2017 IEEE International Conference on Robotics and Automation (ICRA)
Singapore, May 29 - June 3, 2017

Efficient Path Planning for Mobile Robots with Adjustable Wheel Positions

Freya Fleckenstein Christian Dornhege Wolfram Burgard

Abstract— **Efficient navigation planning for mobile robots in complex environments is a challenging problem. In this paper we consider the path planning problem for mobile robots with adjustable relative wheel positions, which further increase the navigation capabilities. In particular we account for changes of these relative wheel positions during planning time, thus fully leveraging the capabilities of the robot. Whereas these additional degrees of freedom increase flexibility, they introduce a more challenging planning problem. The approach proposed in this paper is built upon a search-based planner. We describe how to flexibly integrate joint angle changes in the path planning process and furthermore propose a representation of the robot configuration that substantially reduces the computational burden. In addition, we introduce search guidance heuristics that are particularly useful in environments in which a robot is required to pass over obstacles, such as on agricultural fields. An extensive evaluation on simulated and real-world data with our BoniRob agricultural robot demonstrates the efficiency of our approach.**

I. Introduction

Constructing feasible plans that enable robots to navigate autonomously in a safe and efficient manner has been a field of research in robotics for decades. We consider wheeled robots with advanced capabilities that enable them to change their intrinsic configuration, such as the BoniRob system depicted in Fig. 1. The wheels of this robot can be rotated individually so that the robot can move in arbitrary directions and orientations. In addition, the wheels are mounted on rotatable lever arms around a vertical axis (see Fig. 2). As a result, the BoniRob is able to adjust its wheel positions to pass over or along obstacles in confined areas. We focus on cases, where due to mechanical constraints, changes of the intrinsic configuration are not possible during navigation. For example, the BoniRob would incur damage due to high loads on the mechanism when the lever arms are rotated while driving. Robots with controllable intrinsic configurations obviously are more flexible and envisioned to have advantages in in complex environments with many obstacles, in fields with plant rows or even in rescue scenarios. Incorporating these joints as part of the path planning process guarantees that all possible paths are found and therefore ensures that the full potential of the platform is used.

Adjustable joints introduce additional degrees of freedom (DoF) compared to robots that are only able to change their position and orientation. The higher dimensionality of the planning space makes the problem harder as the number of states is considerably increased. Common approaches are sampling-based or search-based planning. Many sampling-based approaches assume that different states can be connected directly. As we consider robots for which driving and changing specific joint angles at the same time is undesirable, we cannot adopt this assumption, since only a subset of the state values is allowed to change in one motion. Also, customized cost functions are important. For example, unless necessary to reach a target, one wants to avoid the robot going backwards. A flexible search-based approach is well suited in this case.

Fig. 1: The BoniRob agricultural robot with high ground clearance and wheels attached to adjustable arms, here collecting data on a leek field used in the evaluation. Each of the arms can be moved separately around a vertical axis thereby changing the relative wheel positions. Moving all arms to the sides of the robot leads to a large track width, enabling it to pass over extended obstacles. Moving all arms to the front and rear leads to a small track width, enabling it to get through narrow passages.

In this paper we introduce a novel path planning approach that is able to efficiently plan for robots with adjustable wheel positions. This is made possible by the following contributions. We achieve a significantly smaller search space through a concise representation that effectively treats the additional degrees of freedom introduced by the adjustable joints as sets of configurations instead of individual configurations. A key insight here is that these joints only enable different motions, but do not move the robot itself. This allows us to summarize all configurations that enable the same motion into a single state during search. Furthermore, we present heuristics particularly suitable for environments that require the robot to pass over obstacles.

We show the effectiveness of our approach in extensive experiments in simulated and real-world environments evaluating the proposed improvements—namely an efficient state space representation and informed search heuristics. In particular, we point out the necessity of considering the full capabilities of a robot in a demanding environment.

All authors are with the University of Freiburg, Department of Computer Science, Germany. This work has partially been supported by the European Commission under the grant number H2020-ICT-644227-FLOURISH.

II. Related Work

Robots with an adjustable intrinsic configuration are common in the field of search and rescue robotics as the additional DoF allow the robot to overcome obstacles. These are mostly utilized by tele-operation or reactively. Tracked robots with so-called flippers adjust these based on current sensor data to determine whether changing the intrinsic robot configuration is necessary [13]. This does not allow the robot to plan ahead, if a change in configuration is necessary before a critical situation arises.

There are several sampling-based approaches that deal with a high number of DoF by sampling a subset of the configuration space, such as for example rapidly-exploring random trees [10] and variants thereof [9, 6] or probabilistic road maps [7]. Many sampling-based approaches assume that any state is reachable from any other state in a continuous manner, if an obstacle-free transition exists. This is not the case for us, as the robot is not able to change its joint angles and drive at the same time. In addition, producing high-quality plans is important for navigation. It has recently received interest for sampling-based planning [3], but is well understood for search-based planning.

Therefore we aim for a search-based planning approach that is often used for path planning. The simplest method is to perform a 2d Dijkstra search on a grid [2]. State-of-the-art approaches are able to plan for more DoF, for example using state lattices [14]. We also base our approach on this concept. Search-based planning allows for flexible solutions, e.g., footstep planning for four-legged robots [16], navigation planning with 3d collision checks [5] or path planning for parking cars [11]. Besides navigation, search-based planning with state lattices has also been applied to manipulation tasks [1].

The main question we consider in this paper is how to efficiently plan a path for a robot with adjustable joint angles that introduce additional DoF. One way to tackle this issue is by using adaptive dimensionalities for planning. Gochev et al. propose to plan for a set of configuration parameters that are crucial to a resulting path [4] They compute a path in this low-dimensional configuration space and then deduce the values of the remaining configuration parameters from this path. In our case, the joint angles of the arms as well as the robot pose are relevant for the feasibility of a motion, so that a lower-dimensional configuration space does not exist. Another idea in the context of adaptive dimensionalities is to adaptively choose for which part of the configuration space to plan in the current planning phase, which is determined by the distance to the start or goal [11] or also including obstacles [15, 8]. In our case, we might encounter obstacles on the way and thus cannot only focus on the start and goal areas. In addition, while navigating on a field, obstacles in the form of crops might always be close, so that such an approach would always choose the highest dimensionality.

An important technique for speeding up search-based planning are informative heuristics. A commonly used heuristic is the Euclidean distance to the goal. This does not take into account any constraints of the robot and thus often greatly underestimates the true cost. A better estimate is the freespace heuristic [11] that pre-computes the minimal cost based on the possible motions for a robot under the assumption that there are no obstacles. Another heuristic that is often used is the Dijkstra heuristic, induced by the Dijkstra algorithm [2]. For example, Hornung et al. propose to run a Dijkstra search on a map with obstacles inflated by the inner circle of the robot [5]. However, for a robot with high ground clearance, the center being located above an obstacle does not imply a collision, as it is able to pass over an obstacle. Our proposed Wheel Dijkstra heuristic addresses this issue.

III. Problem Statement

We plan for a ground vehicle navigating in a 3-dimensional environment. First, we describe the configuration space of the robot and then the action model that forms valid transitions in the configuration space. We represent the pose of the robot in the world by a rigid body motion relative to a fixed coordinate frame in 3-dimensional space $SE(3)$. This defines the extrinsic configuration space of the robot.

We consider robots that are also able to change their intrinsic configuration. In our case the robot can adjust its relative wheel positions by rotating its arms (see Fig. 2) and thereby can fit through passages or pass over obstacles. We include such joints that are relevant for the feasibility of a robot motion in the configuration space and call the space that describes the controllable configurations of these joints the *intrinsic configuration space* of the robot $\mathcal{C}_R$. The full configuration space is then given by $\mathcal{C} = SE(3) \times \mathcal{C}_R$.

We represent the environment by a traversability map. This is 2.5d grid map that contains elevation and traversability information. The *valid configuration space* $\mathcal{C}_f \subseteq \mathcal{C}$ contains all configurations $c \in \mathcal{C}$ that are safe for the robot according to this map, i. e., the robot is collision-free and stable. As we consider a ground vehicle, the height, roll and pitch in the robot pose cannot be actively controlled. We thus define the *actively controllable configuration* space $\mathcal{C}_a = SE(2) \times \mathcal{C}_R$. Each element in the actively controllable configuration space $\mathcal{C}_a$ is induced by an element in the valid configuration space $\mathcal{C}_f$. Note that a change in the intrinsic configuration of the robot can cause a change in the robot pose in $SE(3)$. For example, when the robot turns an arm onto a bump the attitude and elevation of the robot changes.

We study cases where simultaneous driving and changing of joint angles is undesirable or physically impossible. For example, changing the arm angles of the BoniRob while driving can lead to serious damage in the control mechanism. Therefore there are two kinds of valid motions $\mathcal{M}$ that represent transitions in $\mathcal{C}_a$. Changes in $SE(2)$ correspond to driving, thus we call them *drive motions*. Changes in $\mathcal{C}_R$ correspond to changes in the arm joint angles that we call *arm motions*. A motion is valid, if it does not require a wheel to cross an untraversable cell, and the ground clearance of the robot allows its body and arms to pass over any obstacles. A goal configuration $c_g \in \mathcal{C}_a$ is reachable from a start configuration $c_s \in \mathcal{C}_a$, if there exists a cohesive sequence

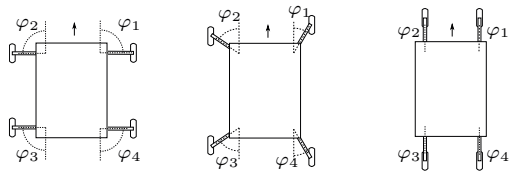


Fig. 2: Schematic of the BoniRob showing its body and the four arms in different joint configurations in a top-down view.

$\pi = (m_i)_{i \in 1,\dots,k}$ of valid motions m_i that starts at c_s and ends at c_g.

We define a cost function $u : \mathcal{M} \to \mathbb{R}^+, m \mapsto u(m)$ that models the time it takes to execute motion m. Additionally, the cost function models user preferences, e. g., we prefer the robot to drive forwards and not backwards. For a sequence π of motions, we define the cost as $cost(\pi) = \sum_{i=1}^{k} u(m_i)$.

We then specify our planning problem as follows. Given a start configuration $c_s \in \mathcal{C}_f$ and a goal configuration $c_g \in \mathcal{C}_f$, determine the induced configurations $c_{s2D}, c_{g2D} \in \mathcal{C}_a$. Find a sequence of valid transitions $\pi = (m_1, m_2, \dots, m_k)$ with minimal $cost(\pi)$ that starts at c_{s2D} and ends at c_{g2D}.

IV. Planning with Adjustable Wheel Positions

In this section, we present our approach for path planning with adjustable relative wheel positions. This is a challenging problem as the additional DoF need to be accounted for during planning to be able to find plans that require these joints for completeness. We aim for a general and flexible formulation that allows us to model robot-specific motions and costs. Therefore, we follow a search-based planning approach that easily optimizes for custom cost functions, and allows us to express drive and arm motions separately.

We first illustrate, how we represent a search space in a state lattice [14] that discretizes states and constructs valid motions that connect these states. They form a graph that is searched with any search algorithm such as A*. We build on formulations for state lattices in $SE(2)$ and then present our extensions to model planning with adjustable wheel positions in state lattices in the controllable configuration space $\mathcal{C}_a$. Finally, we introduce search guidance heuristics that tackle the shortcomings of common path planning heuristics.

A. State Lattice Planning in $SE(2)$

Common approaches for state lattice planning in $SE(2)$ discretize the configuration space, i. e., $SE(2)$. The resulting states are connected with so-called motion primitives and hereby define the search space for planning. Motion primitives represent drive motions that the robot is able to execute and are constructed in a way that the repeated application of motion primitives covers the complete space. The search space is incrementally built during planning. Motion primitives are only applied to a state if they represent a valid motion in $\mathcal{C}_f$. The motion primitives for our robot are shown in Fig. 3. A search in this space results in a plan $\pi \in SE(2)$ consisting of motion primitives.

B. State Lattice Planning with Adjustable Wheel Positions

A state lattice in $SE(2)$ accounts for the pose of a robot in the environment, but arm motions changing joint angles

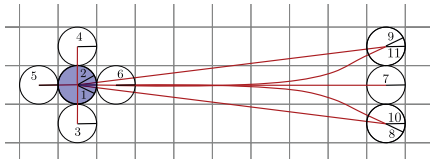


Fig. 3: Motion primitives for our robot shown on a grid. The current pose is highlighted in blue. Possible drive motions are indicated by red lines. The outcomes are shown as numbered robot poses (illustrated by circles for the position and a line for the orientation).

are not considered. To represent the abilities of the robot and enable finding paths even in complex environments, we extend the $SE(2)$ state lattice to include joint angles in $\mathcal{C}_R$. We model *arm motions* and *drive motions* separately. For drive motions we reuse the motion primitives as defined for a state lattice in $SE(2)$. The state space representation also discretizes $SE(2)$ in the same way. In addition, we include the intrinsic robot configuration $\mathcal{C}_R$ in the state lattice and represent configuration changes of the robot by arm motions.

Here, the first step is to integrate the joint angles into the state lattice to get the valid controllable configuration space $\mathcal{C}_a = SE(2) \times \mathcal{C}_R$. This leads to $n = \dim(\mathcal{C}_R)$ additional dimensions in the configuration space. A straightforward way to represent the intrinsic configuration of a robot is to store a single angle per joint. The intrinsic robot configuration is then given by a vector $(\varphi_1, \dots, \varphi_n)^T$.

The second step is to represent the robot motions that change joint angles. One of these arm motions sets a single joint angle to a different configuration value. Like for motion primitives in $SE(2)$, we also test if an arm angle change leads to a configuration in $\mathcal{C}_f$. Otherwise the motion is not applicable. These motions only affect entries in $\mathcal{C}_R$. This is in accordance with our condition that joint angles cannot be changed while driving. This representation increases the dimensionality of the search space by $\dim(\mathcal{C}_R)$.

We therefore introduce a representation that instead of single joint angles uses intervals of joint angles, thus representing sets of configurations in a single state in the state lattice together with the pose of the robot. An intrinsic robot configuration representation then contains an interval $I_i = \left[\varphi_i^{\min}, \varphi_i^{\max}\right]$ for each joint and is given by $(I_1, \dots, I_n)^T$. A state in this representation is valid, if and only if all joint configurations it represents are valid at the given pose, i. e., all combinations of angles in the intervals are valid.

To understand why this works, consider that changes in $\mathcal{C}_R$ do not cause major changes in the pose of the robot. Additionally, in many situations there are multiple joint angle configurations that allow a certain drive motion to be applied. Therefore, it is not necessary to know the exact values in $\mathcal{C}_R$ as long as they lead to valid motions.

However, with a representation over sets of possible configurations, it is necessary to adapt the notion of applying a motion to retain soundness. There are two reasons for the angle interval of a specific joint to change. While arm motions increase intervals, drive motions potentially decrease the represented valid intrinsic configurations, as illustrated in Fig. 4. *Arm motions* expand the joint angle intervals to both sides by a fixed increment, if the corresponding arm rotation

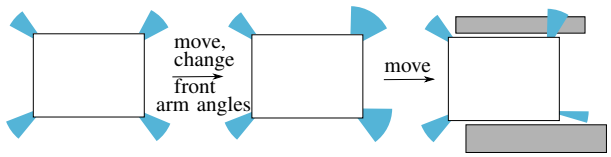


Fig. 4: Changing arm angle intervals due to arm motions and obstacles when moving. The intervals of valid arm angles are shown in blue. Obstacles are shown in gray. The robot starts with the intervals in the leftmost state. It first moves in freespace and changes the angles of its front arms, thus their intervals increase. When driving further towards obstacles the intervals decrease as part of them becomes invalid.

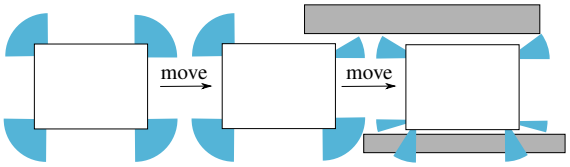


Fig. 5: An environment where splitting intervals happens. Part of the intervals become invalid as the wheels at the end of the arms of the robot would collide with the obstacle. The shown interval parts are valid even when the arms extend over the obstacle as their ground clearance is high enough.

is valid. Applying such a motion thus generates only one successor state representing all combinations of joint angles that are possible results of the applied motion.

Joint angle intervals are also updated when applying a *drive motion*. This is the second reason that intervals can change. If a drive motion is not valid for part of an interval, we remove this part in the resulting state. This state has to represent the resulting set of configurations. The example in Fig. 5 shows that in some scenarios this leads to the interval being split. As our approach supports only single intervals, it cannot represent all possible resulting configurations in a single state. Therefore, in such a case, we create multiple successor states. A new successor is created for the combination of all resulting interval parts for each arm, e.g., in the example in Fig. 5, we get four new states, for the angles of two arms being split in two intervals each. Note that the extreme case of splitting each interval in all individual discretized parts is equivalent to the representation with single arm angles. This is unlikely to happen unless the environment is highly complex. Therefore, in most cases, using the single arm angle representation produces a large number of successor states that the interval representation summarizes into a single state.

C. Heuristics

While the previous section addressed a reduction of the search space to speed up planning, here we aim to guide the search with informative heuristics. Therefore, we present heuristics well suited for robots with high ground clearance, where contact with the ground only occurs with small parts of the robot. In particular we aim for admissible heuristics that underestimate the true cost as these allow the planner to find optimal paths. The commonly used Euclidean distance is not suited for our target applications as it neither considers the cost of the motion primitives nor the structure of the environment. Instead, we use the freespace heuristic [11].

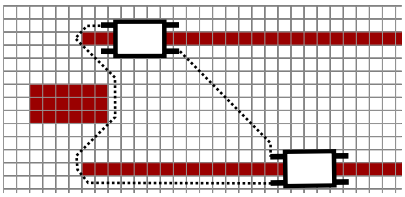


Fig. 6: Example of the Wheel Dijkstra heuristic for the rear wheels of the BoniRob in an 8-neighborhood. Untraversable cells are marked in red. Note that considering only a single point of the robot, e.g., one wheel can greatly underestimate the actual cost.

It precomputes minimal costs in a freespace environment, while considering the actual cost of motion primitives.

The freespace heuristic is a better estimate than the Euclidean distance, but it also does not consider the environment. In environments with many obstacles and little open space, it greatly underestimates the true cost. In such environments, it is imperative to take the structure into account. A heuristic that does this is the Dijkstra heuristic that computes shortest paths from the robot center to all cells in a 2d grid map of the environment and thus gives an estimate of an obstacle-free path that the robot could take. However, it disregards the orientation of the robot and, in our case, intrinsic configuration changes. For robots with high ground clearance, a Dijkstra heuristic on the robot center is not informative, as the robot center is not on the ground.

Instead, we compute a Dijkstra heuristic for the wheels (see Fig. 6) that naturally have ground contact and therefore require traversable locations. For each wheel, given a planning query, we compute the cell in the traversability map that it occupies in the goal configuration. Then, once per planning query, we perform a single source shortest path search on the traversability map with Dijkstra's algorithm from that cell and store the costs. During planning, the heuristic value for a specific wheel is then determined by a fast look-up.

In comparison to a search in the full controllable configuration space of the robot, searching for paths in the 2d space of the traversability map does not require much time, even when performing this search for all wheels. Fig. 6 illustrates the importance of considering all wheels for more realistic estimates, as distances traveled by single wheels to their respective goal positions can differ greatly. The results are informative, as the search avoids collisions of the wheels and represents the minimal distance that each wheel has to travel. We acquire a cost estimate for the complete robot by maximizing over all wheel distances and call this heuristic the Wheel Dijkstra heuristic. We also combine the Wheel Dijkstra heuristic with the freespace heuristic by maximizing.

V. Experiments

In order to show the performance of our planner for robots with adjustable relative wheel positions, we ran extensive experiments on simulated and real-world data. We first investigate the influence of our proposed interval representation for the intrinsic robot configuration. Afterwards we present a comparison of the planner performance with various search guidance heuristics. As an evaluation platform, we model the BoniRob with its four rotatable arms. Each arm is

constrained to an angle between $0°$ and $90°$. The intrinsic configuration space is thus given by $\mathcal{C}_R = \mathcal{C}_l^4$ where $\mathcal{C}_l = [0°, 90°]$ represents the configuration of a single arm.

A *freespace* environment without obstacles serves as the baseline for our experiments. In addition, we created two simulation environments shown in Fig. 7. The *field* environment represents an agricultural field with plant rows. The *rocks* environment is cluttered with tight passages that the robot has to circumvent or rocks that it has to pass over to navigate. It is made intentionally hard to drive the planner and different representations to their limits. Here, adjusting the wheel positions is likely to be required for finding a path. Furthermore, we evaluate on the 3d map of a *leek* field built from real-world data that is shown in Fig. 8 (see Fig. 1 for an impression of the field). This allows us to compare planning on real-world data to the simulation environments.

In all environments, we uniformly sampled 25 poses and constructed planning queries between each pair of poses. In the *leek* environment, we manually aligned pose orientations with the field to generate realistic robot poses. In the *freespace* and *field* environment, we ran the planner on queries where poses were more than 0.5 m and less than 20 m apart and chose a timeout of 30 s. As planning is harder in the *rocks* environment, we ran queries for poses with at least 0.5 m and at most 10 m distance and increased the timeout to 3 min. In the *leek* environment, we allowed for a distance between 0.5 m and 20 m and used a timeout of 90 s. Our implementation uses the search-based planning library with the Anytime Repairing A* as a search algorithm [11, 12]. We set the initial weighting factor to $\varepsilon = 5$. The experiments were run on an Intel i7 4 GHz machine.

We are particularly interested in the time it took the planner to find a first plan, as this allows a robot to start navigating. We plot a cumulative histogram of solved planning queries over time, where, for example, an entry of 50% at 10 s means that after 10 s 50% of all queries returned a plan. Ideally a graph here rises up quickly as this indicates queries being solved fast and reaches a high percentage of solved queries at the timeout. We also investigate the sub-optimality, to determine if the plans are reasonable. The sub-optimality value γ for a plan guarantees that it has at most γ times the cost of the optimal plan [12]. To compare algorithms we plot the percentage of queries that resulted in a plan with sub-optimality of γ or better after a third of the timeout. Ideally a large percentage of plans has a small γ.

A. Evaluation of Configuration Representations

In this section, we analyze the efficiency gain of using joint angle intervals instead of single joint angles for the representation of the intrinsic configuration of the robot. In addition, we compare with a representation that does not allow to change joint angles at all. This serves as a baseline algorithm as its search space is notably smaller, but—although incomplete—should still be able to find plans in most cases. Since preliminary experiments showed that a combination of the Euclidean, freespace and Wheel Dijkstra heuristic leads to the best performance, we used this. We

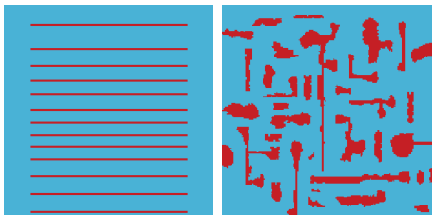


Fig. 7: The environments for our experiments are both 30 m×30 m. Untraversable cells are marked in red, traversable cells in blue. Left: The *field* environment representing a field with plant rows, which are low enough for the robot to pass over them. Right: The *rocks* environment, where changing arm angles multiple times in a single plan can be necessary.

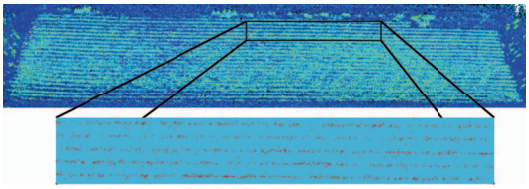


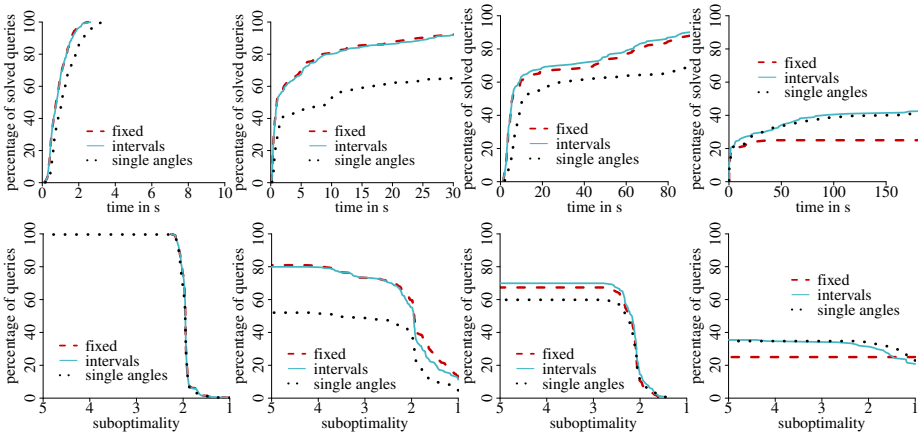
Fig. 8: The real-world data recorded on a leek field that is represented in the *leek* environment (top). The bottom shows the corresponding traversability map that is automatically generated from this data.

ran the planner on the queries of all environments in three different configuration space representations: Once without changing the arm angles during the search (*fixed*), thus representing a configuration space of $SE(2)$, once representing the intrinsic configuration with angle intervals (*intervals*) and once with individual arm angles (*single angles*).

Freespace: We evaluate in the *freespace* environment as a baseline to determine the overhead resulting from planning with joint angles. Fig. 9(a) shows that the *intervals* representation leads to a similar performance as the *fixed* representation, while *single angles* is slightly worse. In this simple environment, we observe no overhead with our proposed *intervals* representation despite the four additional DoF resulting from the adjustable relative wheel positions. Almost all queries also reach a sub-optimality of 2 or better.

Field: This represents a crop field with plant rows as a typical environment for an agricultural robot. Fig. 9(b) illustrates that planning with *intervals* is notably more efficient than using *single angles*. At the timeout, the planner employing the *interval* representation found paths for about 95% of the planning queries, while planning with *single angles* resulted in only 65% of solved planning queries. Again we observe that planning in the more expressive *intervals* representation is just as efficient as planning without joint angles (*fixed*). The sub-optimality for *intervals* is also on par with the *fixed* representation at about 3 for more than 70% of the queries, where at the evaluation point of a third of the timeout 80% of the queries were solved. Again there is a notable difference to *single angles*.

Leek: We validate our results from simulated environments in the *leek* environment based on real-world data. Fig. 9(c) shows a similar performance as in the *field* environment, i. e., planning with *intervals* is more efficient than planning with *single angles* and in this case even slightly



(a) *freespace* (b) *field* (c) *leek* (d) *rocks*

Fig. 9: Performance of the planner using different configuration space representations in the test environments. Plotted are cumulative histograms of the number of solved planning queries over the time until the first plan was found in the top row and the percentage of queries resulting in a certain sub-optimality in the bottom row.

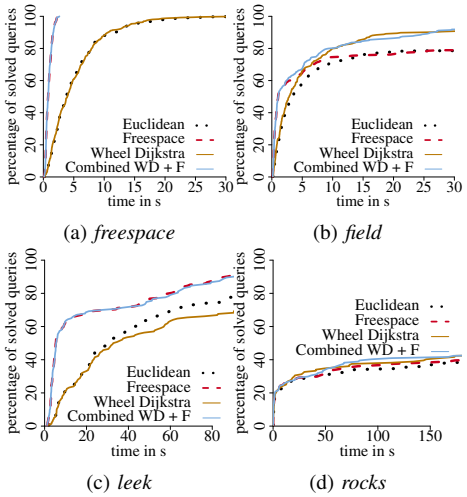
better than planning with *fixed* angles. Sub-optimality here is similar for all representations as the environment is more narrow and thus any found plan is already close to optimal. As the simulated *field* environment has similar properties as the *leek* environment, this confirms the real-world applicability of our configuration representation.

Rocks: The *rocks* environment provides a hard test case enforcing difficult situations. Changing arm angles is a common requirement for finding plans in this environment. Fig. 9(d) demonstrates that planning with *fixed* arm angles is not sufficient any more due to its incompleteness. In particular we see that after about 50 s no new plans are found. Both the *single angles* and the *intervals* representation continue to find plans and at the timeout solved about twice as many queries. The sub-optimality is 1 for *fixed* arm angles for the queries that had solutions. These are mainly the simpler queries that did not require arm reconfiguration actions. The cost for the arm motions is large for our robot in comparison to drive motions, so that the planner prevents these from being executed. Nevertheless there were 1.2 reconfiguration actions on average for *single angles* and *intervals*. This clearly shows that planning with arm angles is necessary in constrained environments.

In addition here we also investigated, how many discretized configurations are represented in a single search state of the *intervals* representation. On average there were 25.2 configurations captured in the intervals of that representation that would be individual search states in the *single angles* representation. This explains, why in the environments, where arm motions are for most queries not necessary, the *intervals* representation shows a performance similar to planning without arm angle changes.

B. Evaluation of Heuristics

To compare the performance of the heuristics, we ran our planner in all environments and evaluated different heuristics. We chose the *intervals* configuration representation. We tested the planner with the Euclidean distance heuristic serving as the baseline and compared with the combination



(a) *freespace* (b) *field* (c) *leek* (d) *rocks*

Fig. 10: Performance of the planner using different heuristics. Plotted are cumulative histograms of the number of solved planning queries over the time until the first plan was found.

of the Euclidean distance and the freespace heuristic, the Wheel Dijkstra heuristic, or the combined Wheel Dijkstra and freespace heuristic (denoted as *Combined WD + F*). We focus the discussion on the percentage of solved queries as the performance differences with regard to the sub-optimality between algorithms were similar to the solved queries.

Freespace: The *freespace* environment provides a measure for how severely heuristics underestimate the true cost in an empty environment. The freespace heuristic is the optimal heuristic for planning with a state lattice planner in this environment. It is thus not surprising that in Fig. 10(a) the freespace heuristic and combinations with it perform notably better than others. Planning with the Wheel Dijkstra heuristic or only the Euclidean heuristic is slower as they both underestimate the true cost of *drive motions* resulting from orientation changes and user preferences.

Field: As we see in Fig. 10(b), initially (up to 5 s) the performance of the freespace heuristic is on par with the *Combined WD + F* heuristic. These are easier planning queries that require driving either along the side of the

field, i. e., mostly in freespace, or along a row of the field, i. e., where obstacles do not influence the preferred path. Larger times indicate more complex queries that, for example, require changing rows. Here, the Wheel Dijkstra heuristic performs best, as it is designed for scenarios like this. The *Combined WD + F* heuristic here covers both scenarios performing similar to the freespace heuristic for shorter queries and similar to the Wheel Dijkstra heuristic for more complex ones. Without the Wheel Dijkstra heuristic only about $80\,\%$ of planning queries were solved. The Wheel Dijkstra heuristic (also in combination with the freespace heuristic) was able to solve about $95\,\%$ of planning queries.

Leek: In the *leek* environment based on real-world data, a row of crops is not one continuous obstacle, but contains gaps between crops that allow a wheel to pass through, although the complete robot cannot. This explains why in Fig. 10(c) we observe that the performance of the Wheel Dijkstra heuristic alone is not as good as in the *field* environment. The freespace heuristic leads to a notably better performance. A reason for this is that rows were quite narrow and when the robot was driving it was constrained by the row to straight-line movements that are modeled well by the freespace heuristic. Nevertheless, the combination thereof with the Wheel Dijkstra heuristic still performed similarly.

Rocks: In this environment evaluated in Fig. 10(d) there are no major differences between heuristics. The *Combined WD + F* heuristic leads to slightly more solved planning queries at the timeout, followed by the Wheel Dijkstra, freespace and the Euclidean heuristic. Here the path planner had to plan complex maneuvers that are not modeled well by any heuristic.

VI. Conclusion

We presented an approach for path planning with adjustable relative wheel positions. Our planner efficiently handles the increased number of degrees of freedom resulting from planning for the additional joint angles. In order to achieve this, we introduced a novel configuration representation for changeable joint angles. Furthermore, we introduced search guidance heuristics that are particularly useful in complex environments with many obstacles. In extensive experiments, we demonstrated that the proposed configuration representation by intervals allows us to plan including joint angles from the intrinsic configuration without a notable decrease in performance compared to planning in $SE(2)$. Note that planning only in $SE(2)$ is incomplete. This is also shown experimentally in constrained environments, where taking joint angles into account during planning is necessary to find paths. The evaluation of the heuristics indicated different behavior of heuristics depending on the environment. However, the combined Wheel Dijkstra and freespace heuristic performs at least as well as other tested heuristics in all environments and situations.

References

[1] B. J. Cohen, S. Chitta, and M. Likhachev. Search-based planning for manipulation with motion primitives. In *Int. Conf. on Robotics & Automation (ICRA)*, 2010.

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

[3] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *Int. Conf. on Robotics & Automation (ICRA)*, 2015.

[4] K. Gochev, B. J. Cohen, J. Butzke, A. Safonova, and M. Likhachev. Path Planning with Adaptive Dimensionality. In *Int. Symposium on Combinatorial Search (SoCS)*, 2011.

[5] A. Hornung, M. Phillips, E. G. Jones, M. Bennewitz, M. Likhachev, and S. Chitta. Navigation in three-dimensional cluttered environments for mobile manipulation. In *Int. Conf. on Robotics & Automation (ICRA)*, 2012.

[6] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Robotics: Science and Systems (RSS)*, 2010.

[7] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[8] D. Kim, Y. Choi, T. Park, J. Y. Lee, and C. Han. Efficient path planning for high-dof articulated robots with adaptive dimensionality. In *Int. Conf. on Robotics & Automation (ICRA)*, 2015.

[9] J. J. Kuffner and S. M. Lavalle. RRT-Connect: An efficient approach to single-query path planning. In *Int. Conf. on Robotics & Automation (ICRA)*, 2000.

[10] S. M. Lavalle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. Technical report, 1998.

[11] M. Likhachev and D. Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *Int. J. of Robotics Research*, 28(8):933–945, 2009.

[12] M. Likhachev, G. J. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Proc. of the Conf. on Neural Information Processing Systems (NIPS)*, 2003.

[13] K. Ohno, S. Morimura, S. Tadokoro, E. Koyanagi, and T. Yoshida. Semi-autonomous control system of rescue crawler robot having flippers for getting over unknown-steps. In *Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

[14] M. Pivtoraiko and A. Kelly. Efficient constrained path planning via search in state lattices. In *Int. Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS)*, 2005.

[15] N. Vahrenkamp, C. Scheurer, T. Asfour, J. J. Kuffner, and R. Dillmann. Adaptive motion planning for humanoid robots. In *Int. Conf. on Intelligent Robots and Systems (IROS)*, 2008.

[16] M. Zucker, J. Andrew, B. Christopher, G. Atkeson, and J. Kuffner. An optimization approach to rough terrain locomotion. In *Int. Conf. on Robotics & Automation (ICRA)*, 2010.