HAL open science

Smooth Local Planning Incorporating Steering Constraints

Freya Fleckenstein, Wera Winterhalter, Christian Dornhege, Cedric Pradalier, Wolfram Burgard

▶ **To cite this version:**

Freya Fleckenstein, Wera Winterhalter, Christian Dornhege, Cedric Pradalier, Wolfram Burgard. Smooth Local Planning Incorporating Steering Constraints. 12th Conference Field and Service Robotics, Aug 2019, Tokyo, Japan. hal-02183027v1

HAL Id: hal-02183027

https://hal.science/hal-02183027v1

Submitted on 14 Jul 2019 (v1), last revised 16 Jul 2019 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Smooth Local Planning Incorporating Steering Constraints

Freya Fleckenstein, Wera Winterhalter, Christian Dornhege, Cédric Pradalier and Wolfram Burgard

Abstract Individually controllable and steerable wheels provide vehicles with greater flexibility and efficiency. A key challenge in the context of such vehicles lies in the effective planning of the motion commands in order to properly deal with the potentially large required changes in the steering angles and to properly incorporate the given motion constraints. Especially large vehicles often perform large steering angle changes only while standing to prevent physical damage to the robot. In this paper we introduce a novel planner for vehicles with individually controllable and steerable wheels and two novel methods to incorporate steering constraints. The latter are designed to provide smooth changes of the instantaneous center of rotation (ICR). Extensive experiments in simulation and on the real robot reveal that our methods allow for an effective planning of steering commands, reduce the travel time and additionally enable the vehicle to accurately follow a given path.

1 Introduction

Mobile autonomous robots have become increasingly important in agricultural and industrial environments. Compared to standard equipment such as tractors, they have the advantage of being more lightweight thus reducing soil compression and being more flexible thus providing the potential to lower the utilization of chemicals. However, the variability of the crop production requires an appropriate flexibility of the robots with respect to wheel adjustments, such as with the robot depicted in Figure 1.

Freya Fleckenstein · Wera Winterhalter · Christian Dornhege · Wolfram Burgard
University of Freiburg, Germany, e-mail: {fleckenf,winterhw,dornhege,burgard}@tf.uni-freiburg.de

Cédric Pradalier
GeorgiaTech Lorraine-UMI 2958, GeorgiaTech-CNRS, Metz, France, e-mail: cedric.pradalier@georgiatech-metz.fr

The core capability of the navigation system is to generate and execute a collision-free trajectory. A common approach is to use a global planner to compute a path consisting of a list of poses the robot should traverse. In addition, a local planner is used to generate motions connecting through these waypoints. Usual requirements are that the motions are smooth, safe and can accurately be executed even at the via-points. The robot then has to translate the motions into concrete steering commands resulting in the proper steering angles of the wheels and the wheel velocities. Our goal in this paper is to create such a local planner that uses all available degrees of freedom to effectively navigate while still accurately tracking the global path.

Fig. 1 These pictures show the agricultural robot BoniRob with four wheels whose orientation and rotation can be changed independently. The picture on the right has been taken during an autonomous run.

In general, there are several hardware limitations that impose constraints on the available control commands. First, there are the overall velocity and acceleration limits. Second, there are steering constraints such as the maximum steering angle and steering velocity, i.e., how fast the robot is able to change its wheel orientations. Our algorithm considers all of these constraints while at the same time generating velocities that lead to high tracking accuracy and at the same time generate efficient overall movements of the vehicle.

The controller we present is based on the principle of trajectory rollouts [7]. It determines and evaluates a set of candidate velocities close to the current velocity. Each candidate is evaluated as to whether it satisfies the hardware acceleration and velocity constraints. It computes a sequence of velocities to reach the desired velocity from the current one. Finally it evaluates each sequence and chooses the best one.

The particular challenge when considering limits on steering angles and steering velocities is that smooth transitions in the velocity of the robot do not necessarily lead to smooth transitions in the wheel angles. For example, to first execute a small forward velocity and then a small sideways velocity, the wheels always need to turn by 90° independent of how small the absolute velocity change is. Therefore it is necessary to consider changes of the wheel angles explicitly. In this paper, we present two methods to incorporate steering constraints in the local planner by transforming the wheel configurations into the space of the instantaneous centers of

rotation (ICR). First, we introduce the constraints in a post-processing filter step on the velocity command that was computed without regard for steering constraints. The second approach is to directly incorporate the steering constraints as ICR constraints in the local planner. To achieve this, we compute a local path of valid ICRs towards a candidate velocity and if such a path exists instantiate the ICRs with velocities to gain a trajectory. In addition, we demonstrate how the steering mechanical constraints can be efficiently expressed in a unified manner.

We evaluate both methods on three trajectories posing different challenges in simulation and verify the observed behavior in real-world experiments. The results show that including the ICR constraints with our method greatly reduces the travel time. At the same time, integrating the constraints directly in the local planner maintains high tracking accuracy.

2 Related Work

Many approaches have been presented that compute velocity commands for tracking a given path. Early methods use a set forward velocities and only control the steering to stay close to the designated trajectory [11]. They were, for example, successfully used in controlling vehicles on a motorway [9]. Similarly, several approaches have been developed that are purely reactive and directly map sensor data to steering corrections [2]. These approaches are not sufficient for the problem we consider, as they might lead to great changes in required steering angles. Other methods restrict the possible paths to be twice differentiable [3]. Our experiments reveal that we do not need comply with this restriction to get a low tracking error.

Some higher-level approaches use model predictive tracking strategies to estimate the future robot state [5, 13]. Fox *et al.* and Gerkey and Konolige [6, 7] sample a set of velocities, predict the robot pose one timestep into the future, compute a score for each velocity and execute the best. Our approach is similar in that we also sample a set of velocities and score them.

There are two major differences to our approach. First, we propose to generate sequences of velocities toward a sampled velocity and include a path prediction in the score. This allows us to plan further ahead and create more stable sequences. In particular, mechanical constraints may require the robot to slow down, which we can incorporate. The pose paths resulting from these velocity sequences are similar to the motion primitives in state lattice planning [12]. Second, we consider mechanical steering constraints in the velocity sequence generation. While most local planner methods incorporate velocity and acceleration limits [10, 6], few of them consider these additional constraints. For integrating the mechanical steering constraints, we leverage previous work [4, 14]. The authors show that the velocity of a ground vehicle corresponds directly to the ICR coordinates expressed in homogeneous coordinates that can be mapped onto a unit sphere in 3D [4]. The advantage of the spherical coordinates is that they are bounded and that any transition between two ICRs can be represented by a segment of a great circle on the sphere. If the

segment crosses the equator, then the robot wheels will temporarily be aligned and the path curvature will change sign at this point. Steering velocity constraints can be expressed by half-plane constraints on the ICR displacement [14]. In this paper, we combine these results and demonstrate how wheel mechanical constraints can be expressed efficiently in a common frame.

3 Integrating Mechanical Steering Constraints

In this section, we demonstrate how the steering mechanical constraints can be efficiently expressed in a unified manner. We consider three constraints: the maximum steering velocity, the distance to a wheel and the rotation limit of the steering joints. In a practical deployment, having an ICR too close to one of the vehicle wheels leads to mechanical instabilities: a small numerical variation in the ICR least-square position estimate [4] will result in large steering angle changes, which will put a continuous strain on the wheel. Additionally, few vehicles have steering joints that can rotate continuously. E.g., in our system it is important to avoid moving from steering angle 89° to 91°, as the wheel will have to be rotated by 178°.

3.1 Formalizing Mechanical Steering Constraints

We describe the mechanical steering constraints in ICR coordinates expressed in the robot coordinate frame. Consider the homogeneous coordinates of an ICR as 3D points. Then, any point P of the homogeneous plane is projected orthogonally on the surface of the unit sphere centered at $(0,0,0)$ by normalizing it as $\Pi(P)$. Points at infinity (i.e., straight line motions) project to the equator of the sphere. The north pole of the sphere corresponds to an ICR representing rotation on the spot. Inversely, from a point (u,v,w) on the sphere, the corresponding ICR can be found as $(u/w, v/w)$ if $w \neq 0$. As a result, diametrically opposite points are equivalent. For any ICR I in homogeneous coordinates, $\Pi^+(I) = I/\|I\|$ and $\Pi^-(I) = -I/\|I\|$ are its equivalent diametrically opposed projections on the sphere. We define $\Pi(I)$ as the tuple $(\Pi^-(I), \Pi^+(I))$. Then, a segment $J = [I_1, I_2]$ in the homogeneous space projects to two great-circle segments $\Pi(J) = ([\Pi^-(I_1), \Pi^-(I_2)], [\Pi^+(I_1), \Pi^+(I_2)])$.

We consider a vehicle with wheels W_i, represented in homogeneous coordinates $(x_i, y_i, 1)$. According to our notation, the line joining the ith wheel to I_1 is $L_i = I_1 \times W_i$. In projective geometry, the cross product of two points is the homogeneous representation of the line going through them, even for points at infinity. Note that a line defined by $ax + by + c = 0$ can be represented in homogeneous coordinates $L_i = (a,b,c)$. Hence, the wheel steering is obtained by $\phi_i = \text{atan2}(b,a)$.

Steering Velocity Constraints Using the results from Schwesinger *et al.* [14], we express the steering velocity constraints by half-plane constraints on the ICR

displacement. Because of the maximum steering velocity, a wheel steering ϕ can change by at most $\pm\delta\phi$ in one timestep. Geometrically, this means that every wheel defines a cone of $\pm\delta\phi$ with boundaries L_i^+ and L_i^- resulting from $W_i \times (-\sin(\phi_i \pm \delta\phi), \cos(\phi_i \pm \delta\phi), 0)^T$. The intersection of the cones defined by all the wheels is the set of feasible ICR changes within one timestep. In the $\mathbb{R}^2$ plane, this intersection is a Nef polygon [1, 8], i.e., a polygon that is defined by the intersection of half-planes and may include points at infinity. The ICR movements and constraints are illustrated in Figure 2. Assuming the ICR needs to move from I_1 to I_2, let us define the line $\mathscr{L} = I_1 \times I_2$ on which the ICR has to move to achieve the requested ICR. The intersection of $\mathscr{L}$ with the cone defined by L_i^+ and L_i^- is the generalized segment $\mathscr{M}_i = [M_i^-, M_i^+]$. A feasible ICR will hence have to belong to the intersection of the n generalized segments $\mathscr{M}_i$ due to all the wheel constraints, or the equivalent great circle segments $\Pi(\mathscr{M}_i)$ on the sphere.

Wheel Distance Constraints If I_1 and I_2 are not both at infinity, we want to forbid the ICR from moving within a radius r from wheel i. This requires computing the intersection C_i^+ and C_i^- of $\mathscr{L}$ with the circle centered on W_i with radius r. This is solved with basic geometry in $\mathbb{R}^2$, without resorting to projective geometry tricks. Contrary to the steering velocity constraints, the wheel distance constraints are exclusive, so the ICR must be outside the $\mathscr{C}_i = [C_i^-, C_i^+]$ generalized segment.

Rotation Limit Constraint of Steering Joints The wheel mechanical constraints define a specific range of steering that the ICR should not enter. Similar to the steering velocity constraints, this defines a set of exclusion cones in the ICR space. The intersection of $\mathscr{L}$ with these cones gives a set of generalized segments $\mathscr{N}_i = [N_i^-, N_i^+]$ out of which the ICR should always be.

3.2 Constraint Unification

All the above ICR constraints have been expressed as inclusion or exclusion segments for the ICR, so the next challenge is to fuse all these segments efficiently. To this end, we use the process depicted in Figure 3: we will project all of them on the unit sphere. In practice, all the segment ends have been computed numerically, so we compute the great circle containing all of them using total least squares. Now, we can represent each ICR P with the coordinates $(x, y, 0)$ of its projection on the great circle, or more efficiently by its angular position $\theta^+(P) = \text{atan2}(y, x)$ which is equivalent to the opposite angle $\theta^-(P) = \text{atan2}(-y, -x)$. In summary, every generalized segment $[p, q]$ in homogeneous coordinates projects on a tuple of great circle segment $\Pi([p, q])$, which is expressed as an angle interval union

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

and the challenge is now to intersect a set of intervals in the most efficient way.

We choose to represent a collection of angle intervals by a sorted list of endpoints and a Boolean flag stating if $\pm\pi$ is included or not in the collection. If it is, the

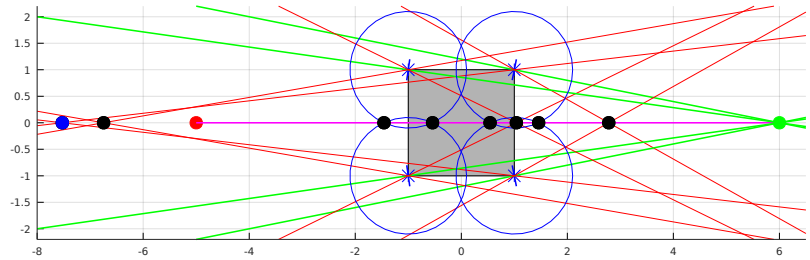

Fig. 2 ICR movement constraints. The robot is depicted in gray, with four wheels. The ICR is initially on the green disk, linked by the green lines to the four wheels, and is requested to move to the red position. The magenta line highlights the ICR path, which crosses the steering velocity constraints in red and the wheel distance constraints in blue. The black dots depict the intersections, the blue disk is the furthest the ICR can move in one timestep.

first value in the list is the end of the first interval, otherwise, it is the start. This representation has a quick complement operator by toggling the flag as well as a quick union operator which just fuses the sorted list in linear time. The intersection is computed using the complement of the union of the complement.

The unified constraints is the intersection $\mathscr{I}$ of all the constraints:

$$\mathscr{I} = \bigcap_{i=1}^{p} \theta\left(\mathscr{M}_i\right) \cap \overline{\theta\left(\mathscr{C}_i\right)} \cap \overline{\theta\left(\mathscr{N}_i\right)} \tag{1}$$

If $\mathscr{I}$ is empty, then the current ICR is too close to the wheels or the mechanical limits and there is no feasible movement to reach a suitable ICR within one timestep. In general, $\mathscr{I}$ will be a collection of intervals, out of which we select the one containing 0, i.e., $\theta(I_1)$. If it also contains $\theta(I_2)$, then I_2 is feasible and safe. Otherwise, the interval endpoint closest to $\theta(I_2)$ is selected. If $\mathscr{I}$ does not include 0, then the current ICR is not acceptable, but there is a feasible ICR at reachable distance. In this case, the selected interval is the one closest to 0. From the optimal value of $\theta(I^*)$, we can easily recover $\Pi(I^*)$ and a safe ICR I^*. In practice, we can run this process in less than $10\mu s$, which is a critical requirement to integrate it into the trajectory rollout presented in Section 4.

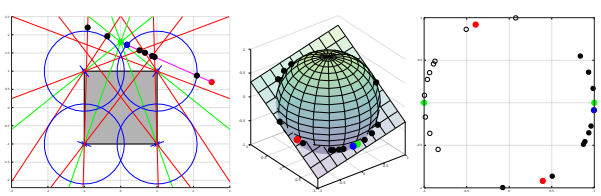

Fig. 3 ICR constraint unification process. Left: the ICR initial and requested configuration in 2D, and the best feasible ICR in blue. Center: projection of the ICR and constraints on the unit sphere, and extraction of the great circle containing all the projections. Right: the ICR constraint on the great circle $\theta(.)$. Black and white disks correspond to opposed constraints.

4 From Paths to Velocity Commands

To follow a path, the robot executes velocity commands $v = (v_x, v_y, \omega)$. At each timestep, we determine a set of candidate velocities. For each candidate, we compute a trajectory rollout of discretized velocities and poses stretching over a constant rollout time. Then we compute a cost for each rollout and choose the best.

Trajectory Rollouts We define a fixed discretization of the velocity space and compute a set of candidate velocities around the current velocity that satisfy the velocity and acceleration constraints of the robot. Assuming maximum acceleration towards the candidate velocity in v_x, v_y and ω, a velocity rollout is generated. If the candidate velocity is reached before the rollout time ends, this velocity is kept for the remaining time. The pose rollout is then computed by successively applying the velocities from the velocity rollout with a given timestep to the current pose.

Velocity Cost During path execution, the robot has to trade off efficiency and accuracy. For example, in a 90° turn, the most accurate execution would be going straight, stopping, turning, and going straight. In contrast, the more efficient, but less accurate solution would be to cut the corner in favor of a smoother and faster execution. As a measure of accuracy, we use the integrated distance $d(P,T)$ of the trajectory rollout T to the path P. For each point in the trajectory rollout, we determine the closest point on the path and compute the distance between those two points. We approximate the integrated distance by the sum over these distances. We model the efficiency by the length of the covered path segment $l(P,T)$. The covered path segment is defined by the first and last trajectory rollout matches onto the path. We balance the effects of these two costs by introducing the *path length scale* $s \in [0,1]$. The velocity cost is then given as $c(P,T) = (1-s) \cdot d(P,T) + s \cdot l(P,T)$. For larger values of s, the length of the covered path segment gains more weight so that the local planner will prefer rollouts that cut corners.

5 Integrating Steering Constraints into the Local Planner

While the approach described above takes into account low-level hardware constraints, it does not consider mechanical steering constraints. As we explained above, these constraints can be expressed in the ICR space in a unified manner. To exploit this model of the steering constraints, we project the velocities (v_x, v_y, ω) into the ICR space expressed in homogeneous coordinates $(-v_y, v_x, \omega)$ [4]. If $\omega \neq 0$, the ICR coordinates in the $\mathbb{R}^2$ plane are $(-v_y/\omega, v_x/\omega)$. Otherwise, the robot is moving straight, which corresponds to an ICR at infinity in the direction $(-v_y, v_x)$, which is exactly what homogeneous coordinates $(-v_y, v_x, 0)$ express. There are two possibilities to integrate ICR constraints into the velocity command execution pipeline. First, the ICR constraints can be enforced in a post-processing *Filter* step. Second, the ICR constraints can be integrated directly into the trajectory rollout computation.

5.1 Mechanical Steering Constraints in a Post-Processing Filter

To filter the velocity commands in a post-processing step, we first compute the trajectory rollouts in the local planner as described in Section 4. The filter then receives the requested velocity command from the local planner as (v_x^r, v_y^r, ω^r). We compute the corresponding ICR $I^r = (-v_y^r, v_x^r, \omega^r)$. Together with the current ICR I^c we find an acceptable ICR $I^* = (i_x^*, i_y^*, w^*)$ that satisfies the mechanical steering constraints and is as close to I^r as possible. We then compute the filtered velocity command $v^* = (v_x^*, v_y^*, \omega^*)$ by projecting I^* back into the velocity space.

However, an ICR is equivalent to a velocity only up to a scale factor. The last stage is to scale the ICR to output velocities as close as possible to the requested command. Two cases must be considered: If $\|(-i_y^*, i_x^*)\| = 0$, then we set $\omega^* = \omega^r$ and $v_x^* = v_y^* = 0$. Otherwise, we scale $(-i_y^*, i_x^*, w^*)$ with a factor α to give $(v_x^*, v_y^*) = \alpha(-i_y^*, i_x^*)$ the same norm and direction as (v_x^r, v_y^r), which means keeping $v_x^r v_x^* + v_y^r v_y^* \geq 0$. This yields a velocity that satisfies the mechanical steering constraints with the same translational velocity as the requested command.

5.2 Mechanical Steering Constraints in the Local Planner

To integrate the steering constraints directly into the local planner, these constraints have to be considered when generating candidate velocities as well as during trajectory rollout computation. First, we discard candidate velocities that do not satisfy the constraints in the ICR space. A more complex problem is generating trajectory rollouts that are compliant with the steering constraints.

Given the current ICR I^c computed from the current wheel configuration and the candidate ICR I of the candidate velocity v, we compute an ICR path with a limited steering angle change in each timestep. If the candidate ICR is not reachable within the rollout time, the candidate velocity is discarded. Given the current velocity $v^c = (v_x^c, v_y^c, \omega^c)$, the candidate velocity $v = (v_x, v_y, \omega)$ and an ICR path $\{I^0 = I^c, I^1, \ldots, I^n = I\}$, we compute a velocity rollout as follows:

For the previous velocity v^{i-1} and the required ICR $I^i = (i_x, i_y, w)$ solve

$$\begin{aligned} \arg\min_\alpha \|\alpha \cdot (i_y, -i_x, w) - v\| \text{ s.t. } & \alpha \cdot i_y \in (v_x^{i-1} - a_x^{\max} \cdot t, v_x^{i-1} + a_x^{\max} \cdot t) \\ & \alpha \cdot -i_x \in \left(v_y^{i-1} - a_y^{\max} \cdot t, v_y^{i-1} + a_y^{\max} \cdot t\right) \\ & \alpha \cdot w \in \left(\omega^{i-1} - a_\omega^{\max} \cdot t, \omega^{i-1} + a_\omega^{\max} \cdot t\right) \end{aligned}$$

where $a_x^{\max}, a_y^{\max}, a_\omega^{\max}$ are the maximum accelerations and t is the length of one timestep. The acceleration limits are set according to the hardware limits of the robot and the timestep is set to the internal control cycle of the firmware. Intuitively, this provides us with a velocity that is as close to the candidate velocity as possible, while being within the acceleration limits and compatible with the next ICR in the ICR path. This yields a velocity rollout with the additional constraint that the steering

angle does not change by more than a given threshold in one timestep. The pose rollout is computed from the velocity rollout as before.

6 Evaluation

We evaluated our approach in simulated and real world experiments to show how an ICR-aware local planner affects the navigation of the robot and which way of integrating ICRs performs better. In particular we investigate if considering ICRs leads to higher efficiency during path execution as the robot does not have to stop to turn the wheels. At the same time we investigate how much the accuracy is impacted. To this end, we considered three different paths to follow: First, a path of smoothly connected *lines and arcs*; second, a *field*-like path with long straight lines and tight 180° turns; and third, a *rectangular wave* path with varying side length, shown in Figures 4, 5, and 7. All experiments were conducted with maximum velocities set to $v_x^{\max} = v_y^{\max} = 0.2\text{m/s}, \omega^{\max} = 0.1\text{rad/s}$ and $v_x^{\max} = v_y^{\max} = 0.4\text{m/s}, \omega^{\max} = 0.2\text{rad/s}$ respectively. We will reference these two settings as maximum velocity 0.2 and 0.4 in this section. We additionally modified the influence of the covered path length on the velocity rating, using path length scales of 0.08, 0.1, and 0.12. All experiments were performed in simulation with three repetitions. In addition, we evaluated the field path with 0.4 maximum velocity and a path length scale of 0.08 in a real-world scenario on our BoniRob robot shown in Figure 1 to verify the results of the simulation experiments. We used the odometry to determine the pose of the robot.

We analyze several criteria that are relevant for the local planner performance: A plot of the path to follow in comparison to the executed trajectory to judge how closely the robot is able to follow a given path; the tracking accuracy along the path; The overall mean execution time and, to gain a closer insight on the last point, the time spent standing to perform wheel orientation changes. As a measure for the tracking accuracy, we use the maximum error between the path and the executed trajectory. We show experiment results for our approach with different variants of how to include ICR constraints: including the ICR constraints directly in the trajectory rollouts of the local planner yielding an ICR-aware local planner (Planner), no ICR-awareness (None), and without ICR-awareness but an additional filter run between the local planner output and the command execution (Filter).

First of all, Table 1 clearly shows that considering the necessary wheel angle changes notably reduces the execution time, both with a post-processing filter and when incorporating the constraints directly into the planner. The approach None is always slower than both other variants, by up to a factor of 1.7 (Field, maximum velocity 0.2). Shown are the results for path length scale 0.08; For other path length scales the results are similar.

The lines and arcs path does not pose a challenge to any of the algorithm variants as it only contains straight lines and wide arcs and thus does not force the robot towards its steering constraints. Figure 4 shows the results for path length scale 0.12, for other path length scales the results are similar. It is already visible in the

Table 1 The mean execution times for path length scale 0.08 for all trajectories and all evaluated algorithms with different velocities.

	Maximum Velocity 0.2			Maximum Velocity 0.4		
	Filter	None	Planner	Filter	None	Planner
Lines and Arcs	385s	492s	383s	182s	214s	182s
Field	169s	257s	152s	147s	192s	122s
Rectangular Wave	235s	324s	221s	292s	374s	302s

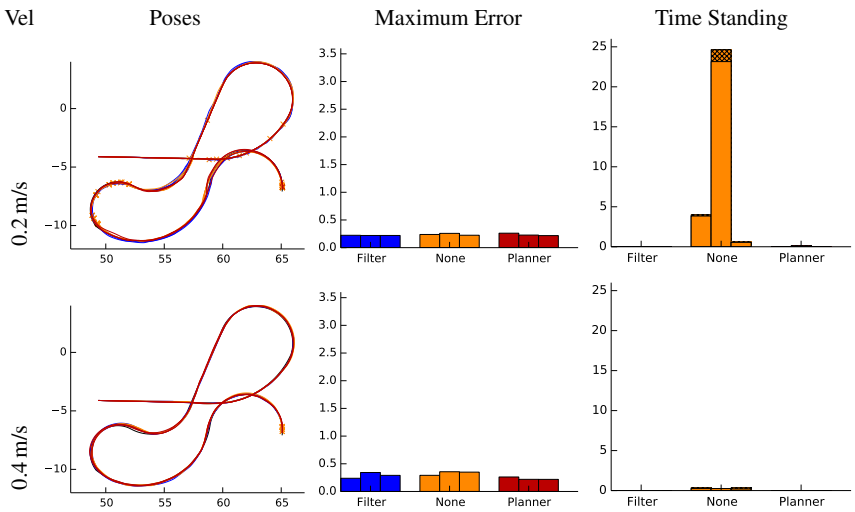

Fig. 4 The results for the lines and arcs path with path length scale 0.12.

trajectory overlay that all variants have a good tracking accuracy. This is confirmed by an overall maximum tracking error of about 35 cm. Neither the filter nor the ICR aware planner spent any considerable time standing, while the planner without regard for its wheel configurations does. All in all, these results demonstrate that on a smooth path with wide arcs, it is beneficial to consider the wheel configuration, but it suffices to avoid unnecessary wheel changes in a post-processing Filter.

The field path contains tight 180 degree turns. Here, the influence of the path length scale becomes more visible as it allows the robot to shortcut the turns (Figure 5). Nevertheless, one might choose a larger path length scale to allow the robot to perform smoother curves, thus reducing standing times. This effect is independent of considering ICR constraints. Figure 6 shows that the maximum error increases when increasing the path length scale, especially for a maximum velocity of 0.4.

We also see that the Filter overshoots the turns on multiple occasions as it prevents turning the wheels to stay within the steering constraints when it would be necessary to do so for the turn. On the one hand this is an undesirable behavior that neither the Planner nor the None variant face. On the other hand Table 1 showed that the Filter succeeds in increasing efficiency. While similar to the Filter and None the Planner is also affected by the reduced accuracy when increasing the path length

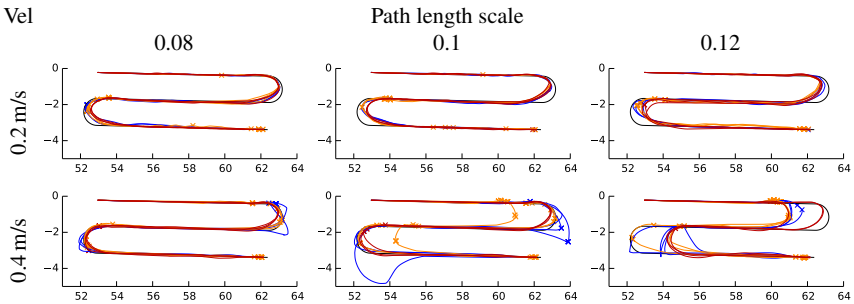

Fig. 5 The evaluated trajectories for the field path. Stops are marked with x.

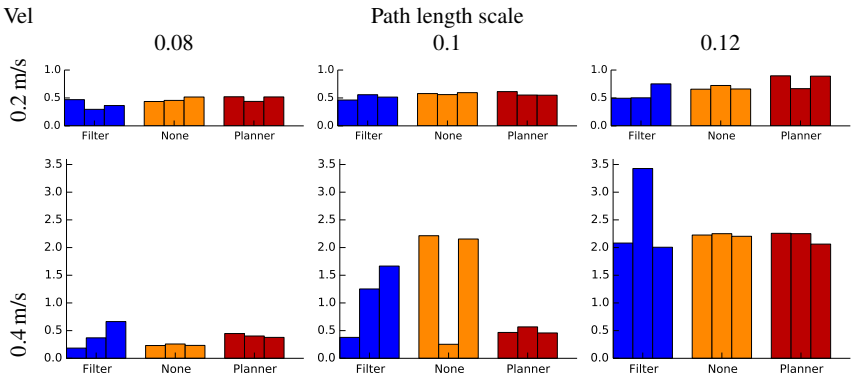

Fig. 6 The maximum error for the field path.

scale, it neither shows overshoots as the Filter nor does it take as long as None. Due to the shortcuts caused by the path length scale, effects on the accuracy due to considering ICRs are not clearly visible. It is still a relevant case as it directly maps to an application in agricultural robotics.

The third experiment on the rectangular wave path poses similar challenges of tight 90 degree turns, but does not provoke shortcuts as the field path does (see Figure 7 for the trajectory overlays). Therefore we did not observe any strong relationship of the path length scale to the accuracy. While all variants appear to be mostly close to the path with some shortcuts, the Filter clearly drifts away from the path in some cases. In this path, one reason for the increased efficiency when considering steering angles becomes apparent in Figure 8. While a larger path length scale does reduce the standing times for each variant, being aware of the ICRs pays off notably as the None variant keeps the robot standing the most. In contrast, in most cases the Filter and Planner show similarly small standing times. The exception is the combination of the smallest path length scale (0.08) with the larger maximum velocity of 0.4 forcing the robot in tight turns. While still better than None, the planner keeps the robot standing more than Filter for two reasons. First, the Planner keeps the

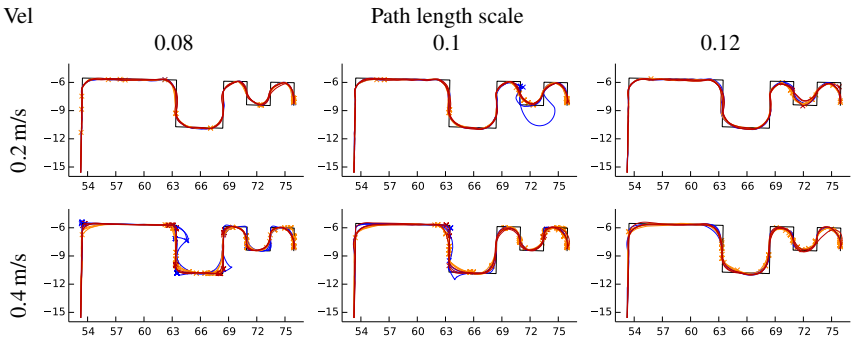

Fig. 7 The evaluated trajectories for the rectangular wave path. Stops are marked with x.

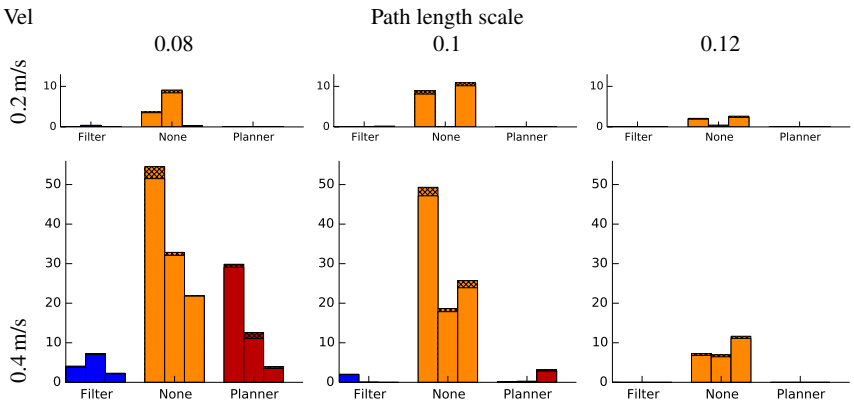

Fig. 8 The total time where the robot was standing in rectangular wave. Times where the robot was standing, but not turning its wheels are shaded with x.

robot standing when necessary, i.e., in the turns, while None also stands on straight lines. Second, the Filter only achieves the lower standing times than Planner at the cost of overshooting the path multiple times. The Filter provides a trade-off tailored towards smooth driving at the cost of following the path accurately.

The real-world experiment shown in Figure 9 shows this even more clearly. Based on the results of the simulation experiments, we set a path length scale of 0.08 for the field path and a maximum velocity of 0.4. The comparison of the three variants on the real robot confirms the results from the simulated data. Filtering the velocity commands using ICR constraints helps to drastically reduce the time spent standing and thus increases efficiency, but leads to large tracking errors mainly when it overshoots the path to keep execution smooth. In contrast, including the ICR constraints in the local planner also reduces the time spent standing, but retains the tracking accuracy of the local planner that does not consider ICR constraints and thus provides the best combination of both options.

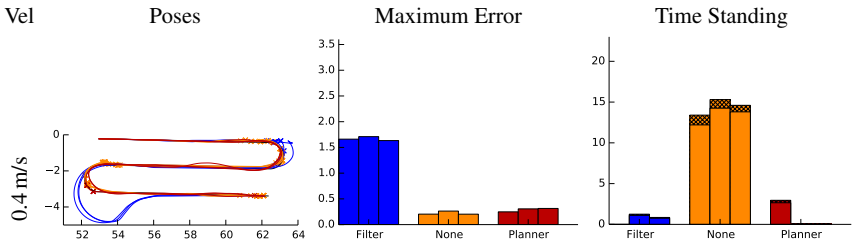

Fig. 9 The results for the real world experiment.

This is an observation that we make throughout all four experiments: Considering ICR constraints is necessary to prevent the robot from standing to perform wheel reconfigurations. As the Filter always prevents excessive wheel rotations it sometimes grossly overshoots the path. The planner is able to make this trade-off of preventing unnecessary wheel movements, while at the same time allowing accurate path tracking.

7 Conclusions

In this paper, we presented novel approaches based on trajectory rollouts for a local planner that computes velocity commands for robots with constraints on steering angles and steering velocities. In extensive experiments, we demonstrated that our approach leads to higher efficiency by minimizing time spent for wheel reconfigurations. At the same time it is able to highly accurately track the planned path when integrating the constraints directly in the planner.

Despite these positive results, we still see room for further improvement. In our future work we will investigate computationally feasible methods that estimate the true error between path and rollouts more precisely to further increase the tracking accuracy.

References

1. Bieri, H.: Nef polyhedra: A brief introduction. In: Geometric modelling, pp. 43–60. Springer (1995)
2. Björn Åstrand and Albert-Jan Baerveldt: A vision based row-following system for agricultural field machinery. Mechatronics **15**(2), 251 – 269 (2005)
3. Cherubini, A., Chaumette, F., Oriolo, G.: Visual servoing for path reaching with nonholonomic robots. Robotica (2011)
4. Clavien, L., Lauria, M., Michaud, F.: Estimation of the instantaneous centre of rotation with nonholonomic omnidirectional mobile robots. Robotics and Autonomous Systems **106**, 47–57 (2018)

5. Conceição, A.S., Moreira, A.P., Costa, P.J.: 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 (2008)
6. Fox, D., Burgard, W., Thrun, S.: The dynamic window approach to collision avoidance. IEEE Robotics Automation Magazine **4**(1), 23–33 (1997)
7. Gerkey, B.P., Konolige, K.: Planning and control in unstructured terrain. In: Workshop on Path Planning on Costmaps, IEEE International Conference on Robotics and Automation (ICRA) (2008)
8. Hachenberger, P., Kettner, L., Mehlhorn, K.: Boolean operations on 3d selective nef complexes: Data structure, algorithms, optimized implementation and experiments. Computational Geometry **38**(1-2), 64–99 (2007)
9. Jurie, F., Rives, P., Gallice, J., Brame, J.: High-speed vehicle guidance based on vision. Control Engineering Practice **2**(2), 289 – 297 (1994)
10. Klančar, G., Škrjanc, I.: Tracking-error model-based predictive control for mobile robots in real time. Robotics and Autonomous Systems **55**(6), 460 – 469 (2007)
11. Micaelli, A., Samson, C.: Trajectory tracking for two-steering-wheels mobile robots. IFAC Symposium on Robot Control **27**(14), 249 – 256 (1994)
12. Pivtoraiko, M., Kelly, A.: Efficient constrained path planning via search in state lattices (2005)
13. Qin, S., Badgwell, T.A.: A survey of industrial model predictive control technology. Control Engineering Practice **11**(7), 733 – 764 (2003)
14. Schwesinger, U., Pradalier, C., Siegwart, R.: A novel approach for steering wheel synchronization with velocity/acceleration limits and mechanical constraints. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2012)