Advanced Techniques for Autonomous Navigation in Precision Agriculture

Freya Fleckenstein

Technische Fakultät
Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Dissertation zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften

Betreuer: Prof. Dr. Wolfram Burgard

Advanced Techniques for Autonomous Navigation in Precision Agriculture

Freya Fleckenstein

Dissertation zur Erlangung des akademischen Grades Doktor der Naturwissenschaften
Technische Fakultät, Albert-Ludwigs-Universität Freiburg

Dekan	Prof. Dr. Roland Zengerle
Erstgutachter	Prof. Dr. Wolfram Burgard
Zweitgutachter	Prof. Dr. Cédric Pradalier
Drittgutachter	Prof. Dr. Joschka Boedecker
Tag der Disputation	02. November 2022

Abstract

A decrease in farmers and the rural exodus call for increased automation in agriculture to feed the growing world population. Furthermore, sustainable and efficient farming technologies are more important than ever to be able to produce a sufficient amount of food. Thus, agriculture is moving from uniform field treatment with large machines towards treatment of individual plants, also called precision farming, using autonomous robots. Two goals of precision farming are to reduce the use of chemicals and thus relieve the burden of chemicals on the environment, and to increase the crop yield by treating each plant as needed.

For any robot to be able to perform a variety of tasks on the field autonomously, it requires a reliable and robust navigation approach for agricultural environments. This poses several challenges: the environment changes quickly and drastically in the context of agriculture due to varying weather conditions and fast plant growth. This makes it hard for a robot to reliably extract semantic information from sensor data. Furthermore, agricultural environments such as fields and greenhouses are quite narrow, and finding and executing safe and efficient paths there is difficult. In this thesis, we present building blocks for a navigation approach suitable for precision farming applications that tackle these challenges.

Various tasks in the context of navigation on a field, such as localization or mapping, employ the position of crops or crop rows as a distinct feature and thus need a reliable detection thereof. Depending on the lighting conditions and the size of plants, either camera or lidar sensors are more suitable for extracting vegetation features. We provide a representation of vegetation features that has the potential to make crop and crop row detection methods independent of the input sensor modality by being able to capture information from both sensor types. We run a crop row detection on the resulting vegetation feature maps and develop a quality measure for the detections that is based on their feature support. This enables us to filter out unreliable detections in applications such as mapping, localization or weed detection based on crop rows. We perform experiments using a crop-row-based localization and show that we are able to increase the robustness and avoid localization pose divergence by applying our quality measure to filter out unreliable detections.

In narrow environments such as fields and greenhouses, movement flexibility is important for navigating efficiently. We therefore consider two elements for path planning that impact the flexibility in motions. First, we factor in the configuration of robot joints that impact possible movement. Second, we take the ground clearance of a robot into account, which enables it to pass over low obstacles. We evaluate a path planning approach that makes use of joint configurations such as manipulator arm angles or adjustable wheel positions to increase the motion possibilities. Our evaluation includes the planning time

and the path efficiency. Additionally, we investigate how a heuristic for planning with high ground clearance affects path planning performance. We perform experiments in simulated and real world environments and test the limits of this planning approach in a particularly challenging environment with many obstacles. The results show that the path planner is able to efficiently incorporate the joints into the planning problem. We also demonstrate the necessity of including these joint configurations, as reaching some planning goals takes considerably more planning time if the joints cannot be adjusted, or they may not be reachable at all. We also establish that the studied planning approach is well suited for real world application in agricultural environments.

Lastly, path execution plays a crucial role in an efficient and safe navigation approach. We present a local planning method to efficiently execute a computed global path with a robot that has independently steerable omnidirectional wheels. An advantage of omnidirectional wheels is that they provide great motion flexibility. However, the wheel angles often have underlying constraints, such as a maximum steering velocity, limiting the speed at which the wheels are able to turn. If these constraints are not considered during motion command computation, the robot may diverge from the planned path or it needs to stop to adjust its wheel angles. We introduce a method to represent common steering constraints in a compact manner and show two techniques to integrate them into local planning, resulting in motion commands that satisfy all constraints. We demonstrate the efficiency of our approach in simulation and real world experiments. In the results, we see that path execution accuracy can be retained despite the additional constraints. At the same time, our approach is able to substantially reduce the path execution time.

The methods presented in this thesis contribute towards a flexible, robust and efficient autonomous navigation approach for precision agriculture. We hope to hereby advance sustainable farming technologies that are critical to ensure sufficient food production for the growing world population.

Zusammenfassung

Die Abnahme an Landwirten und die zunehmende Landflucht erfordern erhöhte Automatisierung in der Landwirtschaft, um die wachsende Weltbevölkerung zu ernähren. Zusätzlich sind nachhaltige und effiziente Methoden für die Agrarwirtschaft wichtiger denn je, um genügend Nahrung zu produzieren. Daher bewegt sich die Landwirtschaft weg von einheitlicher Feldbehandlung mit großen Maschinen. Die Tendenz steigt zur Behandlung einzelner Pflanzen, auch Präzisionslandwirtschaft genannt, unter Verwendung autonomer Roboter. Ziel ist es, die Verwendung von Chemikalien zu reduzieren und dadurch die Umwelt zu entlasten, sowie den Ertrag zu erhöhen, indem jede Pflanze genau die Behandlung erhält, die sie benötigt.

Für die Ausführung verschiedener Aufgaben auf dem Feld benötigt ein Roboter einen verlässlichen und robusten Navigationsansatz für landwirtschaftliche Umgebungen. Dies stellt uns vor verschiedene Herausforderungen: Die Umgebung auf einem Feld ändert sich schnell, durch wechselnde Wetterbedingungen sowie durch das Wachstum der Pflanzen. Dadurch wird es einem Roboter erschwert, aus Sensordaten verlässliche semantische Informationen zu generieren. Des Weiteren finden sich auf Feldern und in Gewächshäusern häufig enge Räume, was es schwierig macht, einen sicheren und effizienten Pfad zu einem gegebenen Ziel zu kalkulieren und auszuführen. In dieser Arbeit stellen wir Bausteine für einen Navigationsansatz vor, der für die Anwendung in der Präzisionslandwirtschaft geeignet ist.

Diverse Aufgaben im Kontext der Navigation auf dem Feld, wie beispielsweise Lokalisierung oder Kartierung, verwenden die Position von Nutzpflanzen oder Pflanzenreihen, und benötigen daher für diese eine akkurate Erkennung. Abhängig von Lichtverhältnissen und von der Pflanzengröße können entweder Kamera- oder LiDAR-Daten bessere Informationen für die Extraktion von Vegetationsmerkmalen bieten. Wir stellen eine Repräsentation für Vegetationsmerkmale vor, die das Potential hat, Methoden zur Nutzpflanzen- und Pflanzenreihenerkennung unabhängig von der Art des Eingabesensors zu machen, indem sie Informationen aus beiden Sensortypen erfasst. Wir führen auf den resultierenden Vegetationskarten eine Pflanzenreihenerkennung aus und entwickeln ein Qualitätsmaß für diese Erkennungen, basierend darauf, wie gut eine Erkennung von den Vegetationsmerkmalen unterstützt wird. Dadurch sind wir imstande, unzuverlässige Erkennungen herauszufiltern, was in verschiedenen Anwendungen wie beispielsweise Kartierung, Lokalisierung oder Unkrauterkennung nützlich ist. In den Experimenten verwenden wir als Anwendungsszenario eine Lokalisierung basierend auf Pflanzenreihen. Wir zeigen, dass wir durch die Anwendung eines Filters, der auf unserem Qualitätsmaß basiert, die Robustheit erhöhen und eine Divergenz der Lokalisierungspose vermeiden können.

In engen Umgebungen wie Feldern und Gewächshäusern ist Flexibilität in Bewegungen wichtig, um effizient navigieren zu können. Für die Pfadplanung betrachten wir dabei

zwei Elemente, die eine wichtige Rolle für Bewegungsflexibilität spielen. Zum einen berücksichtigen wir die Konfiguration mechanischer Gelenke, die mögliche Bewegungen beeinflussen. Zum anderen beachten wir die Bodenfreiheit eines Roboters, die es ihm ermöglicht, über niedrige Hindernisse zu fahren. Wir evaluieren einen Pfadplanungsansatz, der Gelenkkonfigurationen wie etwa Manipulatorarmwinkel oder anpassbare Radpositionen in die Planung mit einbezieht. Dadurch werden die Bewegungsmöglichkeiten eines Roboters erweitert. In die Auswertung beziehen wir die Planungszeit und die Pfadeffizienz mit ein. Zudem untersuchen wir, inwiefern die Verwendung einer Heuristik für Pfadplanung mit hoher Bodenfreiheit die Leistung des Pfadplaners beeinflusst. Wir führen Experimente in simulierten und realen Umgebungen durch und testen die Einschränkungen dieser Methode in einer besonders anspruchsvollen Umgebung mit vielen Hindernissen. Die Ergebnisse zeigen, dass der Pfadplaner die Gelenkkonfigurationen effizient integriert und nutzt. Wir demonstrieren die Notwendigkeit, diese Gelenkkonfigurationen zu berücksichtigen, da manche Planungsziele sonst erst nach sehr viel längerer Planungszeit oder sogar überhaupt nicht erreicht werden können. Zudem zeigen wir, dass der untersuchte Pfadplanungsansatz gut für die Anwendung in realen landwirtschaftlichen Umgebungen geeignet ist.

Die Pfadausführung spielt ebenfalls eine entscheidende Rolle in einem effizienten und sicheren Navigationsansatz. Wir stellen eine Methode vor, um einen globalen Pfad mit einem Roboter mit omnidirektionalen Rädern effizient auszuführen. Der Vorteil von omnidirektionalen Rädern ist, dass sie sehr flexible Bewegungen ermöglichen. Allerdings haben die erreichbaren Radwinkel oft dennoch Beschränkungen, wie beispielsweise eingeschränkte Lenkgeschwindigkeit, d.h. die Räder können sich nur mit einer gewissen Geschwindigkeit drehen. Wenn diese Einschränkungen in der Berechnung von Bewegungsbefehlen nicht berücksichtigt werden, kann es passieren, dass der Roboter vom geplanten Pfad abweicht oder stehen bleiben muss, um die Radwinkel entsprechend anzupassen. Wir präsentieren eine Methode, um übliche Lenk-Einschränkungen in einer kompakten Art und Weise zu darzustellen. Wir zeigen außerdem zwei Techniken, wie sie in einen lokalen Planer integriert werden können, so dass die berechneten Bewegungsbefehle alle Beschränkungen einhalten. Wir demonstrieren die Effizienz unseres Ansatzes in Simulation und in einem realen Szenario. In der Auswertung sehen wir, dass die Genauigkeit der Pfadausführung trotz zusätzlicher Einschränkungen erhalten werden kann. Gleichzeitig kann unser Ansatz die Zeit, die zur Pfadausführung benötigt wird, beträchtlich verkürzen.

Die Methoden, die in dieser Arbeit beschrieben werden, bieten Beiträge zu einem flexiblen, robusten und effizienten autonomen Navigationsansatz für die Präzisionslandwirtschaft. Wir hoffen, damit nachhaltige Technologien in der Landwirtschaft voranzubringen, die notwendig sind, um die Nahrungsmittelproduktion für die wachsende Weltbevölkerung sicherzustellen.

Acknowledgments

I would like to thank my supervisor, all my colleagues, friends, and family for their continued help, support and encouragement in the past years of working on my PhD. My special thanks goes to my supervisor Wolfram Burgard for giving me the opportunity to pursue my PhD at the Autonomous Intelligent Systems lab, for providing a great environment for my research, the time and freedom to conduct said research, and his guidance and advice.

I also want to give special thanks to my friends and colleagues Christian Dornhege and Wera Winterhalter for their invaluable ideas and contributions in interesting discussions, their help with implementation issues and various administrative challenges, the hard work they put into our joint papers, and for the fun we had at work and in after-work activities. Thank you also for the effort you put into our experiments, and to Wera's dad for driving us and the BoniRob to vegetable fields at an ungodly hour several times so we could collect data. Thanks in particular to Wera Winterhalter for our joint venture and her continuous support.

Thanks also goes to Michael Ruhnke for pointing out that the BoniRob is not driving quite as smoothly as you would want it to even before I started my PhD, and to Cédric Pradalier for providing the initial idea on how to fix it. Thank you also for your continued interest in my research and the contribution to our local planning method and the resulting publication.

Thank you also to all my other colleagues from AIS for fun and interesting chats in the kitchen, and to colleagues from the Flourish project for their work in making the project a huge success.

I also want to thank Susanne Bourjaillat, Tatjana Ferl, Evelyn Rusdea and Michael Keser for their assistance with administrative and computer related problems. A big thank you also goes to Noha Radwan, Tim Welschehold, Bastian Steder, Brigitte Fleckenstein, Jonas Fuchs and, once more, Wera Winterhalter, for providing constructive feedback to earlier versions of this thesis.

Finally, I want to thank my friends and family for being there for me in happy and in demanding times, enabling me to push for deadlines and to relax when work was done. Thank you to my parents for supporting my decisions and believing in me no matter what. A big thank you also goes to Jonas for encouraging and motivating me on the hard days and sharing my enthusiasm on the good ones.

This work has been partially supported by the European Commission under the grant number H2020-ICT-644227-FLOURISH.

Contents

Chapter 1

Introduction

A growing world population and the rural exodus increase the need for sustainable automated farming technologies with low impact on the environment. In the past decades, treatment methods such as fertilizing and weed control of crop fields and orchards have shifted from a uniform approach towards individual plant treatment. These methods for treating only parts of the field or trees and only where it is needed are consolidated under the term *precision farming* [14]. They result in saving resources such as herbicides, pesticides, and fertilizer, and by lowering the application thereof also reduce the chemical burden on the environment. Additionally, administering these chemicals only where needed has the potential to increase harvesting yield, as too much fertilizer or herbicides can harm the crop plants [3, 47].

Treatment of individual plants is tedious, time consuming and costly when done manually. Thus, automation in agriculture has been rising from tractors with simple row guided applications for uniform mechanical weeding [55, 62, 84] to fully automated robots that are able to perform complex tasks such as individually spraying plants [8], harvesting fruit [70], or stamping weeds into the ground [92]. A key component for a robot performing any of these tasks without human supervision or intervention is a fully autonomous navigation approach. In this thesis, we present methods that contribute to such a navigation approach for precision farming.

1.1 Challenges for Robots in Agricultural Environments

The domain of precision agriculture poses novel challenges for autonomous navigation that are quite different from those that robots face in urban or indoor environments, which have been explored more extensively [15, 16, 40, 58, 77, 91, 94]. Urban and indoor environments tend to have a clear semantic structure, defined, e.g., by buildings, walls, streets, lamp poles, doors, etc. This structure is easily recognizable and mostly remains unchanged over the course of several weeks or even years. On an agricultural field, detecting and recognizing any semantics is a hard task, since the environment can change within a few hours due to weather conditions or within a few days due to fast growing plants that change their size and shape entirely. In urban and indoor environments, certain fixed structures such as walls, lamp poles, doors or street signs are employed to estimate the robot pose and the semantics of the scene [20, 75, 77, 91]. In indoor environments, additional infrastructure such as bluetooth or wifi emitters is also often used for pose esti-

Figure 1.1: This picture shows our agricultural robot BoniRob on a production vegetable field. It has a high ground clearance and is able to adjust its wheel positions by changing the angles of the lever arms to which the wheels are attached. We use it as an experiment platform to test our approaches in real world scenarios.

mation [1, 4, 95]. While there are sometimes buildings or trees near a field, the commonly used landmarks and infrastracture are not prevalent and thus cannot be used for estimating the semantics of the scene. The only structure available on most crop fields is defined by the crop rows. Tasks that require or benefit from semantic structure, such as localization, mapping, or obstacle avoidance, therefore often make use of the position of individual crops or crop rows [5, 7, 51, 90], for which a robust detection method is needed. Depending on the lighting conditions and the size of plants, different sensor modalities are suitable for detecting vegetation. For small plants in sunlight, a high-resolution camera is better able to capture vegetation, while large plants at dawn are more easily recognizable in lidar data. In order to be flexible regarding the lighting conditions and plant size, a crop or crop row detection method that is independent of the input sensor modality is desirable. An additional challenge is detecting when there are no crops or crop rows visible in the sensor data, as it is often the case near the end of the field. Most crop row detection methods will still produce a detection, even when the sensor data does not show any crop rows [7, 66, 89]. These false detections can easily mislead a localization or row following approach.

Early approaches of automation in farming used a uniform treatment of the field, which requires traversing it entirely. However, for precision farming, the robot may only need to go to specific points in the field. It thus needs to be able to find a path from its current position to a given goal position. Crop fields and greenhouses are narrow environments, which makes it difficult to find a feasible path to a goal position. It is therefore important to consider robot joints that enable certain movements to ensure the greatest possible flexibility for a path planner. This could include the joints of a manipulator arm for harvesting, or joints responsible for changing the track width or wheelbase of a robot, which is possible for a variety of robots such as the Ted robot [67], TREKTOR [81], the robots proposed by Kim et al. [50] and Karamipour et al. [46], or the BoniRob (see Figure 1.1). Taking these joints into account during planning leads to additional degrees of freedom,

which potentially increases the planning time considerably [38]. As an autonomous robot should perform its task efficiently, another challenge is to treat these additional degrees of freedom in a way that minimizes planning time consumption while the resulting path is still efficient.

Once a path has been found to reach a goal position in the field, it has to be executed in a safe, accurate and efficient manner. For accurate and efficient path execution, a robot with omnidirectional wheels provides great flexibility. However, even omnidirectional wheels have some underlying constraints, such as a maximum steering velocity, limiting the speed at which the wheels are able to turn [43]. This especially holds for large, heavy vehicles. These limitations have to be considered when a motion command is computed in a local planner to ensure that the computed global path is executed in an efficient manner. The complexity of these steering limitations increase the planning difficulty.

1.2 Contributions

In this thesis, we present novel techniques for different key tasks in autonomous navigation in agricultural settings. We propose an approach to increase the robustness of crop-row-based applications by providing a vegetation feature representation that is independent of the input sensor type and by evaluating the quality of detected crop rows with respect to the extracted vegetation data. We summarize a planning method from earlier work that allows us to include additional robot joints and takes into account the ground clearance of a robot. We extensively evaluate it with regards to real world applicability and test its limits in a cluttered environment. Lastly, we show how steering constraints for omnidirectional wheels can be formulated and integrated into a local planner to increase path execution efficiency while retaining accuracy. This section outlines the thesis and summarizes its main contributions.

Sensor Independent Vegetation Feature Representation

As discussed before, a robust crop or crop row detection is crucial for several tasks in precision agriculture. Depending on the type and size of crop plants, different sensor modalities are better suited for vegetation feature detection. Small plants in broad daylight are more easily recognizable in camera data, while large overlapping plants in low lighting conditions are better distinguishable in lidar data. Crop row detection methods often rely directly on the raw input data from a sensor, thus lacking the flexibility to adapt to fundamentally different crop fields that require different sensors to gain useful information. In Chapter 2, we propose a representation of vegetation features that can capture data extracted from lidar or camera data, enabling crop row detection methods that are independent of the specific sensor type used to collect the data.

Estimating Crop Row Detection Quality

Approaches that use crop row detections, such as localization, mapping or row following, often rely on the detection robustness and accuracy. Crop row detection methods usually return a set of lines that best explains the sensor data as crop rows, without regard for the quality of the detection [7, 66, 89]. Therefore, the produced crop row detections can be incorrect if insufficient crop row structure is visible in the sensor, e. g., at the edge of the field or when there is high weed pressure. This potentially misleads methods that use these detections in further processing. In the second part of Chapter 2, we thus present a novel approach to estimate the quality of a crop row detection based on its support in our vegetation feature representation. Using this quality measure, we define a classifier that determines unreliable crop row detections and enables us to filter them out. We test the classifier on three crop types of different sizes. The experiments show that our classifier is able to recognize the majority of incorrect pattern detections. We further investigate the impact of filtering out unreliable detections according to our classifier in a crop-row-based localization approach. The results show that our approach increases localization robustness considerably.

Investigating Path Planning with Adjustable Joint Configurations and High Ground Clearance

In the narrow environments of crop fields and greenhouses common in agricultural settings, movement flexibility is key. A high ground clearance ensures flexibility by enabling a robot to pass over plants or other obstacles without any danger to itself or the environment, which is a huge advantage. Path planning approaches often make use of heuristics as a basis to guide the search for a feasible path in a suitable direction. However, heuristics that are commonly used do not cover the possibility of passing over obstacles, e. g., the Dijkstra heuristic based on the Dijkstra algorithm [26], or they do not consider obstacles at all, such as the Euclidean distance heuristic. In previous work, we thus developed a heuristic that considers the positions of all wheels of a robot and aims to find a path for each of them to provide an estimate of the cost to reach a given goal from the start position.

Joints that are not directly linked to steering can still impact which movements are possible in a given situation. For example, an outstretched manipulator arm might need to be retracted to get closer to an obstacle, lever arms connecting the robot wheels to its chassis might need to be adjusted so that the robot is able to pass over a wide obstacle, or prismatic joints for changing the track width could be adjusted for navigating in fields with different inter-row distance. In order to reach the highest possible movement flexibility, these joints have to be considered in path planning. This increases the planning complexity by introducing additional degrees of freedom. In earlier work, we proposed a state space representation for a search-based planning approach that includes the additional joints. Compared to planning when ignoring the additional joints, this representation is able to retain the planning efficiency and optimality despite the higher complexity of the planning problem [29].

In Chapter 3, we investigate the suitability of this planning approach from previous work for real world application. We perform an analysis of the planning time as well as the reached suboptimality bounds in environments with various challenges, including a real world environment of an agricultural field as a common use case. We see that the planning approach with the proposed state space representation and heuristic is well suited for navigation in real world environments, considering a robot with high ground clearance and adjustable joints. Furthermore, we test the limits of this approach in a particularly demanding environment with a high density of obstacles and narrow passages. We show that considering additional joint angles is needed to find feasible paths in cluttered environments.

Smooth Local Planning for Robots with Omnidirectional Wheels

In the context of movement flexibility, omnidirectional wheels have great potential in path execution. However, even with omnidirectional steering, some steering constraints have to be considered to ensure efficient execution. If these constraints are ignored, a robot may have to stop to turn its wheels in order to follow a given path accurately. In Chapter 4, we propose two variants for local planning that take the steering constraints into account. One variant uses motion commands computed solely from velocity and acceleration constraints and filters them to avoid steering constraint violations. The other variant includes steering constraints directly into the motion command computation. We investigate the capabilities of both variants in simulation and real world experiments and compare it with the performance when steering constraints are ignored. We show that there is no loss of accuracy when including the steering constraints in the motion command computation. We further demonstrate that both variants are able to increase path execution efficiency substantially.

The approaches and results presented in this thesis provide agricultural robots with capabilities needed for autonomous navigation, enabling them to perform various tasks on a field and contributing towards more sustainable farming technologies.

1.3 Publications

Parts of this thesis were published in international peer-reviewed conferences and journals. The publications are listed in chronological order below.

- Fleckenstein, F., Dornhege, C., and Burgard, W. (2017). Efficient Path Planning for Mobile Robots with Adjustable Wheel Positions. In *International Conference on Robotics and Automation (ICRA)*.

- Winterhalter, W., Fleckenstein, F., Dornhege, C., and Burgard, W. (2018). Crop Row Detection on Tiny Plants With the Pattern Hough Transform. *Robotics and Automation Letters (RA-L)*, 3(4):3394–3401.

- Fleckenstein, F., Winterhalter, W., Dornhege, C., Pradalier, C., and Burgard, W. (2019). Smooth Local Planning Incorporating Steering Constraints. In *International Conference on Field and Service Robotics (FSR)*.

- Pretto, A., Aravecchia, S., Burgard, W., Chebrolu, N., Dornhege, C., Falck, T., Fleckenstein, F., Fontenla, A., Imperoli, M., Khanna, R., Liebisch, F., Lottes, P., Milioto, A., Nardi, D., Nardi, S., Pfeifer, J., Popović, M., Potena, C., Pradalier, C., Rothacker-Feder, E., Sa, I., Schaefer, A., Siegwart, R., Stachniss, C., Walter, A., Winterhalter, W., Wu, X., and Nieto, J. (2021). Building an Aerial-Ground Robotics System for Precision Farming: An Adaptable Solution. *Robotics & Automation Magazine*, 28(3):29–49.

- Winterhalter, W., Fleckenstein, F., Dornhege, C., and Burgard, W. (2021). Localization for Precision Navigation in Agricultural Fields – Beyond Crop Row Following. *Journal of Field Robotics*, 38(3):429–451.

1.4 Collaborations

Parts of this thesis are results from joint work with other researchers. As the supervisor of this thesis, Wolfram Burgard contributed ideas and suggestions to all of its parts. The remaining collaborations are outlined in the following.

- Chapter 2 is the result of joint work with Wera Winterhalter and Christian Dornhege, leading to two journal publications [89, 90]. Wera Winterhalter developed the Pattern Hough transform, which is the crop row detection method presented in a joint publication [89]. She also proposed the localization correction measures presented in a second joint publication [90]. Wera Winterhalter and Christian Dornhege both contributed in data collection and experiment design for the work presented in this thesis. The author of this work is the main contributor to the generalized vegetation feature representation for capturing data from camera and lidar sensors, as well as the quality measure on detected crop row patterns developed for filtering in crop-row-based approaches to increase robustness. Large parts of Chapter 2 have been previously published in two journal papers [89, 90].

- Chapter 3 builds upon previous work [29], where we presented a path planning approach for robots with adjustable wheel positions and high ground clearance. In this thesis, we investigate the limits of this approach and its suitability for real world application. Christian Dornhege provided ideas and advice regarding the planning space in the previous work as well as for evaluation methods in both previous work and the work presented here. The author of this work is the main contributor to the

idea for a heuristic for robots with high ground clearance as well as implementation and evaluation in both previous and this work. Large parts of Chapter 3 were published at a conference [30].

- Chapter 4 is the result of joint work with research colleagues from the Flourish project, leading to a conference publication [31]. Cédric Pradalier contributed the formulation and unification of steering constraints in the space of the instantaneous center of rotation (ICR) of a robot. He further provided an efficient implementation for the steering constraints. Christian Dornhege helped with data collection and experiment design for the evaluation. Wera Winterhalter also helped with data collection and experiment execution. The author of this work provided the ideas and implementation for velocity rollouts with different constraints, including how to translate an ICR path into a valid velocity path, and performed most of the experiments. Large parts of Chapter 4 have been previously published at a conference [31].

- All work presented in this thesis has been used in a fully autonomous navigation approach developed in collaboration with research colleagues from the Flourish project. Most parts of this navigation approach were presented in a journal publication [73]. The author of this work contributed the global and local planner for the system as presented in Chapter 3 and Chapter 4, respectively. Further improvements were made to the navigation system after the mentioned journal publication, including filtering the crop row detections according to our crop row quality measure presented in Chapter 2 for localization. Fully autonomous navigation on an agricultural field using earlier versions of all components presented in this thesis was successfully showcased at the final review meeting of the Flourish project.

Chapter 2

Features and Quality Measure for Crop Row Detection

In precision agriculture, crop treatment is often performed on single plants. This is tedious and labor-intensive when done manually. Therefore, automated methods are desirable for individual plant treatment. Automating tasks such as weeding, fertilizing, or harvesting requires an autonomous navigation approach. Navigation within the narrow environments of agricultural fields needs to be precise with respect to crop rows in order to avoid damage to the crops. To achieve this, accurately detecting crop rows from sensor data is key. Depending on the growth stage of crops and on weather and lighting conditions, different sensor modalities are more suited for detecting crops or vegetation in general. For small plants in daylight, a high resolution camera yields more decisive data, whereas in the dark or when the plants are large and overlapping, more information can be gained from lidar data. We thus propose a unified representation for vegetation features extracted from either lidar or camera data, so that any crop row detection method using this representation is independent of the type of input sensor.

Irrespective of the type of input data, crop row detection methods may produce inaccurate results or provide false positive detections when there are no crop rows visible in the data. These incorrect detections can lead to errors in algorithms relying on this data, e. g., in a localization, an obstacle avoidance, or a mapping algorithm. To avoid this, we provide a quality measure for crop row detections and formulate a classifier based on this measure. Thus, crop-row-based applications can choose to ignore detections with low quality. In extensive experiments on real world data, we show the accuracy of the crop row classifier that is based on our quality measure, and demonstrate the increased robustness in a crop-row-based localization when employing this classifier.

2.1 Introduction

Precision agriculture entails the individual treatment of single plants or parts of a field, as opposed to a large scale uniform treatment. This allows for more efficient use of resources such as fertilizer or herbicides, which ensures a higher sustainability and less chemical contamination of the soil [10, 35, 68]. Since manual treatment of single plants is tedious and inefficient, automated approaches are required for precision agriculture. Autonomous navigation is a key component for such approaches. To enable various tasks in autonomously navigating a field, accurately detecting crop rows from sensor data is crucial. Common use cases are crop-row-based localization [6, 12, 27, 90] or row following [5, 84]. Considering crop rows in either application ensures that an autonomously navigating robot will not traverse the rows and thereby crush the crops. The information gain for crop row detection from different sensor modalities highly depends on the growth stage and type of plants as well as lighting conditions. While small plants in daylight are more easily recognizable in camera images, large overlapping plants and plants in bad lighting conditions are more distinguishable in lidar data. Therefore, having a range of sensors at hand is helpful for detecting crop rows in a wide span of scenarios. We thus propose a novel representation of vegetation features that can capture both camera and lidar data. As representation we chose a 2d grid map located on the ground plane, where the value of each cell indicates the likelihood that this cell contains vegetation. We call it a vegetation feature map. A crop row detection method is then able to directly operate on this vegetation feature map and does not need to be adapted to lidar or camera input data.

Crop row detection methods usually return the detected crop rows as lines in an image or in 3d space. Most methods assume that there is a crop row present in the sensor data, and return those lines that best explain the sensor data as crop rows [7, 66, 88, 89]. However, when close to the end of a field, the crop rows may not always be visible in the sensor data. In extreme cases, with huge overlapping plants or when some plants are missing from the regular seeding pattern, the crop rows might also not be visible in the middle of the field. This means that detected crop row patterns can potentially be wrong. If those are integrated in any crop-row-based approach regardless, e. g., to correct a localization pose estimate or to correct the steering angle for row following, their accuracy and safety can suffer as a result. We therefore present a method to detect false positive crop row detections in feature maps by determining whether a crop row detection is sufficiently supported by the vegetation features to safely be used by an application for autonomous navigation.

Overall, we pursue novel methods for making crop row detection approaches independent of the type of sensor data, and for ensuring the robustness of crop-row-based applications with regard to unreliable detections. In this chapter, we show two contributions to achieve this. First, we present a representation of vegetation features that is independent of the type of sensor data from which the features are extracted. Second, we provide a quality measure for detected crop rows. This allows for filtering out detections with low quality, thus increasing the robustness of any crop-row-based application.

Substantial parts of the ideas, figures and results presented in this chapter have been previously published [89, 90]. Section 1.4 outlines the author's contribution to this work.

2.2 Related Work

Detecting the semantics of a field, i. e., the crop rows, is useful for several tasks in autonomous navigation in agriculture. In localization and row following they can be used to determine the robot pose relative to the rows, enabling the navigation system to avoid driving over the crops. The first step to crop row detection is to extract vegetation features from the sensor data. Two main data sources are used for this: cameras (RGB or near infrared) and lidar sensors [2, 39, 79]. For extracting vegetation features from RGB vision data, several approaches have been proposed. The triangular greenness index [25], the excess green index [64] and the excess green minus excess red index [64] are commonly used for finding pixels in camera images that likely belong to plants. From lidar data, a 3d point cloud can be generated. Extracting the ground plane and assuming that plants rise above the ground, all remaining points above ground level are likely to be caused by vegetation. Data from a stereo camera allows to build an elevation map of the environment [51], from which vegetation features can be extracted in a similar way. While all these methods provide vegetation features on the given sensor data, any crop row detection that uses these features needs to be adapted to their respective outputs. To remedy this dependency on the sensor type and the vegetation feature detection method, we propose a representation that is able to capture the output from any of these methods.

Running a crop row detection on this representation of vegetation features gives us potential input for crop-row-based localization or row following. In order to avoid causing errors in these applications, it is helpful to filter out these incorrect detections. We therefore present a quality measure for detected crop rows on the representation we developed. Crop row detection approaches usually assume that crop rows are sown in equidistant, straight and parallel lines [5, 7, 63, 82, 89]. All of these assumptions hold when only considering a small local area on most fields. A set of crop rows that satisfies these constraints can be described by their heading, the spacing between two rows, and the offset of one of the rows to a given reference point. This is the output on which we base our quality measure. Some of the mentioned crop row detection methods require specific sensor data. As we aim for a generalized crop row quality measure, we use the Pattern Hough transform [89], which is able to use our unified representation of vegetation features, and is thus compatible with the most common types of sensor input.

2.3 Generalized Vegetation Feature Representation

Crop row detection profits from different sensor modalities, depending on the size and type of plants as well as lighting conditions. In order to make crop row detection independent of the sensor modality, we developed a novel representation of vegetation features that is able to incorporate data from camera images as well as lidar point clouds. The base idea is to have a 2d grid map of the ground plane, where the value of each cell indicates how likely it is that this cell contains vegetation. In the following, we detail how features are extracted from the different sensor modalities and how they are transferred into this vegetation feature map representation.

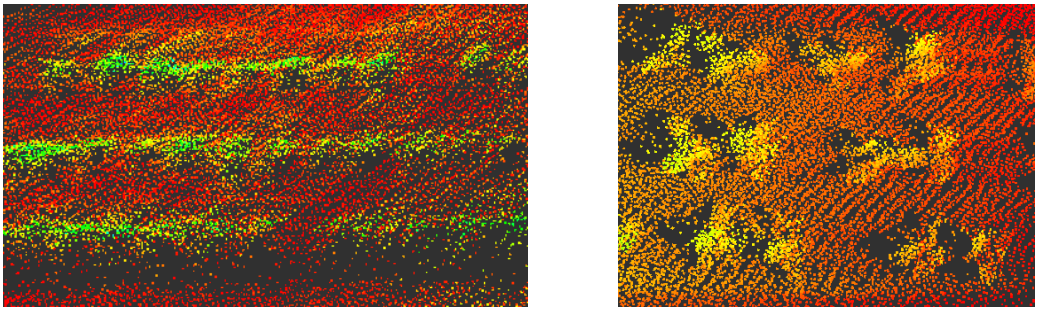


Figure 2.1: Both images show the point cloud of three crop rows from a top down view. On the left, the points are colored according to their intensity values, where green corresponds to a high intensity, and red corresponds to a low intensity. On the right, the points are colored according to their height. Here, red points are low and yellow points are high.

2.3.1 Vegetation Features in Lidar Data

In lidar data, there are two straightforward ways to detect plants. They show a much higher intensity than the soil as can be seen in Figure 2.1 on the left. Thus, one way to extract vegetation features is to set the intensity of a point as the vegetation weight for that point. As intensity values decrease with distance, the values have to be normalized accordingly. A second method is to make use of the fact that crops usually rise above the ground, and thus the height of points above the ground can be employed as indicator of whether vegetation is present as visible in Figure 2.1 on the right. The first step for either method is to transform the points of the point cloud into a coordinate frame that is located on and parallel to the ground plane. The z-axis of this frame will thus point upwards. We call this frame the ground frame. One possibility is to use the robot base frame, which is usually located on the ground. While the ground in the real world is not planar in most cases, especially not in an agricultural setting, it is sufficiently close to being planar in a local environment for our requirements. Once the point cloud has been transformed into the ground frame, a grid map of suitable size considering the field of view of the sensor is created.

If the intensity is used as indicator, the weight of each grid cell is computed as follows: All points falling into this cell are determined. Among all these points, the highest intensity value is chosen. The weight of the cell is the appropriately scaled intensity value.

If the height of the points is used as an indicator, again all points falling into a grid cell are computed. Among these, the highest z value is determined and assigned as weight. To reduce noise, only the highest 10% of grid cell weights are considered as vegetation. Thus, a lower bound is applied dynamically.

Depending on the density of the lidar sensor that is used, the resulting grid map may be rather sparse, which makes it hard for a crop row detection to reliably find the right pattern. To overcome this issue, we suggest integrating the point clouds using robot odometry, if it is available.

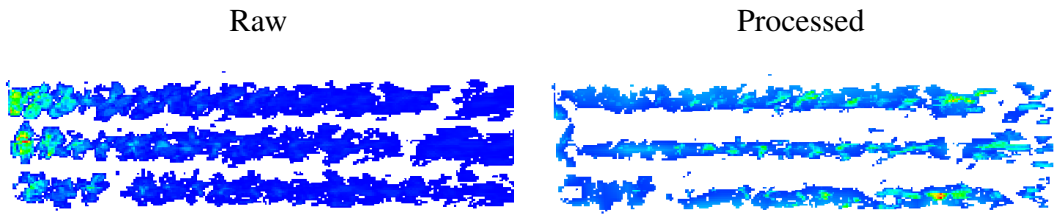


Figure 2.2: On the left, a portion of the raw feature map obtained by applying the triangular greenness index on an image is shown. On the right, we show the feature map after processing by weighting each point with its squared distance to the camera, applying a lower bound and smoothing the weights. The color scheme is a heat map on the rainbow spectrum, where blue corresponds to low feature weights, and red corresponds to high feature weights. The white space has feature weight 0.

2.3.2 Vegetation Features in Image Data

As stated before, there are several methods to detect vegetation in image data, e.g., the triangular greenness index or the excess green index. These both assign a weight to each pixel that indicates the likelihood that this pixel shows vegetation. We apply any of the methods suitable for indicating vegetation in images. Under the assumption that the ground is flat, and given the intrinsic and extrinsic calibration of the camera, we project the camera image to the ground plane. Here we make use of the previously defined ground frame. Just as for the lidar data, we create a grid map with fitting size considering the field of view of the camera. All pixels that are projected into one grid cell are computed. The highest weight among these pixels determines the weight of the grid cell. In order to account for the fact that the size of plants in the image space differs depending on their distance to the camera, the weight of the grid cells is scaled with their squared distance to the camera. While this does not increase the size of plants far away in the grid map, it ensures that any crop row detection algorithm considering the weights has an inclination to incorporate far away plants. As a last step, we only keep the best 25% of features in the vegetation grid map to get rid of noise and smooth the values. A comparison between the raw grid map and the processed version is shown in Figure 2.2.

2.4 Quality Measure for Detected Crop Rows

Any crop row detection method that does not depend on one specific type of sensor input can be run on our generalized representation of vegetation features. Crop row detection algorithms usually output the lines that are best explained as crop rows. They do not provide a measure of how well the data matches the hypothesis that the detected lines are crop rows. Especially near the end of a field, when there is more noise due to grass, bushes or trees visible in the sensor data, the detected rows might not be correct. When they are used for tasks related to autonomous navigation, e. g., by integrating them into a localization or using them to correct the steering angle for row following, this can be misleading and introduce errors. We therefore present a quality measure for a set of crop

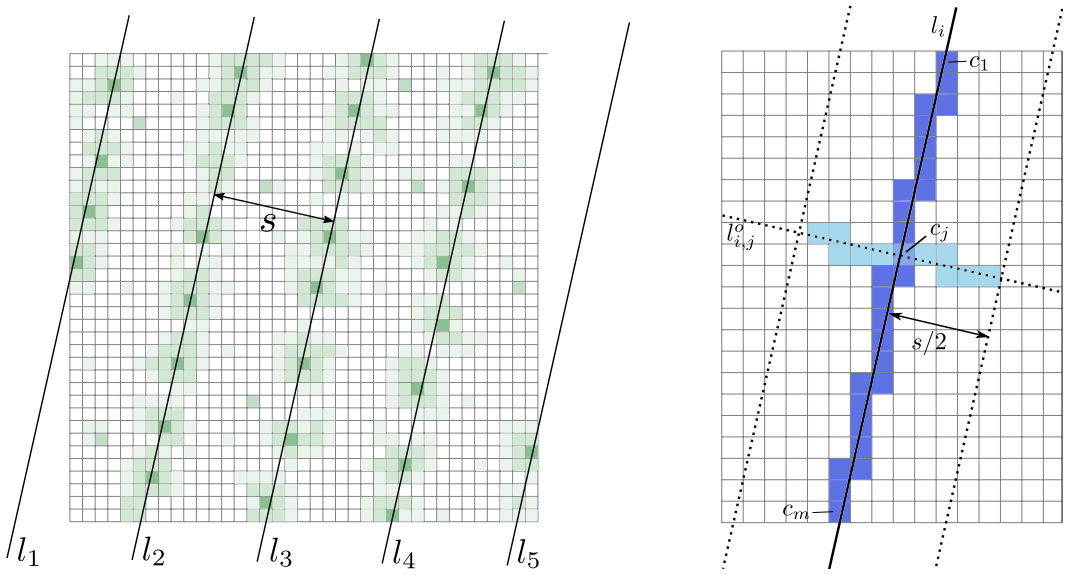


Figure 2.3: The schematic on the left shows a crop row pattern overlaid on a vegetation feature map. Dark green corresponds to a high feature value, lighter shades to lower feature values. In this case, the feature map overlaps with 5 lines $l_1, \ldots, l_5$ of the pattern P. The vegetation feature clusters lying on the pattern lines likely correspond to crops, while vegetation features that diverge from the pattern are presumably weeds.
The schematic on the right illustrates which feature map cells are relevant for computing the feature support of a cell on a pattern line. The pattern line l_i crosses cells $c_1, \ldots, c_m$ (marked in dark blue) in the feature map. For cell c_j on this line, we determine the line $l^o_{i,j}$ that is orthogonal to l_i and goes through c_j. It crosses feature map cells $c^o_1, \ldots, c^o_q$. For computing the feature support of cell c_j, only those cells on $l^o_{i,j}$ that are less than half the pattern spacing s away from c_j are relevant (marked in light blue).

rows extracted from the previously defined vegetation feature map. Using this measure, detected crop rows that are not well supported by the data can be filtered out before using them in other approaches.

Our quality measure is based on the support of the detected crop rows in the feature map data. There are several criteria that are relevant to ensure that detected crop rows are backed up by the obtained data. The first is how well the vegetation features support the theory that crop centers lie on the crop row lines. This incorporates different aspects: It is crucial that there are vegetation features on or close to the crop row lines. Furthermore, they should be distributed among a set of crop row lines, i. e., not all vegetation features should fall on a single line. It is also important that there are few vegetation features far away from crop row lines, otherwise the robot might be standing on grass and not in the field, so that the detected crop rows are just random lines. A second criterion is based on the assumption that the robot is close to the field from which crop rows should be detected: the crop row lines which are supported by vegetation features should be close to the robot. If this is not the case, i. e., all supported crop row lines are far away from the robot, they likely correspond to a different field from the one the robot is driving in.

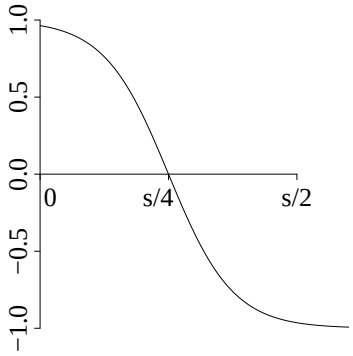


Figure 2.4: This plot shows the sigmoid function we designed for computing the feature support of a cell c_j. When the relevant cells are determined, we weight their feature values according to this sigmoid. We expect high feature values close to c_j, if c_j lies in the center of a crop plant, and low feature values farther away. Thus, small distances get a high positive weight and large distances get a high negative weight. The inflection point is at $(s/4, 0)$ where s is the pattern spacing, reflecting that we expect a crop plant to take up at most $s/2$ in width.

For the following definitions, see Figure 2.3 for an illustration. We define a crop row pattern as a set of parallel equidistant lines $P = \{l_1, \ldots, l_n\}$ in the feature map space, with spacing s. For each line $l_i \in P$ we compute all cells $c_1, \ldots, c_m$ that it crosses in the feature map. In the following, we use the name of a cell c_j equivalently with the 2d coordinates of its center. We are interested in how well the feature map data supports the claim that a line l_i represents a crop row, i. e., it is a line that goes through the center of crop plants. Consequently, we want to know whether a cell c_j on l_i represents the center of a crop plant. In order to determine this, we find the line $l^o_{i,j}$ that is orthogonal to l_i and goes through cell c_j. We omit the indices i, j for simplicity and write l^o from here on. This line l^o crosses cells $c^o_1, \ldots, c^o_q$ within the feature map. If c_j represents the center of a crop plant, we expect high feature values in the feature map close to c_j on l^o, and low values in cells on l^o that are farther away from c_j. The latter only holds for cells that are not closer to a different pattern line, i. e., we only consider cells c^o_k on l^o that are no further than half the pattern spacing away from c_j: $\|c^o_k - c_j\| < s/2$. To model that high feature values close to c_j and no features far away from c_j are desirable, we design a sigmoid function shown in Figure 2.4. It is defined on the distance of a cell c to c_j with values in [-1, 1], where large distances get negative values and small distances get positive values. We set the inflection point of the sigmoid at $s/4$, which implies that a crop plant takes up half of the pattern spacing. Note that different inflection points may yield better results for different plant sizes, but as shown in our experiments, this value works well for different crop sizes. The sigmoid is defined by

$$\text{sigm}_{c_j}(c_k^o) = -2\left(-\frac{1}{2} + \left(1 + \exp\left(\frac{-2\left(\frac{\|c_j - c_k^o\|}{s} - \frac{1}{4}\right)}{\frac{1}{4}\cdot\frac{1}{2}}\right)\right)^{-1}\right). \tag{2.1}$$

Multiplying the feature value $f(c_k^o)$ of a cell c_k^o with its sigmoid value indicates how well this cell supports the claim that c_j is the center of a crop plant. Summing over all relevant cells on the orthogonal line l^o results in the *local feature support* of c_j, a measure of how well c_j is supported as crop center by the feature map:

$$g(c_j) = \sum_{c_k^o, k\in\{1\ldots q\}, \left\|c_j - c_k^o\right\| < s/2} f(c_k^o)\cdot \text{sigm}_{c_j}(c_k^o) \tag{2.2}$$

In the real world, crops are not always sown in exact lines. Thus, we allow for some deviation d of the crop center to the left and right and compute the *feature support* of cell c_j as

$$h(c_j) = \max_{c_k^o, k\in 1\ldots q, \left\|c_j - c_k^o\right\| < d} g(c_k^o).$$

We say a cell is supported if its feature support is higher than a given threshold. We then define a *reference cell*

$$c_j^r = \arg\max_{c_k^o, k\in 1\ldots q, \left\|c_j - c_k^o\right\| < d} g(c_k^o).$$

It is the cell on line l^o with the highest evidence that it is a crop center and is sufficiently close to l_i to belong to it (see Figure 2.5 for an illustration). Intuitively, if the crop is slightly off the pattern line, this cell is likely its center. We call its distance to the pattern line the *reference cell offset*.

Once we have determined the feature support and reference cell for each cell on the detected crop row pattern, we check for segments of pattern lines that are supported by the feature maps. This indicates how the feature support is distributed among the pattern lines, which is another criterion for the data supporting the detected crop rows. Following each pattern line through the cells of the feature map, we create supported line segments as follows: If a cell c is supported, a supported segment of this line exists and it is not too far from the considered cell, add c to the supported segment, along with all cells in between c and the supported segment. Therefore, a sequence of crop cells is still recognized as a continuous supported crop line segment if there are gaps between the crops, as long as they are close enough together. If the last supported segment is too far away or none exists, we create a new supported segment that only contains c. All cells between the last and the new supported segment form an unsupported line segment. This might be the case when the robot is near the end of the field and perceives the end of crop rows (a long supported line segment) as well as weeds growing on the headlands (a short supported line segment that is not connected to the longer one). We perform some additional checks to see if a supported line segment is valid. First, it has to be longer than

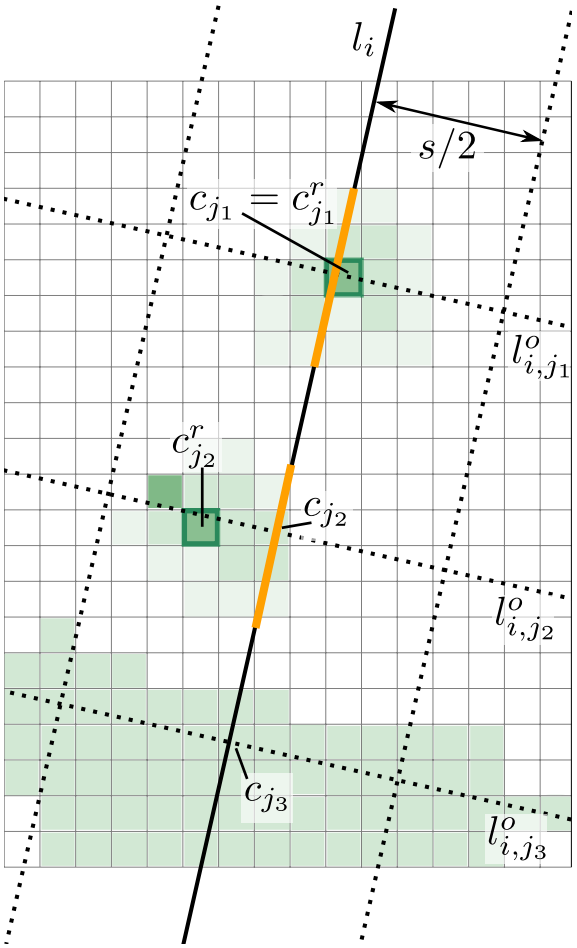


Figure 2.5: This schematic shows examples for supported line segments (marked in orange) and reference cells (marked with a dark green border) to pattern line cells $c_{j_1}, c_{j_2}, c_{j_3}$. The vegetation features in the feature map are shown in different shades of green, where a darker shade represents a higher feature value. For cell c_{j_1}, there appears to be a crop directly on c_{j_1}. Within the distance threshold of $s/2$ along the line l^o_{i,j_1}, c_j is the cell with the highest local feature support, i. e., it is its own reference cell, $c^r_{j_1} = c_{j_1}$.
For cell c_{j_2}, while there are two cells with high feature values on the line l^o_{i,j_2}, one of them has more cells with feature support closeby and thus gets chosen as reference cell $c^r_{j_2}$. In this case, the center of the crop might be slightly off the pattern line, or the detected vegetation features were caused by a weed.
For cell c_{j_3}, vegetation features are similar in all cells within the distance threshold. When computing the local feature support for each cell on l^o_{i,j_3}, they cancel each other out due to multiplication with the sigmoid function. Thus, c_{j_3} has no feature support. This is desired behavior, since a vegetation distribution like this contains no evidence for crop rows.

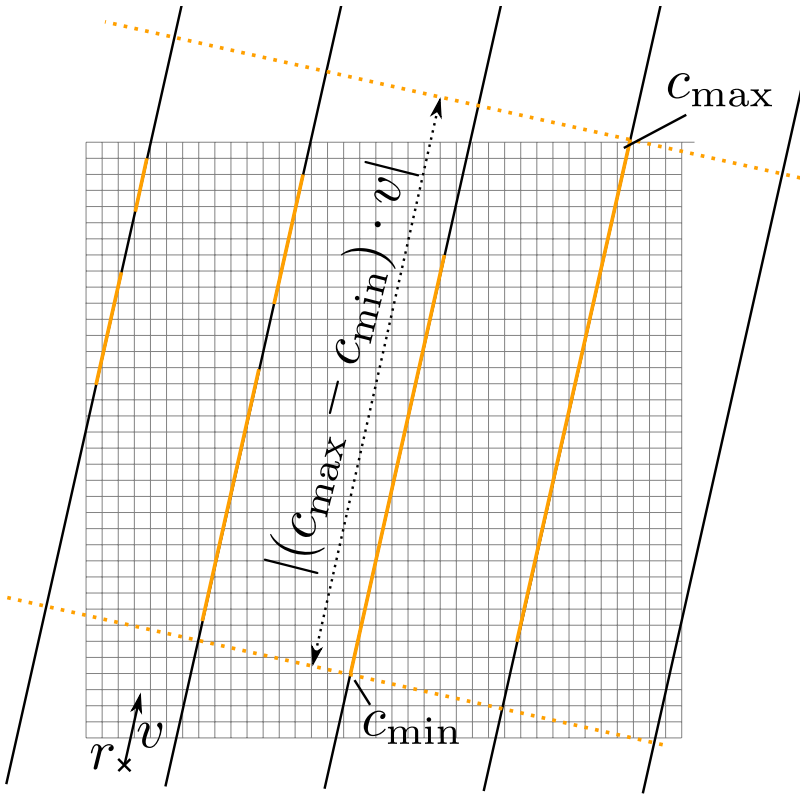


Figure 2.6: This graphic illustrates how we compute pattern extensions. Pattern lines are shown in black, supported line segments are represented by solid orange line segments. The robot pose r is in the lower left of the picture, and v is the direction of the pattern lines. $c_{\min}$ is computed as the cell in all supported segments over all pattern lines that is closest to r in direction v. Similarly, $c_{\max}$ is the cell that in all supported line segments is farthest from r in direction v. The pattern extension is then given by the distance between those two cells along the pattern, i. e., $|(c_{\max} - c_{\min}) \cdot v|$.

a given threshold. This allows us to prune supported segments generated by weeds, which are usually short because weeds do not grow in straight lines. Second, we check that the density of supported cells is high enough, i. e., the number of supported cells divided by the total number of cells in a segment is higher than a threshold.

When the valid supported segments of each pattern line have been determined, we extract the length of the supported pattern part, i. e., how well the feature support is distributed along the pattern lines. If only a short part of the pattern is supported, it is likely that only a small part of the field of view of the sensor covers crop row data, making the estimated crop rows more prone to errors. We find the cells $c_{\min}$ and $c_{\max}$ belonging to a valid supported line segment that are closest to and farthest from the robot pose along the pattern lines (see Figure 2.6 for an illustration). Given the direction of the pattern lines v with $\|v\| = 1$, the distance of these two cells in direction v defines the pattern extension, $e(P) = |(c_{\max} - c_{\min}) \cdot v|$.

All previously computed values impact how well the feature map data fits the detected

crop row pattern. We compute a likelihood that the detected pattern is correct using the following steps.

First, we determine whether a sufficient amount of supported pattern lines was detected. A pattern line is supported if it contains at least one valid supported segment. We denote the number of supported pattern lines by a. The threshold t_1 depends on the field of view of the sensor and the resulting size of the feature map. It expresses the minimum of how many crop rows we expect to see in the sensor data. We here account for the fact that the robot might be at the edge of the field so that not all sensor data lies in the field, e. g., only half of the sensor data contains crop rows. We define

$$p_1(P) = \begin{cases} 0 & \text{for } a < t_1 \\ 1 & \text{else} \end{cases}$$

as the likelihood that the pattern is correct, given the number of supported pattern lines.

Second, we check whether the pattern is off-center, i. e., whether there are only supported lines to the left or to the right of the robot. If this is the case, the robot is likely at the edge of the field. During in-field navigation, if all supported crop rows are far away from the robot, they are likely incorrect or belong to a different field than the one the robot is driving on. We compute the offset o of the robot position to the closest supported pattern line and model the likelihood that the detected pattern is correct given this offset as

$$p_2(P) = \begin{cases} 1 - t_2 \cdot o & \text{for } o < 1/t_2 \\ 0 & \text{else} \end{cases}$$

where t_2 is the maximum distance we expect the crop rows to have from the robot.

If the pattern is not off-center, i. e., there are supported pattern lines to the left and right of the robot, it is likely at a position in the field. In most cases, we expect the same amount of crop rows to be visible to the left and right of the robot, given the field of view of the sensors is symmetrical with regard to the robot position. We thus determine the number of supported lines to the left and to the right of the robot. The minimum b of these two values is the symmetry of the pattern support. We define the likelihood that the pattern is correct given its symmetry as

$$p_3(P) = \begin{cases} \frac{1}{t_3} \cdot b & \text{for } b < t_3 \\ 1 & \text{else} \end{cases}$$

where t_3 is the number of crop rows we expect to see on either side of the robot.

Lastly, we investigate how far the assumed crop centers deviate from the pattern lines, i. e., how much the reference cells for the pattern lines deviate from those lines. For each supported pattern line l_i, we compute the mean μ_i and variance σ_i of the reference cell offset across all supported segments. We scale both to values in [0,1] and call the results μ_i^s and σ_i^s. We then define the likelihood that the pattern is correct given the reference cell offset mean as

$$p_4(P) = \sum_i \mu_i^s$$

and given the reference cell offset variance as

$$p_5(P) = \sum_i \sigma_i^s.$$

Overall, we compute the likelihood that the pattern is correct or the *pattern quality* as

$$q(P) = \prod_{l=1}^{5} (\alpha_l + (1 - \alpha_l) \cdot p_l(P))$$

where the $\alpha_l \in [0, 1]$ are introduced for numerical stability and to allow for weighting the different factors.

To achieve higher robustness in crop-row-based applications, such as localization, we implement a classifier based on thresholding the pattern quality. Patterns P with a quality higher than threshold t_q, $q(P) > t_q$, are classified as valid patterns, patterns with equal or lower quality are classified as invalid. In a crop-row-based localization, mapping or row following approach, the detected crop row patterns can be filtered using this classifier to avoid feeding them misinformation.

2.5 Experiments

In order to evaluate our quality measure for crop row detection, we performed two different experiments. In the first experiment, we determine how well we are able to filter out wrong crop row detections using the classifier based on our quality measure. This gives us a baseline on the potential increase in robustness for any application using the classifier. In the second experiment, we test how filtering the detected crop rows according to our quality measure impacts a localization method. This shows how we are able to increase the robustness of the localization method in practice. Both experiments were run on real world data from a production vegetable field. There were three types of crops on the field: kohlrabi, Chinese cabbage, and pointed cabbage (see Figure 2.7 for some examples of the data). The crops have varying sizes, allowing us to test the flexibility of our approach with regard to plant size. We recorded data of the same field twice in order to test reproducability and from here on refer to the two runs as *Run 1* and *Run 2*, respectively. We performed Run 1 in the morning and Run 2 in the afternoon, to ensure at least slightly different lighting conditions applied.

2.5.1 Evaluation of Pattern Classification using our Crop Row Pattern Quality Measure

For the first experiment, we manually labeled crop rows in vegetation feature maps. We ran the Pattern Hough transform [89] as crop row detection on the same feature maps. By using a threshold on the lateral and angular error between the labeled and the detected crop rows as introduced by Winterhalter et al. [89], we determine the ground truth of whether a detected crop row pattern is valid. We then compute the quality measure on the

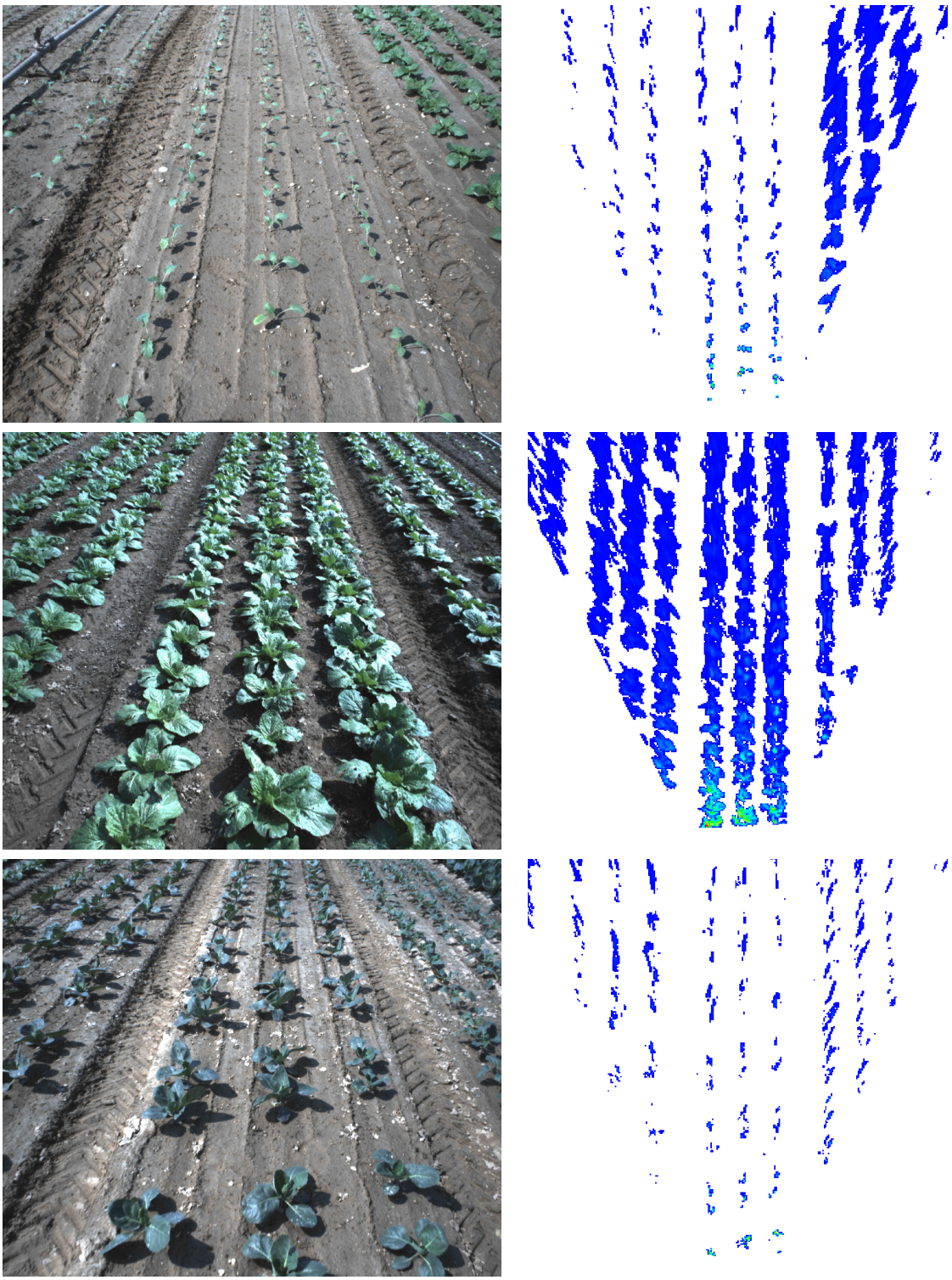


Figure 2.7: On the left, we show example images for the three crop types present in our data set. From top to bottom we see kohlrabi, Chinese cabbage, and pointed cabbage. On the right, the corresponding vegetation feature maps are shown. The color scheme is a heat map on the rainbow spectrum, where blue corresponds to low feature weights, and red corresponds to high feature weights. The white space has feature weight 0. Images and vegetation feature maps show the variety of plant sizes in the different crop types.

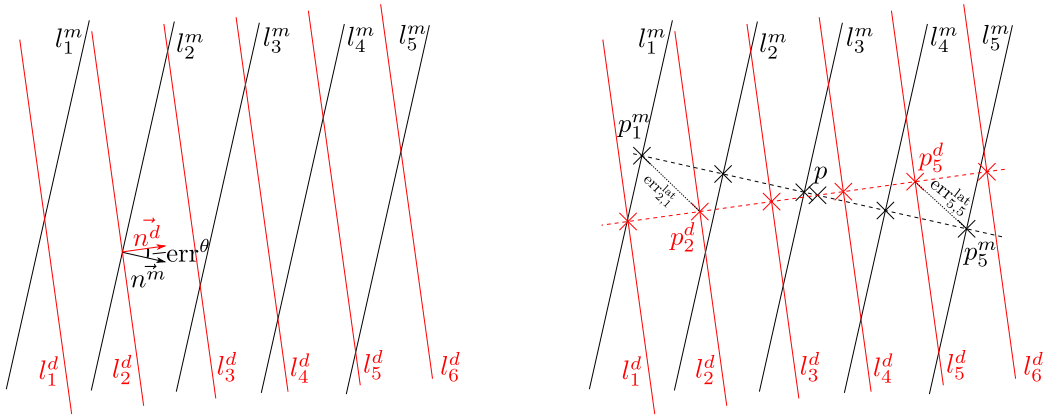


Figure 2.8: These two figures illustrate the computation of the angular and lateral crop row pattern error according to Winterhalter et al. [89]. The detected pattern is shown in red, the manually labeled pattern in black. The angular error err^θ is defined as the angular difference between the normals of the detected and labeled pattern, as illustrated on the left. On the right, our reference point p is orthogonally projected onto the pattern lines, creating new points p_i^d or p_j^m, respectively. As examples, only p_1^m, p_5^m, p_2^d and p_5^d and the error between the line pairs (l_2^d, l_1^m) and (l_5^d, l_5^m) are labeled.

detected crop row pattern. Applying our classifier by thresholding the quality measure gives us a classification as valid pattern for patterns with high quality, and invalid pattern for those with low quality. Comparing the classification results with the ground truth determined from labels, we establish which detected patterns were correctly classified as valid or invalid.

The detected crop row pattern is given as

$$P^d = \left\{ l_i^d \,|i \in \{1, \ldots, n\} \right\}, l_i^d = \left\{ \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^2 \,\middle|\, \vec{n^d} \cdot \left(\begin{pmatrix} x \\ y \end{pmatrix} - \vec{a_i^d} \right) = 0 \right\}$$

where the normal vector $\vec{n^d}$ is the same for all pattern lines because they are parallel to each other, and a_i^d is a support vector for line l_i^d. Analogously, the manually labeled crop row pattern is given as $P^m = \{ l_i^m \,|i \in \{1, \ldots, n\} \}$ with equivalent definitions for the manually labeled pattern lines l_i^m. The error measures between the manually labeled ground truth pattern and the detected crop row pattern were introduced by Winterhalter et al. [89]: The angular error is the angle difference between the normal vectors, i. e.,

$$\text{err}^\theta(P^d, P^m) = \arccos(\vec{n^d} \cdot \vec{n^m}).$$

See also Figure 2.8 on the left for an illustration. For computing the lateral error, we define a reference point p. We project the robot position 1 m forward and define this as the reference point. This is a point a row following approach could use for steering correction. We project the reference point onto all pattern lines l_i^d and l_j^m from the detected as well as the labeled pattern (see Figure 2.8 on the right), which gives us projected points for each line, p_i^d and p_j^m. We then create all pairs of lines from the detected pattern and the labeled

pattern. For each line pair (l_i^d, l_j^m), we compute the distance between the reference point projections,

$$\text{err}_{i,j}^{\text{lat}}(P^d, P^m) = \left\| p_i^d - p_j^m \right\|.$$

We then define the lateral error for the detected pattern as

$$\text{err}^{\text{lat}}(P^d, P^m) = \min_{i,j} \text{err}_{i,j}^{\text{lat}}(P^d, P^m).$$

A threshold on both error values determines our ground truth label on whether a pattern is valid (both values are below or equal to a threshold) or invalid (at least one value is higher than the threshold).

We divide each run into an *in row*, a *transition* and an *out of field* data set. The in row data set contains only feature maps where crop rows are visible in the whole feature map. Examples are shown in Figure 2.7. The transition data set consists of feature maps where crop rows are only partially visible, i. e., the robot is leaving or entering the field. This poses a more challenging scenario. Figure 2.9 shows examples for this data set. Lastly, the out of field data set is comprised of feature maps where crop rows are not visible at all. Here, the scenarios range from grassy headlands to pure earth headlands as shown in Figure 2.10. In the transition and the out of field data set, we evaluate on feature maps from every image. In the in row data set, we only consider feature maps every 5 s, because here the data changes more slowly due to low driving speed and almost no rotation.

Table 2.1 presents the results of this experiment. The out of field data set for each run has more patterns than the other two, because we evaluate every feature map and it contains almost all feature maps of when the robot is turning at the end of the field. At the top of both tables, the confusion matrices are given. In this scenario, a true positive is a valid pattern according to ground truth that our classifier also identifies as valid. A false positive is an invalid pattern that is classified as valid, thus potentially misleading an approach such as localization that uses it in further processing. A true negative is an invalid pattern that is classified as such. It provides incorrect data that will be filtered out. A false negative is a valid pattern that is classified as invalid. These are patterns that would be filtered out in an application such as localization, even though they are valid. A high number here would indicate that we lose a lot of correct data. We particularly aim for a low false positive rate to avoid misleading approaches using the filtered crop row detections.

The pattern quality estimate has the best classification results in the in row data set. In this data set in both runs, the pattern classification achieves more than 95% of true positives. Only less than 2% of the patterns are false positives. These are the patterns that potentially mislead an application that uses them, since they are wrong but will not be filtered out. Additionally, less than 3% of the patterns yield false negatives.

The transition data sets are more challenging. Only about 49% and 58% of the detected patterns are true positives in Run 1 and Run 2, respectively. Around 40% and 25% of the patterns are false negatives. This is correct data that we lose when filtering. Less than 4% of patterns are false positives. We will show in the second experiment that these false positive and false negative rates are still sufficient to enable an accurate pose estimation. Lastly, about 10% and 14% of the patterns are correctly filtered out as true negatives.

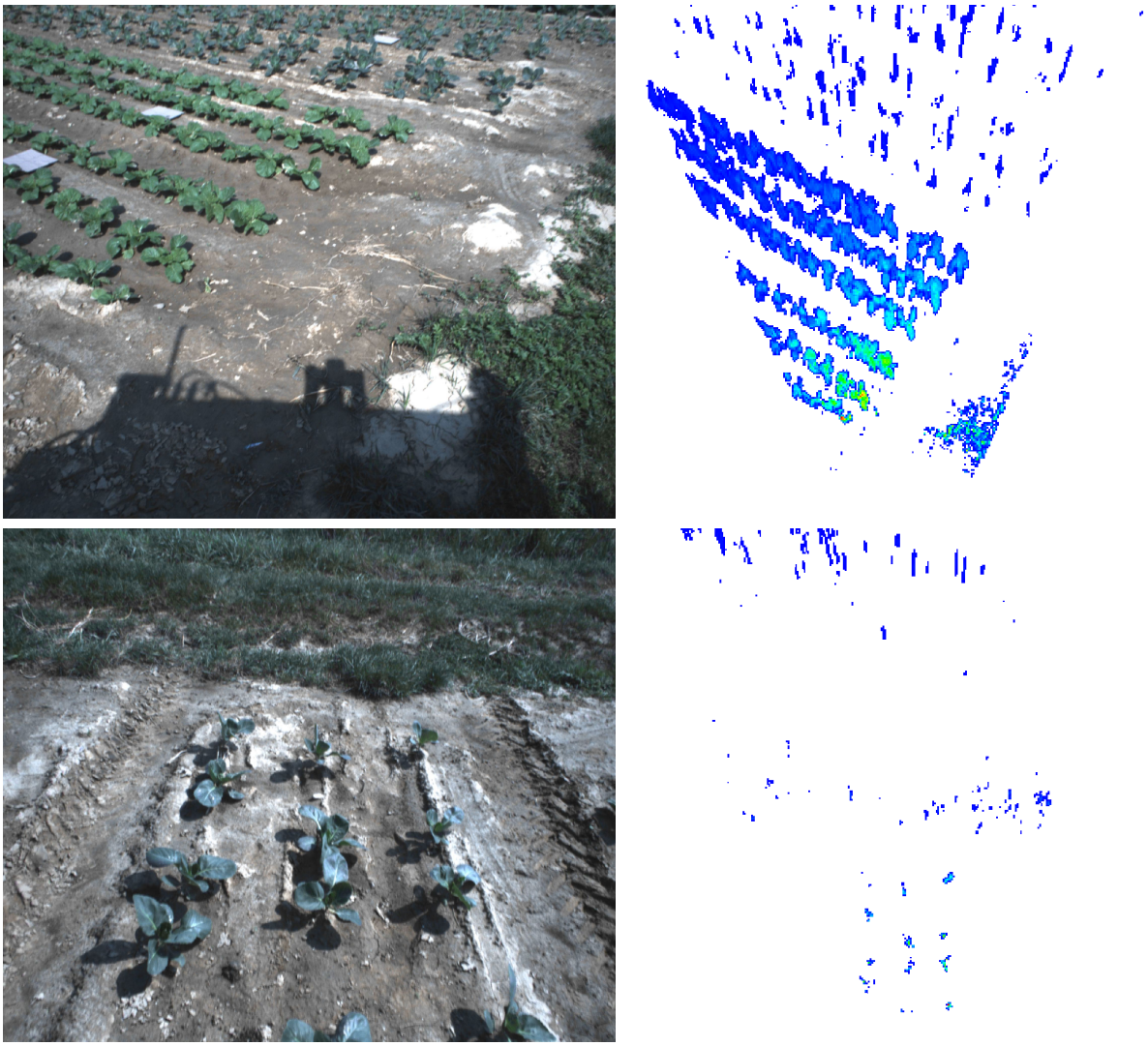


Figure 2.9: On the left, two example images from the transition data set are shown. On the right, we show the corresponding vegetation feature maps. The color scheme is a heat map on the rainbow spectrum, where blue corresponds to low feature weights, and red corresponds to high feature weights. The white space has feature weight 0. The top image was taken when the robot entered a crop row, while the bottom one was taken when it was leaving a crop row.

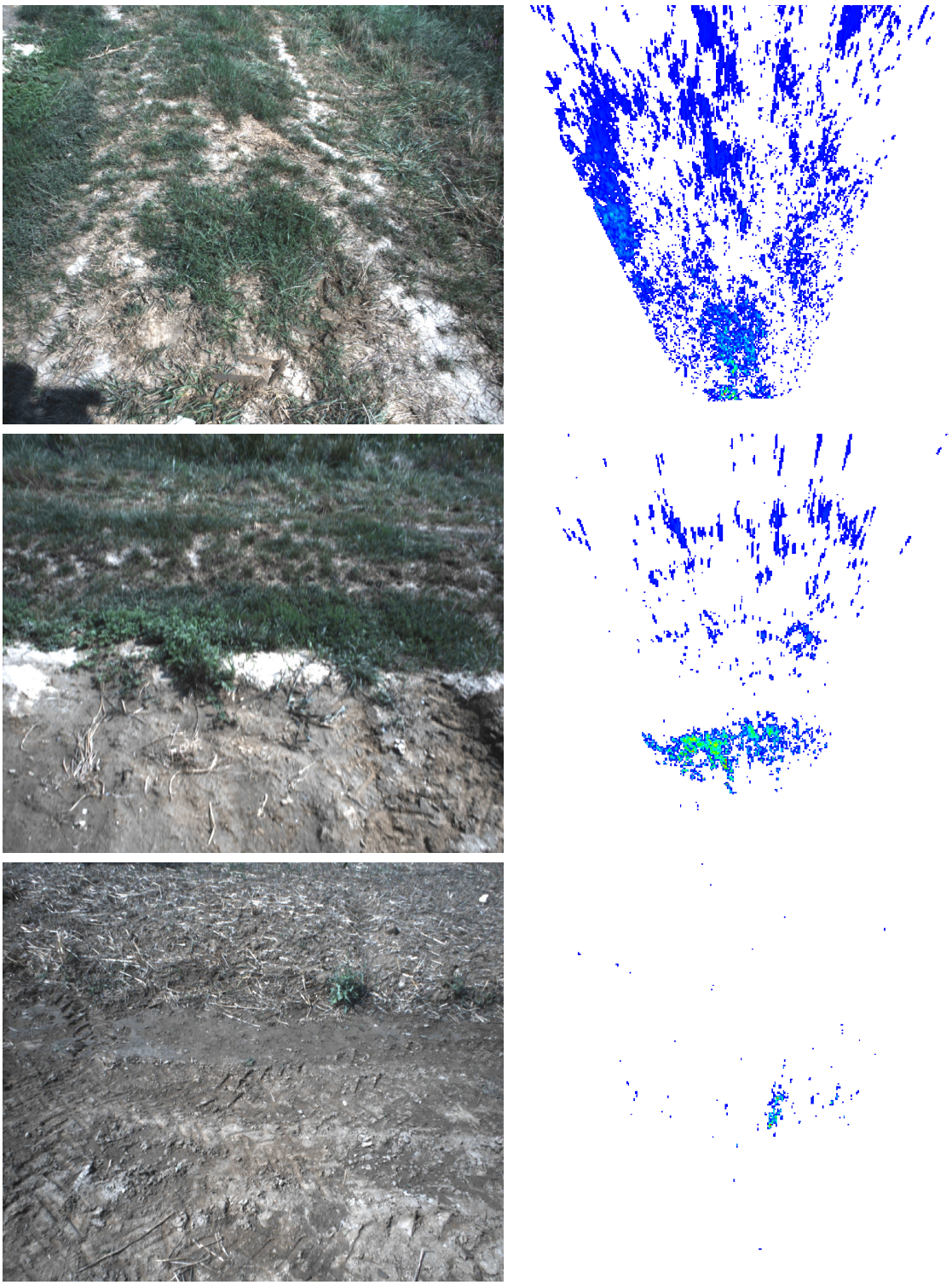


Figure 2.10: On the left, images from the out of field data set are shown with their corresponding vegetation feature maps on the right. The color scheme is a heat map on the rainbow spectrum, where blue corresponds to low feature weights, and red corresponds to high feature weights. The white space has feature weight 0. The top image shows grassy headlands which produces a rather dense feature map. The middle image shows headlands with a mixture of grass, other larger weeds and bare soil, resulting in a feature map that has dense parts and some features distributed in the whole field of view of the camera. The bottom image shows mostly bare soil with a few weeds, giving us a sparse feature map with some feature clusters from large weeds.

Table 2.1: These tables show the confusion matrices and the precision and recall values for the detected crop row classification according to our quality measure. Results are given for both runs on all data sets.

Run 1

	in row (267 patterns)		transition (337 patterns)		out of field (1456 patterns)	
	classified as		classified as		classified as	
	valid	invalid	valid	invalid	valid	invalid
valid	98.13%	1.12%	48.66%	39.17%	0.00%	0.00%
invalid	0.75%	0.00%	1.78%	10.39%	23.76%	76.24%
precision	99.24%		96.47%		-	
recall	98.87%		55.41%		-	

Run 2

	in row (273 patterns)		transition (462 patterns)		out of field (1117 patterns)	
	classified as		classified as		classified as	
	valid	invalid	valid	invalid	valid	invalid
valid	95.97%	2.56%	58.01%	24.46%	0.0%	0.00%
invalid	1.47%	0.00%	3.90%	13.64%	23.90%	76.10%
precision	98.50%		93.71%		-	
recall	97.40%		70.34%		-	

The out of field data sets provide a special case, as there are no crop row patterns visible in the data and thus all detected crop row patterns are invalid. About 76% of the patterns are true negatives in both runs and would be filtered out for usage in a crop-row-based application. However, the remaining 24% would be integrated and potentially cause errors in localization estimates or mapping approaches based on crop rows. We will present a solution for this issue in the second experiment.

In the lower part of the tables in Table 2.1, we show the precision and recall values. Note that, for the out of field data sets, these values carry no information since there are no correct patterns, which is why we omit them. We aim for a high precision value. A high precision implies a high true positive to false positive ratio. Thus, when filtering according to our quality measure, for example in a localization approach, the ratio of valid to invalid patterns that will be used is high. The higher this value, the less likely it is that the localization pose diverges – a lot of valid patterns will be used, but only few invalid patterns will be integrated. The recall value indicates how many correct patterns are filtered out, i. e., how much correct data we lose. While it is desirable to keep as much correct data as possible, it is not as crucial as filtering out wrong data. In the in row data sets, precision and recall are both high with values above 97%. In the transition data set, the precision values are still high (about 96% and 94%, respectively), but the recall drops

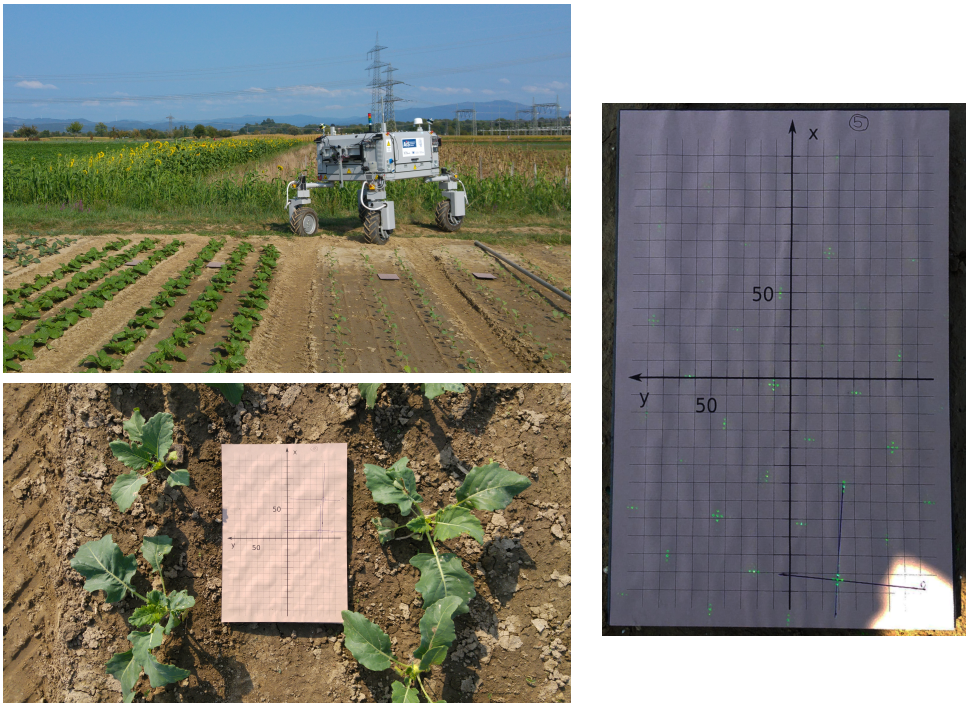


Figure 2.11: These images show how we acquired the ground truth poses of the robot. We placed markers at the beginning, mid and end of each crop row (left). On the robot, we rigidly mounted a laser pointer with a clearly visible mid point and orientation. Whenever the robot passed a marker, we marked its position and orientation on the marker using the laser pointer (right). The coordinate system enables us to accurately determine the robot pose.

to about 55% and 70%. This means that we filter out about 45% and 30% of correct data, respectively.

Overall, the detected pattern classification using our quality measure yields high precision. While the recall is not that high in more challenging data sets, the results of the following experiment show that enough correct data remains to robustly localize a robot in the field.

2.5.2 Evaluation of the Impact of Filtering Patterns in a Crop-Row-Based Localization Method

To evaluate how filtering detected crop row patterns according to our quality measure impacts the performance of a localization method, we used the following setup. We placed markers with coordinate systems (see Figure 2.11, bottom left) at the front, mid and end of each row. We rigidly installed a laser pointer at the bottom of the robot, pointing downwards. The laser pattern has a clear midpoint and orientation. Whenever the robot passed one of the markers, we marked its pose on the marker, using the laser pattern (see Figure 2.11, right). We additionally marked the corresponding time stamp. This gives us

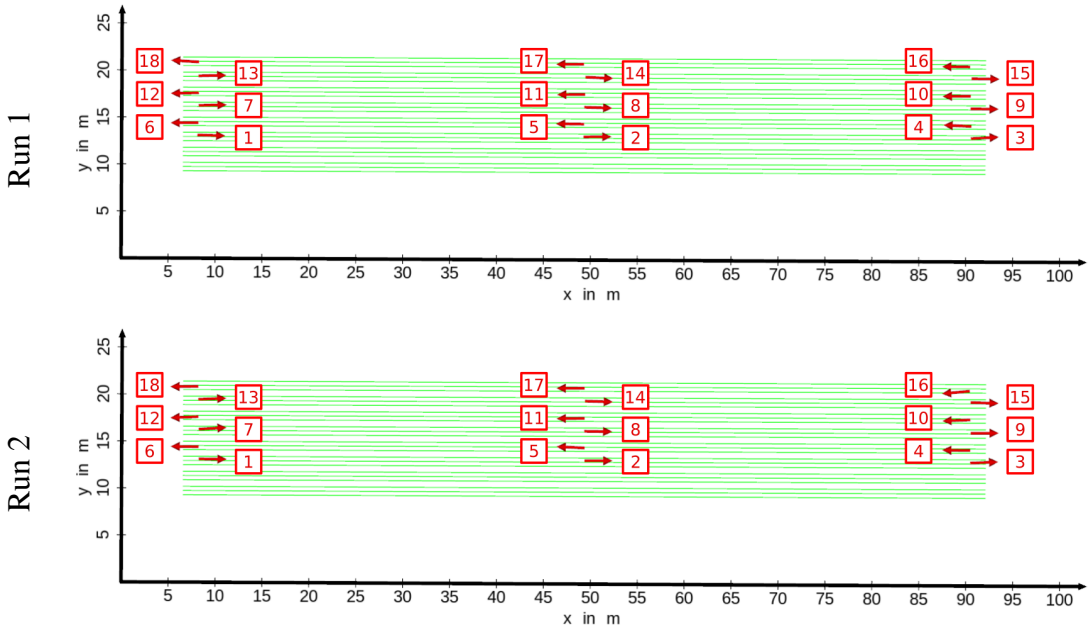


Figure 2.12: In this schematic, we indicate the ground truth poses of the robot for both runs as red arrows. The red numbers indicate the marker index of the respective position. The crop rows are represented as green lines. These correspond to the semantic map of the field that the localization algorithm uses. The images were taken from Winterhalter et al. [90].

the ground truth poses of the robot at certain timestamps. We estimate the measurement error of this method to 5 cm in either translational direction, and 3 ° for the angle. We then run a crop-row-based localization as presented by Winterhalter et al. [90] on the data collected during the runs.

The localization approach uses a semantic map of the field, where crop rows are given as lines (see Figure 2.12). In this figure, we added the ground truth poses of the robot at the marker positions in both runs. The localization algorithm uses the odometry of the robot as initial guess for its movement. It employs the detected crop row patterns for lateral and angular pose correction with respect to the crop rows in the semantic map. In addition, the detected pattern extension as explained in Section 2.4 is used to provide a longitudinal correction whenever the end of the field is visible. Whenever the end of the field is not visible, a longitudinal correction is computed using a global navigation satellite system (GNSS) sensor. All these corrections are applied in an Extended Kalman Filter [83].

Filtering the detected crop row patterns according to our quality measure helps to prevent integrating wrong patterns in the localization algorithm. Patterns with low quality are not used for pose correction and thus cannot mislead the localization method. As the robot approaches the end of the field, the crop rows leave its field of view. Thus, once the end of field was detected and, according to the localization pose and extrinsic sensor calibration, the crop rows cannot be visible in the sensor data anymore, we disable pattern

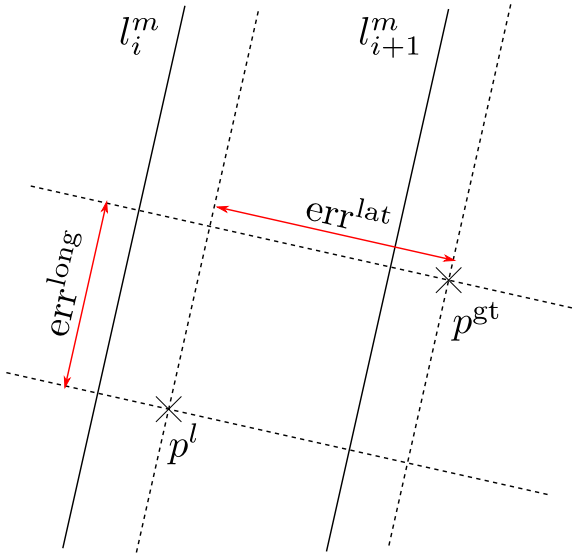


Figure 2.13: This schematic shows how the lateral and longitudinal localization errors are computed as introduced in Winterhalter et al. [90]. The ground truth position of the robot is denoted by p^{gt}, the localization position estimate by p^l. The lateral and longitudinal errors are computed with respect to crop rows in the semantic map, here shown as lines l_i^m and l_{i+1}^m. The lateral error is the difference from the position estimate to the ground truth position orthogonal to the crop rows, denoted by $\mathrm{err}^{\mathrm{lat}}$. The longitudinal error of the position estimate is its difference to the ground truth position in the direction of the crop rows, denoted by $\mathrm{err}^{\mathrm{long}}$. In this example, the localization position estimate is roughly one crop row off to the side.

integration in the localization. It is re-enabled once the robot has turned so that it should be able to see the crop rows again. This solves the issue we discovered in the previous experiment, that about 23% of wrong patterns are not filtered out in the out of field data set. The lateral and longitudinal localization errors are computed with respect to the crop row direction in the map as proposed in Winterhalter et al. [90] and shown in Figure 2.13. We provide lateral and longitudinal error separately instead of providing a translational error that contains both, because the lateral error with respect to crop rows is more relevant for in-field navigation. A lateral error of 15 cm means that the robot is potentially driving over crops, which we want to avoid. On the other hand, a longitudinal error of 15 cm is not relevant in this scenario. A localization pose estimate is sufficiently accurate for our purpose, when the lateral error is below 10 cm and the angular error is below 10 °.

The localization errors are shown in Figure 2.14 for localization with filtering out unreliable crop row detections according to our quality measure (*quality*) and without filtering (*no quality*). We do not provide longitudinal localization errors for markers in the middle of the field, as it is difficult to obtain a longitudinal ground truth estimate relative to the beginning or end of the crop rows when they are far away. To investigate the cause of localization errors in greater detail, we show when and how patterns were integrated close to specific marker positions in Figure 2.15. To give an intuition about how to read

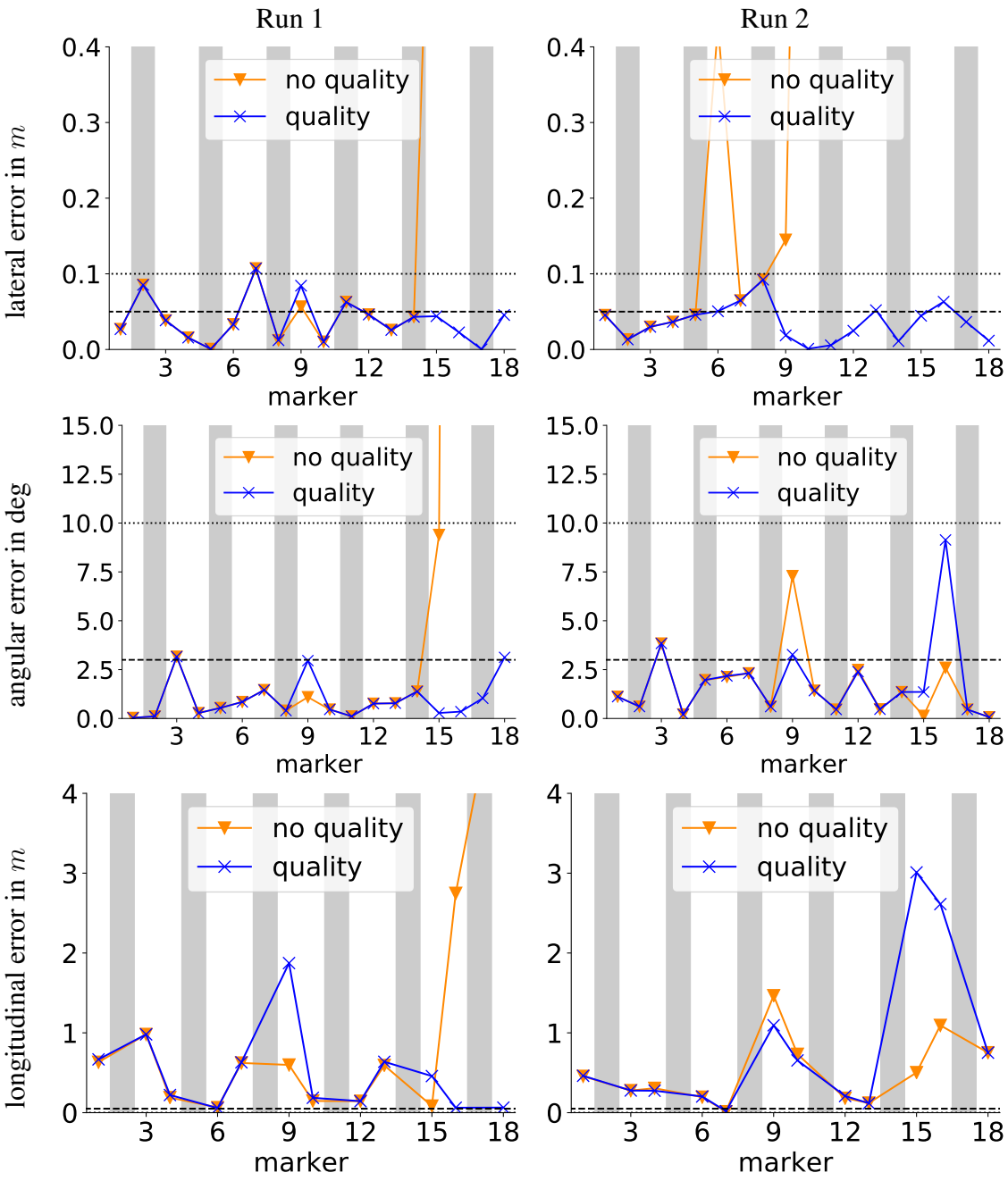
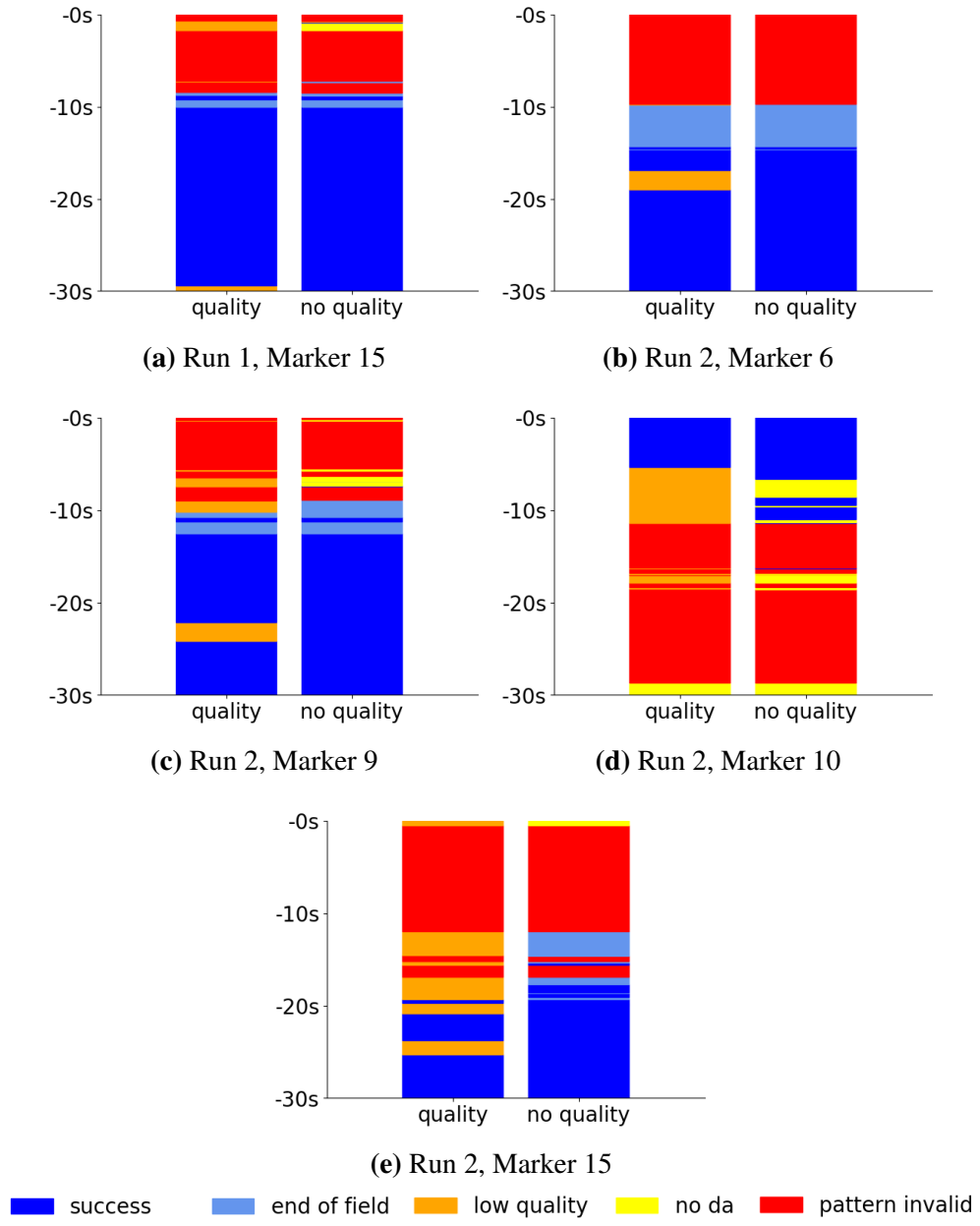


Figure 2.14: These graphs show the lateral (top), angular (middle) and longitudinal (bottom) localization errors for both runs. Markers in the middle of a row are marked in light gray to provide a better overview. Dashed lines indicate the measurement error, dotted lines indicate the threshold used for successful localization. The graphs labeled *quality* show the respective localization pose errors when our crop row pattern quality filter was applied. The graphs labeled *no quality* show the errors when the filter is not applied.



(a) Run 1, Marker 15

(b) Run 2, Marker 6

(c) Run 2, Marker 9

(d) Run 2, Marker 10

(e) Run 2, Marker 15

Figure 2.15: These bar plots show if and how the detected crop row pattern was integrated into the localization estimate in the last 30 s before a marker was reached. The left bar shows the result when we apply our quality measure, the right one when we don't use it to filter out patterns. Success (dark blue) indicates that a pattern was integrated using lateral and angular correction. End of field (light blue) indicates that the end of the field was detected using the pattern extension, and a corresponding longitudinal correction was applied in addition to lateral and angular correction. Low quality (orange) are cases where the detected crop row pattern was filtered out by our quality measure and no pose correction from the pattern was applied. No data association (no da, yellow) indicates that the localization algorithm did not find crop rows in the map that would match the detected crop rows. Pattern invalid (red) means that the detected pattern has too few lines. Thus, red, orange and yellow all indicate that the pattern was not integrated into the localization estimate, for different reasons.

these plots: In Run 2, Marker 9 (Figure 2.15c) at about -23 s, the plot for using quality shows *low quality* (orange). This indicates that about 23 seconds before the marker was reached, the quality of the detected pattern was so low that it did not get integrated into the localization pose estimate.

The most distinct observation in Figure 2.14 is that in both runs, the localization pose diverges after a time when low quality patterns are not filtered out, especially in terms of the lateral error (top). In Run 1, this happens at Marker 15, where first the lateral and angular error are high and then the longitudinal error rises as well. Studying the pattern integration before Marker 15 was reached in Run 1 (Figure 2.15a), we observe that some detected crop row patterns get filtered out by our quality measure, but they get integrated if we do not apply our classifier. For example, around 30 s, 8 s and 1 s before the marker was reached, patterns are filtered out. Some of these are not integrated into the localization pose when the quality filter is not applied because no data association is found. Of those few that do get integrated, some are wrong. This suffices to mislead the localization and causes the pose estimate to diverge.

In Figure 2.14 on the top right we observe that, when the quality filter is not applied, the lateral error goes up to about 15 cm in Run 2 at Marker 9 and higher than 40 cm at Marker 10. The localization pose gets lost irrevocably. Similar to what we saw in Figure 2.15a, some patterns that are integrated into the localization pose when the quality filter is not active are classified as invalid and ignored when the filter is applied, as shown in Figure 2.15c and Figure 2.15d. This again causes divergence of the localization pose when the quality filter is not applied. The angular error remains below our threshold for successfull localization for Markers 11 to 18 (Figure 2.14, middle right) and the longitudinal error remains below 2 m (Figure 2.14, bottom right). This indicates that the localization pose estimate moves parallel to the ground truth pose and follows a different crop row starting at Marker 11.

In Run 2, the lateral error already has a peak at Marker 6 when our quality measure is not used, but the localization algorithm is able to recover the pose. Figure 2.15b shows that some detected crop row patterns are filtered out about 18 s before reaching the marker and then again a single one about 10 s before reaching it when the filter is applied. All of them are integrated when this is not the case. This causes the high lateral error in Run 2 at Marker 6 when the quality filter is not applied.

Interestingly, at Markers 15 and 16 in Run 2, the longitudinal error is larger when filtering patterns than without filtering (Figure 2.14, bottom right). In the pattern integration (Figure 2.15e), we observe that the end of the field is detected in some of the crop row patterns, as indicated by the light blue parts in the right bar. However, these patterns are filtered out by our classifier when it is applied due to their low quality score. Thus, the localization algorithm is missing some longitudinal correction data that is evidently correct.

In conclusion, integrating even only a few patterns with low quality in a crop-row-based localization method can cause the irrevocable loss of the correct localization pose as observed in Figure 2.14. Filtering out patterns with low quality prevents the localization pose from diverging. However, we also discovered that the loss of data with low quality that might still be correct can cause errors in the localization pose.

When the quality filter is applied, both lateral and angular localization errors are low enough to ensure safe navigation in a crop field without crushing any crops. Overall, filtering the detected crop row patterns according to our quality measure is able to prevent the localization pose from diverging and decreases the localization error to suitable bounds.

2.6 Conclusions

Multiple tasks in autonomous navigation in agriculture require reliable crop row detections. Examples include localization, row following, or mapping approaches. Many crop row detection methods require input from a specific sensor type. In this chapter, we presented a vegetation feature representation that generalizes over different types of sensor data. This enables crop row detection approaches to be independent of the input sensor type and thus be more flexible in their application to various plant types, plant sizes, and lighting conditions. Our vegetation feature representation is a 2.5-dimensional grid map located on the ground plane, where the value of each cell indicates the likelihood that this cell contains vegetation.

Furthermore, we aim to prevent crop-row-based methods such as a localization or row following approach from integrating incorrect or unreliable crop row detections, thus saving them from divergence. To achieve this, we provided a novel quality measure for crop row detections and defined a binary classifier on it. Applying the classifier allows crop-row-based applications to filter out potentially incorrect detections. The quality measure analyses the support of the detected crop rows in the vegetation feature map. Two key aspects are that there is a high vegetation feature density close to the detected crop rows, and low feature density farther away from the crop rows, which corresponds to crop centers being close to the detected crop rows.

We tested our crop row detection classifier in extensive experiments with different plant types and sizes and applied it in a crop-row-based localization as a common use case. We determined that our vegetation feature representation is suitable for usage in this scenario. More importantly, we showed that our approach is able to reliably filter out most invalid crop row detections and applying it in a crop-row-based localization prevents the estimated pose from diverging. We used this method in a fully autonomous navigation approach on the BoniRob presented by Pretto et al. [73].

We also observed that, while the lateral and angular localization errors improve by filtering out crop row detections with low quality, the longitudinal pose estimate sometimes suffers when correct patterns are not integrated due to low quality. A next step would be exploring methods to improve longitudinal localization errors, for example by mapping and localizing with respect to the positions of individual plants. Additionally, using the crop row quality as a confidence score instead of employing a binary classifier on it should be investigated.

The presented crop row classification method helps robustify crop-row-based localization algorithms which provide a key component for autonomous navigation in agriculture. An estimate of the current pose is needed for global planning to reach a given goal pose

and it is crucial for executing a computed path accurately. We cover global path planning and local planning for path execution in the following chapters.

Chapter 3

Evaluation of a Path Planner for In-Field Navigation

For precision farming, it is often unnecessary to traverse the whole field, as only certain parts of it need to be treated. Thus, a pure row following approach that proceeds row by row is suboptimal. A global path planner enables a robot to find an efficient path to any point in the field. In previous work, we developed a global planner that considers robots with high ground clearance and additional intrinsic configuration parameters that are relevant for the feasibility of some motions. Examples for such configuration parameters are settings of joints for adjusting wheel positions or manipulator arm joints that have to be considered for collision free navigation. Robots that are able to actively control these joints are particularly useful for navigation in narrow spaces with many obstacles like greenhouses or on uneven paths to a field. In this chapter, we summarize the technical contribution of our previously developed approach [29] and present a new evaluation with regard to real world applicability. We create a particularly challenging environment in order to determine the limits of this approach. In addition, we determine the performance of our planner by evaluating the planning time consumption and the cost of the resulting plans.

3.1 Introduction

While classical methods for automated treatment of an agricultural field include a uniform appliance of chemicals such as herbicides and pesticides, precision agriculture aims for treating individual plants to minimize the usage of chemicals and reduce the resulting burden on the environment. Thus, a vehicle is no longer required to traverse the whole field. Instead, a robot with an autonomous navigation module should only drive to locations where treatment or data collection is needed. Consequently, a path planner is necessary to reach a given goal position in an efficient manner. While many planning approaches exist, a method that takes the specific abilities of a robot into account is desirable to increase efficiency. We presented such an approach in earlier work [29]. The planner presented

therein is based on a state lattice planner [72], which takes into account the specific driving motions a robot is able to perform. In our earlier work, we extended the classical state lattice planner to incorporate intrinsic joint configuration changes in an efficient manner. It employs a novel state space representation that uses joint configuration intervals. These joints could for example control the wheel positions. Changing them can enable a robot to drive through narrow spaces or pass over obstacles, if the ground clearance is high enough. A different example where considering the intrinsic configuration benefits path planning is a robot with a manipulator arm. Such a robot might be able to drive through a tight space with its arm retracted, but not when it is outstretched. We further presented a novel heuristic in our earlier work, the Wheel Dijkstra heuristic, that is useful for robots with high ground clearance. It finds a collision free path for each wheel and combines the resulting path lengths to estimate the minimum path length the robot has to travel.

In this chapter, we extensively evaluate our planner introduced in previous work. We perform two baseline experiments in simple, non-cluttered simulated environments. We further investigate its suitability for path planning in real world agricultural environments and verify the performance in a real world field environment in comparison to the performance on a simulated field environment. Additionally, we create a cluttered environment with many narrow passages to test the limits of this approach and the benefit of including intrinsic configuration parameters into the planning considerations. Moreover, we inspect the planning performance with different heuristics, namely the Euclidean heuristic and the freespace mechanism heuristic [59] as two baselines, the Wheel Dijkstra heuristic, and a combination of the Wheel Dijkstra with the freespace mechanism heuristic.

The experiments demonstrate that the planning performance with our state space representation from earlier work transfers well from a simulated to a real world environment. While the investigated heuristics lead to varying performances in the different environments, a joint result in all environments is that combining the Wheel Dijkstra heuristic with the freespace mechanism heuristic yields the best results among all considered heuristics.

Substantial parts of the ideas, figures and results presented in this chapter have been previously published [30]. Section 1.4 outlines the author's contribution to this work.

3.2 Related Work

Path planning in narrow spaces requires considering intrinsic joint configurations for maximum movement flexibility. This implies using additional degrees of freedom (DoF) in planning instead of limiting planning to the robot pose. These additional DoF make the planning problem more complex.

In the field of search and rescue robotics, robots with additional DoF from adjustable joints are often used to overcome obstacles. Some approaches adjust so-called flippers of tracked robots based on current sensor data when needed [69]. While such a reactive method may help to overcome some obstacles, in some scenarios it is necessary to plan ahead for changing the intrinsic configuration in time. Other approaches that do not actively control the adjustable joints [37] have the same shortcomings. Other approaches

rely on partial autonomy and require tele-operation in some scenarios [18]. We aim for an approach that does not involve human intervention.

Several sampling based approaches have been proposed to deal with many DoF. They sample a subset of the configuration space. Examples are rapidly-exploring random trees [57] and variants thereof [45, 54] or probabilistic road maps [48]. More sophisticated approaches use additional data for generating more informative or goal oriented samples of the configuration space [9, 17, 76, 86]. Sampling based approaches usually assume that, if an obstacle free transition between two states exists, one is reachable from the other in a continuous manner. Some robots are unable to change their intrinsic configuration while driving, or would cause great mechanical stress when doing so. Therefore, we pursue an approach that does not rely on this assumption. In narrow environments, sampling based approaches are prone to sampling configurations and poses that are in collision or unreachable from a different sampled configuration. Thus, they require a lot of sampling until a valid configuration that connects with an already sampled configuration is found. Furthermore, producing efficient plans is important for navigation. This has recently received interest for sampling-based planning [33], but is well understood for search-based planning [53].

We therefore aim for a search based planning approach. A simple grid based method is a Dijkstra search on a grid map of the environment [26]. Search based planning is adaptable to various scenarios with different DoF, such as footstep planning for four-legged robots [96], navigation planning with 3d collision checks [41] or path planning for parking cars [59].

The main challenge for path planning in an agricultural environment for a robot with adjustable joints is how to efficiently find a high quality path. One possibility to deal with the additional DoF from the adjustable joints is using adaptive dimensionalities for planning. Gochev et al. [38] propose planning for a subset of configuration parameters that are relevant for a resulting path. The path computed in the lower dimensional configuration space is later enhanced by deducing the values of the remaining configuration parameters. As we are considering a scenario where all configuration parameters are relevant for the feasibility of a motion, the space containing only crucial configuration parameters is the same as the original planning space. A different idea in the context of adaptive dimensionalities is to change the relevant configuration space according to the current planning phase. The planning phase is determined by the distance to the start or goal [59] or the distance to obstacles [49, 87]. For our planning challenge, obstacles can be encountered at any point and changing joint configurations might be required at any time. Thus, we cannot only focus on the start and goal areas. In addition, while navigating on a field, obstacles in the form of crops are always close to the robot. An approach that adapts the dimensionality dependent on the distance to an obstacle would therefore always choose the highest dimensionality. We thus decided against an approach with adaptive dimensionalities and instead extended a state lattice planner [72] to fit our needs in earlier work [29], which this chapter is based on.

Guiding a search based planner towards the goal by using an informative heuristic is an important tool for increasing planning efficiency [13, 23, 61]. The Euclidean distance to the goal position is often used as a heuristic. However, it does not take into account any

motion constraints or obstacles and thus tends to underestimate the true cost considerably. The freespace mechanism heuristic [59] remedies the first shortcoming of the Euclidean distance heuristic by explicitly modeling the motion capabilities of the robot. It still does not consider obstacles. For taking into account obstacles, the Dijkstra heuristic based on the Dijkstra algorithm [26] is often used [41]. When using the Dijkstra heuristic, algorithms usually check whether the robot center is located on an obstacle cell on a grid map. For a robot with high ground clearance, collision with an obstacle is only immediately implied when one of the wheels is located on a cell containing an obstacle, as it is able to pass over low obstacles. The Wheel Dijkstra heuristic we proposed in previous work employs this information [29].

3.3 Planning with High Ground Clearance and Adjustable Joints

Here, we will briefly summarize the concept of our previous work on path planning to make the evaluation more accessible to the reader. The planner we developed is suitable for ground robots with high ground clearance and intrinsic configuration parameters that are relevant for the feasibility of some motions, such as angles of specific hinge joints or the configuration of prismatic joints. A high ground clearance enables a robot to drive over certain obstacles, and adjustable joints make certain motions in tight spaces possible that would result in collisions if the joints remain fixed. Relevant joint configurations are for example those that control the wheel positions or width of a robot as for the BoniRob (see Figure 1.1), the Ted robot [67], TREKTOR [81], or a robot designed for space exploration presented by Karamipour et al. [46]. The Ted and TREKTOR robots are additionally able to change their height, which can be another important setting. For any robots with manipulator arms, like the PR2 [42], the arm configuration may also be relevant for collision free navigation.

In the evaluation, we concentrate on the BoniRob as an experimental platform, which has individually adjustable wheel positions. The approach is also viable for other robots with different joints for changing wheel positions or other configurations that are relevant for collision free planning, such as adjustable robot height. Some valid joint settings for two robot types are shown in Figure 3.1. For the BoniRob, we consider the angles of the lever arms to which the wheels are attached as relevant joint configuration in the planning problem. Similarly, for a robot with prismatic joints for changing wheel positions, these joints have to be considered. Figure 3.2 illustrates how different joint settings enable certain motions.

Our robot navigates in a 3-dimensional environment, where the only relevant information in the third dimension (up/down) is the height of obstacles and the ground. Therefore, we represent the environment as a 2.5-dimensional grip map containing elevation and traversability information.

Those parts of the robot configuration that are relevant for planning are its 3d pose in the environment and the intrinsic configuration parameters that enable or disable certain

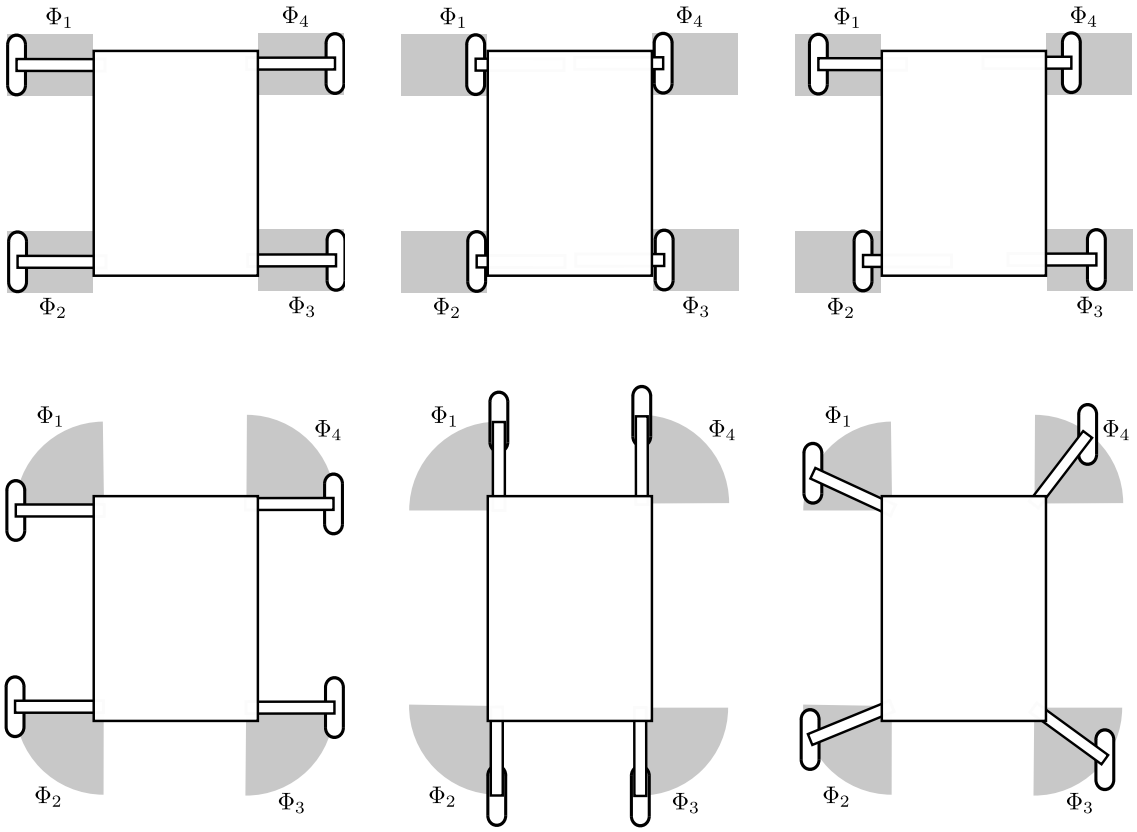


Figure 3.1: Some robots are able to change their wheel positions. At the top, possible configurations for a robot with prismatic joints for adjusting wheel positions are shown. At the bottom, possible configurations for robots like the BoniRob that can change wheel positions by adjusting lever arm angles are shown. The leftmost schematics show the widest possible track width setting, the schematics in the middle show the most narrow track width setting, and on the right we show an example setting for individually controllable joints. The light gray background shows the extent of valid joint settings, $\Phi_1, \ldots, \Phi_4$.

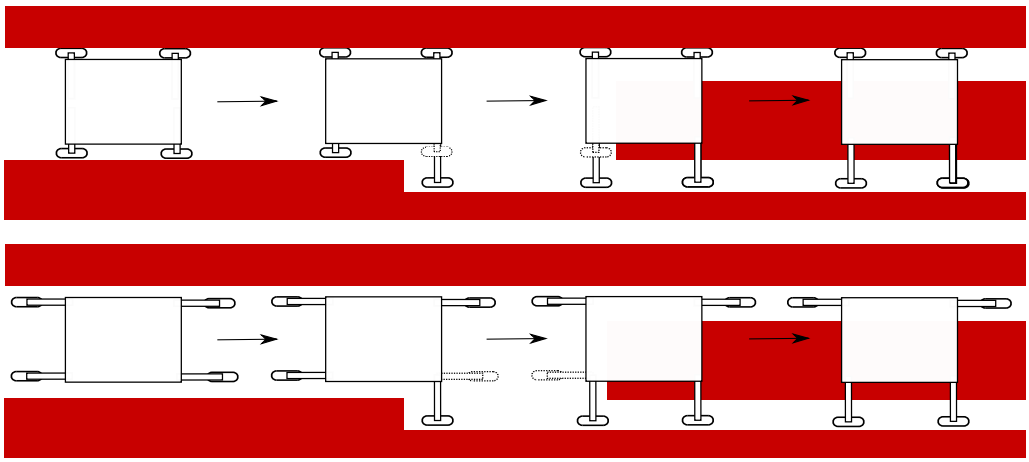


Figure 3.2: This schematic illustrates how changing joint configurations enables certain motions. Obstacles are shown in red. In order to keep driving in this narrow passage, robots have to first change one of their front wheel positions by changing the corresponding joint, and then one of their back wheel positions. The top shows the procedure for a robot with prismatic joints for adjusting wheel positions. The bottom shows the same procedure for robots that change their wheel positions by adjusting lever arm angles, such as the BoniRob.

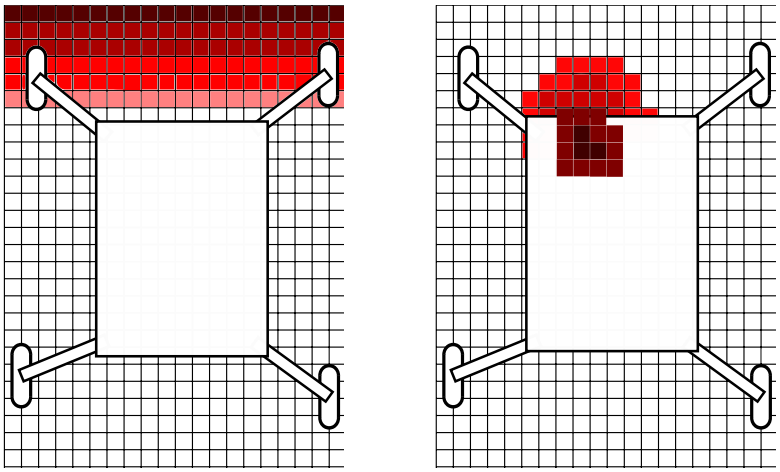


Figure 3.3: Examples of invalid robot configurations for robots like the BoniRob. Untraversable grid map cells are marked in red, where a darker shade represents a higher elevation. On the left, two of the wheels are in collision with an obstacle. On the right, while the robot is able to pass above the lower parts of an obstacle due to its high ground clearance, it collides with the higher parts of the obstacle.

motions. We describe the robot pose as rigid body motion in $SE(3)$ relative to a fixed coordinate frame in the environment. We denote the relevant intrinsic configuration space by C_R. The space that represents the robot state relevant for planning, called the configuration space, is therefore defined by $C = SE(3) \times C_R$. For a robot with adjustable wheel positions, the relevant intrinsic configuration is comprised of the joint settings that control the wheel positions. We use the BoniRob as an example. The set of feasible joint angles for all four lever arms construct the relevant intrinsic configuration space $C_R = \Phi_1 \times \Phi_2 \times \Phi_3 \times \Phi_4$, where Φ_i is the set of valid angles for the i-th joint as shown in Figure 3.1 at the bottom. Likewise, for a robot with prismatic joints to adjust the wheel positions, C_R contains all valid settings of these joints. It is noteworthy that the robot pose can be changed by modifying the intrinsic configuration. For example, moving one wheel to a lower part of the ground can change the roll or pitch of the robot.

The set of valid configurations $C_f \subset C$ contains all configurations that are safe for the robot in the given environment. This entails that the robot is not in collision and is stable. Figure 3.3 shows examples of invalid robot configurations for the BoniRob.

For ground vehicles, the roll, pitch and height are not directly controllable. We define the *actively controllable configuration space* C_a of the robot as $C_a = SE(2) \times C_R$. Given these values, the roll, pitch and height can be computed from the traversability grid map of the environment.

A motion m can be applied to a robot configuration c_1 and yields a new robot configuration c_2. A feasible motion given the environment and a robot configuration $c \in C_f$ is a motion where the robot configuration remains in the valid configuration space C_f at all times during the execution of the motion. For the BoniRob, this means that the wheels do not cross untraversable cells and the robot is not in collision, considering the ground clearance of the robot and the height of obstacles.

A given goal robot configuration c_g is reachable from a given start configuration c_s if there exists a sequence of feasible motions $m_1, \ldots, m_n$ that yields configuration c_g when applied to c_s.

Some robots are unable to change their joint configuration while driving, either due to physical constraints or to avoid mechanical stress on these joints. We therefore distinguish between *drive motions* that only change the robot pose, and *configuration change motions* that change the intrinsic configuration and may change the roll, pitch or height of the robot.

We aim for a planning approach that is flexible with regard to the robot being used and to the task at hand. As such, we chose a search based method, where the cost function can be easily adjusted to the current needs. Since we model drive motions and configuration change motions separately, we employ a state lattice planner [72] and extend it to represent the configuration change motions.

In order to use a state lattice planner, we discretize the actively controllable configuration space and construct predefined actions, so called motion primitives, between them. These motion primitives are short standard actions the robot is able to perform and thus have to be defined for each robot individually. Examples for motion primitives are driving one grid cell forwards, backwards or sideways, or turning on the spot by a certain amount. Together with the configuration change motions, the motion primitives form the

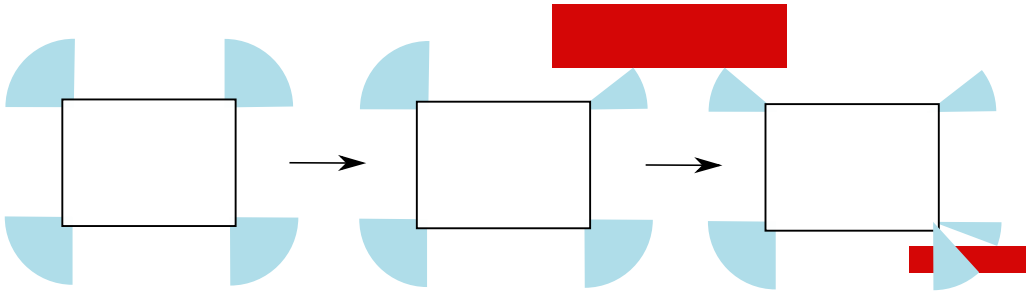


Figure 3.4: This schematic shows how, for robots like the BoniRob, valid joint angle intervals get reduced and sometimes split by certain motions in a given environment. This concept of reducing and splitting valid joint setting intervals by moving transfers to robots with other joint types and different motion capabilities. The currently valid angle intervals are shown in light blue, obstacles in red. In the first step, the robot encounters an obstacle on the front left, due to which the valid angle interval for the front left lever arm gets reduced accordingly. Equivalently, the back left wheel potentially encounters the same obstacle in the next step and the corresponding joint angle interval gets reduced. Additionally, there is an obstacle on the front right of the robot. It is still valid to have the wheel left or right of the obstacle (provided the ground clearance of the robot is high enough to pass over the obstacle), only the part of the interval that would lead to a wheel colliding with the obstacle has to be removed. The result is an interval split for the front right wheel.

set of valid motions M. We build a directed search graph where the valid robot configurations are represented as nodes. A node n_i is connected to node n_j by an edge, if and only if there is a valid motion $m \in M$ that is also feasible and connects the robot configurations represented by n_i and n_j, respectively. This edge represents the motion that connects the two configurations.

We further define a cost function u on the graph edges, which gives us for any motion m the cost to execute it in the current state s: $u : C_a \times M \to \mathbb{R}; (s, m) \mapsto u(s, m)$. It is based on the time it takes to execute a certain action and user preference, e. g., it might be prefered if the robot drives forwards instead of backwards.

For any given start configuration c_s and goal configuration c_g, the goal of our planner is to find a sequence of feasible motions from c_s to c_g with the lowest cost. Thus, we find the nodes in our graph that correspond to c_s and c_g, respectively. Running a search algorithm on the graph between these two nodes provides us with a sequence of feasible motions, represented by the edges in the graph.

In order to increase the efficiency of our planning approach, we extended it twofold. The first extension concerns the representation of the intrinsic robot configuration in the state space. Initially, each robot state $s \in C_a$ includes a single value for each of the joints. If we consider a robot with four wheels where the position of each is controlled individually by a single joint, a robot state is given by $s = (x, y, \theta, \varphi_1, \varphi_2, \varphi_3, \varphi_4)$, where the φ_i are the settings for each joint. We proposed to instead include a whole interval of valid joint settings in the state space representation. The new state space is thus defined as

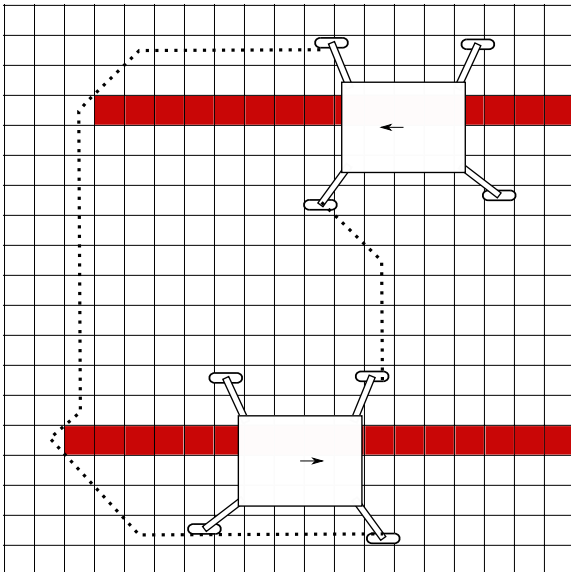


Figure 3.5: The Wheel Dijkstra heuristic provides an estimate of how long it takes to move all wheels from their start positions to their desired goal positions. Obstacles are shown in red. They are low enough so that the robot can pass over them. We assume an 8-connected grid and perform a Dijkstra search from each wheel start position to their respective goal position. For two of the robot wheels, a path that is a possible result of this search is shown as a dotted line. The path lengths for each wheel are quite different in this example. As the robot has to move all wheels to their desired goal positions, we use the maximum over all wheel path lengths as the most informative value.

$C_s = SE2 \times C_I$ where C_I includes all sets of valid joint setting intervals. In our example, this means

$$C_I = \{[\varphi_{1,1}, \varphi_{1,2}][\varphi_{1,1}, \varphi_{2,2}][\varphi_{3,1}, \varphi_{3,2}][\varphi_{4,1}, \varphi_{4,2}], \varphi_{i,j} \in \Phi_i\}.$$

As the current robot configuration is fixed when planning starts, the initial configuration contains only a single value in the interval for each joint. Applying a configuration change motion increases the intervals of all joints by one discretization step in either direction or to the boundary of valid joint settings. This represents that when a configuration change occurs, we do not consider each joint and its setting separately. Instead, we only need to know that any or all joint settings may have changed to either direction. Drive motions may reduce or even split intervals when the robot is close to obstacles, as illustrated in Figure 3.4. This results in two or more successor nodes in the search graph. We expect that including intervals of joint setting in the robot state leads to less node creations, as there will be a single successor node for all possible joint changes instead of a set of up to $2 \cdot J$ nodes, where J is the number of joints (changing any joint in either direction). For the BoniRob, this would yield up to 8 successor nodes – two for each of the four arms.

Our second contribution to increase the efficiency in solving the planning problem at hand was a novel heuristic function. It makes use of the high ground clearance of the robot and considers that the robot is able to drive over certain obstacles. Given a certain start

configuration c_s and goal configuration c_g of the robot, the wheel positions of the robot are fixed on the traversability grid map of our environment, i.e., we know for each wheel where it starts moving and where it is supposed to go. As the wheels should never collide with any obstacle, we run a Dijkstra search for each wheel from its start to its goal cell, where untraversable cells in the traversability grid map cannot be crossed. This yields the minimum path length that each wheel has to travel (with the given discretization and connectivity between grid cells), i.e., we get four values $h_i(c_s, c_g) \in \mathbb{R}, i \in \{1, \ldots, 4\}$. The most informative value is the maximum distance any of the wheels has to travel, $h(c_s, c_g) = \max h_i(c_s, c_g), i \in \{1, \ldots, 4\}$. Introducing a factor b that encodes how long it takes the robot to move a wheel through a cell, we get the Wheel Dijkstra heuristic $h_b(c_s, c_g) = b \cdot h(c_s, c_g)$ as an estimate of how long the robot will take to move from the start to the goal configuration. An illustration is shown in Figure 3.5. While this heuristic gives us an estimate of the minimum distance a wheel has to travel, it is not a given that the Wheel path computed by the Dijkstra search is executable. Since each wheel is considered separately, there may not be valid wheel positions for all other wheels following one wheel path. Furthermore, the robot chassis and possible collisions with it are not considered at all. Thus, a wheel path is not suitable to be executed, it only serves as an estimate of how long it takes to get to the goal.

In the following, we investigate how the planning performance with these extensions transfers to a real world environment and to more challenging environments that are difficult to navigate.

3.4 Experiments

In previous work [29], we evaluated our path planner on simulated environments that were relatively easy to navigate and did not require a lot of intrinsic configuration changes. In order to test the applicability in a real world scenario, we extend the evaluation to include planning requests in a real world environment. We computed the corresponding traversability grid map from a data set recorded on a leek field. To test the limits of the approach, we also created a particularly challenging environment with a multitude of obstacles and narrow passages. In this environment, the usefulness of changing joint configurations becomes apparent. Furthermore, we include an analysis of the suboptimality that is reached in the plans to investigate the efficiency gain from representing the intrinsic configuration as intervals instead of single values. We also include an analysis of the time it takes to get a first plan, the number of node expansions until a first plan is found, as well as the amount of successfully handled planning requests.

Overall, we determine the performance of the planner on four different environments, using our experiment platform BoniRob with the motion primitives shown in Figure 3.6. The three simulated environments all have sizes of 30 m × 30 m. The first is a *freespace* environment. It has no obstacles and serves as a baseline for our experiments. The second is a *field* environment, showing a common use case for the planner on our robot. The third is the *rocks* environment, which has a lot of obstacles and narrow passages, pushing the planning algorithm to its limits and showing the necessity of considering the joint con-

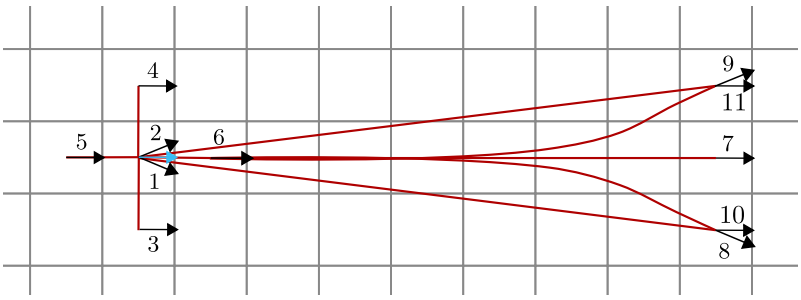


Figure 3.6: This schematic shows the motion primitives we use for planning on the BoniRob. The current robot pose is indicated as a light blue arrow. The movement patterns for all motion primitives are represented as red lines, with arrows at the end to illustrate the resulting robot pose. While the robot is mostly symmetric and able to drive in any direction due to its omnidirectional wheels, its sensors are mostly located at the front. We therefore decided against completely symmetric motion primitives. They are defined as follows. **1**: Turn 22.5° clockwise. **2**: Turn 22.5° counter-clockwise. **3**: Move sideways, one grid cell to the right. **4**: Move sideways, one grid cell to the left. **5**: Move backwards one grid cell. **6**: Move forwards one grid cell. **7**: Move forwards eight grid cells. **8**: Move eight grid cells forwards and one to the right in a curve, turn 22.5° clockwise on the way. **9**: Move eight grid cells forwards and one to the left in a curve, turn 22.5° counter-clockwise on the way. **10**: Diagonally move eight grid cells forwards and one to the right without changing orientation. **11**: Diagonally move eight grid cells forwards and one to the left without changing orientation.

Figure 3.7: This picture was taken on the leek field we used as real world environment.

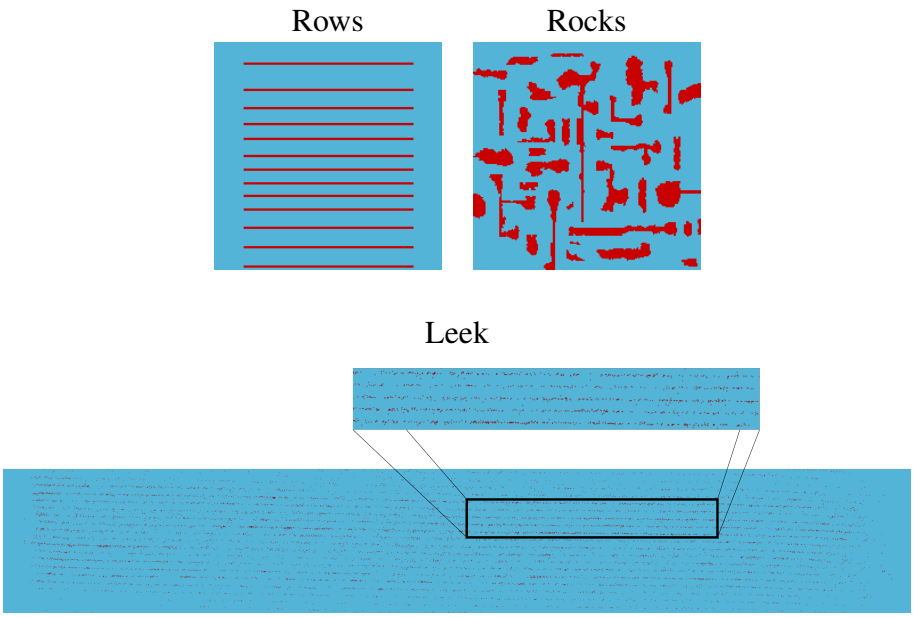


Figure 3.8: These are the rows, rocks and leek environments we use for testing the planner performance. Traversable grid map cells are marked in light blue, untraversable cells in dark red. The freespace environment is not shown, as it is just an empty map. The bordered box shows a magnification of the leek map, as it is rather sparse.

figurations. The real world environment is the *leek* environment, which is a traversability grid map that was created from real world data recorded on a leek field. The map size is roughly 15 m × 90 m. An example picture of the leek field is shown in Figure 3.7. This environment provides a sense of the real world applicability of the planning approach. The traversability grid maps corresponding to all environments are displayed in Figure 3.8. For the discretization of the robot position, we chose a resolution of 15 cm in the simulated environments and 5 cm in the real world environment. The resolution in the real world environment is higher due to the noise in the data. The robot orientation and the four arm angles are discretized with a resolution of 22.5 °. As search algorithm, we used the Anytime Repairing A* algorithm [60], which provides an upper bound for the suboptimality of a computed plan. We used an initial heuristic inflation factor of 5, which means that the first path that is found may have a cost that is up to five times higher than the cost of the optimal path.

We uniformly sampled 25 poses in each of the environments and created planning queries between each of those. To create realistic robot poses in the leek environment, we aligned them with the rows of the field. Here, we only ran planning queries for start and goal poses with a distance between 0.5 m and 20 m and used a timeout of 90 s. This often implies that the robot has to drive to the end of a crop row in the field, turn at the end, and then reenter the correct row. In the freespace and the field environment, we reduced the timeout to 30 s, as the environments are smaller and thus we expect a plan to be found faster. For the rocks environment, we reduced the maximum allowed distance between start and goal to 10 m and increased the timeout to 3 min, as planning is a lot harder there.

We investigate two main points: The impact of different heuristics, and the influence of employing an interval representation for the intrinsic configuration. In particular, we are interested in how well our results on simulated planning environments transfer to a real world environment, and how the planner behaves in extremely hard conditions with a multitude of obstacles and narrow gaps.

Evaluation of the Impact of Heuristics on Planner Performance

We study the effect of four different heuristics. The first is the freespace mechanism heuristic [59]. It was especially designed for state lattice planning and employs the provided motion primitives. It computes the path with the lowest cost assuming that there are no obstacles present in the environment. This was used as a first baseline. The second heuristic is the standard Euclidean heuristic on the robot base frame, which serves as another baseline. The third is the Wheel Dijkstra heuristic that we proposed in previous work [29] and explained in the previous section. The last is a combination of the Wheel Dijkstra and the freespace mechanism heuristic. We combine them by maximizing over their values, so that we get a better estimate of the true cost. Note that for two admissible heuristics, the maximum of both is again admissible. For all planning runs in this section, we use the interval representation for joint configurations.

We consider the percentage of solved planning queries over time and the reached suboptimality after the timeout. The percentage of solved planning queries indicates how fast the planner was able to find an initial plan. This initial plan is not necessarily optimal. This measure is important to know how much time passes until the robot could start moving towards its goal. The reached suboptimality indicates the efficiency of the best path computed within the timeout. A path with suboptimality 1 is an optimal path. The suboptimality value corresponds directly to the increase in cost, i. e., it is the factor by which the cost of the path is higher than the cost of the optimal path.

Figure 3.9 shows an overview of solved planning queries over time as well as the reached suboptimality in all four environments. In the freespace environment, the freespace heuristic shows better planner performance than either Euclidean and Wheel Dijkstra heuristic, as can be seen at the top of the figure. The planner is able to solve all planning queries in less than 5 s, and the reached suboptimality is no more than 2.2 for all queries. The Euclidean and Wheel Dijkstra heuristic yield planner performances similar to each other in the freespace environment. Both result in longer planning times, and only for about 10 % of queries, a suboptimality below 2 is reached. The similar performance of these two heuristics in the freespace environment can be explained by the fact that when there are no obstacles, there is a straight path from each wheel start position to each wheel goal position as well as from the robot start position to the robot goal position. Thus, the wheels have to travel roughly as far as the robot base. The main difference in this environment is that the Wheel Dijkstra heuristic indirectly considers orientation changes, which the Euclidean heuristic does not do. To illustrate the timing difference between the freespace mechanism and the Wheel Dijkstra heuristic further, we directly compare the time until a first plan is found in Figure 3.10 on the left. In this scenario, the freespace mechanism heuristic yields faster planning times. This can be explained by the next graph

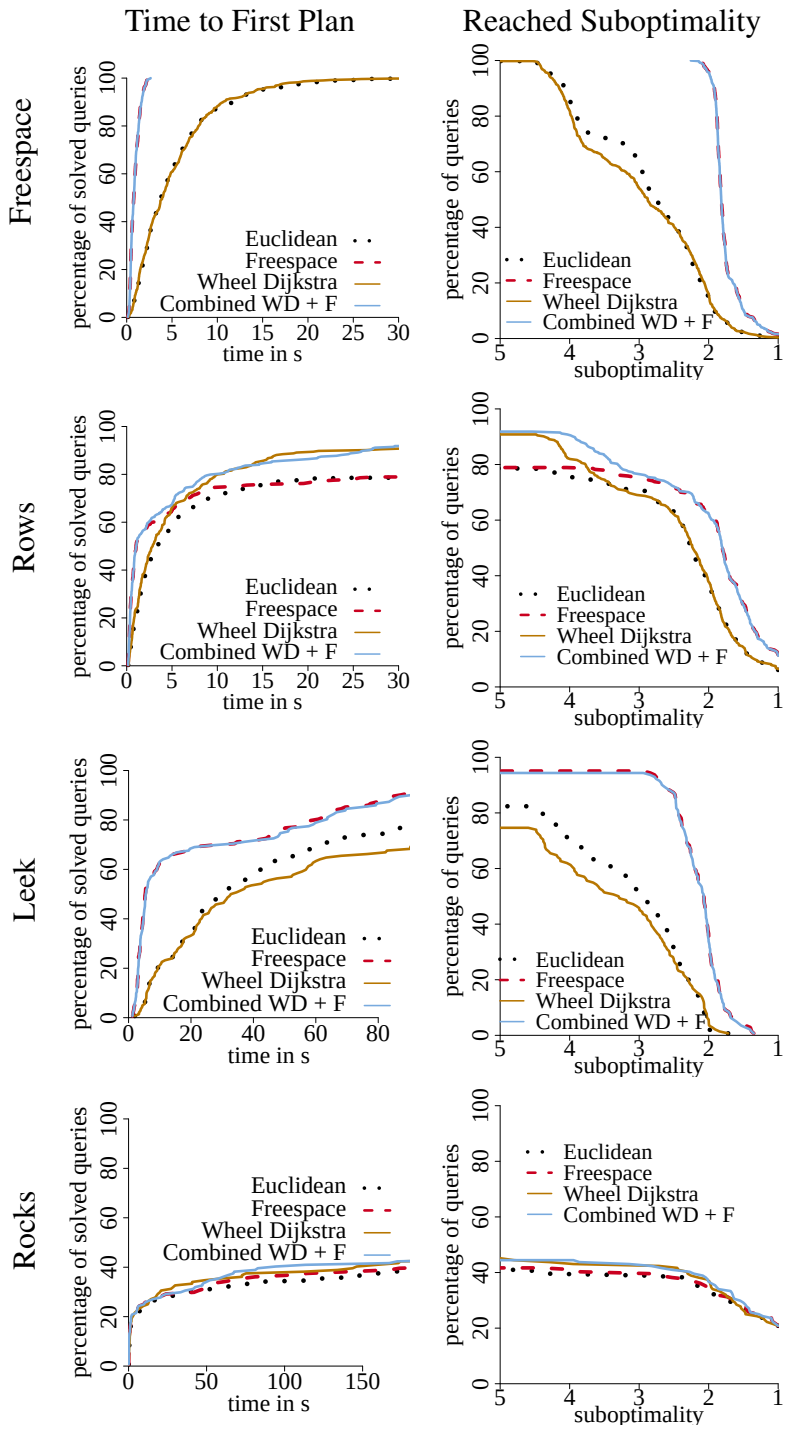


Figure 3.9: These graphs show the performance of the planner in all four environments with different heuristics. The left column shows the percentage of solved planning queries over time until the timeout is reached. A planning query counts as solved when a first plan is found, irrespective of the suboptimality of the plan. The right column shows a cumulative histogram of which path suboptimality bound was reached for which percentage of planning queries. The suboptimality indicates the upper bound of the factor by which the computed path cost is higher than that of the optimal path. A low suboptimality for a high percentage of planning queries is desirable.

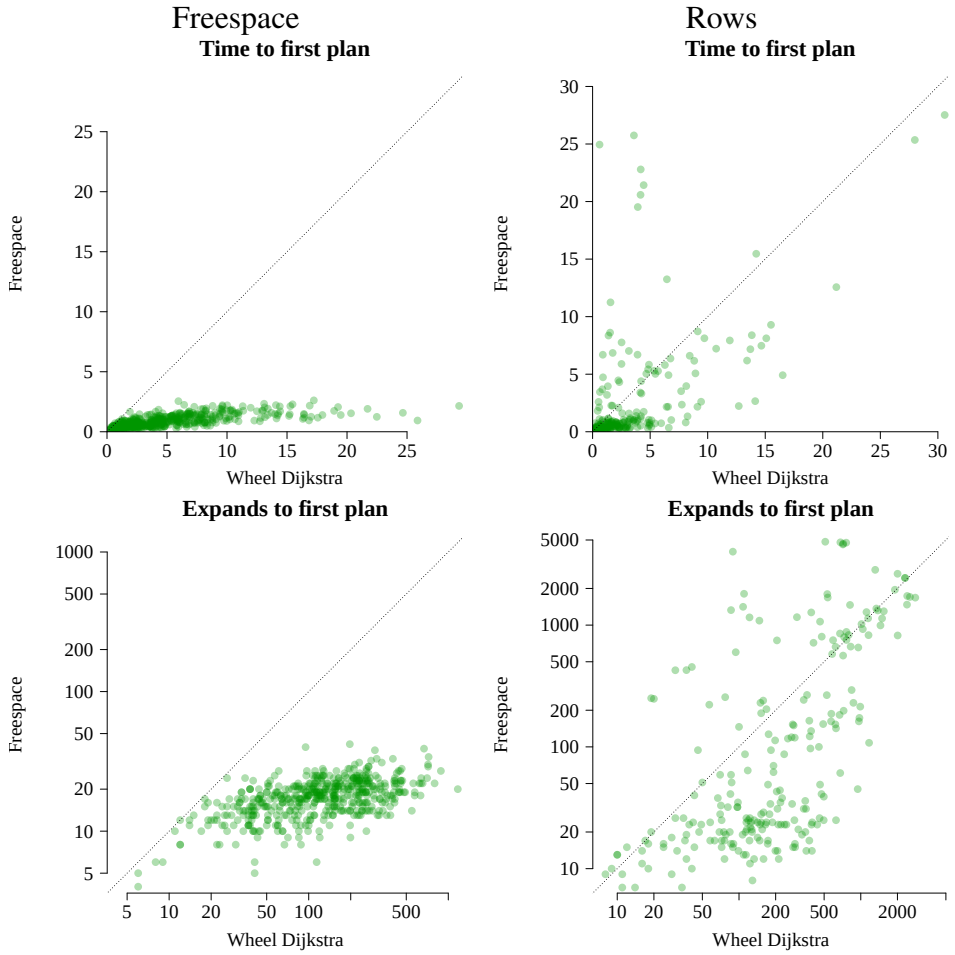


Figure 3.10: These graphs directly compare the time until a first plan is found and the number of node expansions between the Wheel Dijkstra and the freespace mechanism heuristic. Results are shown in the freespace (left) and the rows environment (right).

in this figure: It displays the number of nodes that were expanded until a first plan was found. Using the freespace mechanism heuristics means that fewer nodes need to be expanded in this environment. This is expected, as the freespace mechanism heuristic is the optimal heuristic in this space. Finally, the combined Wheel Dijkstra Freespace heuristic shows a performance that is close to the freespace mechanism heuristic in the freespace environment (Figure 3.9, top). Here, it is evident that the freespace mechanism heuristic and a combination with it outperforms other heuristics.

In the rows environment, planning with the freespace mechanism or the Euclidean heuristic is only able to solve about 80 % of the queries, while planning with the Wheel Dijkstra or the combined Wheel Dijkstra Freespace heuristic is able to solve 90 % of the queries. While planning is faster with the freespace mechanism heuristic for around 70 % of the planning queries, the Wheel Dijkstra heuristic yields plans faster for the remaining queries that it is able to solve. This is also depicted in Figure 3.10 in the top right graph, where the timings are compared directly between freespace mechanism and Wheel

Dijkstra heuristic in the rows environment. Mostly, the freespace mechanism heuristic produces faster results, but there are some outliers where the Wheel Dijkstra heuristic is faster. Again, this is reflected in the number of node expansions until a first plan is found as shown in the bottom right graph of this figure. In Figure 3.9, we observe that the combined Wheel Dijkstra Freespace heuristic takes the best from both heuristics, as it is always on par with the heuristic that yields faster results. Looking at the suboptimality, it appears that planning with the freespace mechanism heuristic has slightly better results than with the Wheel Dijkstra or Euclidean heuristic for those queries that were solved with all these heuristics. Again, the combined Wheel Dijkstra Freespace heuristic outperforms all other heuristics.

In the leek environment, which is our real world environment, more queries get solved faster and with better suboptimality when using the freespace mechanism heuristic or the combined Wheel Dijkstra Freespace heuristic than either of the other two heuristics. The reason that the Wheel Dijkstra heuristic performs worse in this scenario is likely that the map is rather sparse (see Figure 3.8). The Wheel Dijkstra heuristic finds paths for individual wheels through tiny gaps in the crop rows, which correspond to movements that are not part of our motion primitives or would require other wheel to pass over plants, which are treated as obstacles. Thus, the planner is not able to use these tiny gaps, which is reasonable as we do not want the BoniRob to switch between crop rows while driving in the field. The performance of the Wheel Dijkstra heuristic is better in a simulated field environment (rows) than in a real field environment (leek) and thus does not transfer directly due to the aforementioned reason. However, the performance of the combined Wheel Dijkstra Freespace heuristic does transfer.

The rocks environment provides difficulties for all four tested heuristics. All of them solve around 45 % of the planning queries in a similar time frame and with similar suboptimality results. Clearly, the planner reaches its limits in this environment. However, it is possible that some of the planning queries are not solvable, as we sampled the start and goal poses randomly and did not check for reachability.

Note that both in the leek and the rocks environment, new plans get found until the timeout is reached, so it is likely that even more would be found if we allowed for more planning time.

Evaluation of the Joint Configuration Interval Representation

To study the influence of the joint configuration interval representation, we run the same planning queries once with a single value representation for the joint angles (*single angles*), once with the interval representation (*intervals*), and as a baseline we also run them with a representation that does not allow any joint value changes (*fixed*). We expect the latter one to find less plans overall, as it is limited in its motion capabilities, but if a goal can be reached, the plan will usually be computed faster due to the much smaller search space. As heuristic, we use the combined Wheel Dijkstra Freespace heuristic.

We depict the percentage of solved planning queries over time as well as the reached suboptimality bound in Figure 3.11. The first thing to note is that the interval representation has very similar results to the fixed representation in the freespace, rows, and leek

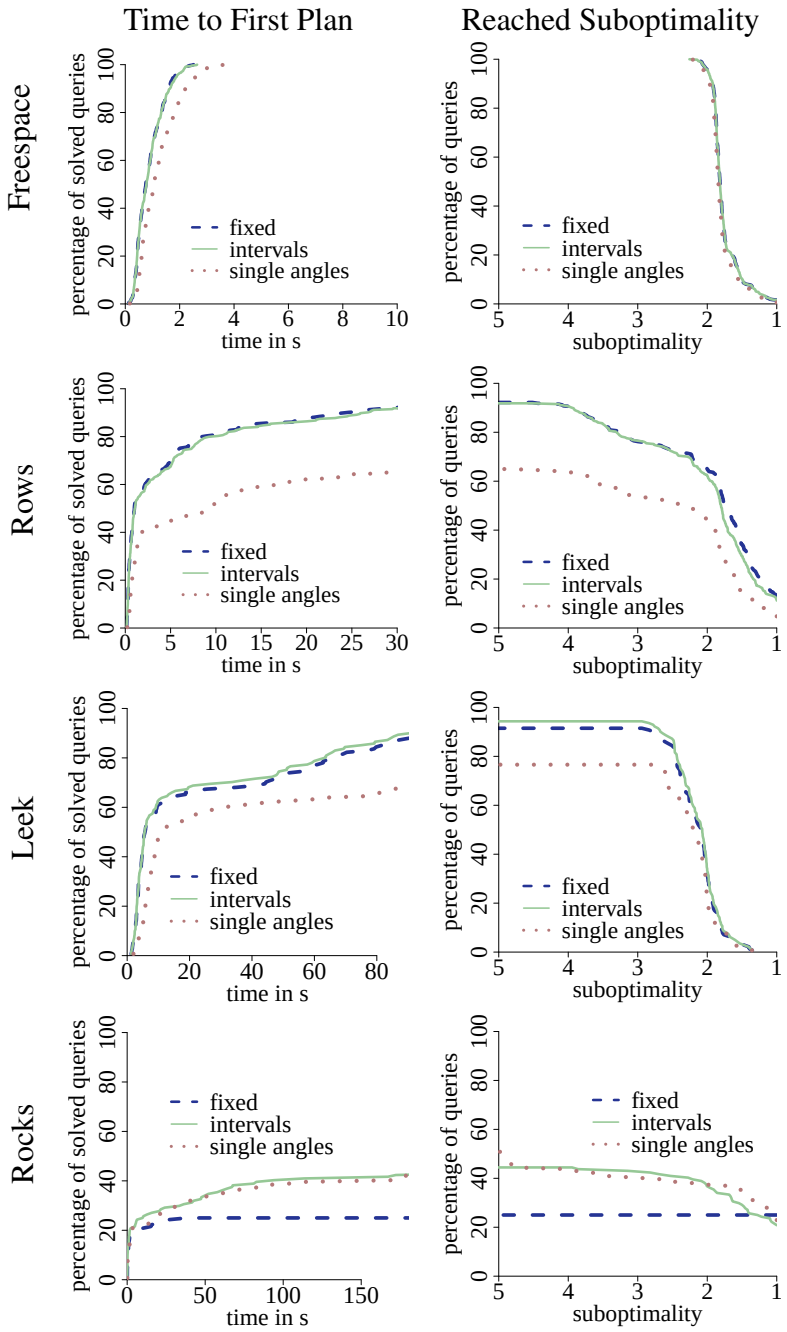


Figure 3.11: These graphs show the performance of planning with different configuration representations in all four test environments. The left column displays the percentage of planning queries solved over time. The right column shows cumulative histograms of the final upper suboptimality bound that was reached by the planner.

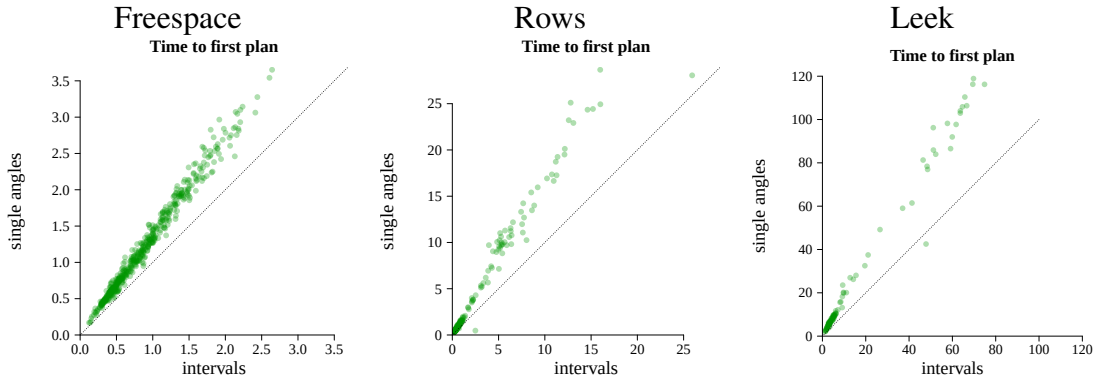


Figure 3.12: These scatter plots compare the time in s until a first plan was found between a single joint configuration representation and a joint configuration interval representation in the freespace, rows and leek environment.

environment, both with regard to timings and suboptimality. So even though the interval representation has a much bigger state space, the efficiency of the representation ensures a performance that is comparable to planning in the much smaller state space with fixed joint angles. We also see that in the named environments, the single value representation results in much slower planning overall. This becomes even more obvious in Figure 3.12, where we directly compare the time until a first plan was found for the single value and the interval representation. A first plan is found faster using the interval representation in all three environments. While the suboptimality results are comparable among all three representations in the freespace environment (see Figure 3.11, top right), the single value representation gets worse suboptimality bounds in the rows and leek environment. This is closely linked to the increased planning time. It is likely that a higher timeout for the single value representation would result in lower suboptimality bounds. The performance for all three representations are similar between the simulated rows environment and the real world leek environment, within the given time limits. While planning does take longer in the leek environment, this is to be expected as the leek map is more than a factor 10 bigger than the rows environment. Overall, the performance transfers well from a simulated to a real world environment.

In the rocks environment, it first becomes apparent that being able to change the joint configuration is necessary in some environments to find any plans at all. While planning with the fixed representation that does not allow for any joint setting changes is only able to solve 25 % of the planning queries, employing the intervals and single value representation result in about 45 % and 50 % of solved queries, respectively. Interestingly, both representations including joint configurations show similar performance in this environment. Presumably, the environment is so narrow that most of the time only one joint setting is viable and the joint setting intervals include only a single value. This negates the advantage of this representation over the single value representation and makes them mostly equivalent. These two representations find new plans until the end of the timeout and are thus likely to solve more planning queries if we increase the time limit. In con-

trast, the representation with fixed joints does not find any new plans after roughly 30 s. Thus, it is likely that planning with fixed joints is inherently unable to solve the remaining queries at all. This shows the necessity of considering joint configuration changes for planning in very cluttered environments.

3.5 Conclusions

In this chapter, we extensively evaluated a planning approach from previous work with regard to real world applicability in precision farming as well as testing its limits and its versatility. Firstly, we investigated its performance with different heuristics, among them the Wheel Dijkstra heuristic, which considers robots with high ground clearance that are able to pass over some obstacles. We combined it with the freespace mechanism heuristic, which is specifically designed for state lattice planning, which the planner we evaluated is based on. As expected, each of the heuristics individually have advantages in different environments and the performance of the planner when using one of them thus depends on the environment. We determined that the performance for the individual heuristics does not transfer well from a simulated crop field environment to a real crop field environment, due to the sparsity of the real world map. However, using a combination of the two heuristics improves the results so that the planner performs at least as good as with either one of the heuristics used separately in any environment. Thus, we are able to employ the advantages of different heuristics in this planning approach by combining them.

Secondly, we tested the impact of representing the valid configurations of one-dimensional joints as intervals in the planning state space. We found that this representation consistently yields more efficient planning performance than when planning with single values for each joint. The planner performance with regard to single value versus interval representation transfers well from a simulated to a real world environment. Furthermore, we established that including the intrinsic configuration of a robot, i. e., its joint settings, is crucial for planning in narrow and cluttered environments.

Chapter 4

Smooth Local Planning for Robots with Four-Wheel Independent Steering

Navigation in precision agriculture requires that a computed path is executed safely, accurately and efficiently. Some robots are equipped with four-wheel independent steering, i. e., they have four omnidirectional individually steerable wheels, which provides high movement flexibility. They thus have the potential to satisfy these requirements. To ensure that a path is executed in the desired fashion, any restrictions on robot movement have to be considered when generating velocity commands in order to follow a given path. Usually, local planning approaches consider the velocity and acceleration constraints of a robot. However, omnidirectional wheels have certain steering constraints. Few approaches take steering constraints into account, which especially for large robots with slow turning wheels has a large impact on movement efficiency. In this chapter, we present a local planner approach that accounts for velocity and acceleration limits as well as steering constraints stemming from a limited steering velocity, a non-continuous steering mechanism and singularities in steering angles. We present a unified representation of all steering constraints in the space of the instantaneous center of rotation of a robot. We further propose two methods of integrating these constraints into a local planner, once as a post-processing filter and once directly in the motion command computation. In extensive simulation and real world experiments, we demonstrate how including steering constraint information improves local planning performance in terms of efficiency without loss of accuracy.

4.1 Introduction

The purpose of any autonomous navigation approach is to safely and efficiently reach a given goal pose or configuration. After a feasible and efficient path is found, it has to be executed accurately and efficiently in a safe manner. A global planner often provides

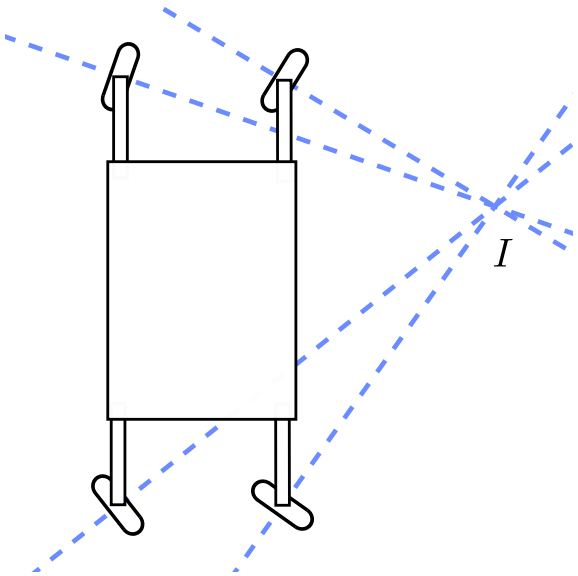


Figure 4.1: The ICR I of a robot with four-wheel independent steering is defined by the interception point of all wheel normals. The wheel normals are indicated as blue dashed lines.

the path to a goal as a series of robot poses or positions [85]. In order for a robot to follow this path, it has to be translated into motion commands that the robot is able to execute by a local planner. Robots with four-wheel independent steering allow for high movement flexibility and can execute a wide variety of velocity commands [56]. For example, they can move sideways, diagonally, or turn on the spot. However, if the desired direction of motion changes quickly, a lot of wheel turning can be required. In agriculture, vehicles with large wheels are commonly used to perform several tasks, as they provide high stability and low slippage on uneven ground. Large wheels in combination with high friction on uneven ground result in slow turning of wheels, often purposely controlled to avoid mechanical stress on the steering mechanism. Thus, reaching the desired motion direction may take some time. During this time, the robot either moves in the wrong direction, or it has to stop to adjust its wheel angles. Especially on agricultural fields with soft soil, this can have undesirable effects like soil compression and the robot digging itself into the ground. Therefore, to ensure the accuracy and efficiency of path following, considering the magnitude of necessary wheel angle changes when generating velocity commands is beneficial and some planning ahead is crucial.

In this chapter, we describe a novel local planner for robots with four-wheel independent steering that is able to follow a global path accurately, using the available motion flexibility while avoiding large wheel movements. In order to account for wheel movement, we consider several common constraints for robots with omnidirectional wheels, such as maximum steering angles and steering velocities, i. e., how fast the wheel angles can be adjusted. To express these constraints in a compact manner, we transfer them into the space of the instantaneous center of rotation (ICR) of the robot (see Figure 4.1). Using the overall velocity and acceleration limits, we compute sequences of velocity commands

that are executable within a certain time frame. We score the different velocity command sequences according to the accuracy and efficiency of the predicted path execution. We also provide a scoring mechanism based on safety and show how to include it in the total score computation. We propose two methods to include the steering constraints into our planning considerations: First, we include the anticipated wheel movement into the computation of velocity rollouts and their scoring. Second, we present a post-processing filter that alters computed velocity commands in a manner that avoids large wheel movement.

We compare these two methods in simulation experiments with our agricultural robot BoniRob and verify the results in a real world scenario. The simulation includes three different global paths to follow. One of them is also used for our real world experiment. As a baseline, we perform the same experiments with a local planner that is not aware of steering constraints. As including the steering constraints mostly presents a trade off between efficiency and accuracy, we perform our experiments in an environment without any obstacles. This enables us to directly observe the influence of including steering constraints, without any disruption caused by safety concerns. Since we expect a collision free path from a global planner, following it accurately should suffice for staying clear of obstacles. All experiments show that incorporating steering constraints into the local planner leads to smoother and more efficient path execution. If steering constraints are included in the planning phase, efficiency is improved without loss of accuracy.

Substantial parts of the ideas, figures and results presented in this chapter have been previously published [31]. Section 1.4 outlines the author's contribution to this work.

4.2 Related Work

Computing velocity commands for a robot to follow a given path is a challenge that has been approached with a variety of ideas. Early methods set a fixed forward velocity and send only steering commands to attempt to follow a path accurately [65]. A possible application for such a method is controlling a vehicle on a motorway [44]. Some purely reactive approaches compute steering commands directly from sensor data [5]. Neither of them consider the steering angle changes. We aim for an approach that explicitly models these changes in order to reduce the time spent for adjusting the wheel angles. A step in this direction is presented by Cherubini et al. [21], where the authors restrict the path that the robot has to follow to be twice differentiable. While this results in a smooth path, following the path exactly may still result in steering constraint violations. We show in our experiments that satisfying the constraint on the path to be twice differentiable is not necessary for accurate and efficient path execution.

Some approaches predict the future robot state by employing model predictive tracking [24, 74] and are thus able to act considering a prediction of the future instead of only reacting to current sensor data. Fox et al. [32] and Gerkey and Konolige [36] sample a set of possible reachable velocities for the next time step. They compute scores for each sampled velocity and execute the one with the highest score. We chose a similar approach, as we also sample and score a set of velocities.

Our approach is set apart from these velocity sampling and scoring approaches in two

ways. First, instead of only considering the velocity for the next planning time step, we contemplate a sequence of velocity commands for a few time steps into the future. Thus, we plan further ahead and the resulting velocity sequences are more stable. For example, reaching specific velocities may require the robot to stop or slow down, which we can plan for by considering several time steps. Second, we incorporate steering constraints in the velocity command generation by the local planner. Common approaches for computing velocity commands include velocity and acceleration constraints of a robot [32, 52], but few of them consider the mechanical constraints of the steering mechanism. For integrating these additional constraints, we leverage previous work [22, 78]. Clavien et al. [22] provide an estimate of the ICR of a ground vehicle from its wheel angles and positions. From velocity readings from a robot, we can directly compute the homogeneous coordinates of the ICR. The homogeneous ICR coordinates can then be mapped onto a unit sphere in 3d. We use this representation to express steering constraints in a compact manner, using the advantage that all ICR coordinates are bounded in this representation. Schwesinger et al. [78] show how to express steering velocity constraints as half plane constraints on the ICR position. We combine the results of these two papers and demonstrate how to express mechanical steering constraints in an efficient and unified manner.

4.3 Local Planning for Smooth Trajectory Execution

A local planner that computes velocity commands from a provided global pose or position path has to follow several requirements. Most obviously, the computed velocity commands should be feasible regarding the overall maximum acceleration and maximum velocity of the robot. Another straightforward requirement is that it needs to accurately follow the given path. Most local planners take these requirements into account [11, 71, 80]. Additionally, a smooth path execution is desirable: sudden changes of direction or sudden stops can lead to high mechanical stress on the robot chassis or joints and lead to unnecessary time consumption [28, 34]. This is especially true for large, heavy robots. To increase efficiency, stopping for a long time should also be avoided, unless it is absolutely necessary to follow a given path.

In particular, for some maneuvers, stopping the robot is necessary to avoid mechanical stress. One example is shown in Figure 4.2, where the robot first goes straight and then switches to performing a turn on the spot. While the wheels turn from one command to the next, the ICR is not well defined. If this command switch would be performed while the robot is moving, the wheels would work against each other, putting a lot of strain on the steering mechanism. Furthermore, the result is difficult to predict due to slippage. Thus, the robot is usually stopped while the wheels are turned to their new steering orientation for the given command. As this is time consuming and in agricultural settings leads to soil compression and to the robot digging itself into the ground, these maneuvers should be avoided, if possible without loss of accuracy.

A smooth and efficient path execution requires small wheel angle changes, as small wheel angle changes imply small direction changes. There are some common constraints on the steering mechanism that may cause large wheel movement [19, 93]. First, wheeled

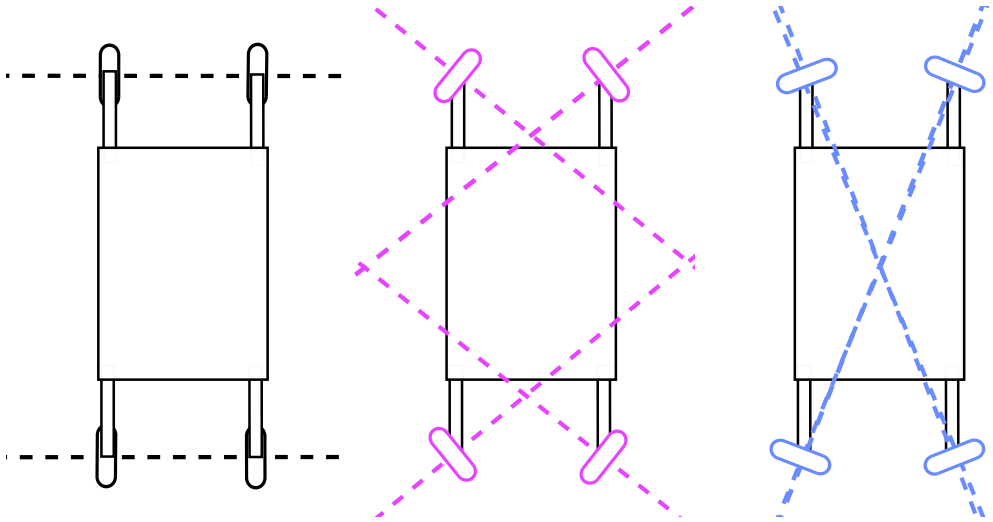


Figure 4.2: This graphic shows the procedure when a robot with four-wheel independent steering drives straight and receives the command to turn on the spot. Wheel normals are indicated as dashed lines. The robot first moves straight forwards and the ICR is at infinity (left). When the next steering command is turning on the spot, turning all wheels is required. While the wheels are being turned, the ICR is not clearly defined (middle, purple). Attempting to drive in this wheel configuration would lead to the forces applied by each wheel working against each other. This puts a considerable amount of stress on the robot chassis and on the steering mechanism. Furthermore, predicting the robot movement with this wheel configuration is difficult due to slippage and friction factoring in. Once the wheels have reached their goal orientation, the robot can turn on the spot (right, light blue).

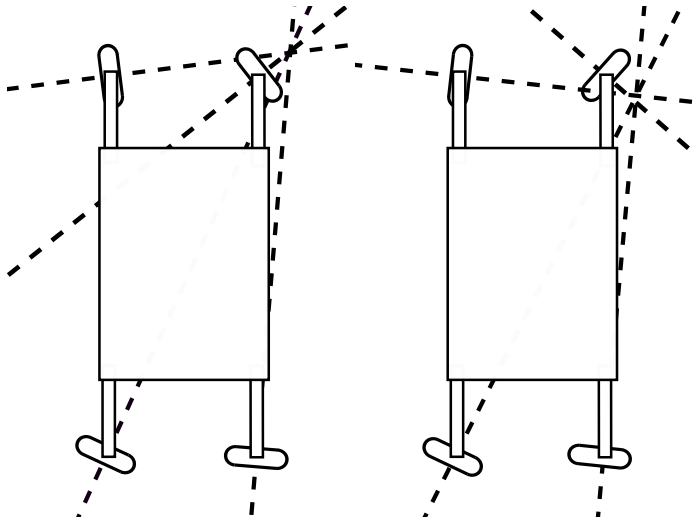


Figure 4.3: While the ICR only changes position slightly from the left to the right image, the wheel which is close to it needs to turn a lot to correct it steering angle.

robots usually do not have continuous steering, e. g., to change a wheel steering angle from 89 ° to 91 °, the wheel might need to turn 178 ° instead of just turning by 2 °. Second, there is a steering angle singularity for a wheel if the robot tries to turn exactly on that wheel. The steering angle for this wheel is then not clearly defined. If the turning spot changes minutely, e. g., due to numerical instabilities, the wheel might have to turn a lot as shown in Figure 4.3. To avoid that the robot stops, we also consider the steering velocity for each wheel. If a wheel cannot turn fast enough to reach a desired angle in a given time step, the robot will either not perform the correct motion, or it would have to stop, adjust the wheel angles and then resume the motion. Overall, we have the robot velocity and acceleration constraints, the non-continuous steering constraint, the steering angle singularity, and the steering velocity to take into account.

As the robot requires velocity commands to execute a path, we aim to generate a sequence of velocity commands that are reachable within certain time steps. To plan ahead, we perform a velocity rollout over a time limit, i. e., we compute a sequence of velocity commands that comply with our constraints and test the predicted robot pose path for proximity to the desired path. The velocity rollouts are scored according to safety, efficiency, and proximity to the desired path. We choose the one with the best score to execute.

4.3.1 Steering Constraints in ICR Space

All three steering constraints (non-continuous steering, limited steering velocity, and steering angle singularities) are complicated to express in velocity space, but straightforward to express in terms of the ICR. We therefore translate velocity commands into ICR space to handle these constraints. We define the coordinate system of the ICR space as the robot base coordinate frame. For a robot with individually steerable, omnidirectional wheels, the ICR is defined as the point where the normals of all wheels intersect, as illustrated in Figure 4.1. In a real world scenario, they will never intersect exactly in one point due to slight angular variations. Thus, methods for estimating the ICR from wheel positions and orientations have been developed. We use the least-square ICR position estimation approach by Clavien et al. [22] to tackle this issue.

We express the steering constraints in ICR space as proposed by Fleckenstein et al. [31]. Turning on a wheel means that the ICR lies exactly on that wheel. To avoid the singularity and numerical instabilities when the ICR is on or close to a wheel, we constrain the ICR to stay outside a circular area around the wheels, as indicated in Figure 4.4 on the left. The steering velocity constraints can be formulated as half plane constraints on the ICR, as presented by Schwesinger et al. [78]. As shown in Figure 4.4 in the middle, the valid normal angles of each wheel after time step t create cones of valid ICR positions according to the steering velocity constraints on that wheel. The set of valid ICR positions considering all wheels is represented by the intersection of all cones. The non-continuous steering constraint in our case is that the steering angle of each wheel can only be in $[-90\,°, 90\,°]$. To avoid numerical instabilities, we forbid the steering angles to enter a certain range around these values. In ICR space, this gives us cones that the ICR is not allowed to enter, as illustrated in Figure 4.4 on the right.

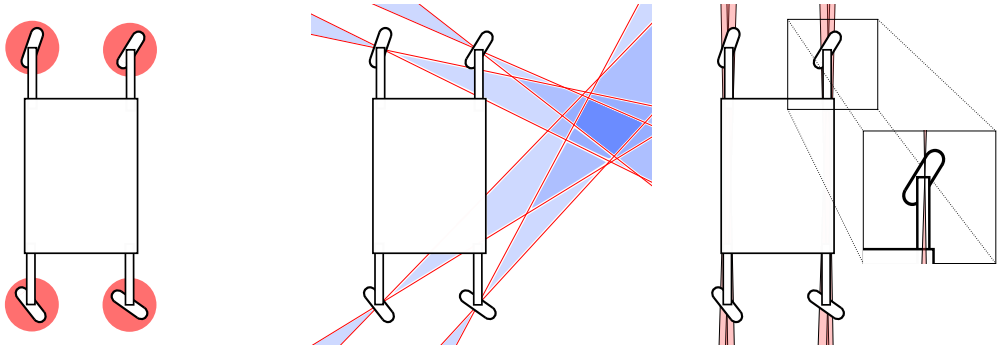


Figure 4.4: These images show all steering constraints we consider in our local planner. Having an ICR close to one of the wheels causes numerical instabilities in the required steering angles, leading to large wheel movements. The first image illustrates the areas around the wheels that the ICR is not supposed to enter in red. The steering velocity constraints affect the possible wheel angle change. In the second image, an example for the sets of wheel normal angles that are reachable in one time step are indicated as light blue cones. The area where all cones intersect represents the set of ICR positions that is reachable within the given time step. The third constraint is the non-continuous steering constraint. For the BoniRob, the wheel angle can not cross the line at $\pm 90°$. To avoid numerical instabilities, a certain area around these values is forbidden. This gives us the cones in the third image. For more clarity, we included a closeup of the front right wheel, where only the cone corresponding to this wheel is shown. Other robots may have different limits for non-continuous steering.

In order to efficiently determine whether reaching a given goal velocity from the current velocity without violating any steering constraints is possible, we aim for a compact and unified representation of these constraints as introduced by Fleckenstein et al. [31]. First, to avoid issues with ICR positions at infinity in any direction, which corresponds to driving in a straight line, we project the ICR space onto a unit sphere centered at $(0, 0, 0)$. The result is bounded coordinates for any ICR position. This is done as follows. Given a finite 2d ICR position (x, y), its homogeneous coordinates are defined as $(x, y, 1)$. Orthogonal projection onto the unit sphere yields coordinates

$$\Pi(x, y, 1) = (\frac{x}{\sqrt{x^2 + y^2 + 1^2}}, \frac{y}{\sqrt{x^2 + y^2 + 1^2}}, \frac{1}{\sqrt{x^2 + y^2 + 1^2}}).$$

For an ICR at infinity, we find the direction (x, y) in which it lies. The homogeneous coordinates for the ICR position are $(x, y, 0)$, and the orthogonal projection onto the unit sphere gives us

$$\Pi(x, y, 0) = (\frac{x}{\sqrt{x^2 + y^2}}, \frac{y}{\sqrt{x^2 + y^2}}, 0),$$

i. e., an ICR at infinity projects to the equator of the unit sphere.

From a point (u, v, w) on the unit sphere, we can recover the 2d ICR position as $\left(\frac{u}{w}, \frac{v}{w}\right)$ if $w \neq 0$. If $w = 0$, the ICR lies in direction (u, v) at infinity. Thus, diametrically opposite points on the unit sphere are equivalent representations for the same

ICR position, $(u, v, w) \sim (-u, -v, -w)$ as $\left(\frac{-u}{-w}, \frac{-v}{-w}\right) = \left(\frac{u}{w}, \frac{v}{w}\right)$. For an ICR position $I = (x, y, z)$ in homogeneous coordinates we define $\Pi^+(I) = \frac{I}{\|I\|}$ and $\Pi^-(I) = -\frac{I}{\|I\|}$ as the equivalent projection points on the unit sphere. We further define the tuple of equivalent representations as $\Pi^{\pm}(I) = (\Pi^-(I), \Pi^+(I))$. A segment $J = [I_1, I_2]$ in homogeneous space is projected onto two great circle segments on the unit sphere, $\Pi(J) = ([\Pi^-(I_1), \Pi^-(I_2)], [\Pi^+(I_1), \Pi^+(I_2)])$. Thus, moving the ICR in a straight line from a start position to a goal position in 2d space corresponds to moving it on a great circle segment on the unit sphere, even if the motion includes points at infinity. To check whether a steering constraint gets violated by an ICR movement, we only need to check whether the corresponding great circle segment crosses any constrained areas on the unit sphere. We therefore project our steering constraints onto the unit sphere. In the following, we assume the ICR should move from a start position I_s to a goal position I_g in homogeneous 2d space. We use the notation $L = I_s \times I_g = (a, b, c)$ where (a, b, c) is the homogeneous representation of a line, $ax + by + c = 0$. L is a straight line that connects I_s and I_g. It projects to a great circle on the unit sphere $\Pi(L)$.

The steering velocity constraints define how fast the wheels can turn, i. e., how much the steering angles can change in a given time step. In 2d space, these constraints for each wheel are given as half plane constraints as proposed by Schwesinger et al. [78]. These half plane constraints form cones of valid ICR positions from each wheel: If the current steering angle of wheel i is ϕ_i and the maximum steering velocity is ϑ, the steering angle after time step t is within $[\phi_i - \vartheta \cdot t, \phi_i + \vartheta \cdot t]$. Correspondingly, the valid wheel normal angles that define the ICR position also form cones as shown in Figure 4.4 in the middle. Considering a wheel position in homogeneous coordinates, $W_i = (x_i, y_i, 1)$, the line connecting the wheel position to the ICR position I is defined by the cross product $L_i = W_i \times I = (a_i, b_i, c_i)$. The wheel normal points in the direction of this line, i. e., the wheel normal angle is given by $\alpha_i = \arctan2(-a_i, b_i)$. The wheel steering angle is then defined by $\phi_i = \arctan2(b_i, a_i)$. The boundaries of each cone representing valid ICR positions according to steering velocity constraints for one wheel are lines

$$L_i^+ = W_i \times (W_i + (-\sin(\vartheta \cdot t), \cos(\vartheta \cdot t), 0))$$

and

$$L_i^- = W_i \times (W_i + (-\sin(-\vartheta \cdot t), \cos(-\vartheta \cdot t), 0)).$$

ICR positions that are valid considering the steering velocity of all wheels are given by the intersection of these cones. To move the ICR from a start position I_s to a goal position I_g, we move it in a straight line given by $L = I_s \times I_g$. Intersecting L with the cone defined by L_i^+ and L_i^- gives us a generalized segment $M_i = [M_i^+, M_i^-]$. The intersection of L with all n cones representing steering velocity constraints represents the set of valid ICR positions considering steering velocity constraints for all wheels when moving the ICR from I_s to I_g in a straight line. It is again a generalized segment $M = \cap_i M_i$, which translates to a great circle segment $\Pi(M)$ on the unit sphere.

The non-continuous steering constraints are given as a range of steering angles that cannot or should not be executed. They also form cones in 2d ICR space, as shown in Figure 4.4 on the right. With a procedure analogous to dealing with the steering velocity

constraints, we obtain segments N_i of the line L for each wheel corresponding to these constraints. Again, they project to great circle segments on the unit sphere. These are areas that have to be avoided, i. e., they have to be excluded from the great circle segment of feasible ICR positions.

Lastly, the steering singularity constraints dictate that the ICR should not enter a certain area around the wheels to avoid excessive wheel turning due to numerical instabilities. In 2d ICR space, these constraints are represented as circles with radius r around the wheels W_i as shown in Figure 4.4 on the left. Intersecting each of these circles with the line L gives two points C_i^+ and C_i^-, if they intersect. The ICR cannot enter the line segments $[C_i^-, C_i^+]$. Projecting them onto the unit sphere gives us great circle segments that have to be excluded from the set of feasible ICR positions.

An illustration of the steering velocity and steering angle singularity constraints is shown in Figure 4.5, including intersection points of the constraint representations with the line L through I_s and I_g. We omit the non-continuous steering constraints for better visibility. The first image shows the situation in 2d space. The second image shows the projection to the unit sphere, where the plane cutting through goes through start and goal ICR and defines the great circle $\Pi(L)$ on which the ICR moves. We transform the great circle to lie in the (x, y) plane, such that the start ICR I_s lies at $(1, 0, 0)$. The third image in Figure 4.5 shows the transformed great circle with the constraints, the start ICR and the goal ICR. We represent each point $P = (x, y, 0)$ on the transformed great circle by its angular position, $\theta^+(P) = \arctan2(y, x)$, which is equivalent to the opposite angle $\theta^-(P) = \arctan2(-y, -x)$. We define the tuple of equivalent representations of P as $\theta(P) = (\theta^-(P), \theta^+(P))$. A generalized segment $[p, q]$ is expressed as union of angle intervals

$$\theta([p, q]) = [\theta^-(p), \theta^-(q)] \cup [\theta^+(p), \theta^+(q)].$$

We now aim to find the great circle segments that satisfy all constraints. This is achieved by computing

$$S = \bigcap_{i=1}^{n} \theta(M_i) \cap \overline{\theta(C_i)} \cap \overline{\theta(N_i)}.$$

We use the complement of $\theta(C_i)$ and $\theta(N_i)$, as the corresponding singularity and non-continuous steering constraints define areas the ICR should avoid.

In order to check whether the goal ICR I_g is safe and reachable from I_s within a time limit, we proceed as follows. If the set of feasible ICR positions S is empty, the start ICR is too close to a steering limit or a wheel position and there is no possible movement to achieve a valid ICR in one time step. This also implies that I_g is not reachable. In general, S is a set of intervals. We choose the one that includes $\theta^+(I_s) = 0$. If $0 \notin S$, the start ICR is not valid, but a suitable ICR can be reached in one time step. We then choose the interval closest to 0. If the chosen interval also includes either value in $\theta(I_g)$, I_g is safe and reachable in a single time step. If I_g is not contained in the chosen interval, we select the interval endpoint $\theta^* = \theta(I^*)$ that is closest to $\theta(I_g)$. We recover the coordinates $\Pi(I^*)$ on the unit sphere and from that the ICR coordinates in homogeneous 2d space I^*. The ICR I^* is safe and reachable within one time step, and it lies between I_s and I_g on the line L. Note that we are in homogeneous space, so that it may be faster or even the

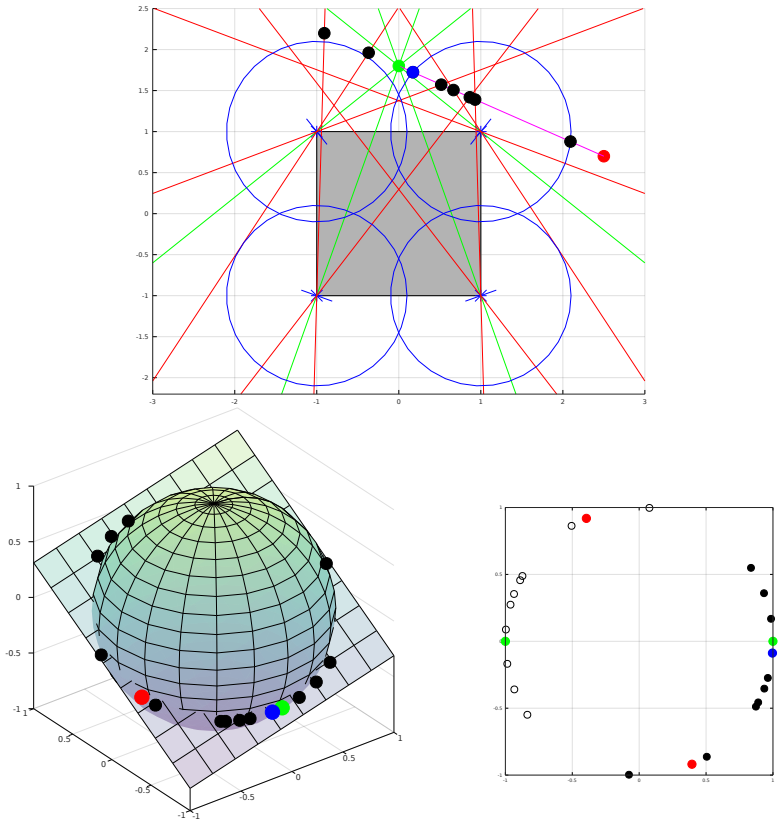


Figure 4.5: The top image illustrates the computation of a feasible ICR from a given current and desired ICR. The robot chassis is indicated as a gray rectangle with wheels positioned at each corner. Green lines indicate the current wheel normals, intersecting in the green dot, which is the current ICR. The desired ICR is represented by the red dot. Steering velocity constraints are given by red lines: for each wheel, the normal angle can only change a certain amount in one time step, thus the next ICR has to lie within the cones given by each of the wheel normal ranges that are reachable within one time step. The steering angle singularities lie on each of the wheels, so that a certain area around each wheel is excluded from the valid ICR positions, indicated as blue circles. The goal is to find an ICR that lies on the line from the current ICR (green) to the desired ICR (red) without violating any of the constraints. We thus find the line through the current and the desired ICR, indicated by the magenta line segment. Starting from the current ICR, we move towards the desired ICR until the line intersects with any of the constraints. The intersection points are shown as black dots, the one that is hit first is shown by the blue dot. This gives us a feasible ICR that is reachable in the next time step and is closer to the desired ICR than the current one.
On the bottom left, we show the same situation projected onto the unit sphere. The line through the current and the desired ICR projects to a great circle on the unit sphere. In this representation, diametrically opposite dots are equivalent concerning the 2d ICR position, $(x, y, z) \equiv (-x, -y, -z)$, so that each intersection of the line from the current to the desired ICR with any constraint shows up twice on the great circle. The same goes for the current and the goal ICR.
On the right, we see this great circle transformed into the (x, y) plane such that the current ICR lies on $(1, 0, 0)$. Intersection points with constraints are again indicated by black dots, and their diametrically opposite counterparts as unfilled dots. All images were taken from Fleckenstein et al. [31].

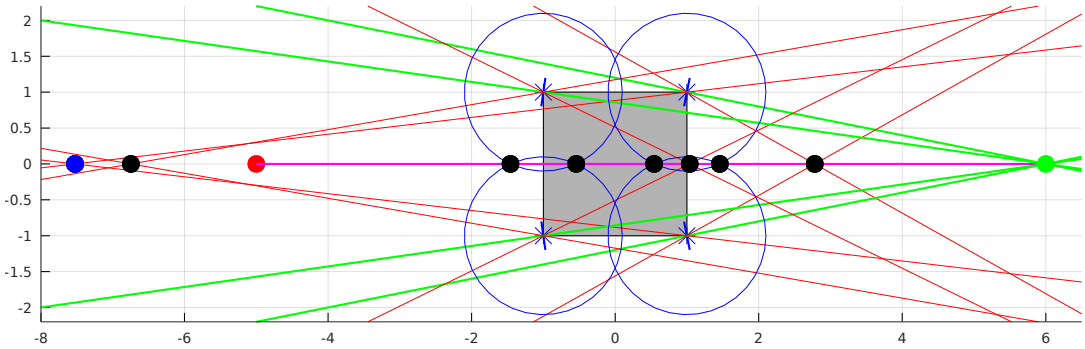


Figure 4.6: This image illustrates the computation of a feasible ICR from a given current and desired ICR, where moving through infinity gets us closer to the desired ICR than staying in finite range. The robot chassis is indicated as a gray rectangle with wheels positioned at each corner. Green lines indicate the current wheel normals, intersecting in the green dot, which is the current ICR. The desired ICR is represented by the red dot. Steering velocity constraints are given by red lines: for each wheel, the normal angle can only change a certain amount in one time step, thus the next ICR has to lie within the cones given by each of the wheel normal range that is reachable within one time step. The steering angle singularities lie on each of the wheel angles, so that a certain area around each wheel is excluded from the valid ICR positions, indicated as blue circles. The goal is to find an ICR that lies on the line from the current ICR (green) to the desired ICR (red) without violating any of the constraints. We thus compute the line through the current and the desired ICR, indicated by the magenta line segment. Starting from the current ICR, we move towards the desired ICR until the line intersects with any of the constraints. The intersection dots are shown as black dots, the one that is encountered first when going through infinity is shown by the blue dot. This results in a feasible ICR that is reachable in the next time step and is closer to the desired ICR than the current one. The image was taken from Fleckenstein et al. [31].

only way to reach I_g from I_s by crossing infinity instead of staying in finite range. An example is shown in Figure 4.6. We choose I^* as next ICR in the ICR path from I_s to I_g. We then set I^* as start ICR for the next time step. Due to the changed wheel steering angles, the reachable ICR positions may change for the next time step. Recomputing the ICR constraints accordingly allows us to iteratively find a feasible ICR path from I^s to I^g. When the time steps exceed the given timeout without reaching the goal ICR, it is not reachable within the time limit. If I^g is safely reachable from I^s, we call the resulting ICR path an ICR rollout from I^s to I^g. This is a feasible sequence of ICR positions that can be executed within the time limit without violating any steering constraints. The next step to smooth path execution is to compute velocity commands matching this ICR rollout.

4.3.2 Incorporating Steering Constraints in the Local Planner

We present two methods for including the steering constraints in order to increase path execution efficiency and smoothness. For the first approach, we initially determine se-

quences of velocity commands that comply with the overall acceleration and velocity constraints and compute a score for each sequence. The sequence with the best score is chosen to be executed. We add a post-processing filter that adjusts the next velocity command at each time step, such that the steering constraints are also satisfied. In the second approach, we directly integrate the steering constraints into the velocity sequence generation, such that the velocity sequences follow the steering constraints as well as the acceleration and velocity constraints. Again, the velocity sequences get a score and the best one is chosen to be executed.

Velocity Rollout Generation

We present two different methods to generate a velocity rollout, i. e., a sequence of velocity commands to execute. The first only takes the overall velocity and acceleration constraints into account, while the second additionally considers steering constraints. In both cases, we discretize the velocity space with a given resolution. Since our goal is to develop a local planner for ground robots, we only consider the translational velocities in x and y direction, v_x, v_y, and the rotational velocity, ω. Out of all velocity commands within the velocity limits, we determine those that are reachable from the current velocity within the time limit while satisfying the acceleration constraints. These are our candidate goal velocities which we aim to reach in the velocity rollouts.

The first method to generate a velocity rollout to reach a candidate goal velocity does not consider any steering constraints. We simulate acceleration from the current velocity towards the candidate goal velocity. To ensure a smooth transition, we do not simply use the maximum acceleration in each velocity component. Instead, we determine for each component of the candidate goal velocity command (v_x^g, v_y^g, ω^g) how long it would take to reach it from the current velocity (v_x^s, v_y^s, ω^s) given the respective acceleration limits. The maximum time over the components $t^{\max}$ is the time it takes to reach the candidate goal velocity. We then accelerate in all velocity components in proportionate steps over time steps δt, such that all goal velocity components are reached at the same time:

$$v_i = v^s + i \cdot \frac{\delta t}{t^{\max}} \cdot (v^g - v^s), i = 1, \ldots, \left\lfloor \frac{t^{\max}}{\delta t} \right\rfloor$$

and define $v_i = v^g$ for $i = \left\lceil \frac{t^{\max}}{\delta t} \right\rceil, \ldots, n$ where $n = \frac{T}{\delta t}$ is the number of time steps until the timeout T is reached. This gives us a velocity rollout $v_1, \ldots, v_n$ with commands that are feasible considering acceleration and velocity constraints. For each velocity rollout, we compute the predicted robot path incrementally using the current robot position, the velocities and the time step length. We call this predicted path the pose rollout.

The second method for generating velocity rollouts accounts for steering constraints as well as velocity and acceleration constraints. Once we have our candidate goal velocities $v^g = (v_x^g, v_y^g, \omega^g)$, we determine the candidate goal ICRs in homogeneous coordinates, $I^g = (-v_y^g, v_x^g, \omega^g)$. We then go into ICR space and compute an ICR rollout from the current ICR to the candidate goal ICR that satisfies the steering constraints as detailed in Section 4.3.1. To increase computation efficiency, we first determine whether the goal ICR is reachable at all within the time limit by looking for an ICR path with larger time

steps. If a path exists, it is refined in a second step. After we found a feasible ICR path, we compute velocity commands that correspond to the ICR rollout as explained in the following.

Note that a velocity corresponds to an ICR up to a scale, i. e., velocities $v = (v_x, v_y, \omega)$ and $s \cdot v = (s \cdot v_x, s \cdot v_y, s \cdot \omega)$ correspond to the same ICR $I = (-v_y, v_x, \omega) \equiv (-s \cdot v_y, s \cdot v_x, s \cdot \omega)$, since this ICR representation uses homogeneous coordinates. Intuitively, the ICR only indicates which point the robot is turning around, but not how fast it is moving. Thus, we cannot directly compute a velocity rollout from the ICR rollout. Instead, we need to determine the scale s to calculate the next velocity command from the next ICR position in each time step of the rollout.

At each point in the ICR rollout, we know the previous ICR I_{i-1} and the next ICR $I_i = (x, y, z)$ as well as the overall candidate goal velocity v^g and the previous velocity command v_{i-1}, starting with the current velocity. We define an unscaled next velocity command $\widehat{v_i} = (-y, x, z) = (\widehat{v_{i_x}}, \widehat{v_{i_y}}, \widehat{\omega_i})$. We now want to find a scaling factor s such that $v_i = s \cdot \widehat{v_i}$ is reachable within one time step considering acceleration and velocity limits. Ideally, v_i should be closer to v^g than v_{i-1}. As extensively searching for the best scaling factors over all time steps is too computationally expensive, we propose testing only a subset of scaling factors that are computed as follows.

We determine the velocity window for the next time step, i. e., the minimum and maximum values of each velocity component reachable with maximum acceleration, $v_x^{\min}, v_y^{\min}, \omega^{\min}$ and $v_x^{\max}, v_y^{\max}, \omega^{\max}$. Potential scales are then given by $\frac{v_x^{\min}}{\widehat{v_{i_x}}}$, $\frac{v_x^{\max}}{\widehat{v_{i_x}}}$ and values equivalently computed from the v_y and ω components. This results in scaling such that one component is set to its minimum or maximum reachable value, respectively, i. e., we assume maximum acceleration in one component. Additionally, we compute scales $\frac{v_x^g}{\widehat{v_{i_x}}}$ and the according scales for the other components, i. e., we scale so that one component takes on its goal value. This produces velocities that are close to the goal velocity, if possible. Lastly, we scale one component to its previous value by using scales $\frac{v_{i-1_x}}{\widehat{v_{i_x}}}$ and corresponding values for the v_y and ω components. These scales likely require the least change in velocity, so that the velocity is likely reachable. Overall, we get twelve different scales to test. We scale the unscaled next velocity command $\widehat{v_i}$ with each of the values for s presented and check whether the resulting potential next velocity command $\widetilde{v_i} = s \cdot \widehat{v_i}$ lies within the velocity window, i. e., all velocity components are reachable within one time step given the acceleration limits and satisfy the velocity limits. Scales that result in velocities which do not satisfy this constraint are discarded. We compare the translational part of each potential next velocity with the translational part of the goal velocity and choose v_i with the smallest difference $\|(v_{i_x} - v_x^g, v_{i_y} - v_y^g)\|$. If none of the scaled velocity commands satisfy the acceleration and velocity limits, the candidate velocity to which the rollout is being computed is also discarded.

Performing these steps for every part of the ICR rollout results in a velocity rollout that satisfies velocity and acceleration constraints as well as all steering constraints.

Velocity Rollout Scoring

To chooose which of the computed velocity rollouts should be executed, we determine which of them is best regarding safety, efficiency and accuracy using the following scores.

For the safety score, we use a traversability map of the environment. A traversability map is a 2d grid map, where each cell c contains a traversability weight $t(c)$. A traversability weight of 1 means the cell contains an obstacle and is untraversable for the wheels. A value of 0 indicates that the cell is completely safe to traverse. Values between 0 and 1 may indicate uneven terrain or proximity to obstacles. We define a safety threshold t_{safe} and declare cells with traversability weights below this threshold as safe. A second threshold t_{crit} defines the traversability weight above which safety concerns are considered critical, i. e., we want to avoid these cells if possible. We define the cost of crossing a cell c with a wheel as

$$f(c) = \begin{cases} 0 & \text{if } t(c) < t_{\text{safe}} \\ \frac{t(c)-t_{\text{safe}}}{t_{\text{crit}}-t_{\text{safe}}} & \text{if } t_{\text{safe}} < t(c) < t_{\text{crit}} \\ 1 & \text{else.} \end{cases}$$

For every velocity rollout, we compute the path of each wheel w_i. We sum up the cost of traversing all wheel path cells $c_{i,j}$ with a decay weight $\gamma_1 \in \mathbb{R}, \gamma_1 > 0$ for cells further ahead, and divide by the sum of decay weights to normalize the result between 0 and 1,

$$c_{\text{trav}}(w_i) = \frac{\sum_j \gamma_1^j t(c_{i,j})}{\sum_j \gamma_1^j}.$$

For $\gamma_1 < 1$, cells further along the rollout get lower weight, while $\gamma_1 > 1$ gives higher weight to cells later in the rollout.

We define the overall traversability cost as maximum of the traversability cost over all wheels,

$$c_{\text{trav}} = \max_i c_{\text{trav}}(w_i)$$

To translate it into a safety score, we subtract the cost from 1,

$$s_{\text{safe}} = 1 - c_{\text{trav}}.$$

The path following accuracy score is computed as follows. The local planner keeps track of its current waypoint in the global path and defines a local goal. At first, the current waypoint is the start point of the global path. The local goal is another pose on the global path, which is at a given distance from the current waypoint along the path. After a motion was executed, the new current waypoint to the current robot pose r is computed as the pose on the global path between the previous waypoint and the previous local goal that is closest to r. For each pose $p = (x_p, y_p, \theta_p)$ on the pose rollout, we determine the waypoint $w_p = (x_{w_p}, y_{w_p}, \theta_{w_p})$ on the global path between the current waypoint and the local goal that is closest to p. Then, we compute the translational and rotational difference between the rollout pose and its corresponding waypoint and cap them with upper bounds $d_{\max}^{\text{trans}}, d_{\max}^{\theta}$:

$$d^{\text{trans}}(p, w_p) = \min\left\{\|(x_p - x_{w_p}, y_p - y_{w_p}))\|, d^{\text{trans}}_{\max}\right\}$$

$$d^{\theta}(p, w_p) = \min\left\{|r(\theta_p, \theta_{w_p})|, d^{\theta}_{\max}\right\}$$

where $r(\theta_p, \theta_{w_p})$ is the rotational difference defined such that values lie in $[-\pi, \pi]$. Capping the translational and rotational difference with an upper bound allows us to normalize them to values in $[0, 1]$ as $\frac{d^{\text{trans}}(p,w_p)}{d^{\text{trans}}_{\max}}$, $\frac{d^{\theta}(p,w_p)}{d^{\theta}_{\max}}$. The respective costs over the entire pose rollout with poses $p_i, i = 1, \ldots, N$ are then given by the average

$$d^{\text{trans}} = \frac{1}{N}\sum_{i=1}^{N}\frac{d^{\text{trans}}(p_i, w_{p_i})}{d^{\text{trans}}_{\max}} \in [0, 1]$$

$$d^{\theta} = \frac{1}{N}\sum_{i=1}^{N}\frac{d^{\theta}(p_i, w_{p_i})}{d^{\theta}_{\max}} \in [0, 1]$$

We define the overall path distance cost with a scaling factor $\gamma_2 \in [0, 1]$,

$$c_{\text{dist}} = (1 - \gamma_2) \cdot d^{\text{trans}} + \gamma_2 \cdot d^{\theta} \in [0, 1].$$

Since this gives us the path following inaccuracy, we define the accuracy score

$$s_{\text{acc}} = 1 - c_{\text{dist}} \in [0, 1].$$

For ensuring path execution efficiency, we add a score based on the length of the global path segment that is covered by the pose rollout. We find the path waypoint corresponding to the end pose of the rollout. We define the length l of the path that was covered as the number of path poses between this end waypoint and the current waypoint to account for the coverage of rotational as well as translational changes in the path. The maximum path length $l_{\max}$ is the number of poses between the current waypoint and the local goal on the path. We normalize the path length by the maximum and get a score for the covered path length, which we define as efficiency score

$$s_{\text{eff}} = \frac{l}{l_{\max}} \in [0, 1].$$

In order to combine the safety, accuracy and the efficiency score, we introduce scaling factors $\alpha_1, \alpha_2, \alpha_3 \in [0, 1], \alpha_1 + \alpha_2 + \alpha_3 = 1$ and define the rollout score as

$$s_{\text{rollout}} = \alpha_1 \cdot s_{\text{safe}}\alpha_2 \cdot s_{\text{eff}} + \alpha_3 \cdot s_{\text{acc}}.$$

Examples of trajectory rollouts with their scores in an environment without any obstacles are shown in Figure 4.7.

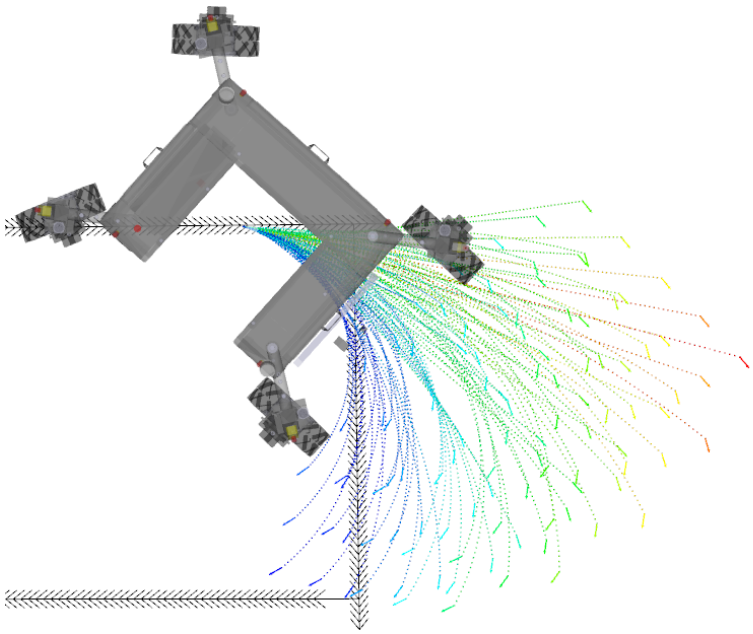


Figure 4.7: This image shows the simulation model of our experimental platform, the BoniRob, following a pose path (black arrows). The viable velocity rollouts are shown by colored dotted lines with an arrow at the end to indicate the orientation at the final position of a rollout. The color scheme is a heat map on the rainbow spectrum applied to the score, where red indicates a low score and blue indicates a high score.

Choosing a Velocity Rollout

We choose the velocity rollout with the highest score as the potential next velocity sequence to execute. However, we do not blindly follow a computed velocity rollout to the end, as localization corrections, changes in the environment or inaccurate velocity command execution may require different velocity sequences to ensure safe, accurate and efficient path execution. Instead, we compute the best velocity rollout in each execution time step and compare its score with the score of the previously chosen velocity rollout. As the score of the previous velocity rollout may change during execution, we recompute it. In the computation, we remove the velocity commands at the beginning of the sequence that were already executed. Since all velocity rollouts need to have the same length in order for the scores to be comparable, we add the final velocity command that is reached in this rollout as often as needed to compensate the removal of the already executed velocity commands. The pose rollout and the corresponding score are then computed. This simulates executing the previously chosen velocity rollout to its end given the new localization pose and environment information.

The new best rollout is only chosen if it has a score that is better than the score of the previously chosen rollout by a given factor. Otherwise, we keep the previous rollout and execute the next velocity in that sequence. While it may be more intuitive to use the new best rollout in every time step, this results in numerous direction changes, especially

when the robot is moving very slowly or standing. To ensure a more stable and smooth execution, we thus give a preference to keeping the previous velocity rollout.

Integrating Steering Constraints in a Post-Processing Filter

We explained how to generate and choose velocity rollouts that conform only to velocity and acceleration constraints as well as how to do the same for rollouts that also satisfy steering constraints. Here, we present how to generate a velocity command v^* that satisfies steering constraints from a desired command $v^d = (v_x^d, v_y^d, \omega^d)$ that only meets velocity and acceleration constraints. The procedure is similar to the procedure for generating velocity rollouts that consider steering constraints. We determine the current ICR I^c and the desired ICR I^d and perform a single step of finding an ICR path from I^c to I^d. This gives us an ICR $I^* = (i_x^*, i_y^*, i_z^*)$ that is reachable from I^c in one time step, lies on the line from I^c to I^d and is as close to I^d as possible. We then aim to find a velocity that is as close to the original command v^d as possible. The unscaled velocity is given by $\widehat{v^*} = (-i_x^*, i_x^*, i_y^*)$. Here, we differ from the scaling procedure described above. If the translational part of the unscaled velocity is zero, $\|(-i_y^*, i_x^*)\| = 0$, no scaling will have an impact on that, so we set the rotational part to the desired value, $v^* = (0, 0, \omega^d)$. Otherwise, we aim to keep the translational magnitude and direction of the desired velocity command. Thus, we set $v^* = \alpha \cdot \widehat{v^*} = (v_x^*, v_y^*, \omega^*)$ such that $\|(v_x^*, v_y^*)\| = \|(v_x^d, v_y^d)\|$ and $v_x^d v_x^* + v_y^d v_y^* \geq 0$.

Using this method, we generate velocity rollouts that only satisfy the velocity and acceleration constraints, and filter the result such that the velocity command that gets executed also satisfies steering constraints. We expect this to result in smoother movements than if we directly execute the velocity commands that were computed without any awareness of steering constraints. Note that in this case, we only perform a single ICR path computation. In constrast, integrating the steering constraints directly into the velocity rollout computation requires computing ICR paths for all candidate goal velocities. Therefore, this method is less computationally expensive. However, it changes the computed velocity command after scoring. Therefore, the path rollout that was computed may no longer be accurate and the executed velocity may be suboptimal. This potentially results in inaccurate path following behavior. In the next section, we evaluate the execution efficiency and the resulting accuracy loss.

4.4 Experiments

We evaluate the performance of three different planner behaviors and name them by their ICR awareness level. As a baseline, we use the planner that generates velocity rollouts considering only velocity and acceleration constraints. Its ICR awareness level is *None*. We investigate how performance changes when adding a post-processing filter that integrates steering constraints after the velocity rollout computation. This yields an ICR aware *Filter*. Lastly, we examine the performance when integrating the steering constraints into the velocity rollout computation, which gives us an ICR aware *Planner*.

We set the BoniRob to follow different paths with the different local planner behaviors, repeating each run three times to test for replicability and consistency. Furthermore, we tested two maximum velocity settings: The first had a maximum of 0.2 m/s in either translational component, $|v_x| \leq 0.2, |v_y| \leq 0.2$ and a maximum rotational velocity of 0.1rad/s, $|\omega| < 0.1$. We refer to it as the 0.2 maximum velocity setting. The local goal in this setting is 3 m from the current waypoint along the path. The second had maximum translational components of 0.4 m/s and a maximum rotational component of 0.2rad/s. We refer to it as the 0.4 maximum velocity setting. The local goal here is 6 m from the current waypoint along the path. The distance to the local goal was chosen such that it is reachable given the velocity limits, but driving with maximum velocity cannot overshoot the local goal.

In order to directly observe the trade off between accuracy and efficiency without disruptions by any obstacles, we investigate only paths in obstacle-free environments. Thus, we set the safety scale $\alpha_1 = 0$ for the velocity scoring, such that the score is given by $s_{\text{rollout}} = \alpha_2 \cdot s_{\text{eff}} + (1 - \alpha_2) \cdot s_{\text{acc}}$. We call α_2 the *path length scale* (PLS), since the efficiency score is based on the covered path length. Note that α_2 presents a trade off between efficiency and accuracy: for larger values of α_2, efficiency gains importance, while accuracy loses weight. This encourages cutting corners. We tested different PLS values to investigate its influence on path following accuracy and efficiency.

We perform experiments in three simulation scenarios and one real world scenario. The corresponding paths are shown in Figure 4.8, Figure 4.11 and Figure 4.14. The first simulation scenario (Figure 4.8) is a path that has wide, smooth curves that do not require many direction changes. We expect that smooth and therefore efficient path execution is easily attainable while following the path accurately. We call this the *lines and arcs* path. The second is a path that is similar to a field traversal as a common use case for autonomous navigation in precision agriculture (Figure 4.11). It simulates following the crop rows and turning at the end of each row to follow the next one. We made the rows very short, since we are primarily interested in the turns at the end of each row. This is a more challenging scenario, since it involves tight 180 ° turns. We use the same scenario in our real world experiment and call it the *field* path. Lastly, we chose a path that follows a rectangular wave with varying side length (Figure 4.14). We expect that the wheels turn excessively in the corners of this path in order to accurately follow the path. We call this path the *rectangular wave* path.

For a qualitative analysis of the accuracy and efficiency of each planner behavior, we

Table 4.1: This table shows the mean overall execution times on the lines and arcs path for all planner behaviors with the different velocity and path length scale settings.

PLS	0.2 m/s			0.4 m/s		
	Filter	None	Planner	Filter	None	Planner
0.08	385s	492s	383s	182s	214s	182s
0.1	370s	493s	380s	180s	231s	177s
0.12	327s	488s	355s	175s	208s	178s

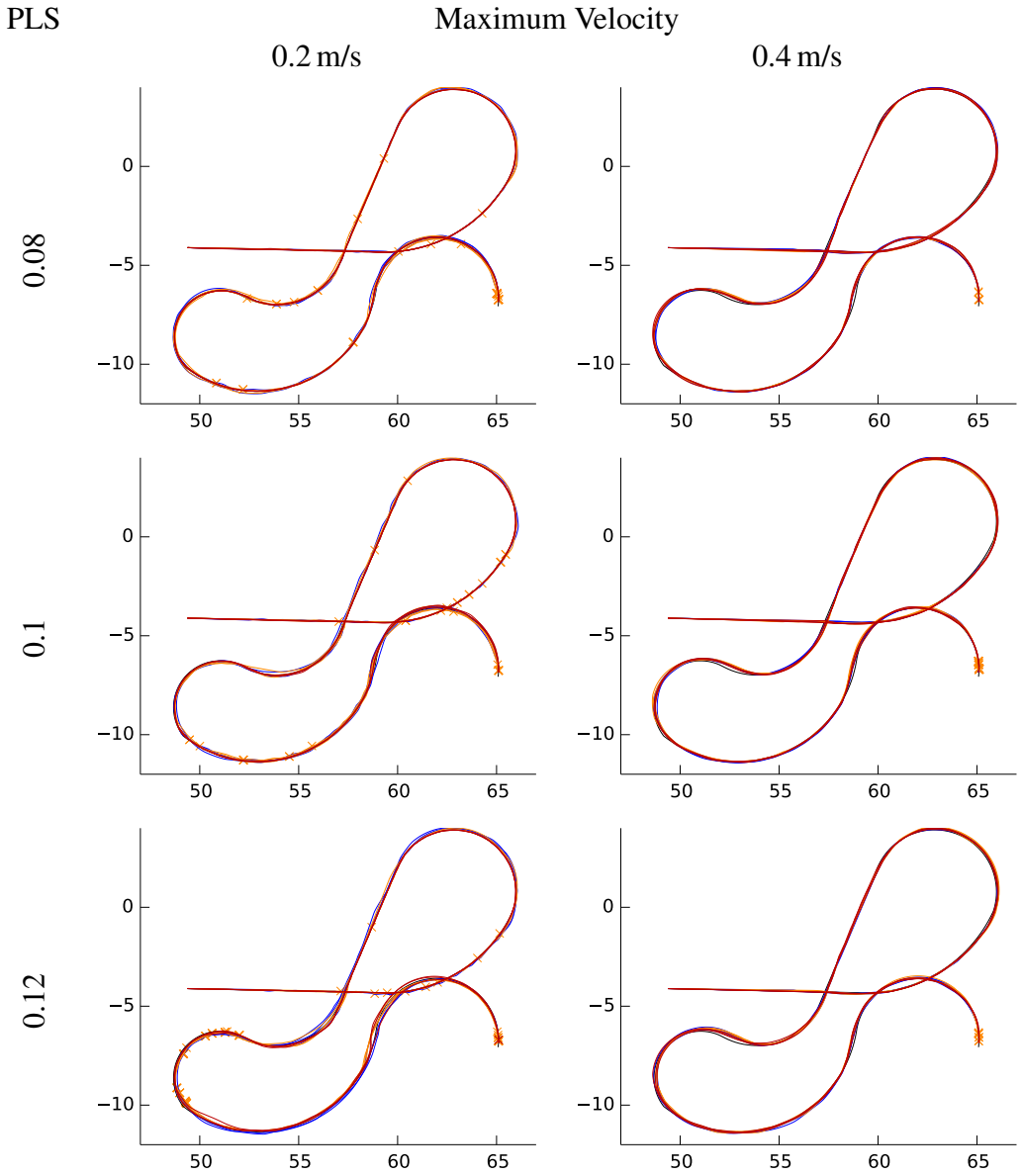


Figure 4.8: Shown are the original lines and arcs path (black) and the paths the different planner behaviors took in each of their three runs (yellow: None, blue: Filter, red: Planner) for both maximum velocity settings and all three path length scales. Points where the robot stopped are marked with a cross in the corresponding color. Units are m relative to a fixed frame on the ground plane.

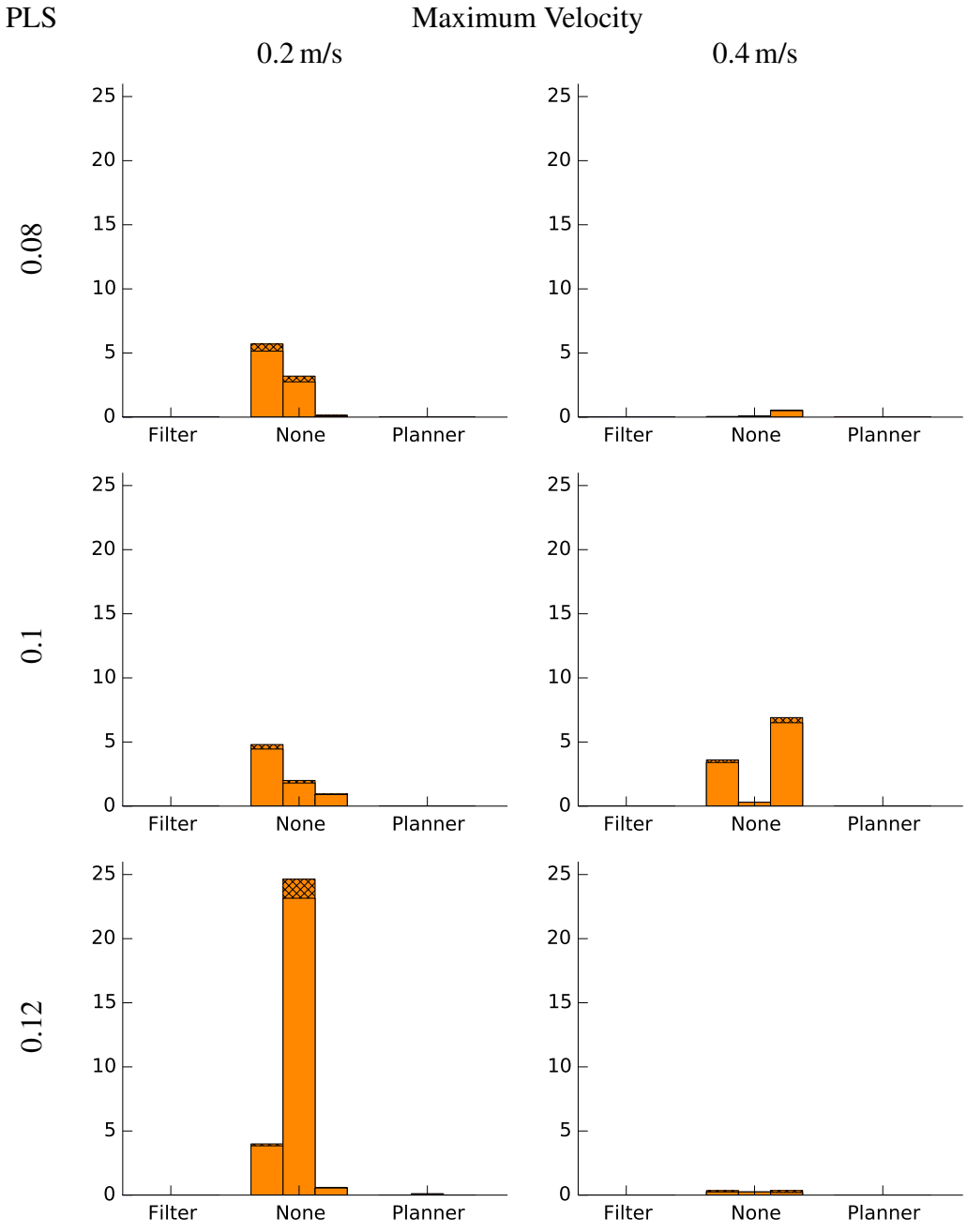


Figure 4.9: These plots depict the time spent standing in s on the lines and arcs path for each of the three runs with each planner behavior, the different maximum velocity settings and path length scales. The shaded areas represent standing without turning the wheels.

Table 4.2: This table shows the mean overall execution times on the field path for all planner behaviors with the different velocity and path length scale settings.

PLS	0.2 m/s			0.4 m/s		
	Filter	None	Planner	Filter	None	Planner
0.08	169s	257s	152s	147s	192s	122s
0.1	159s	279s	150s	136s	204s	104s
0.12	159s	217s	150s	118s	196s	87s

show the paths of the robot as well as where it stopped. For the lines and arcs path, the result is shown in Figure 4.8. All planner behaviors are able to follow the path quite accurately. The planning behavior with ICR awareness None stops often in order to adjust the wheel angles, as visible by the cross marks on the pose path. Both the Filter and Planner awareness help increase the efficiency by avoiding maneuvers with large wheel changes that would require the robot to stop for adjusting its wheel angles. This is shown in Figure 4.9 as well: while planner ICR awareness None results in standing times of up to 25 s, with Filter or Planner ICR awareness, the standing times are negligible or the robot does not stop at all. Regarding the mean overall execution times (see Table 4.1), ICR awareness Filter and Planner show similar efficiency. For a path length scale of 0.08 and maximum velocity setting 0.2, both are able to follow the path in about 380 s, while a planner with ICR awareness None takes roughly 490 s. Behavior None consistently takes this long with low maximum velocity and any path length scale. On the other hand, the planner behaviors Filter and Planner are able to use the higher path length scale to further reduce execution time with low maximum velocity. With a higher maximum velocity, the difference between behavior None and the other two is not as large anymore, but still notable. All behaviors result in lower execution times with higher maximum velocity.

In order to further investigate the accuracy of each planner behavior, we determine the maximum error between poses during path execution and the original path. The error plots in Figure 4.10 confirm that the maximum error is very similar between all planning behaviors for this path and the maximum velocity and the path length scale have low impact on the path execution accuracy.

The field path poses more challenges for path execution. The poses and stop points are shown in Figure 4.11. The most striking observation is that with larger maximum velocity, the planner is more likely to cut or overshoot the original path. Cutting the path is likely caused by the local goal being further ahead with larger maximum velocity. On this path, cutting the turns brings high reward in the efficiency score. This is especially the case for larger path length scales. This could be avoided in two ways. First, we could set the local goal closer to the current waypoint. However, this would lead to high velocities overshooting the local goal consistently, so that the high velocities will never be used because they would lead to overshooting the local goal. A second possibility would be to add crop rows as obstacles and include the safety score in the velocity rollout scoring process. The overshooting done by the Filter behavior is a result of insufficient wheel angle changes when large changes are needed, combined with the high velocity.

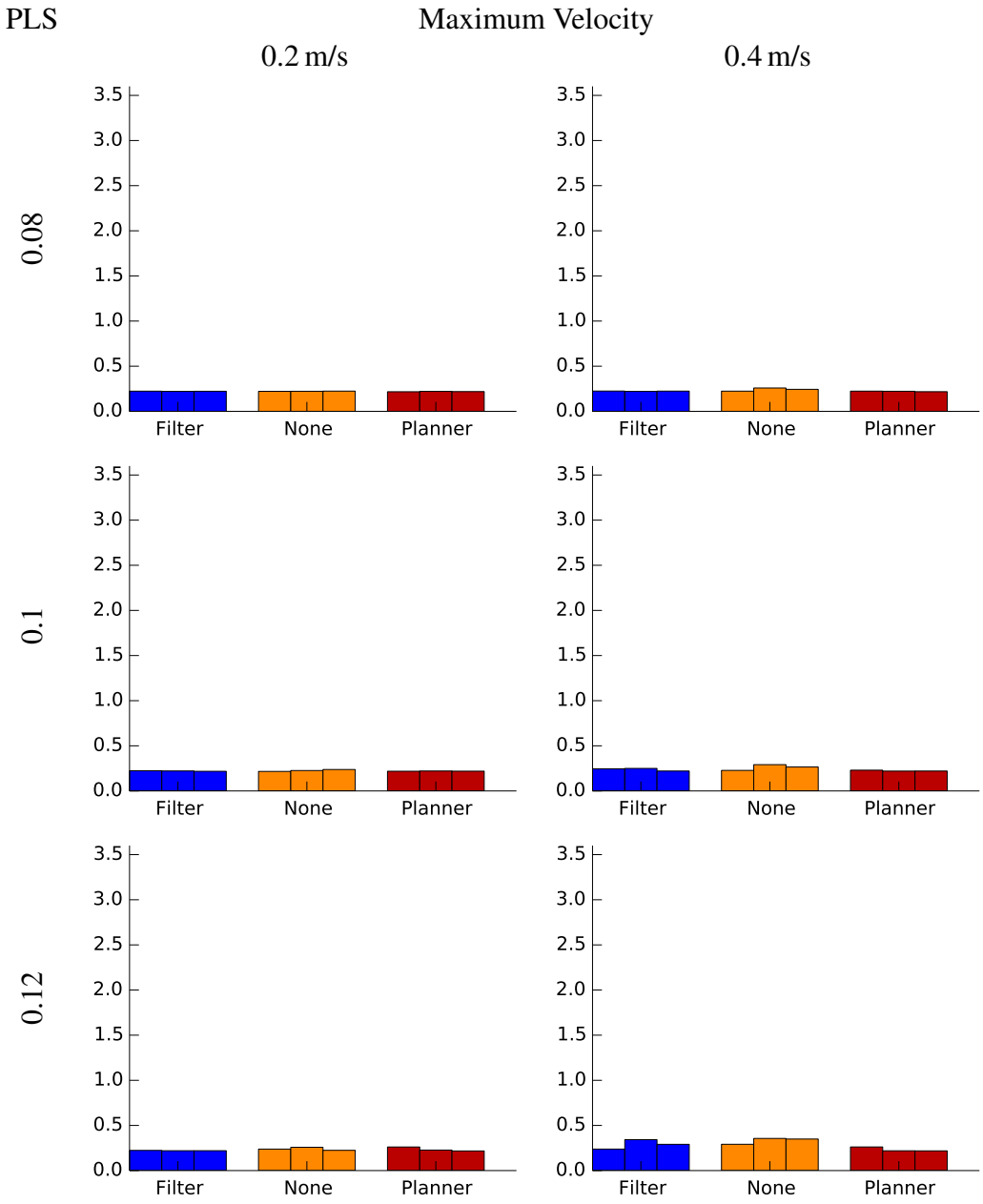


Figure 4.10: These plots depict the maximum error in m between each pose on the path the planner took and its corresponding waypoint on the original lines and arcs path. The error is shown for all three runs with each planner behavior, with the different maximum velocity settings and path length scales.

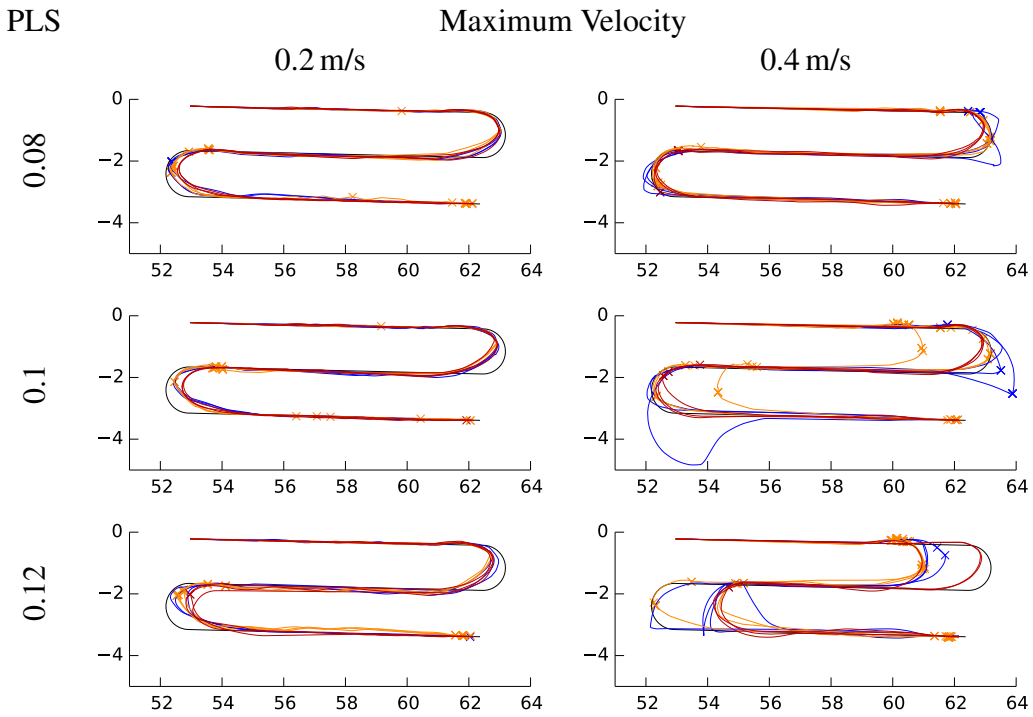


Figure 4.11: Shown are the original field path (black) and the paths the different planner behaviors took (yellow: None, blue: Filter, red: Planner) for both maximum velocity settings and all three path length scales. Points where the robot stopped are marked with a cross in the corresponding color. Units are m relative to a fixed frame on the ground plane.

In the right middle plot, the ICR aware Filter overshoots at the first turn and has to stop as continuing would bring it too far away from the original path. At the second turn, it overshoots in one out of the three runs, but is able to recover without having to stop. As shown by the cross marks, the None ICR aware planner stops often on this path to turn its wheels. This is reflected in Figure 4.12 in the time spent standing. Here we also observe that planning without ICR awareness and a higher maximum velocity results in more time spent standing. While the Filter behavior also spends some time standing here, it still outperforms the time needed with ICR awareness None. For either type of ICR awareness (Filter or Planner), higher path length scales lead to less time spent standing, especially with high maximum velocity. This is to be expected, since a high path length scale reduces the need for accuracy and increases the reward for efficiency. It is further shown in Figure 4.11 that higher path length scales lead to cutting turns. Table 4.2 shows the quantitative results for execution times. For a maximum velocity setting of 0.4, the Filter behavior reduces execution time from about 150 s to about 120 s, and the Planner behavior reduces it from about 120 s to about 90 s. Again, the None behavior takes considerably longer than the other two behaviors.

Interestingly, the path following accuracy shown in Figure 4.13 is similar between all planner behaviors in most cases. For the higher maximum velocity, we get larger errors

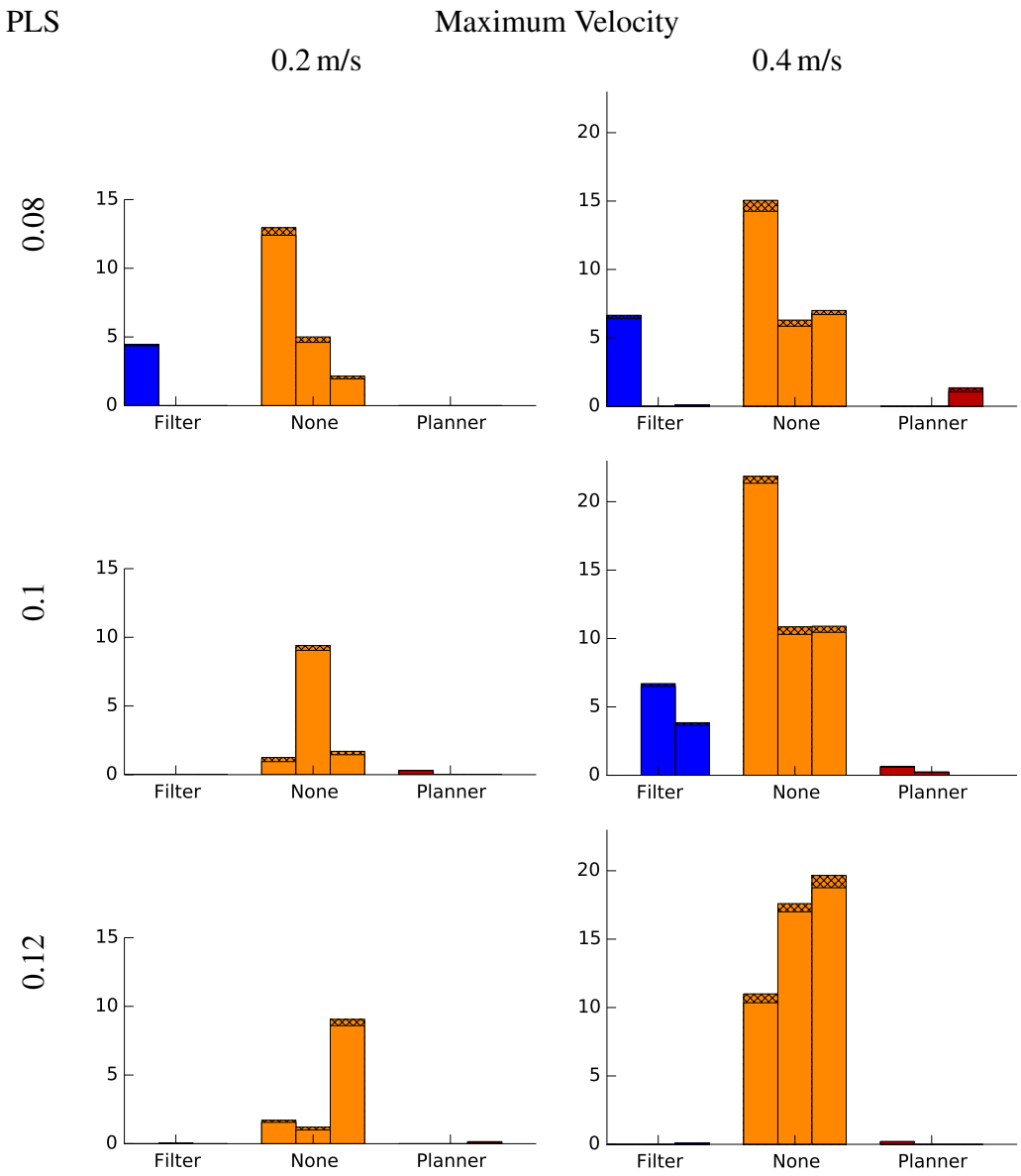


Figure 4.12: These plots show the time spent standing in s on the field path for each of the three runs with each planner behavior, the different maximum velocity settings and path length scales. The shaded areas represent standing without turning the wheels.

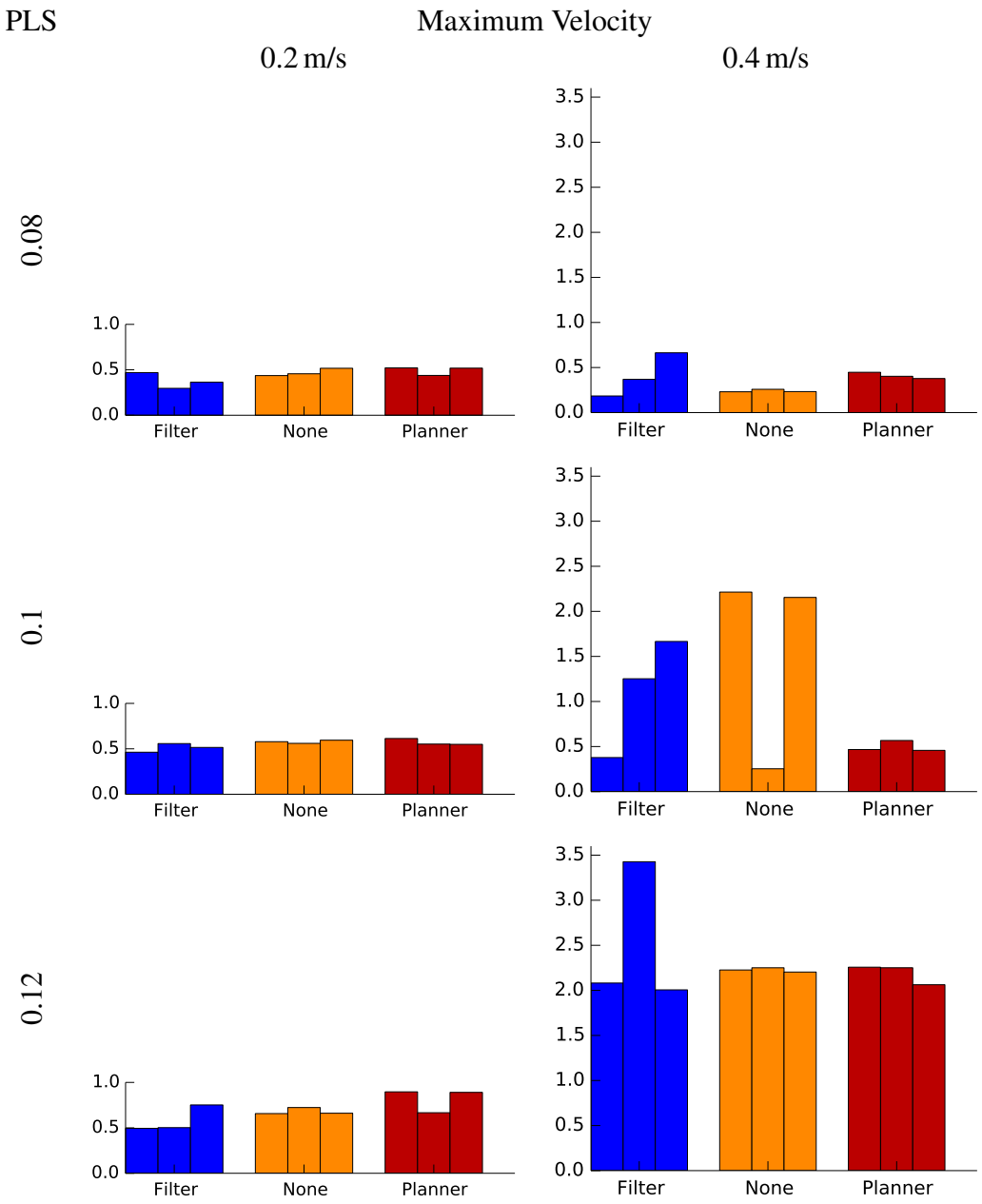


Figure 4.13: These plots show the maximum error in m between each pose on the path the planner took and its corresponding waypoint on the original field path. The error is shown for all three runs with each planner behavior, with the different maximum velocity settings and path length scales.

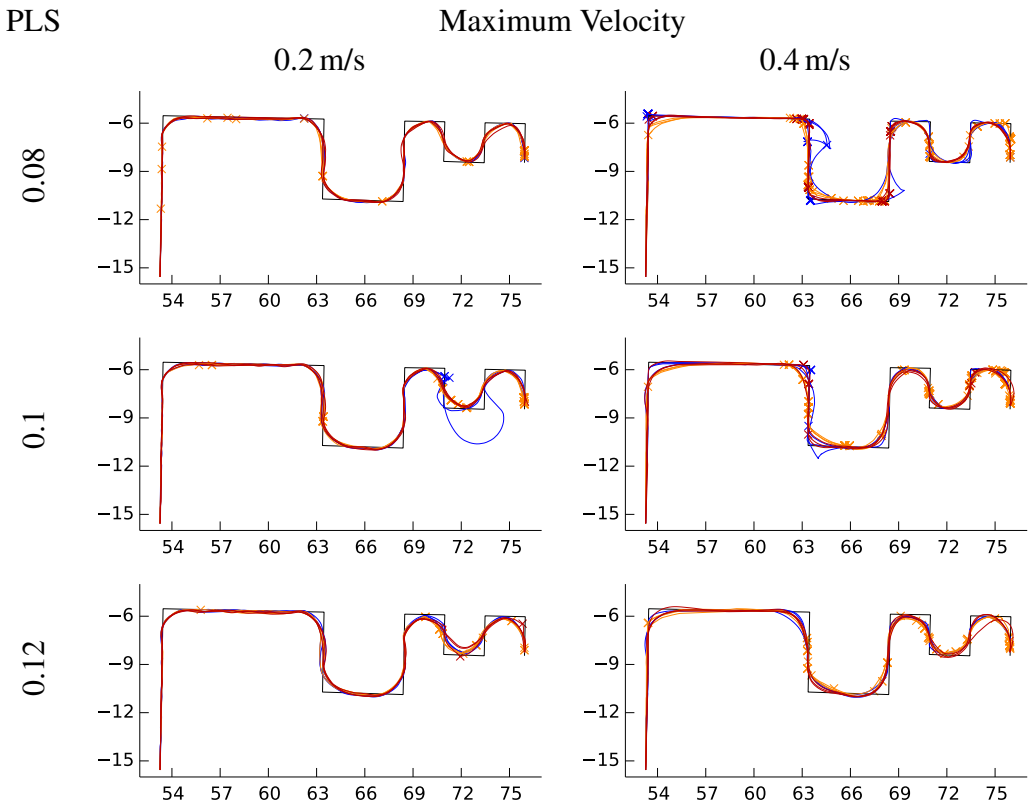


Figure 4.14: Shown are the original rectangular wave path (black) and the paths the different planner behaviors took (yellow: None, blue: Filter, red: Planner) for both maximum velocity settings and all three path length scales. Points where the robot stopped are marked with a cross in the corresponding color. Units are m relative to a fixed frame on the ground plane.

Table 4.3: This table shows the mean overall execution times on the rectangular wave path for all planner behaviors with the different velocity and path length scale settings.

PLS	0.2 m/s			0.4 m/s		
	Filter	None	Planner	Filter	None	Planner
0.08	235s	324s	221s	292s	374s	302s
0.1	257s	319s	214s	197s	380s	178s
0.12	225s	275s	211s	143s	266s	135s

especially for higher path length scales. While for the medium path length scale, large errors in the Filter behavior are due to overshooting, behavior None is already cutting the turns here. For the highest path length scale, all behaviors tend to use shortcuts and thus show high errors.

The rectangular wave path is by far the most challenging due to the 90 ° corners. In Figure 4.14 we show that the filter overshoots and stops a few times in different maximum velocity and path length scale settings. The planner behavior without ICR awareness stops

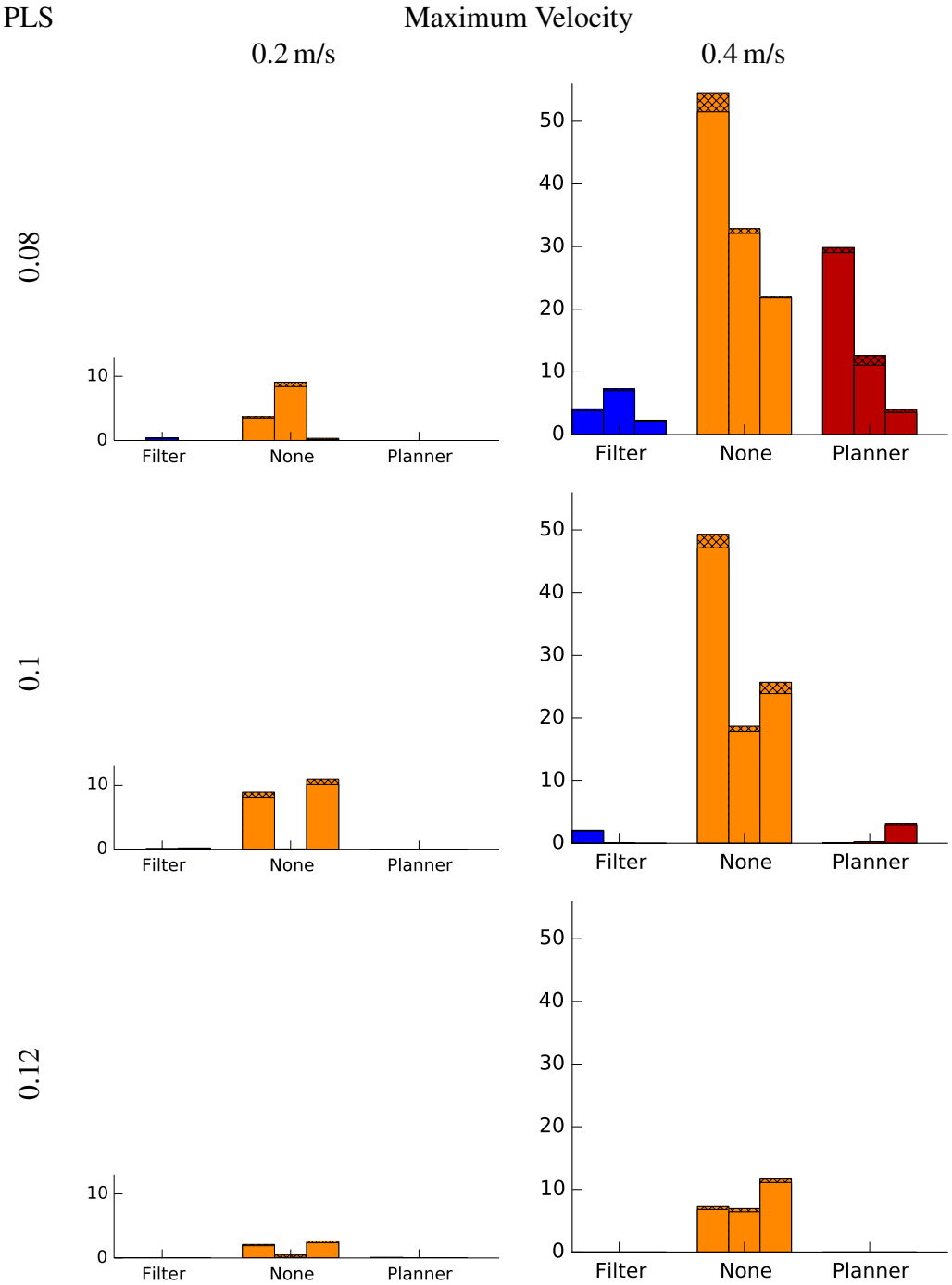


Figure 4.15: These plots show the time spent standing in s on the rectangular wave path for each of the three runs with each planner behavior, with the different maximum velocity settings and path length scales. The shaded areas represent standing without turning the wheels.

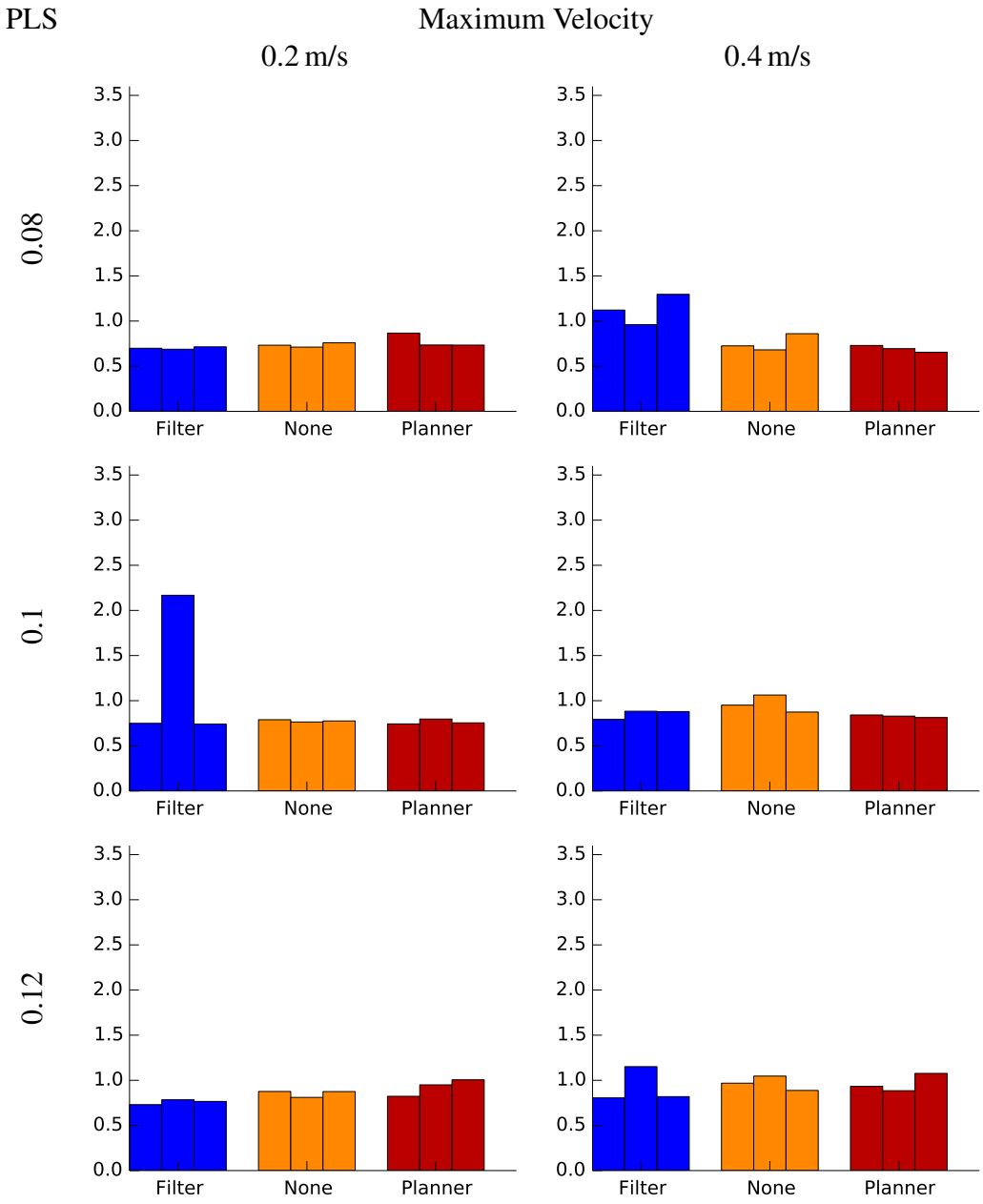


Figure 4.16: These plots show the maximum error in m between each pose on the path the planner took and its corresponding waypoint on the original rectangular wave path. The error is shown for all three runs with each planner behavior, with the different maximum velocity settings and path length scales.

a lot on this path. We observe that, in particular for high maximum velocity with low path length scale, the ICR aware Planner stops multiple times. Following this path accurately without a lot of wheel angle changes is a difficult task. This is also reflected by the time spent standing (see Figure 4.15): It is the first time the ICR aware Planner spends a notable amount of time standing, up to roughly 30 s with high maximum velocity and low path length scale, i. e., it tries to follow the given path fast, but accurately. It is also the path where ICR awareness None leads to the largest amounts of standing time by far with up to roughly 60 s, compared to the other paths. From Table 4.3 we observe that for low path length scale, all behaviors take longer when using the higher maximum velocity setting than when moving more slowly. This is because for the straight line segments of the path, the planner uses maximum velocity to cover as much of the path as possible in the given time frame, but then has to slow down drastically to avoid overshooting and turn the wheels to stay close enough to the original path. Since we keep the previously chosen velocity rollout until a different one has a considerably better score for stability reasons, the maximum velocity is kept for too long. For higher path length scales, i. e., lower expectations regarding accuracy, the execution time reduces to about half for the Filter and Planner behaviors, and about 3/4 for the None behavior. The path following accuracy shown in Figure 4.16 is similar among all three ICR awareness levels, with some outlier where the Filter overshoots.

Overall, we observed in our simulation experiments that Filter and Planner ICR awareness greatly increase path execution efficiency by avoiding excessive wheel turning. However, the Filter tends to overshoot the path since the filtered velocity is determined after the score computation and is thus not tailored to staying close to the original path. We further observed that a high path length scale tends to result in taking shortcuts. This leads to a high inaccuracy especially with a large maximum velocity.

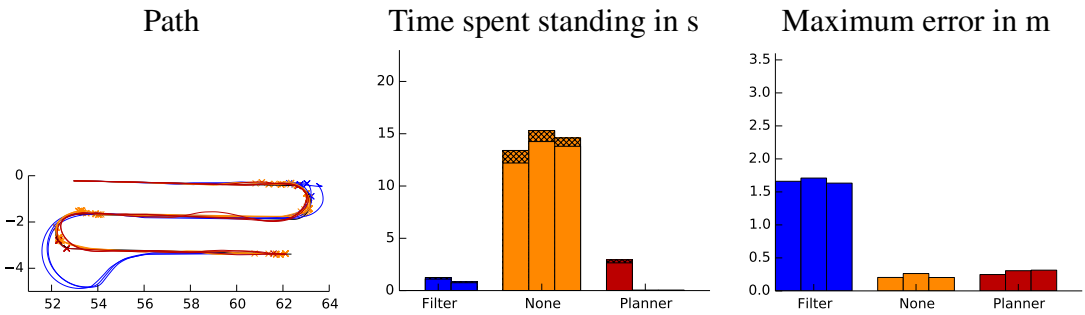


Figure 4.17: These plots show the performance of the different planning behaviors in a real world experiment on the field path. We used the maximum velocity setting 0.4 and a path length scale of 0.08. On the left, we see the original field path (black) and the paths the different planner behaviors took (yellow: None, blue: Filter, red: Planner). Points where the robot stopped are marked with a cross in the corresponding color. The plot in the middle shows the time the robot spent standing, where the shaded areas represent standing without turning the wheels. On the right, we see the maximum error of each pose on a path the robot took to its corresponding waypoint on the original path.

Table 4.4: This table shows the mean overall execution times on the field path for all planner behaviors in our real world experiment.

Filter	None	Planner
133s	146s	113s

For our real world experiment, we choose a path length scale of 0.08 and maximum velocity of 0.4 m/s based on the results of our simulation experiments. We use the field path as our most common use case in agricultural environments. In Figure 4.17 on the left, we show the paths taken by the different planner behaviors and the stops. This plot confirms our simulation results, in particular that ICR awareness None stops a lot more than either of the other two behaviors. This becomes even more apparent in the middle plot showing the time spent standing. The tendency of the filter to overshoot in tight turns shows again, which is also reflected in the maximum error plot on the right. In Table 4.4, we show the mean overall execution times for all planner behaviors. The Filter and Planner execution times are very similar to the execution times in simulation (see Table 4.2, top right for comparison). The Filter behavior takes longer than the planner behavior due to its overshooting and the resulting longer path. The None behavior performs slightly better in the real world setup than in simulation, but still takes longer than both other behaviors. In conclusion, these results validate that the Planner behavior is able to minimize time spent standing while retaining accuracy in path following. Overall, this confirms our observations from simulation data.

4.5 Conclusions

Robots with four-wheel independent steering have great movement flexibility, but are subject to some steering constraints. For example, large robots may only be able to turn their wheels slowly, to avoid mechanical stress. This leads to robots stopping to turn their wheels when attempting to follow a path accurately. We presented a novel approach to integrate steering constraints into local planning and thus avoiding unnecessary stops for wheel turning. We used a unified compact representation of different steering constraints in ICR space developed by Fleckenstein et al. [31] and proposed two methods to compute velocity commands that satisfy these constraints. The first method uses velocity commands computed considering only velocity and acceleration constraints. It applies a post-processing filter to make the commands conform with steering constraints. The second method directly integrates steering constraints into the velocity command computation.

In our experiments, we investigated path following accuracy and execution efficiency. While the first method, which applies a post-processing filter on the computed velocity commands, already increases path execution efficiency, it tends to overshoot in tight turns. In contrast, we showed that integrating steering constraints directly into the velocity command computation does not affect the path execution accuracy. Furthermore, we demonstrated that considering the steering constraints in local planning greatly improves

the efficiency of path execution by reducing the time spent standing to change wheel angles.

Chapter 5

Conclusions and Outlook

This thesis presented advanced techniques for various components of autonomous navigation in the context of precision agriculture. There are two main challenges in this scenario: First, determining the semantics of an agricultural environment is a hard task due to rapid changes and a wide variety of plant shapes and sizes. Plant positions are the only reliable naturally occuring feature on a crop field. Second, the narrow spaces in greenhouses and crop fields make navigation through them difficult and require high movement flexibility. We contributed to semantics computation in fields on the sensor input side as well as on the semantics output side. We introduced a vegetation feature representation that is able to capture different sensor data types and has the potential to make any task based on crop perception independent of the type of input sensor. Moreover, we established a quality measure for crop row detections, which are commonly used in localization or row following on agricultural fields. With regard to movement flexibility in narrow spaces, two components have to be considered: A global path planner that determines a feasible path to a given goal, and a local planner to execute the computed path. We showed the suitability of an approach for path planning with an adjustable intrinsic configuration that enables high motion flexibility for navigation in narrow spaces. Furthermore, we developed a local planning method that incorporates steering constraints for four-wheel independent steering that retains path following accuracy while ensuring efficient path execution.

First, we proposed a method for representing vegetation feature data extracted from camera or lidar sensors in a unified manner. As the information gain from each sensor type depends on the size of plants and the lighting conditions, being able to draw on information from various sensor modalities is a considerable advantage for estimating semantics in a crop field. In most cases, crop or crop row detection methods highly depend on the type of sensor data. Representing camera and lidar data in a unified form provides higher flexibility regarding the type and size of crops, the time of day and the weather conditions. Our representation is based on a 2d grid map located on the ground plane where each grid cell holds a weight that depicts how likely it is that there is vegetation present in this cell. It can be computed from either type of sensor data by projecting it onto the ground plane, using intrinsic and extrinsic calibration. Any crop or crop row detection method using this representation is then independent of the type of sensor that was used to capture the data.

Running a crop row detection method on this vegetation feature map yields crop row

patterns that best explain the data, but are not necessarily correct. High weed pressure and vegetation at the edge of the field can mislead the detection approach. Thus, we additionally proposed a quality measure for crop row detections that can be used for classifying detections as reliable or unreliable. Unreliable detections can then be filtered out, for example in a localization approach. The quality measure depends on different factors, all connected to the support of the detected crop row pattern in the vegetation feature map. We showed the performance of a classifier based on our quality measure in extensive experiments on real world data from different crop types and sizes. Furthermore, we used the classifier in a crop-row-based localization approach and evaluated the performance gain in terms of pose estimate accuracy. The experiments show that a classifier based on our quality measure is able to filter out a considerable amount of incorrect detections. They also demonstrate that by applying this filter, we increase the localization robustness and are able to prevent the pose estimate from diverging.

For global path planning in agriculture, we applied methods from our previous work [29]. We used the search-based planner presented therein, which efficiently includes adjustable joint configurations that allow for higher movement flexibility. At the same time, these adjustable joints increase the size of the planning space. To keep a high planning efficiency despite the larger planning space, we introduced an interval representation for valid joint configurations. Another contribution in this earlier work was the Wheel Dijkstra heuristic. We investigate this heuristic developed for vehicles with high ground clearance further in this thesis. We evaluate the performance of this path planner in a variety of different settings, with special attention to a real world scenario and a very cluttered scenario that shows the limits of planning without considering the adjustable joints. While heuristics show differing outcomes in all environments, one consistent result is that combining the Wheel Dijkstra heuristic with the freespace mechanism heuristic that assumes an obstacle free environment leads to the best planner performance. The experiments illustrate that the performance of the used planner transfers well from a simulated to a real world field environment. Moreover, planning time consumption using the joint interval representation is comparable to planning without the adjustable joints. In a cluttered environment, the advantage of including the adjustable joints in the planning problem becomes apparent, as the planner is able to solve only a small portion of the planning queries without them.

To execute the computed path with a local planner, we proposed an approach based on velocity rollouts that incorporates steering constraints in two different ways. The first computes a velocity rollout based only on velocity and acceleration constraints and applies a post-processing filter so as to not violate any steering constraints. The second directly integrates steering constraints into the velocity rollout computation. As steering constraints are complex to formulate in velocity space, we employed a compact and unified representation in ICR space. We presented how to compute an ICR path to a given goal velocity and how to determine a feasible velocity rollout from an ICR path. We performed simulation and real world experiments on global paths with different challenges. When the constraints are directly included in the velocity rollout computation, the path following accuracy remains equal to path execution when steering constraints are ignored. The experiments further show that considering steering constraints in local planning can

greatly improve the efficiency of path execution, as the robot spends less time standing for turning its wheels.

In this work, we presented an approach for depicting either camera or lidar data in a common representation. Since we may not know beforehand which sensor provides more information on vegetation, a method for fusing the input of both sensor types into one vegetation feature map would be worthwhile to explore.

Additionally, we tested the impact of filtering crop row detections with the quality measure we proposed and of using the result in a crop-row-based localization. It would be interesting to explore the impact in other applications such as mapping or row following. Furthermore, we used a binary classifier for detection filtering. As the quality measure provides a continuous value, it could be used as a confidence score. In localization, this could help to avoid large longitudinal errors at the end of the field as crop row detections in this area tend to have low quality while still carrying valuable information about the detected end of the field.

The path planner we used from earlier work can theoretically include any adjustable one-dimensional joints. It would be intriguing to evaluate the impact of our interval representation for joint configurations on more complex systems such as manipulator arms with several joints that form a kinematic chain.

For our local planning approach, we observed that including the steering constraints helps considerably with reducing standing time for wheel turning. However, we see room for further improvement. Currently, the accuracy score for a velocity rollout is based on the distance of each pose in the pose rollout to the corresponding waypoint on the global path. This ignores when parts of the global path are skipped. A more comprehensive error estimation between the pose rollouts and the global path would be beneficial in increasing the path following accuracy, specifically with regard to cutting corners and tight turns.

Overall, we provide contributions to advance autonomous navigation in precision agriculture in various areas. We hope to progress more sustainable and efficient farming technologies to ensure sufficient food production for the future.

List of Figures

List of Tables

Bibliography

[1] Abbas, M., Elhamshary, M., Rizk, H., Torki, M., and Youssef, M. (2019). WiDeep: WiFi-based Accurate and Robust Indoor Localization System using Deep Learning. In *International Conference on Pervasive Computing and Communications (PerCom)*.

[2] Agapiou, A. (2020). Vegetation Extraction Using Visible-Bands From Openly Licensed Unmanned Aerial Vehicle Imagery. *Drones*, 4.

[3] Albornoz, F. (2016). Crop responses to nitrogen overfertilization: A review. *Scientia Horticulturae*, 205:79–83.

[4] Altini, M., Brunelli, D., Farella, E., and Benini, L. (2010). Bluetooth indoor localization with multiple neural networks. In *International Symposium on Wireless Pervasive Computing*, pages 295–300.

[5] Åstrand, B. and Baerveldt, A.-J. (2005). A vision based row-following system for agricultural field machinery. *Mechatronics*, 15(2):251–269.

[6] Bakker, T., van Asselt, K., Bontsema, J., Müller, J., and van Straten, G. (2011). Autonomous navigation using a robot platform in a sugar beet field. *Biosystems Engineering (BE)*, 109(4):357–368.

[7] Bakker, T., Wouters, H., van Asselt, K., Bontsema, J., Tang, L., Müller, J., and van Straten, G. (2008). A vision based row detection system for sugar beet. *Computers and Electronics in Agriculture (CEA)*, 60(1):87–95.

[8] Baltazar, A. R., Santos, F. N. d., Moreira, A. P., Valente, A., and Cunha, J. B. (2021). Smarter Robotic Sprayer System for Precision Agriculture. *Electronics*, 10(17).

[9] Berenson, D., Srinivasa, S. S., Ferguson, D., Collet, A., and Kuffner, J. J. (2009). Manipulation planning with workspace goal regions. In *International Conference on Robotics and Automation (ICRA)*.

[10] Berenstein, R. and Edan, Y. (2017). Human-robot collaborative site-specific sprayer. *Journal of Field Robotics (JFR)*, 34(8):1519–1530.

[11] Bestaoui, Y. (1989). On line motion generation with velocity and acceleration constraints. *Robotics and Autonomous Systems (RAS)*, 5(3):279–288.

[12] Biber, P., Weiss, U., Dorna, M., and Albert, A. (2012). Navigation System of the Autonomous Agricultural Robot "BoniRob" . In *International Conference on Intelligent Robots and Systems (IROS)*.

[13] Bonet, B. and Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1):5–33.

[14] Bongiovanni, R. and Lowenberg-DeBoer, J. (2004). Precision Agriculture and Sustainability. *Precision Agriculture (PA)*, 5:359–387.

[15] Boniardi, F., Valada, A., Mohan, R., Caselitz, T., and Burgard, W. (2019). Robot Localization in Floor Plans Using a Room Layout Edge Extraction Network. In *International Conference on Intelligent Robots and Systems (IROS)*.

[16] Burgard, W., Brock, O., and Stachniss, C. (2008). Map-Based Precision Vehicle Localization in Urban Environments. In *Robotics: Science and Systems III*, pages 121–128.

[17] Burns, B. and Brock, O. (2005). Single-Query Entropy-Guided Path Planning. In *International Conference on Robotics and Automation (ICRA)*.

[18] Calisi, D., Nardi, D., Ohno, K., and Tadokoro, S. (2008). A semi-autonomous tracked robot system for rescue missions. In *Society of Instrument and Control Engineers (SICE) Annual Conference*.

[19] Chamberland, S., Beaudry, E., Clavien, L., Kabanza, F., Michaud, F., and Lauriay, M. (2010). Motion planning for an omnidirectional robot with steering constraints. In *International Conference on Intelligent Robots and Systems (IROS)*.

[20] Chen, Z., Zou, H., Jiang, H., Zhu, Q., Soh, Y. C., and Xie, L. (2015). Fusion of WiFi, Smartphone Sensors and Landmarks Using the Kalman Filter for Indoor Localization. *Sensors*, 15(1):715–732.

[21] Cherubini, A., Chaumette, F., and Oriolo, G. (2011). Visual servoing for path reaching with nonholonomic robots. *Robotica*, 29(7):1037–1048.

[22] Clavien, L., Lauria, M., and Michaud, F. (2018). Estimation of the instantaneous centre of rotation with nonholonomic omnidirectional mobile robots. *Robotics and Autonomous Systems (RAS)*, 106:47–57.

[23] Cohen, B., Chitta, S., and Likhachev, M. (2014). Single- and dual-arm motion planning with heuristic search. *International Journal of Robotics Research (IJRR)*, 33(2):305–320.

[24] Conceição, A. S., Moreira, A. P., and Costa, P. J. (2008). A nonlinear model predictive control strategy for trajectory tracking of a four-wheeled omnidirectional mobile robot. *Optimal Control Applications and Methods*, 29(5):335–352.

[25] Constantin, D., Rehak, M., Akhtman, Y., and Liebisch, F. (2015). Detection of Crop Properties by Means of Hyperspectral Remote Sensing from a Micro UAV. *Bornimer Agrartechnische Berichte*, 88:129–137.

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

[27] English, A., Ball, D., Ross, P., Upcroft, B., Wyeth, G., and Corke, P. (2013). Low Cost Localisation for Agricultural Robotics. In *Australasian Conference on Robotics and Automation (ACRA)*.

[28] Fang, Y., Hu, J., Liu, W., Shao, Q., Qi, J., and Peng, Y. (2019). Smooth and time-optimal S-curve trajectory planning for automated robots and machines. *Mechanism and Machine Theory*, 137:127–153.

[29] Fleckenstein, F. (2016). Planning for Outdoor Vehicles with Adjustable Wheel Positions. [Master's thesis, University of Freiburg].

[30] Fleckenstein, F., Dornhege, C., and Burgard, W. (2017). Efficient Path Planning for Mobile Robots with Adjustable Wheel Positions. In *International Conference on Robotics and Automation (ICRA)*.

[31] Fleckenstein, F., Winterhalter, W., Dornhege, C., Pradalier, C., and Burgard, W. (2019). Smooth Local Planning Incorporating Steering Constraints. In *International Conference on Field and Service Robotics (FSR)*.

[32] Fox, D., Burgard, W., and Thrun, S. (1997). The dynamic window approach to collision avoidance. *Robotics & Automation Magazine*, 4(1):23–33.

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

[34] Gasparetto, A. and Zanotto, V. (2007). A new method for smooth trajectory planning of robot manipulators. *Mechanism and Machine Theory*, 42(4):455–471.

[35] Geiger, F., Bengtsson, J., Berendse, F., Weisser, W. W., Emmerson, M., Morales, M. B., Ceryngier, P., Liira, J., Tscharntke, T., Winqvist, C., Eggers, S., Bommarco, R., Pärt, T., Bretagnolle, V., Plantegenest, M., Clement, L. W., Dennis, C., Palmer, C., Oñate, J. J., Guerrero, I., Hawro, V., Aavik, T., Thies, C., Flohre, A., Hänke, S., Fischer, C., Goedhart, P. W., and Inchausti, P. (2010). Persistent negative effects of pesticides on biodiversity and biological control potential on European farmland. *Basic and Applied Ecology*, 11(2):97–105.

[36] Gerkey, B. P. and Konolige, K. (2008). Planning and control in unstructured terrain. In *Workshop on Path Planning on Costmaps, IEEE International Conference on Robotics and Automation (ICRA)*.

[37] Gianni, M., Gonnelli, G., Sinha, A., Menna, M., and Pirri, F. (2013). An augmented reality approach for trajectory planning and control of tracked vehicles in rescue environments. In *International Symposium on Safety, Security and Rescue Robotics*.

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

[39] Guo, X., Coops, N. C., Tompalski, P., Nielsen, S. E., Bater, C. W., and John Stadt, J. (2017). Regional mapping of vegetation structure for biodiversity monitoring using airborne lidar data. *Ecological Informatics*, 38:50–61.

[40] Hata, A. Y., Osorio, F. S., and Wolf, D. F. (2014). Robust curb detection and vehicle localization in urban environments. In *Intelligent Vehicles Symposium*.

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

[42] IEEE (2021). PR2. https://robots.ieee.org/robots/pr2/. Accessed: 2021-12-30.

[43] Indiveri, G., Nuchter, A., and Lingemann, K. (2007). High speed differential drive mobile robot path following control with bounded wheel speed commands. In *International Conference on Robotics and Automation (ICRA)*.

[44] Jurie, F., Rives, P., Gallice, J., and Brame, J. (1994). High-speed vehicle guidance based on vision. *Control Engineering Practice*, 2(2):289–297.

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

[46] Karamipour, E., Dehkordi, S. F., and Korayem, M. (2020). Reconfigurable Mobile Robot with Adjustable Width and Length: Conceptual Design, Motion Equations and Simulation. *Journal of Intelligent & Robotic Systems*, 99:797–814.

[47] Katan, J. and Eshel, Y. (1973). Interactions between herbicides and plant pathogens. *Residue Reviews*, 45:145–177.

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

[49] Kim, D., Choi, Y., Park, T., Lee, J. Y., and Han, C. (2015). Efficient path planning for high-DOF articulated robots with adaptive dimensionality. In *International Conference on Robotics and Automation (ICRA)*.

[50] Kim, I., Jeon, W., and Yang, H. (2017). Design of a transformable mobile robot for enhancing mobility. *International Journal of Advanced Robotic Systems*, 14(1).

[51] Kise, M., Zhang, Q., and Más, F. R. (2005). A Stereovision-based Crop Row Detection Method for Tractor-automated Guidance. *Biosystems Engineering*, 90(4):357–367.

[52] Klančar, G. and Škrjanc, I. (2007). Tracking-error model-based predictive control for mobile robots in real time. *Robotics and Autonomous Systems (RAS)*, 55(6):460–469.

[53] Korf, R. E. (1988). Optimal Path-Finding Algorithms. In *Search in Artificial Intelligence*, pages 223–267. Springer New York.

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

[55] Kunz, C., Weber, J., Peteinatos, G., Sökefeld, M., and Gerhards, R. (2018). Camera steered mechanical weed control in sugar beet, maize and soybean. *Precision Agriculture (PA)*, 19:708–720.

[56] Lam, T. L., Qian, H., and Xu, Y. (2010). Omnidirectional steering interface and control for a four-wheel independent steering vehicle. *IEEE/ASME Transactions on Mechatronics*, 15(3):329–338.

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

[58] Levinson, J. and Thrun, S. (2010). Robust vehicle localization in urban environments using probabilistic maps. In *International Conference on Robotics and Automation (ICRA)*, pages 4372–4378.

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

[60] Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., and Thrun, S. (2005). Anytime Dynamic A*: An Anytime, Replanning Algorithm. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

[61] Liu, X. and Gong, D. (2011). A comparative study of A-star algorithms for search and rescue in perfect maze. In *International Conference on Electric Information and Control Engineering*.

[62] Machleb, J., Peteinatos, G. G., Kollenda, B. L., Andújar, D., and Gerhards, R. (2020). Sensor-based mechanical weed control: Present state and prospects. *Computers and Electronics in Agriculture (CEA)*, 176.

[63] Marchant, J. (1996). Tracking of row structure in three crops using image analysis. *Computers and Electronics in Agriculture (CEA)*, 15(2):161–179.

[64] Meyer, G. E. and Neto, J. C. (2008). Verification of Color Vegetation Indices for Automated Crop Imaging Applications. *Computers and Electronics in Agriculture (CEA)*, 63(2):282–293.

[65] Micaelli, A. and Samson, C. (1994). Trajectory tracking for two-steering-wheels mobile robots. *IFAC Symposium on Robot Control*, 27(14):249–256.

[66] Montalvo, M., Pajares, G., Guerrero, J., Romeo, J., Guijarro, M., Ribeiro, A., Ruz, J., and Cruz, J. (2012). Automatic detection of crop rows in maize fields with high weeds pressure. *Expert Systems with Applications*, 39(15):11889–11897.

[67] Naïo Technologies (2021). Ted. https://www.naio-technologies.com/en/ted/. Accessed: 2021-12-30.

[68] Oberti, R., Marchi, M., Tirelli, P., Calcante, A., Iriti, M., Hočevar, M., Baur, J., Pfaff, J., Schütz, C., and Ulbrich, H. (2013). Selective spraying of grapevine's diseases by a modular agricultural robot. *Journal of Agricultural Engineering*, 44:149–153.

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

[70] Onishi, Y., Yoshida, T., Kurita, H., Fukao, T., Arihara, H., and Iwai, A. (2019). An automated fruit harvesting robot by using deep learning. *ROBOMECH Journal*, 6.

[71] Pamosoaji, A. K., Piao, M., and Hong, K.-S. (2019). PSO-based minimum-time motion planning for multiple vehicles under acceleration and velocity limitations. *International Journal of Control, Automation and Systems*, 17(10):2610–2623.

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

[73] Pretto, A., Aravecchia, S., Burgard, W., Chebrolu, N., Dornhege, C., Falck, T., Fleckenstein, F., Fontenla, A., Imperoli, M., Khanna, R., Liebisch, F., Lottes, P., Milioto, A., Nardi, D., Nardi, S., Pfeifer, J., Popović, M., Potena, C., Pradalier, C., Rothacker-Feder, E., Sa, I., Schaefer, A., Siegwart, R., Stachniss, C., Walter, A., Winterhalter, W., Wu, X., and Nieto, J. (2021). Building an Aerial-Ground Robotics System for Precision Farming: An Adaptable Solution. *Robotics & Automation Magazine*, 28(3):29–49.

[74] Qin, S. and Badgwell, T. A. (2003). A survey of industrial model predictive control technology. *Control Engineering Practice*, 11(7):733 – 764.

[75] Radwan, N., Tipaldi, G. D., Spinello, L., and Burgard, W. (2016). Do you see the bakery? Leveraging geo-referenced texts for global localization in public maps. In *International Conference on Robotics and Automation (ICRA)*.

[76] Rosell, J., Suárez, R., and Pérez, A. (2013). Path planning for grasping operations using an adaptive PCA-based sampling method. *Autonomous Robots*, 35(1):27–36.

[77] Schaefer, A., Büscher, D., Vertens, J., Luft, L., and Burgard, W. (2021). Long-term vehicle localization in urban environments based on pole landmarks extracted from 3-D lidar scans. *Robotics and Autonomous Systems*, 136:103709.

[78] Schwesinger, U., Pradalier, C., and Siegwart, R. (2012). A novel approach for steering wheel synchronization with velocity/acceleration limits and mechanical constraints. In *International Conference on Intelligent Robots and Systems (IROS)*.

[79] Sha, Z. and Yu, M. (2008). Remote sensing imagery in vegetation mapping: A review. *Journal of Plant Ecology*, 1:9–23.

[80] Simmons, R. (1996). The curvature-velocity method for local obstacle avoidance. In *International Conference on Robotics and Automation (ICRA)*.

[81] SITIA (2016). TREKTOR: The hybrid autonomous robot for agriculture. https://www.sitia.fr/en/innovation-2/trektor/. Accessed: 2021-12-30.

[82] Søgaard, H. and Olsen, H. (2003). Determination of crop rows by image analysis without segmentation. *Computers and Electronics in Agriculture (CEA)*, 38(2):141–158.

[83] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. MIT Press.

[84] Tillett, N., Hague, T., and Miles, S. (2002). Inter-row vision guidance for mechanical weed control in sugar beet. *Computers and Electronics in Agriculture*, 33(3):163–177.

[85] Tsardoulias, E., Iliakopoulou, K., Kargakos, A., and Petrou, L. (2016). A Review of Global Path Planning Methods for Occupancy Grid Maps Regardless of Obstacle Density. *Journal of Intelligent & Robotic Systems*, 84:829–858.

[86] Urmson, C. and Simmons, R. G. (2003). Approaches for heuristically biasing RRT growth. In *International Conference on Intelligent Robots and Systems (IROS)*.

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

[88] Vidović, I., Cupec, R., and Ž. Hocenski (2016). Crop row detection by global energy minimization. *Pattern Recognition*, 55:68–86.

[89] Winterhalter, W., Fleckenstein, F., Dornhege, C., and Burgard, W. (2018). Crop Row Detection on Tiny Plants With the Pattern Hough Transform. *Robotics and Automation Letters (RA-L)*, 3(4):3394–3401.

[90] Winterhalter, W., Fleckenstein, F., Dornhege, C., and Burgard, W. (2021). Localization for Precision Navigation in Agricultural Fields – Beyond Crop Row Following. *Journal of Field Robotics*, 38(3):429–451.

[91] Winterhalter, W., Fleckenstein, F., Steder, B., Spinello, L., and Burgard, W. (2015). Accurate indoor localization for RGB-D smartphones and tablets given 2D floor plans. In *International Conference on Intelligent Robots and Systems (IROS)*.

[92] Wu, X., Aravecchia, S., Lottes, P., Stachniss, C., and Pradalier, C. (2020). Robotic weed control using automated weed and crop classification. *Journal of Field Robotics*, 37(2):322–340.

[93] Ye, Y., He, L., and Zhang, Q. (2016). Steering Control Strategies for a Four-Wheel-Independent-Steering Bin Managing Robot. *IFAC-PapersOnLine*, 49(16):39–44. 5th IFAC Conference on Sensing, Control and Automation Technologies for Agriculture AGRICONTROL 2016.

[94] Zafari, F., Gkelias, A., and Leung, K. K. (2019). A Survey of Indoor Localization Systems and Technologies. *Communications Surveys & Tutorials*, 21(3):2568–2599.

[95] Zhuang, Y., Yang, J., Li, Y., Qi, L., and El-Sheimy, N. (2016). Smartphone-based indoor localization with bluetooth low energy beacons. *Sensors*, 16(5).

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