^2) z_i}{\sum_{i \in I_j} w_{i,j} G(d_i^m - z_i, s^2)}. \quad (2)$$

The iterative process determines a set of unique cluster centers $\{d_k\}_1^K$ from the converged iterates $\{d_i^N\}_{i \in I_j}$. The kernel density of cluster center d_k is given by,

$$v_k = \frac{\sum_{i \in I_j} w_{i,j} G(d_k - z_i, s^2)}{\sum_{i \in I_j} w_{i,j}}. \quad (3)$$

We rank the clusters with respect to the kernel density estimates and the cluster centers,

$$s_k = v_k + \frac{D}{d_k}. \quad (4)$$

Here, the weight D rewards clusters that are near the camera. It is set such that foreground noise and occluded points are not rendered. We chose the optimal cluster as $\bar{k} = \arg \max_k s_k$ and set the depth value of pixel j to the corresponding cluster center $d_{\bar{k}}$. The feature value is calculated as the weighted average, where the weight is determined by the proximity to the chosen cluster,

$$\mathbf{c}_{\bar{k}} = \frac{\sum_{i \in I_j} w_{i,j} G(d_{\bar{k}} - z_i, s^2) \mathbf{c}_i}{\sum_{i \in I_j} w_{i,j} G(d_{\bar{k}} - z_i, s^2)}. \quad (5)$$

Since the indices $i \in I_j$ of the contributing points i are stored, it is trivial to map the semantic segmentation scores produced by the CNN back to the point cloud itself.

An example of the rendering output is shown in Figure 2.

3.2 Deep Multi Stream Image Segmentation

Following the current success of deep learning algorithms we deploy a CNN-based algorithm for performing semantic segmentation on the rendered images. We consider using multiple input modalities, which are combined using a multi-stream architecture [23]. The predictions from the streams are fused in a sum layer, as proposed in [4]. The full multi stream network can thus be trained end-to-end. However, note that our pipeline is agnostic to the applied image semantic segmentation approach.

In our method, each stream is processed using a Fully Convolutional Network (FCN) [15]. However, as previously mentioned, any CNN architecture can be employed. The FCN is based on the popular VGG16 network [22]. The weights in each stream are initialized by pre-training on the ImageNet dataset [2]. In this work, we investigate different combinations of input streams, namely color, depth, and surface normals. While

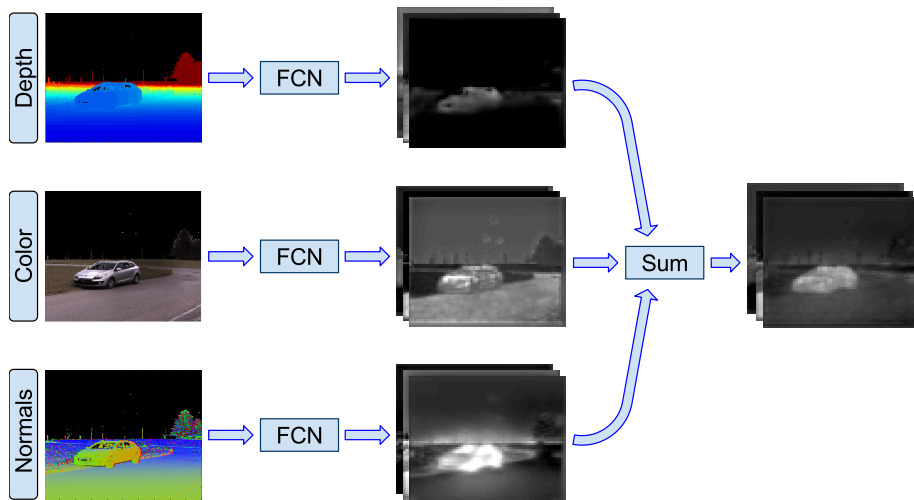


Fig. 3: Illustration of the proposed multi-stream architecture for 2D semantic segmentation. Each input stream is processed by a Fully Convolutional Network[15]. The prediction scores from each stream are summed to get the final prediction.

the RGB-stream naturally benefits from pre-training on ImageNet, this is also the case for the depth stream. Previous work [3] has shown that a 3-channel jet colormap representation of the depth image better benefits from pre-training on RGB datasets, such as ImageNet. Finally, we also consider surface normals as input to a separate network stream. For this purpose, we deploy an efficient algorithm for approximate normals computation, which is based on direct differentiation of the depth map.

3.3 Score fusion

The deep network outputs a prediction score for each class for every pixel in the image. The scores from each rendered view are mapped to the corresponding 3D points using the indices $i \in I_j$ as described in section 3.1. We fuse the scores by computing the sum over all projections. Finally, the points are assigned the labels corresponding to the largest sum of scores.

4 Experiments

4.1 Dataset

We conduct our experiments on the dataset Semantic3D [6], which provides a set of large scale 3D point clouds of outdoor environments. The point clouds were acquired by a laser scanner and include both urban and rural scenes. Colorization was performed using a cube map generated from a set of high-resolution camera images. In total, the

dataset contains 30 separate scans and over 4 billion 3D-points. The points are labeled with 8 different semantic classes: man-made terrain, natural terrain, high vegetation, low vegetation, buildings, hard scape, scanning artifacts, and cars.

4.2 Experimental setup

View selection In order to fully cover the point clouds in the rendered views, we collect images by rotating the camera 360° around a fix vertical axes. For each 360° rotation, we use 30 camera views at equally spaced angles. For each point cloud, we generate four such scans with different pitch angles and translations of the camera, resulting in a total of 120 camera views. To maintain a certain amount of contextual information, we remove images where where more than 10% of the pixels have a depth less than five meters. Furthermore, images with less than 5% coverage were discarded.

Network setup and training For the training we generated ground truth label images by selecting the most commonly occurring label in the optimal cluster from section 3.1. An example is shown in Figure 2. In addition to the 8 provided classes, we also included a 9th background class to label empty pixels, i.e pixels without any intersecting point spread functions. We generated training data from the training set provided by Semantic3D [6], consisting of 15 point clouds from different scenes. Our training data set consists of 3132 labeled images including color, jet visualization of the depth, and surface normals.

We investigate the proposed multi stream approach using color, depth and surface normals streams as input. In order to determine the contribution of each input stream we also evaluate network configurations with a single stream. Since some point clouds may not have color information we also investigate a multi stream approach without the color stream. All network configurations are listed in table 1.

Table 1: Network configurations with input streams in the left column

	RGB	D	N	RGB+D+N	D+N
Color	X			X	
Depth jet		X		X	X
Surface normals			X	X	X

All network configurations were trained using the same training parameters. We trained for 45 epochs with a batch size of 16. The initial learning rate was set to 0.0001 and divided by two every tenth epoch. Following the recommendations from [14], we used a momentum of 0.99. The networks were trained using MatConvNet [25].

4.3 Results and Discussions

We evaluated our method for the different network configurations on the reduced test set provided by Semantic3D. The test set consists of four point clouds, containing 80

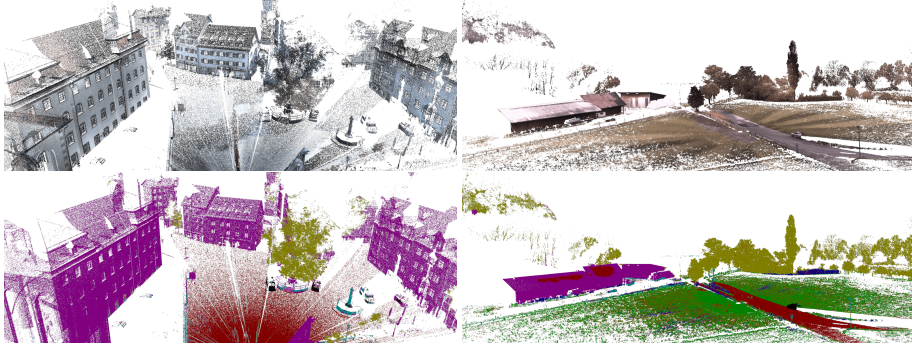


Fig. 4: Qualitative results. Top: input point clouds. Bottom: Segmentation output using our proposed **RGB+D+N** network.

million points in total. All points are assigned a class label j , which is compared to the ground truth label i . A confusion matrix C is constructed, where each entry c_{ij} denotes the number of points with the ground truth label i that are assigned the label j . The quantitative measure provided by the benchmark [6] is the intersection over union for each class i , given by

$$\text{IoU}_i = \frac{c_{ii}}{c_{ii} + \sum_{j \neq i} c_{ij} + \sum_{k \neq i} c_{kj}}. \quad (6)$$

The over all accuracy is also provided and is given by

$$\text{IoU} = \frac{\sum_i c_{ii}}{\sum_j \sum_k c_{jk}}. \quad (7)$$

The evaluation results are shown in table 2. The single-stream network with RGB and surface normals as input performs significantly better than the single-stream depth network. However, the three streams seem to provide complementary information, and give a significant gain in performance when used together. Our best multi-stream approach significantly improves over the previous state-of-the-art method [8]. Also our multi-stream approach without the color stream obtains results comparable to the previous state-of-the-art, showing that our method is applicable even if color information is absent. Interestingly, even our single-stream approaches with only RGB or surface normals as input achieves a remarkable gain compared to the 3D-CNN based VoxNet [6]. Figure 4 shows some qualitative results on the test set using our multi-stream **RGB+D+N** network.

Note that we are using a simple heuristic for generating camera views, and a basic segmentation network trained on limited data. Yet, we obtain very promising results. Replacing these blocks with better alternatives should improve the results even further. However, this is outside the scope of this paper.

Table 2: Benchmark results on the reduced test set in Semantic3D [6]. IoU for categories (1) man-made terrain, (2) natural terrain, (3) high vegetation, (4) low vegetation, (5) buildings, (6) hard scape, (7) scanning artefacts, (8) cars.

	Avg IoU	OA	IoU1	IoU2	IoU3	IoU4	IoU5	IoU6	IoU7	IoU8
TML-PCR[18]	0.384	0.740	0.726	0.730	0.485	0.224	0.707	0.050	0.000	0.150
DeepNet[6]	0.437	0.772	0.838	0.385	0.548	0.085	0.841	0.151	0.223	0.423
TLMC-MSR[8]	0.542	0.862	0.898	0.745	0.537	0.268	0.888	0.189	0.364	0.447
Ours RGB	0.515	0.854	0.759	0.791	0.720	0.335	0.857	0.209	0.123	0.326
Ours D	0.262	0.662	0.281	0.468	0.395	0.179	0.763	0.006	0.001	0.000
Ours N	0.511	0.846	0.815	0.622	0.679	0.164	0.903	0.251	0.186	0.470
Ours RGB+D+N	0.585	0.889	0.856	0.832	0.742	0.324	0.897	0.185	0.251	0.592
Ours D+N	0.543	0.872	0.839	0.736	0.717	0.210	0.909	0.153	0.204	0.574

5 Conclusion

We propose an approach for semantic segmentation of 3D point clouds that avoids the limitations of 3D-CNNs. Our approach first projects the point cloud onto a set of synthetic 2D-images. The corresponding images are then used as input to a 2D-CNN for semantic segmentation. Consequently, the segmentation results are obtained by re-projecting the prediction scores to the point cloud. We further investigate the impact of multiple modalities in a multi-stream deep network architecture. Experiments are performed on the Semantic3D dataset. Our approach outperforms existing methods and sets a new state-of-the-art on this dataset.

References

1. Anguelov, D., Taskar, B., Chatalbashev, V., Koller, D., Gupta, D., Heitz, G., Ng, A.Y.: Discriminative learning of markov random fields for segmentation of 3d scan data. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA. pp. 169–176 (2005)
2. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR09 (2009)
3. Eitel, A., Springenberg, J.T., Spinello, L., Riedmiller, M., Burgard, W.: Multimodal deep learning for robust rgb-d object recognition. In: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. pp. 681–687. IEEE (2015)
4. Feichtenhofer, C., Pinz, A., Zisserman, A.: Convolutional two-stream network fusion for video action recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. pp. 1933–1941 (2016), <http://dx.doi.org/10.1109/CVPR.2016.213>
5. Gupta, S., Girshick, R., Arbeláez, P., Malik, J.: Learning rich features from rgb-d images for object detection and segmentation. In: European Conference on Computer Vision. pp. 345–360. Springer (2014)
6. Hackel, T., Savinov, N., Ladicky, L., Wegner, J.D., Schindler, K., Pollefeys, M.: Semantic3d. net: A new large-scale point cloud classification benchmark. arXiv preprint arXiv:1704.03847 (2017)

7. Hackel, T., Wegner, J.D., Schindler, K.: Fast semantic segmentation of 3d point clouds with strongly varying density. In: ISPRS Annals - ISPRS Congress, Prague (2016)
8. Hackel, T., Wegner, J.D., Schindler, K.: Fast semantic segmentation of 3d point clouds with strongly varying density. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Prague, Czech Republic 3, 177–184 (2016)
9. Huang, J., You, S.: Point cloud labeling using 3d convolutional neural network. In: International Conference on Pattern Recognition (ICPR) (2016)
10. Johnson, A.E., Hebert, M.: Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Trans. Pattern Anal. Mach. Intell.* 21(5), 433–449 (1999)
11. Kähler, O., Reid, I.D.: Efficient 3d scene labeling using fields of trees. In: IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013. pp. 3064–3071 (2013)
12. Kazhdan, M., Hoppe, H.: Screened poisson surface reconstruction. *ACM Transactions on Graphics (TOG)* 32(3), 29 (2013)
13. Kim, B., Kohli, P., Savarese, S.: 3d scene understanding by voxel-crf. In: IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013. pp. 1425–1432 (2013)
14. Liu, W., Rabinovich, A., Berg, A.C.: Parsenet: Looking wider to see better. arXiv preprint arXiv:1506.04579 (2015)
15. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3431–3440 (2015)
16. Martinovic, A., Knopp, J., Riemenschneider, H., Gool, L.J.V.: 3d all the way: Semantic segmentation of urban scenes from start to end in 3d. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015. pp. 4456–4465 (2015)
17. Maturana, D., Scherer, S.: Voxnet: A 3d convolutional neural network for real-time object recognition. In: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. pp. 922–928. IEEE (2015)
18. Montoya-Zegarra, J.A., Wegner, J.D., Ladický, L., Schindler, K.: Mind the gap: modeling local and global context in (road) networks. In: German Conference on Pattern Recognition. pp. 212–223. Springer (2014)
19. Ogniewski, J., Forssén, P.E.: Pushing the limits for view prediction in video coding. In: 12th International Conference on Computer Vision Theory and Applications (VISAPP'17). Scitepress Digital Library, Porto, Portugal (February-March 2017)
20. Qi, C.R., Su, H., Nießner, M., Dai, A., Yan, M., Guibas, L.J.: Volumetric and multi-view cnns for object classification on 3d data. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. pp. 5648–5656 (2016)
21. Salti, S., Tombari, F., di Stefano, L.: SHOT: unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding* 125, 251–264 (2014)
22. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR abs/1409.1556 (2014)
23. Simonyan, K., Zisserman, A.: Two-stream convolutional networks for action recognition in videos. In: Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada. pp. 568–576 (2014), <http://papers.nips.cc/paper/5353-two-stream-convolutional-networks-for-action-recognition-in-videos>
24. Szeliski, R.: *Computer Vision: Algorithms and Applications*. Springer-Verlag New York, Inc. (2010)

25. Vedaldi, A., Lenc, K.: Matconvnet – convolutional neural networks for matlab. In: *Proceeding of the ACM Int. Conf. on Multimedia* (2015)
26. Wolf, D., Prankl, J., Vincze, M.: Fast semantic segmentation of 3d point clouds using a dense CRF with learned parameters. In: *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*. pp. 4867–4873 (2015)
27. Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J.: 3d shapenets: A deep representation for volumetric shapes. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. pp. 1912–1920 (2015), <http://dx.doi.org/10.1109/CVPR.2015.7298801>
28. Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., Torralba, A.: Scene parsing through ade20k dataset. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017)
29. Zwicker, M., Pfister, H., Van Baar, J., Gross, M.: Surface splatting. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. pp. 371–378. ACM (2001)