2011 IEEE International Conference on Robotics and Automation
May 9-13, 2011, Shanghai, China

Efficient Kinodynamic Trajectory Generation for Wheeled Robots

Marcell Missura and Sven Behnke

Abstract— **Planning dynamic motion is computationally demanding and thus can hardly be done in real-time onboard robots. In this paper, we present an analytic approximation to predict the dynamic state of wheeled robots with non-holonomic constraints, given a start state and a sequence of piecewise constant controls. Our approximations are accurate and fast to calculate. They can be used to replace numerical integrators in kinodynamic planning algorithms. The predictions are differentiable and allow us to utilize gradient descent methods to solve the inverse dynamics as well and generate trajectories connecting arbitrary points in state space.**

I. Introduction

Motion planning in dynamic environments is a challenging problem and even more so for car-like vehicles with non-holonomic constraints. Despite some insights from optimal control theory [1], existing algorithms do not deliver the runtimes needed for real-time motion planning in dynamic environments with moving obstacles.

Thanks to their computational efficiency, the most simple reactive approaches have the leading edge regarding applicability on real systems. These include potential field methods [2], Lyapunov candidate function-based parking controllers [3], and path tracking methods, as for example used by the DARPA Grand Challenge winner "Stanley" [4]. As reactive controllers only obey ad-hoc heuristics and do not follow a global plan to reach a given target, they cannot produce optimal trajectories. They are also likely to get stuck in local minima and can produce oscillating behavior. This also applies to the Dynamic Window Approach [5], even if it does plan a small amount of time into the future.

A more extensive method to global planning is to determine a geometrical path that adheres to continuous curvature and minimal turning radius restrictions, such that in principle it can be followed by a vehicle with a steering wheel [6] [7]. This setting is purely kinematic and ignores important physical aspects of motion planning: velocity and acceleration. The produced paths are mostly suitable for vehicles with a relatively slow and steady pace. The kinematic setting, however, reduces the amount of dimensions to consider and helps some of the proposed path planners to retain real time capabilities, such as the Ariadne's Clew Algorithm [8].

In full-fledged kinodynamic planning [9], the state of a vehicle includes not only the Cartesian coordinates and the orientation, but also the translational and angular velocities. Planning is mostly performed directly in control space where velocity and acceleration bounds can inherently be taken into account. The increase in dimensionality makes discrete cell decomposition methods impractical. Thus, randomized approaches [10] are often used that sample possible control inputs and project them into the state space by numerical integration. This way, the open-loop controls to execute a calculated trajectory arise naturally. The avoidance of moving obstacles is also possible, as demonstrated in [11], by extending the state space by the time dimension in a similar fashion as it was already suggested in [12]. In [11] successful experiments on a real system are reported. The setting was reduced to holonomic motion planning, but the obstacles were moving and the robot managed to maneuver in between them to reach a given target. Except for this simplified setting, the computation times reported by the works cited above do not allow the application of kinodynamic planning on real non-holonomic systems in rapidly changing environments.

With our proposed method, we address the computationally expensive projection of controls into state space. Our analytic approximation allows fast and accurate calculation of future vehicle states. Replacing standard Euler or Runge-Kutta integration modules can significantly speed up kinodynamic motion planning. Furthermore, the differentiability of our formulas allows us to exploit gradient descent methods to determine the controls needed to connect two arbitrary states in the absence of obstacles, which is known as the steering problem.

The remainder of this work is organized as follows. First, we introduce the unicycle model, which is a simple unifying model for non-holonomic vehicles without drift. We show how cars and differential drives both can be mapped to the unicycle model. Then we present our analytical model that predicts the dynamic state of a unicycle given a set of piecewise constant controls. This is followed by experimental results comparing the accuracy and performance of our approach to standard numerical integration techniques. Finally, we present our approach to solve the steering problem with a gradient descent algorithm.

II. The Unicycle Model

We consider vehicles with the following dynamic model:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\omega} \end{pmatrix} = \begin{pmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} a + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} b. \quad (1)$$

The state vector $\mathbf{s} = (x, y, \theta, v, \omega) \in \mathbb{R}^2 \times [-\pi, \pi] \times \mathbb{R}^2$ of this system is five-dimensional and includes the Cartesian

All authors are with the Autonomous Intelligent Systems Group, University of Bonn, Germany. Email: missura@ais.uni-bonn.de This work has been supported partially by grant BE 2556/2-3 of German Research Foundation (DFG)

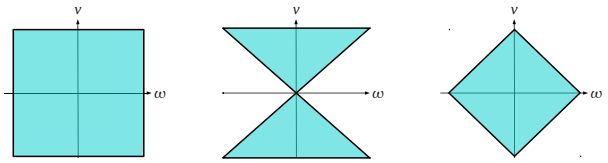

Fig. 1. Velocity domains of the unicycle (left), a car (center) and a differential drive (right). The permitted translational and angular velocity combinations are inside the shaded areas.

coordinates (x, y), the orientation θ, the translational velocity v, and the angular velocity ω.

The non-holonomic constraint can be expressed with the equation

$$\dot{y}\cos\theta - \dot{x}\sin\theta = 0, \tag{2}$$

or in simple words: the robot always moves in the direction it is currently facing. To change the system state, two control parameters a and b can be used. $a \in [-A, A]$ is a bounded acceleration of the translational velocity v and $b \in [-B, B]$ is a bounded acceleration of the angular velocity ω. We consider t to be a third control parameter that describes for how long the accelerations a and b are applied to the system. The mobile robot is controlled by a set of piecewise constant control triples $\mathbf{U} = \{(a, b, t)\}$. This representation not only makes analytic approximation feasible, but also allows to approximate any non-linear control function by choosing a large amount of control triples with a small t.

Despite their conceptual differences, cars and differential drive vehicles can both be mapped to the unicycle (1). Cars are controlled with an acceleration pedal that influences the rolling speed ν of the wheels and a steering wheel that changes the steering angle ζ. Assuming a distance of 1 between the front and rear axles, the controls of the car can be mapped to (1) with the equations

$$v = \nu\cos\zeta, \tag{3}$$

$$\omega = \nu\sin\zeta. \tag{4}$$

As the steering angle is bounded by the constraint $|\zeta| \leq \zeta_{max}$, a car always has to respect a minimal turning radius

$$\frac{v}{\omega} \geq r_{min}. \tag{5}$$

This constraint excludes a subset of the velocity domain, as depicted in Fig. 1. Differential drives do not have a steering wheel, but the velocities of the wheels on the left side ν_l and on the right side ν_r can be controlled independently. Let l be the distance between the wheels on the left and the right. Then the mapping of the differential drive controls to (1) is given by

$$v = \frac{1}{2}(\nu_l + \nu_r), \tag{6}$$

$$\omega = \frac{1}{l}(\nu_l - \nu_r). \tag{7}$$

The velocity limits of a differential drive are imposed by the maximum velocity of the wheels. If a robot wants to exploit its maximum translational velocity, both wheels have to roll forward (backward) with their maximum velocities and the angular velocity is restricted to zero. The analogous condition applies to maximizing the angular velocity. This is also illustrated in Fig. 1. The different limitations on the velocity domain do not affect our predictions, as our formulas are valid for the unbounded $v \times \omega$ space.

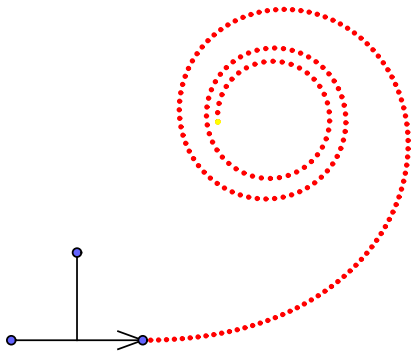

Fig. 2. An example non-holonomic trajectory for constant controls $a = 0.05\,m/s^2$, $b = 0.95\,rad/s^2$ for $t = 5.55\,s$ with a start state $v = 1.5\,m/s$ and $\omega = 1.5\,rad/s$.

The curves described by the unicycle can be characterized as follows. The fraction $\frac{v}{\omega}$ yields the instantaneous radius of curvature for a given state $\mathbf{s}$. If no controls are applied, the vehicle travels on a circular arc, or on a straight line if $\omega = 0$. If any controls a and b are applied for a time t, the radius of curvature r changes smoothly according to the law

$$r = \frac{v + at}{\omega + bt}. \tag{8}$$

As time grows to infinity, the radius of curvature approaches $\frac{a}{b}$. Generally, we are dealing with spirals that converge to a circle with the radius $\frac{a}{b}$ with time. Unlike common spirals, these spirals do not revolve around a center and their evolute is not a single point. Figure 2 shows an example. Clothoids are a special case of these spirals when $a = 0$ and $b \neq 0$. Clothoids have the nice property that the curvature is a linear function of the distance traveled along the path. However, this advantage can only be fully exploited if a vehicle travels with constant velocity. We argue that, for ease of control, acceleration should be applied linear in time rather than linear in the distance traveled.

III. Analytic Approximation

As it was described in the previous section, the robot is controlled by a sequence of triples $\mathbf{u_i} = (a_i, b_i, t_i)$. When the system state $\mathbf{s_i} = (x_i, y_i, \theta_i, v_i, \omega_i)$ is known, the transformation induced by the control input $\mathbf{u_i}$ is described by the transfer function $\mathbf{s_{i+1}} = T(\mathbf{s_i}, \mathbf{u_i})$. For more convenient notation we will drop the index i in the sequel, unless explicitly needed for disambiguation. We obtain the transfer

function by integrating the equations of motion (1) over time

$$T(\mathbf{s},\mathbf{u}) = \begin{pmatrix} x \\ y \\ \theta \\ v \\ \omega \end{pmatrix} + \begin{pmatrix} \Delta X(\mathbf{s},\mathbf{u}) \\ \Delta Y(\mathbf{s},\mathbf{u}) \\ \frac{b}{2}t^2 + \omega t \\ at \\ bt \end{pmatrix}. \tag{9}$$

The change of Cartesian coordinates

$$\Delta X(\mathbf{s},\mathbf{u}) = \int_0^t v(\tau)\cos\theta(\tau)d\tau, \tag{10}$$

$$\Delta Y(\mathbf{s},\mathbf{u}) = \int_0^t v(\tau)\sin\theta(\tau)d\tau \tag{11}$$

cannot be integrated in closed form. However, the antiderivatives can be written as

$$\begin{aligned} F_x(\mathbf{s},\mathbf{u}) = {} & \frac{\sqrt{\pi}}{b^{\frac{3}{2}}}(bv - a\omega)(\sigma S + \gamma C) \\ & + \frac{a}{b}\sin(\frac{b}{2}t^2 + \omega t + \theta), \end{aligned} \tag{12}$$

$$\begin{aligned} F_y(\mathbf{s},\mathbf{u}) = {} & \frac{\sqrt{\pi}}{b^{\frac{3}{2}}}(bv - a\omega)(\gamma S - \sigma C) \\ & - \frac{a}{b}\cos(\frac{b}{2}t^2 + \omega t + \theta), \end{aligned} \tag{13}$$

where

$$\gamma = \cos(\frac{b}{2}\omega^2 - \theta), \tag{14}$$

$$\sigma = \sin(\frac{b}{2}\omega^2 - \theta), \tag{15}$$

$$S = S(\frac{\omega + bt}{\sqrt{b\pi}}), \tag{16}$$

$$C = C(\frac{\omega + bt}{\sqrt{b\pi}}). \tag{17}$$

The antiderivatives are expressed in terms of the Fresnel integrals $S(x)$ and $C(x)$:

$$S(x) = \int_0^x \sin(\frac{\pi}{2}t^2)dt, \tag{18}$$

$$C(x) = \int_0^x \cos(\frac{\pi}{2}t^2)dt. \tag{19}$$

The Fresnel integrals can also not be evaluated in closed form, but efficient and accurate approximations exist. We are using a C implementation of the method described in [13]. Other implementations are readily available in mathematical tools, e.g. Matlab and Mathematica. Now, the Cartesian coordinates of the new state $\mathbf{s_{i+1}}$ can be calculated using

$$\Delta X(\mathbf{s},\mathbf{u}) = F_x(\mathbf{s},\mathbf{u}) - F_x(\mathbf{s},\mathbf{u_0}), \tag{20}$$

$$\Delta Y(\mathbf{s},\mathbf{u}) = F_y(\mathbf{s},\mathbf{u}) - F_y(\mathbf{s},\mathbf{u_0}), \tag{21}$$

where $\mathbf{u_0} = (a, b, 0)$.

There is still a number of special cases to consider. Equations (12) and (13) are defined only for $b > 0$. If b is negative, a symmetry has to be exploited to avoid the negative square roots appearing in (12), (13), (16), and (17). By substituting b with $-b$ and ω with $-\omega$, an axially symmetric pair of ΔX and ΔY can be computed using (20) and (21). Then the vector $(\Delta X, \Delta Y)$ has to be mirrored back on the axis defined by the initial angle θ, before it can be used in Equation (9).

Furthermore, to avoid the singularity at $b = 0$, we integrate the equations of motion (1) assuming $b = 0$ and obtain a second set of antiderivatives

$$F_x(\mathbf{s},\mathbf{u}) = \frac{a\cos(\omega t + \theta)}{\omega^2} + \frac{(at + v)\sin(\omega t + \theta)}{\omega}, \tag{22}$$

$$F_y(\mathbf{s},\mathbf{u}) = \frac{a\sin(\omega t + \theta)}{\omega^2} - \frac{(at + v)\cos(\omega t + \theta)}{\omega}. \tag{23}$$

Now using (22) and (23) in (20) and (21) respectively, the Cartesian coordinates can be calculated for the case $b = 0$.

If b and ω are both zero, the vehicle travels on a straight line and (10), (11) can be calculated directly as

$$\Delta X(\mathbf{s},\mathbf{u}) = (\frac{a}{2}t^2 + vt)\cos\theta, \tag{24}$$

$$\Delta Y(\mathbf{s},\mathbf{u}) = (\frac{a}{2}t^2 + vt)\sin\theta. \tag{25}$$

To integrate a sequence $\mathbf{U}$ of n control inputs $\mathbf{u_i} \in \mathbf{U}$, the transfer function can be concatenated as follows:

$$T(\mathbf{s_0},\mathbf{U}) = \begin{pmatrix} x_0 \\ y_0 \\ \theta_0 \\ v_0 \\ \omega_0 \end{pmatrix} + \sum_{i=0}^{n-1} \begin{pmatrix} \Delta X(\mathbf{s_i},\mathbf{u_i}) \\ \Delta Y(\mathbf{s_i},\mathbf{u_i}) \\ \frac{b_i}{2}t_i^2 + \omega_i t_i \\ a_i t_i \\ b_i t_i \end{pmatrix}. \tag{26}$$

IV. Experimental Results

To verify the accuracy of our method, we compare its output with standard numerical integration methods. We generated a set of 10,000 start state and control pairs with each dimension uniformly sampled from the range $[-10, 10]$, except for t which was sampled from the range $[0, 10]$. The data set includes highly dynamic situations with velocities up to 10 m/s, angular velocities up to 10 rad/s and adequately strong accelerations of up to 10 m/s^2 and 10 rad/s^2 projected up to 10 seconds forward in time. For each case, we calculate the Cartesian coordinates (x, y) using our transfer function (9), Euler's method and fourth-order Runge-Kutta integration. The other three state components do not need to be evaluated, since they are calculated analytically. Unfortunately, all three methods can only deliver approximate results and there is no ground truth to compare to, so we can only compare the results with each other. We express the quality of our predictions by their closeness to the results of numerical integration. In Fig. 3, the pairwise average distances between the three integration methods are presented with standard deviations. The time step of the numerical integrations was set to 10^{-6} seconds and distances are calculated with the Euclidean norm. The results can be interpreted as follows. Obviously, the high-precision numerical integrations and our

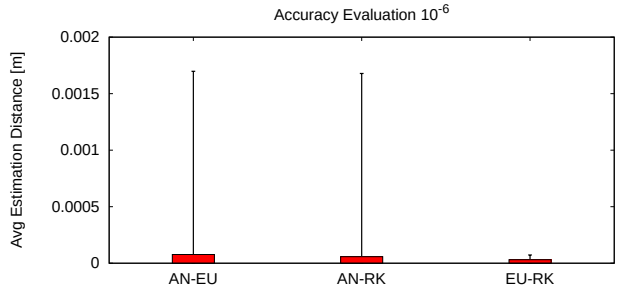

Fig. 3. Average distances and their standard deviations between our estimations and Euler's method (AN-EU), between our method and fourth order Runge-Kutta integration (AN-RK) and between Euler's method and Runge-Kutta integration (EU-RK). The numerical integrations were performed with a time step of 10^{-6} seconds.

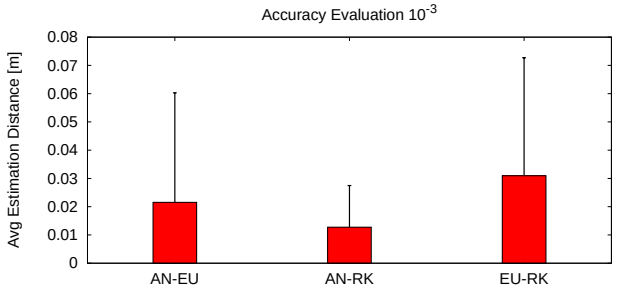

Fig. 4. Average distances and their standard deviations between our estimations and Euler's method (AN-EU), between our method and fourth order Runge-Kutta integration (AN-RK) and between Euler's method and Runge-Kutta integration (EU-RK). The numerical integrations were performed with a time step of 10^{-3} seconds.

formulas agree strongly. While Euler's method and Runge-Kutta integration predict almost exactly the same coordinates for all test cases, our method shows some deviation, which can result only from a few cases, as the average distance between our method and the numerical integrators is near zero. By examining the worst cases, we came to the conclusion that very small values of b ($|b| < 0.001$) coupled with large translational or angular velocities v and ω result in very large factors that amplify the error of the Fresnel integral approximation and lead to numerical problems (see (12) and (13)). These cases can be circumvented by setting $b = 0$. When lowering the precision of the numerical integration to a time step of 10^{-3} seconds, larger distances between the three methods can be observed, as shown in Fig. 4. Please note the difference in scaling. As the output of our equations stays the same, it must be the two numerical methods that "moved away" from the high-precision results. It can also be clearly seen that Euler's method and Runge-Kutta integration do not agree as much as they did with a time step of 10^{-6} seconds. We can conclude that the precision of our results is comparable with numerical integration with a time step of 10^{-6} seconds.

Focusing on the computational costs, the following observation can be made. While the computational costs for any incremental integrator grow linear with simulated time, the calculation time of our method stays constant. On our reference machine (Intel 2.4GHz, 32 bit) the calculation of one forward transformation takes 10^{-3} milliseconds. During this time approximately 10 Euler integration steps can be performed. Consequently, our method outperforms the Euler integration unless only tiny portions of time need to be simulated. On the same scale of precision this time limit is as small as 0.01 milliseconds. With a time step of 10^{-3} seconds, only up to 0.01 seconds of simulation time can be calculated with Euler integration faster than with our formula by sacrificing an order of magnitude in precision. We propose our method to replace time consuming numeric integrators in kinodynamic planning algorithms for wheeled robots. Especially if larger amounts of times can be integrated in one heap, tremendous speed-ups can be achieved when using our formulas.

V. Steering Method

The steering problem consists of finding a suitable path to connect two arbitrary states in the absence of obstacles. While originally stated only in configuration space [1], the extension to dynamic state space means finding an open-loop trajectory from a start state $\mathbf{s_s}$ to a target state $\mathbf{s_t}$, including the translational and angular velocities. The velocities in the start and the end states are not necessarily zero. The focus of applicability on real systems requires frequent replanning to correct errors and to cope with changes in the environment. This not only dictates a high computational performance, but also the ability to update a trajectory for a vehicle already in motion – and the velocity of a moving vehicle is, of course, not zero. Additionally, the target state could be a via point that has to be touched with non-zero velocity, or even a moving target that the vehicle is supposed to follow.

The steering problem is the inverse problem to the forward transformation $\mathbf{s_{i+1}} = T(\mathbf{s_i}, \mathbf{u})$. While the forward transformation is easy to solve, it is difficult to determine the control sequence that transfers a wheeled robot from $\mathbf{s_s}$ to $\mathbf{s_t}$. This can be regarded as a task of inverse kinematics, which is often solved with gradient descent methods. Utilizing the differentiability of T (9), we partially differentiate T with respect to the control parameters a, b, and t to obtain the Jacobian matrix

$$\mathbf{J} = \left(\frac{\partial T}{\partial a}, \frac{\partial T}{\partial b}, \frac{\partial T}{\partial t} \right). \tag{27}$$

For only one control triple, $\mathbf{J}$ is a 5×3 matrix. When a sequence of more than one control inputs is used, (26) needs to be differentiated with respect to each control parameter and the Jacobian grows to a size of 5×6, 5×9, and so on. For the sake of brevity, a complete Jacobian cannot be given here, but it can easily be obtained from (26) using an algebraic tool. The derivatives of the Fresnel integrals are given in closed form by

$$S'(x) = \sin(\frac{\pi}{2}x^2), \tag{28}$$

$$C'(x) = \cos(\frac{\pi}{2}x^2). \tag{29}$$

For the gradient descent itself, we evaluated a few inverse kinematics motivated algorithms, such as the Jacobian Transposed, the Pseudoinverse, and the Damped Least Squares algorithms, which are comprehensively described in [14]. Additionally, we also tried Resilient Propagation and the Nelder-Mead algorithm that does not require the gradient. Any method we expect to reach a precision of at least $\|T(\mathbf{s_s}, \mathbf{U}) - \mathbf{s_t}\| < 0.01$. Most of these algorithms seem to have difficulties with the strong non-linear search space and can only slowly converge towards a solution with very small steps. The Damped Least Squares method, however, clearly excels among the candidates and converges up to ten times as fast as the others. Therefore, we used the Damped Least Squares method exclusively in all experiments.

The two central determinants of solving the steering problem with this approach are the number of control triples required and what values to choose to initialize the gradient descent. We found that when using three control triples, we were able to solve each of 10K examples of start and target state situations that we randomly sampled from a $20\,m^2$ square area with orientations in $[-\pi, \pi]\,rad$, linear velocities up to $10\,m/s$ and angular velocities in $[-\pi, \pi]\,rad/s$. For initialization, we tried random values and a base of canonical controls $(\{-A, 0, A\}, \{-B, 0, B\}, 1.0)^3$, where A and B are the acceleration limits of a_i and b_i. We are using values of $A = 5.0\,m/s^2$ and $B = 5.0\,rad/s^2$. The acceleration limits were enforced during gradient descent such that the absolute values of a_i and b_i were not permitted to exceed A and B. The initial value of 1.0 second for t_i is allowed to change during descent, but each t_i is enforced to be always ≥ 0. Velocity limits, however, were not enforced. Both initialization methods worked fairly well in the sense that they successfully lead to convergence of the gradient descent after a few initialization attempts. For each of the 10K samples, we performed an exhaustive search through the 729 possible canonical controls and found that more than one can lead to convergence in a specific situation and they can result in different solutions. We recorded the control triples that yielded the minimal time solution and built a corpus of 10K (s_s, s_t) pairs and their controls U. In further experiments, we found that trying the k-nearest neighbors from the corpus for initialization lead to faster convergence of the algorithm improving both, the number of initialization attempts and the number of iterations that our gradient descent method takes to find a solution. By selectively adding new situations to the corpus in places where more than 5 neighbors had to be tried before convergence could be achieved, we observed a continuous improvement of convergence times due to less and less iterations required by the gradient descent method. This is depicted in Figure 5. Unfortunately, it is increasingly more difficult to retrieve the k-nearest neighbors from a growing, randomly scattered point cloud and eventually the saved gradient descent iteration cycles are canceled out by the k-nearest neighbor retrieval times. We stopped growing the corpus when it reached the size of 100K entries, which take approximately 10 MB space in memory. As an alternative, we also generated structured grid data sets by dividing each dimension of the search space into equal parts of three, four and five. In the structured grids, the data points may not be located as beneficial as in the selectively grown corpus, but the retrieval of the k-nearest neighbors can be performed in constant time. Figure 6 shows the result of an experiment where we compared different initialization methods with respect to the runtime of the resulting gradient descents. The methods we compared were random initialization, initialization from the canonical base, initialization from a selectively grown corpus with sizes of 20K, 50K and 100K (SEL20K, SEL50K, SEL100K), and structured data grids of 3, 4, and 5 intersections in each dimension (GRID3, GRID4, GRID5). Obviously, the gradient-descent approach is fast and robust, as even random initialization leads to steering problem solutions in less than 10 milliseconds. The data set-driven initializations significantly outperform the random and the canonical base methods. The selectively grown data set does not offer an advantage over the structured grid. Also, it appears to be sufficient to use relatively small data sets of 20K points, as larger data sets do not result in a genuine increase of performance.

Unfortunately, we are lacking a method to assess the quality of the found trajectories, but this is subject to change in the foreseeable future. For now, human judgment has to be applied. Fig. 7 shows a selection of solution trajectories.

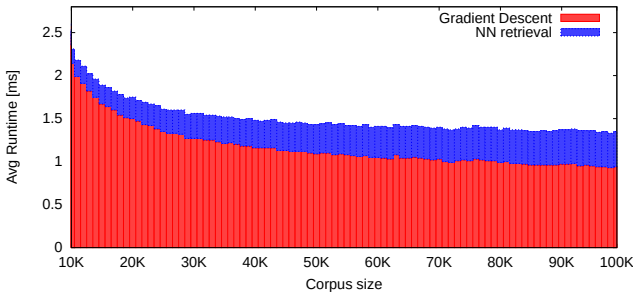

Fig. 5. Combined average runtime of the nearest neighbor retrieval and the gradient descent algorithms versus the size of the corpus. The larger the corpus, the better the gradient descent can be initialized, but the more effort it takes to retrieve the nearest neighbors.

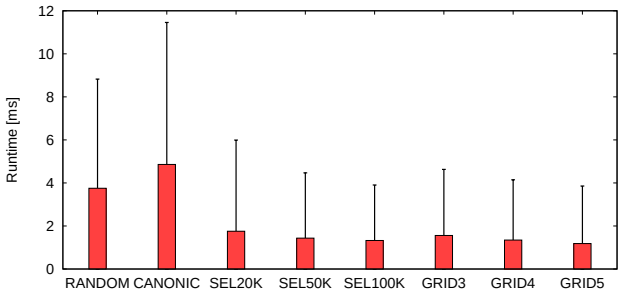

Fig. 6. Comparison of the gradient descent runtimes caused by different methods of initialization. Random initialization, initialization from a canonical base, and initialization from selectively grown data sets and grid structured data sets of varying sizes are compared.

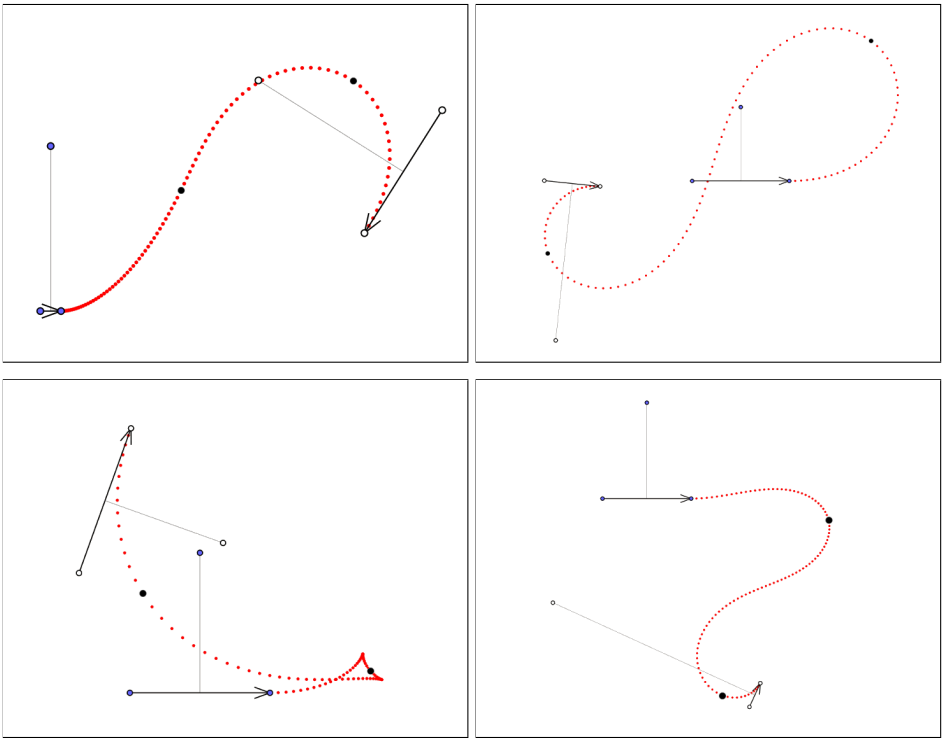

Fig. 7. A selection of trajectories generated by our steering method. The start state is marked with a blue arrow, the target state with a white arrow. The length of the arrows indicates the translational velocity and the length of the orthogonal indicates the magnitude of the angular velocity. The trajectories are sampled with a fixed frequency, so the distance between the red dots reflects the speed of the vehicle. The black dots indicate the end of a link, where the next control is applied. One of the presented trajectories (bottom left) includes cusps.

Some of the solutions shown contain cusps, where the simulated robot comes to a brief stop and continues in reverse.

VI. Conclusions

We have presented an analytical approximation to calculate future dynamic states of wheeled mobile robots given a start state and a set of controls. Our estimations are accurate and fast to calculate. Our approach aims to replace numerical integrators in kinodynamic planning algorithms. Since our formulas are differentiable, they can be utilized by a gradient descent method to generate the necessary controls to connect arbitrary points in state space. We demonstrated that this is possible by finding a solution for over 100,000 randomly generated cases. In contrast to discrete cell decomposition methods and numerical integration based algorithms, the computational time needed to find a solution for the steering problem with our method does not depend on the distance between the start state and the target state, nor on the time required to drive from one state to the other. Using a relatively small corpus of precalculated solutions to initialize gradient descent, our method can find a solution to the steering problem in less than 10 milliseconds.

In future work, we will investigate methods to evaluate the quality of the generated trajectories. Additionally, we plan to implement enforcement of velocity limits. This includes a minimal turning radius required for car-like vehicles.

References

[1] J.-P. Laumond et al., "Robot Motion Planning and Control", Springer, ISBN 3-540-76219-1.

[2] O. Khatib, Real-time obstacle avoidance for robot manipulator and mobile robots. *The International Journal of Robotics Research*, pp. 90-98, 1986

[3] M. Aicardi, G. Caslino, A. Bicchi, A. Balestrino, Closed Loop Steering of Unicycle-like Vehicles via Lyapunov Techniques, *IEEE Robotics and Automation Magazine*, 2(1):27-35, March 1995

[4] S. Thrun et. al., Stanley, the robot that won the DARPA Grand Challenge, *Journal of Field Robotics*, 2006.

[5] D. Fox, W. Burgard, S. Thrun, The Dynamic Window Approach to Collision Avoidance, *IEEE Robotics and Automation Magazine*, 4(1):23-33, 1997

[6] F. Lamiraux and J.-P. Laumond, Smooth motion planning for car-like vehicles, *IEEE Transactions on Robotics and Automation*, vol. 17, 2001, pp. 498-502.

[7] A. Scheuer, Th. Fraichard, Collision-Free and Continuous-Curvature Path Planning for Car-Like Robots, *Proc. IEEE Int Conf. on Robotics & Automation*, 1997, pp. 867-873.

[8] E. Mazer, J. M. Ahuactzin, P. Bessiere, The Ariadne's Clew Algorithm, *Journal of Artificial Intelligence*, pp. 295 - 316, 1998

[9] B. Donald, P. Xavier, J. Canny, J. Reif, Kinodynamic Motion Planning, *J. ACM*, vol. 40, 1993, pp. 1048-1066.

[10] S. M. LaValle, J. J. Kuffner Jr., Randomized Kinodynamic Planning, *I. J. Robotic Res.*, vol. 20, 2001, pp. 378-400.

[11] D. Hsu, R. Kindel, Robert, J.-C. Latombe, S. Rock, Randomized Kinodynamic Motion Planning with Moving Obstacles, *The International Journal of Robotics Research*, vol. 21, 2002, pp. 233-255.

[12] Th. Fraichard, Trajectory planning in a dynamic workspace: a 'state-time space' approach, *Advanced Robotics*, vol. 13, 1998, pp. 75-94.

[13] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, "Fresnel Integrals, Cosine and Sine Integrals.", 6.79 in *Numerical Recipes in FORTRAN: The Art of Scientific Computing*, 2nd ed. Cambridge, England: Cambridge University Press, pp. 248-252, 1992.

[14] Samuel R. Buss, Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods, 2009.