Efficient Evaluation of Collisions and Costs on Grid Maps for Autonomous Vehicle Motion Planning

Georg Tanzmeister, Martin Friedl, Dirk Wollherr, and Martin Buss

***Abstract*—Collision checking is the major computational bottleneck for many robot path and motion planning applications, such as for autonomous vehicles, particularly with grid-based environment representations. Apart from collisions, many applications benefit from incorporating costs into planning; cost functions or cost maps are a common tool. Similar to checking a single configuration for collision, evaluating its cost using a grid-based cost map also requires examining every cell under the robot footprint. This work gives theoretical and practical insights on how to efficiently check a large number of configurations for collision and cost. As part of this work, configuration space costs are formulated, which can be seen as generalization of configuration space obstacles allowing a complete configuration check incorporating the robot geometry to be done using a single lookup. Furthermore, this paper presents two efficient algorithms for their calculation: FAMOD, an approximate method based on convolution, which is independent of the size and the shape of the robot mask, and vHGW-360, an exact method based on the van Herk–Gil–Werman morphological dilation algorithm, which can be used if the robot shape is rectangular. Both algorithms were implemented and evaluated on graphics hardware to demonstrate the applicability and benefit to real-time path and motion planning systems.**

***Index Terms*—Autonomous vehicles, collision checking, configuration space costs, cost evaluation, grayscale dilation, motion planning.**

I. Introduction

A SUBSTANTIAL computational part of current path and motion planning algorithms deals with collision detection [1]–[3]. Collision detection has already been studied for decades [4], and many approaches have been proposed to solve it. However, computational performance is still an issue as collision checking is often the major bottleneck in real-world motion planning applications. In addition to checking for collisions comes the problem of determining costs of paths, as often a mere collision-free path is not sufficient, but one that exhibits low cost is required. Hence, cost calculations further increase the computational complexity. Since for real-time path planning applications, e.g., for autonomous vehicles, computation time is essential, the problem of checking for collisions and evaluating costs efficiently for a large number of configurations is addressed in this paper.

Originally coming from the robotics community, robot path and trajectory planning found their way into advanced driver assistance systems, such as obstacle avoidance systems [5] or even road course estimation methods [6]. For autonomous vehicles [7], which are essentially robots, local path and trajectory planners [8]–[11] are even more important and an essential component. Contrary to the global planner, which plans a route from a start state to a goal state based on a stored map, the local planner operates in the perceptual field of the vehicle sensors to avoid obstacles and may require that a large amount of trajectories have to be evaluated to find a safe and comfortable one. Sometimes, the stored map may be invalid, such as in road construction sites, or localization in the map may not be possible, such as in tunnels, and thus, there may not even be a local goal to plan toward, which additionally increases the planning complexity [12].

Collision detection and cost calculation depend on how the environment is represented. Collisions and costs can be evaluated against the static world, such as obstacles, and against the dynamic world, such as other vehicles. Often, a polygon-based list of objects is used to represent the dynamic environment [13], whereas the static environment is often represented using occupancy grid maps [14], [15]. Apart from different representations, static and dynamic collision tests are in their essence different. In collision avoidance with, usually few, dynamic objects, prediction is a key concern, and uncertainty in the prediction requires a defensive collision checking strategy. Hence, overapproximations are not only acceptable but even desired. On the other hand, checking against the static world typically requires a high precision, e.g., in parking scenarios, and due to the grid representation, a large number of cells have to be evaluated instead of a few polygons. This paper focuses on the problem of evaluating collisions and costs against the static environment using a grid representation.

Occupancy grid maps are popular as they are well suited to be derived from sensor data, do not require any assumption about the shape of the obstacles, and can be used to derive cost maps. Checking a single configuration for collision, when using grid maps, requires that every cell that lies within the current robot footprint is checked against the occupancy grid. Depending on the resolution of the grid, the size of the robot, and the number of configurations to be checked, the computational complexity can dramatically be reduced by precalculating configuration space obstacles, instead of checking each new configuration individually for collision. This allows a single collision check to be done by a simple point check or an array lookup. The

Manuscript received October 21, 2013; revised February 5, 2014; accepted March 11, 2014. Date of publication May 8, 2014; date of current version September 26, 2014. This work was supported by BMW Group Research and Technology. The Associate Editor for this paper was H. Jula.

G. Tanzmeister and M. Friedl are with BMW Group Research and Technology, 80992 Munich, Germany (e-mail: georg.tanzmeister@bmw.de; martin.mf.friedl@bmw.de).

D. Wollherr and M. Buss are with the Institute of Automatic Control Engineering (LSR), Technische Universität München, 80333 München, Germany (e-mail: dw@tum.de; mb@tum.de).

Digital Object Identifier 10.1109/TITS.2014.2313562

drawback is that, for noncircular robots, such as autonomous vehicles, every possible robot rotation has to be accounted for, and with every new grid, the configuration space obstacles need to be recalculated.

Instead of accounting for all robot rotations, bounding volume methods try to quickly process a large number of collisions by using shape approximations and only perform a detailed check when the check with the approximate shape remains inconclusive. In [16], only two slices of the configuration space obstacles are computed, i.e., a map dilated by a circle whose width is equal to the robot inner radius and a map dilated by a circle whose width is equal to the robot outer radius, in order to reduce the number of exact collision checks needed. While such an approach works well if the robot shape is rather uniform, it may become ineffective. Consider, for example, a corridor whose width is smaller than the outer radius, e.g., an autonomous vehicle in a narrow road construction site. In such a scenario, every check with the approximate shape remains inconclusive, and expensive detailed checks are required. In a different method, which also exploits the property of circles being rotation invariant, the robot shape is decomposed into overlapping disks [17]. The authors precompute configuration space obstacles using the disks and perform the checks at the individual disk centers. They further propose to decompose the circles into axis-aligned rectangles and use a summed area table to quickly check a rectangle for collision. Although the precomputation is reduced to a single summed area table, which can be quickly calculated, the robot-to-circle decomposition and, in turn, the circle-to-rectangle decomposition need a considerable amount of primitives to achieve high accuracy.

As aforementioned, in addition to collision checking, calculating costs is also essential to many path planning algorithms. Often, grid-based workspace cost maps are used in order to represent the cost of traversing a certain discretized area. An occupancy grid can directly serve as such a cost map, as a single cell represents the probability for it to be occupied, which can easily be mapped to cost values, so that a planner favors traversing areas that are more likely to be free.

However, there is very little literature on how costs for particular robot configurations can be efficiently extracted out of workspace cost maps, since, often, the robot is assumed to be a point [18], or the costs are already required to be defined in the configuration space [19]. However, similar to collision checking, calculating the cost of a certain configuration, given a cost map, also requires that every cell under the current robot footprint is examined, and a structure similar to configuration space obstacles, which we term configuration space costs, can be constructed to allow the cost calculation for a particular configuration to be done by a single lookup. The only work that could be found that concerns the calculation of configuration space costs is that of [20]. They presented an algorithm to compute a slice from the configuration space costs that is equal to a direct implementation of morphological grayscale dilation known from image processing [21].

This paper presents two approaches, namely, Fast Approximate MOrphological grayscale Dilation (FAMOD) and vHGW-360, to calculate the configuration space costs for a grid-based cost map on a graphics processing unit (GPU) both in real time, thus applicable for autonomous vehicles. Using the precomputed configuration space costs, cost evaluations of robot configurations, which include checking for collisions, can be done by simple lookups, potentially speeding up motion planning algorithms significantly. This paper is structured as follows. First, in Section II, the fundamentals of collision checking are given, and it is clarified how binary dilation and convolution relate to each other. This will become essential, when, later in Section III, configuration space obstacles are generalized and configuration space costs are introduced. In Section IV, FAMOD, a fast algorithm to calculate an approximation of these costs that works for arbitrary robot shapes, is proposed, and details about a parallel implementation on graphics hardware is given. Since apart from circular shapes rectangular shapes are common, Section V proposes vHGW-360, an exact method building upon the van Herk–Gil–Werman (vHGW) dilation algorithm. It can be applied when given a rectangular mask. Section VI then covers how the cells of a continuous path can be extracted, and finally, in Section VII, the algorithms are evaluated using grids coming from a vehicle equipped with a laser scanner.

II. Fundamentals

Here, the basic concepts of collision checking are quickly revisited, since they are essential for the rest of this paper. A path $\tau : [0, l] \to \mathcal{C}$ is collision free if all of its configurations q from some configuration space $\mathcal{C}$ are collision free, i.e., $\forall \lambda \in [0, l] : \tau(\lambda) \notin \mathcal{C}_{\text{obs}}$, where

$$\mathcal{C}_{\text{obs}} = \{q \in \mathcal{C} | S_q \cap B \neq \emptyset\} \tag{1}$$

denotes the configuration space obstacles [3] for the obstacles B, as given by a binary occupancy grid, and the robot region S_q, at position and orientation q, as given by a binary mask. Throughout this paper, 3-D configuration spaces are used, i.e., (x, y, θ).

For a certain fixed orientation θ of the robot S, $\mathcal{C}_{\text{obs}}$ can be calculated using the Minkowski difference $\ominus$ of the obstacle regions B and S [3], i.e.,

$$B \ominus S_\theta = \{b - s | b \in B, s \in S_\theta\} \tag{2}$$

or, equivalently, using the Minkowski sum $\oplus$, i.e.,

$$B \oplus -S_\theta = \{b + s | b \in B, s \in -S_\theta\}. \tag{3}$$

Mathematically equivalent to the Minkowski sum is morphological binary dilation known from image processing [22], [23]. Since obstacles are often represented using discretized grid structures, such as in occupancy grids, it is useful to define the calculation of a slice of $\mathcal{C}_{\text{obs}}$ using the dilation of the grid B with the robot footprint as binary mask S, i.e.,

$$B \oplus R_\pi(S_\theta) = \bigcup_{n \in B} R_\pi(S_{\theta,n}). \tag{4}$$

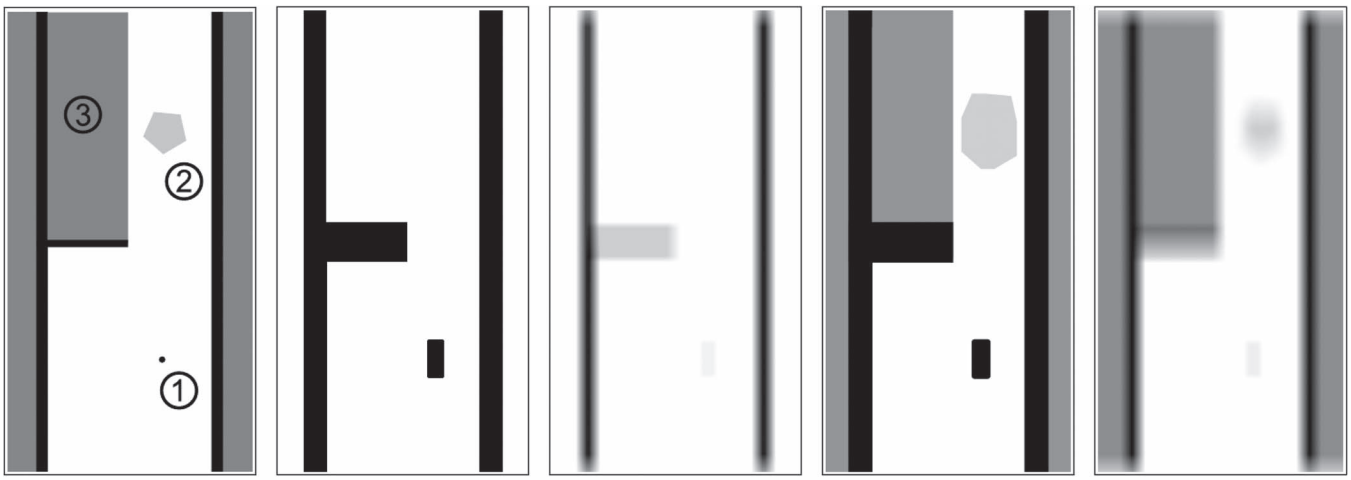


Fig. 1. Comparison between binary dilation, binary convolution, grayscale dilation, and grayscale convolution of an occupancy grid with a vertically oriented rectangular mask. Low occupancy probability is visualized by bright gray values, and high occupancy probability is visualized by dark values. Binary dilation and binary convolution were performed with a thresholded grid containing the cells where $p(x) > 0.5$. Three interesting areas can be seen: a small circular-shaped obstacle ① where $p(x) = 1$, an uncertain pentagon-shaped area ② where $p(x) = 0.25$, and an unknown area ③ with $p(x) = 0.5$. The convolutions are visualized by linearly mapping the values back to the original range. (a) Occupancy grid. (b) Binary dilation. (c) Binary convolution. (d) Grayscale dilation. (e) Grayscale convolution.

The rotation R_π of the binary mask by 180° around the mask origin is, in the 2-D case, equivalent with inverting every set element as in (3), which can be easily verified as

$$R_\pi(s) = \begin{pmatrix} \cos\pi & -\sin\pi \\ \sin\pi & \cos\pi \end{pmatrix} s = -s. \tag{5}$$

It is, however, only required for nonsymmetrical shapes. Without loss of generality, symmetrical shapes will be assumed in the following to simplify notation, and R_π will be dropped.

In binary dilation, the obstacle region is expanded by placing the origin of S at every obstacle cell n of B. The mask S is also known as structuring element, kernel, or footprint; and the terms are interchangeably used in this paper.

The common notation of configuration space obstacles from (1) can be now written with the use of binary dilation, i.e.,

$$\mathcal{C}_{\text{obs}} = \bigcup_\theta B \oplus S_\theta. \tag{6}$$

The process of calculating $\mathcal{C}_{\text{obs}}$ is consistently denoted by convolution throughout the literature, e.g., [16] and [24]–[27], as these operations are similar. In contrast to the union operation used in binary dilation, the 1-D discrete convolution of two functions f_1 and f_2, i.e.,

$$(f_1 * f_2)(n) = \sum_{i \in D} f_1(i) f_2(n-i) \tag{7}$$

is, however, given by their weighted average.

In Fig. 1, the two operations are compared side by side. An occupancy grid, as shown in Fig. 1(a), is first thresholded to include values >0.5. Then, binary dilation, as shown in Fig. 1(b), and convolution, as shown in Fig. 1(c), are performed using the same binary mask, which is a vertically oriented rectangular shape. Binary dilation correctly produces a collision map that is independent of the number of individual cell collisions under the footprint.

However, the result of the convolution can easily be mapped to equal the result of binary dilation by applying a simple function h, i.e.,

$$B \oplus S = h(B * S), \text{with} \tag{8}$$

$$h(n) = \begin{cases} 0 & n = 0 \\ 1 & n > 0. \end{cases} \tag{9}$$

In this paper, the convention is used that functions that take whole grids as input, as in (8), process them by applying a function individually on each cell of the grid, as in (9). For simplicity, the same name is used.

Calculating the configuration space obstacles by convolution was already shown in [28]. Its great benefit comes from the convolution theorem reducing the convolution to a single multiplication in frequency space, thus making the calculation independent of the size and the shape of the mask [24]. Although differentiating between dilation and convolution may appear merely of formal nature without practical relevance, it will become crucial in the more general case of costs, as will be shown in the following section.

III. From Collisions to Costs

When given a grid-based cost map, calculating the cost of a particular robot configuration is similar to checking it for collision, since it also requires examining all the cells from the cost map that fall under the current robot footprint and reduce them to a single cost value. In this work, we use cost maps that are defined in the workspace or the world $\mathcal{W}$ of the robot. In this work, $\mathcal{W} = \mathbb{R}^2$. It is assumed that $\mathcal{W}$ is discretized into grid cells. The cost map

$$m_c : \mathcal{W} \to \mathbb{R}_0^+ \tag{10}$$

holds, for each cell of a grid map, a cost value. Workspace cost maps represent the costs of traversing a certain area in the world

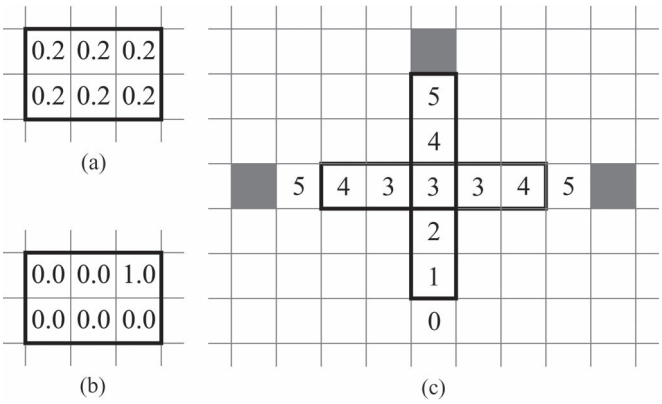


Fig. 2. Comparison of cost reduce operators on cells under the robot footprint. (a) $\sum c_i = 1.2$, $\max(c_i) = 0.2$. (b) $\sum c_i = 1$, $\max(c_i) = 1$. (c) Vertical orientation: $\sum c_i = 15$, $\max(c_i) = 5$. Horizontal orientation: $\sum c_i = 17$, $\max(c_i) = 4$.

and may be, e.g., occupancy grids directly, or other costs such as potential fields applied on a binary obstacle grid to represent clearance to obstacles.

Although sometimes the sum is used to reduce the cells under the footprint to single configuration cost, e.g., in [29], the maximum operator is used in this work. It is the aim of this section to motivate the choice. Note that configuration costs extracted out of a cost map are not to be confused with the overall cost of a path, where different choices are possible and the sum of individual configuration costs is in fact the more reliable criterion than the maximum or the average cost [19]. In fact, there exists a variety of different cost measures. The cost may also represent the path length or the path smoothness. However, such costs are intrinsic to the path and do not depend on the geometrical footprint of the robot, and they can well be added to the costs from the workspace cost map.

Fig. 2 shows the difference between the maximum and the sum on the cost values. First, consider the case of cost values in the range of [0, 1]. When using the sum, Fig. 2(a) has a higher resulting cost than Fig. 2(b), although, intuitively, it does not make sense that traversing a *cheap* area, e.g., one that is likely to be free from obstacles, is more expensive than traversing an area that contains the maximum cost value, e.g., corresponding to a certain collision. This example should not suggest, however, that overall collision probabilities are computed, but the aim of this work is the more general case of arbitrary costs, albeit it could be used as approximation. The second example shows costs that correspond to the inverse distance transform on an obstacle map, i.e., clearance is preferred over proximity to obstacles. The cost values are in the range of [0, 5], and the distance metric corresponds to the Manhattan distance. The vertically oriented shape in Fig. 2(c) has a lower sum value than the horizontally oriented one, although it is closer to an obstacle. Hence, the sum is not the appropriate operator.

In contrast to the set-based definition of $\mathcal{C}_{\text{obs}}$ as a subset of $\mathcal{C}$, a functional of all configurations $q \in \mathcal{C}$ is used. Let M denote a grayscale map and S_q the binary robot footprint S at q; the configuration space costs

$$\mathcal{C}_{\text{costs}} : \begin{cases} \mathcal{C} & \to \mathbb{R}_0^+ \\ q & \mapsto (M \oplus S_q) \end{cases} \tag{11}$$

are defined with the use of grayscale dilation

$$M \oplus S_{\theta,n} = \max\left(M(n-x) + S_{\theta,n}(x) | x \in S_{\theta,n}\right) \tag{12}$$

with a flat structuring element. A flat structuring element is a mask that only holds values of 0. Grayscale dilation with a flat structuring element can be therefore more compactly written, i.e.,

$$M \oplus S_{\theta,n} = \max_{x \in S_{\theta,n}} M(n-x). \tag{13}$$

Morphological grayscale dilation is a generalization of binary dilation, and its use for the calculation of $\mathcal{C}_{\text{costs}}$ comes naturally in that costs are interpreted as a generalization of collisions by simply associating the highest cost value to represent collisions. Note that binary dilation can be also represented with (13) and, thus, also

$$\mathcal{C}_{\text{obs}} : \begin{cases} \mathcal{C} & \to \mathbb{B} \subset \mathbb{R} \\ q & \mapsto (B \oplus S_q) \end{cases} \tag{14}$$

where $\mathbb{B}$ denotes the space of binary numbers.

Fig. 1(d) shows the result of grayscale dilation of an occupancy grid with a binary rectangular mask. The advantages of $\mathcal{C}_{\text{costs}}$ over $\mathcal{C}_{\text{obs}}$ are manifold. Apart from the traversal cost, it is often difficult to define a collision threshold, and although this problem remains when using costs, it is mitigated by the possibility of raising the threshold to areas that are more likely to be occupied, as the occupancy probability information remains represented by the cost. Another benefit is that the information about the unknown areas is not lost and can be thus used in the design of the planner.

From the discussion about which operator should be used for configuration space costs, it is straightforward to see that, in contrast to the binary case, the differentiation between dilation and convolution is crucial. Due to the sum operator, it is not possible to differentiate between few cells with high cost values or many cells with low-to-medium values. Similar to the example shown in Fig. 2, it can be observed that, with grayscale convolution, as shown in Fig. 1(e), the small circular obstacle leads to a collision since $p(x) = 1$ but wrongly results in lower values, visualized by brighter values, than the uncertain pentagon-shaped area, where $p(x) = 0.25$, or the unknown area, where $p(x) = 0.5$. Fig. 3 shows a visualization of the configuration space obstacles versus the configuration space costs for different orientations of a rectangular robot. After having motivated the choice of the maximum operator for reducing workspace cost values to the configuration cost and the definition of $\mathcal{C}_{\text{costs}}$, a fast algorithm for their computation using an approximation of grayscale dilation is presented next.

IV. Fast Approximate Grayscale Dilation for Arbitrary Robot Footprints (FAMOD)

Here, it is demonstrated how grayscale dilation and, thus, the configuration space costs can be calculated by using convolution and how the proposed algorithm is implemented on graphics hardware. By using the convolution theorem, the complexity in frequency space reduces to a single multiplication and is thus independent of the size of the mask and independent

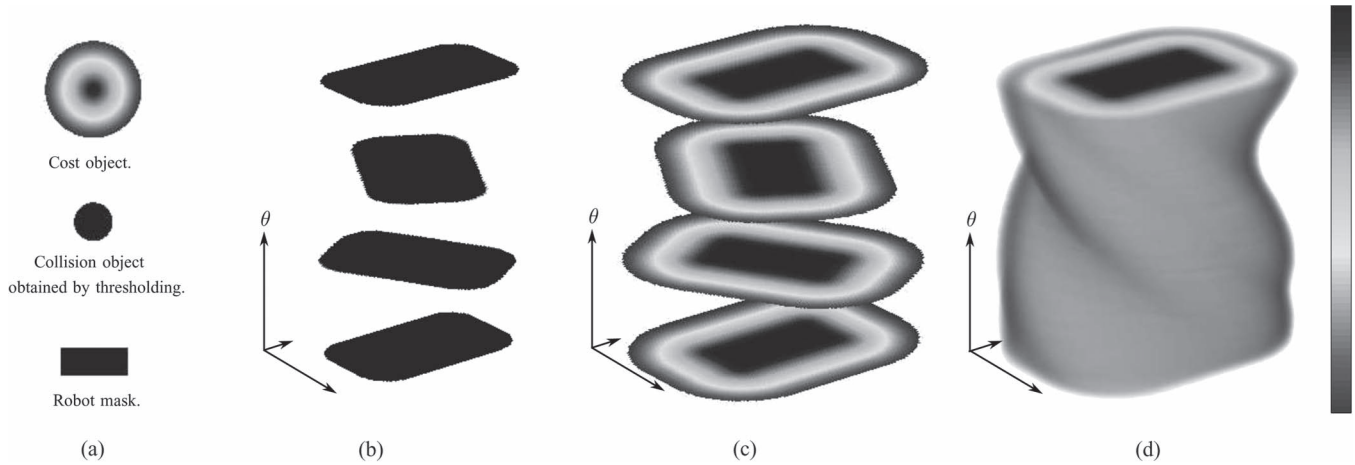


Fig. 3. Visualization of the configuration space obstacles and the configuration space costs of a circular cost object and a rectangular robot footprint. The cost object is expanded with the robot mask rotated at different orientations θ. (a) Scene elements. (b) Slices at 0°, 60°, 120°, and 180° of robot orientation of $\mathcal{C}_{\mathrm{obs}}$. (c) Slices at 0°, 60°, 120°, and 180° of robot orientation of $\mathcal{C}_{\mathrm{costs}}$. (d) Complete configuration space costs.

of the shape of the robot. In particular, the last property is of great advantage, since for certain common shapes, such as axis-aligned rectangles, efficient algorithms already exist, as will be shown in the next section.

A. Calculating Grayscale Dilation by Convolution

Morphological dilation belongs to the class of nonlinear image filters, which are filters that cannot be represented by a pure convolution [21]. This is due to the $\max$ operator, which is a nonlinear operator and therefore cannot be represented as sum in general. However, motivated by (8) from Section II, it is shown how, with the use of a mapping function, grayscale dilation can be also expressed by the convolution operator.

Let k denote the number of set bits, i.e., the number 1s of the binary mask $S = \{0,1\}^*$ that represents the robot footprint; then the lowest and highest possible values of the output of the convolution of an arbitrary binary grid with S are

$$\begin{aligned} \max(B * S) &= k \\ \min(B * S) &= 0 \end{aligned} \tag{15}$$

if the binary values of B are $\{0,1\}$. Consider now a binary grid $B^{\{u,v\}}$ with two arbitrary values u and v, where $u < v$; then the maximum and the minimum result in

$$\begin{aligned} \max\left(B^{\{u,v\}} * S\right) &= kv \\ \min\left(B^{\{u,v\}} * S\right) &= ku. \end{aligned} \tag{16}$$

This observation is used to derive a mapping so that the maximum cost value can be reextracted from the sum. Let the cost map exhibit r different cost values $c_0, \ldots, c_{r-1}$, with $c_0 < \cdots < c_{r-1}$, and ε be a small positive number. Then, by mapping the cost values with the recurrence relation

$$\begin{aligned} g(c_0) &= c_0 \\ g(c_i) &= kg(c_{i-1}) + \varepsilon \end{aligned} \tag{17}$$

and by remapping the convolution result using

$$h(n) = \begin{cases} c_0 & g(c_0) \le n < g(c_1) \\ \vdots & \\ c_{r-2} & g(c_{r-2}) \le n < g(c_{r-1}) \\ c_{r-1} & n > g(c_{r-1}) \end{cases} \tag{18}$$

grayscale dilation of an arbitrary grayscale map M with a flat structuring element S, i.e.,

$$M \oplus S = h\left(g(M) * S\right) \tag{19}$$

is still achieved by convolution. Next, practical considerations and implementation details are examined.

B. Approximating Exact Grayscale Dilation

For a practical implementation, it has to be considered that the resulting values of the mapping function g from (17) exponentially grow, and the number of individual cost values r is therefore limited. Hence, depending on the data types used and on the range of the original values, it might be necessary to reduce the quantization resolution. This can be done either uniformly in order to best approximate an exact grayscale dilation with a random cost map or nonuniformly with the use of additional information to achieve a better fit.

For cost maps from occupancy grids, this limitation is, however, not as restricting as it may appear, and a nonuniform mapping leads to good approximations even with a low quantization. Occupancy values, if they are not already converged toward 0 or 1, are usually volatile and change with every cell update, thus leading to slightly different path costs in every time frame. Hence, a coarser discretization is acceptable if not even desirable due to the resulting paths being less affected by this noise. In addition, with cost maps from occupancy grids, one is primarily interested in differentiating between cells that are free with a high probability, cells that are occupied with a high probability, and cells where the measurements do not lead to a clear result or have not been observed yet. Hence, with three or four classes, depending on whether yet unobserved

cells ought to be discriminated from unsure cells, a cost map from an occupancy grid can be approximated well, as will be shown in Section VII.

C. GPU Implementation Details

Since general-purpose programming on the GPU and toolkits such as NVIDIA CUDA [30] have become available, the use of graphics hardware for non-graphics-related problems has become increasingly popular. Today's graphics cards are highly optimized for massively parallel computation-intensive operations, and their use, depending on how well a specific algorithm fits into the single-instruction–multiple-data paradigm, can lead to a significant speedup over CPU implementations. Grayscale dilation and convolution are highly parallelizable since for every cell the same operations are individually and independently performed.

Algorithm 1 describes the basic outline of our FAMOD. CUDA already provides a fast Fourier transform library, i.e., cuFFT [31], which was used in the implementation. All functions are implemented as individual kernels and are parallelized over the grid cells. Regarding performance, the functions g and h from (17) and (18) are not defined in a beneficial way since they require to place loops and conditionals inside CUDA kernels. Assuming that $c_0 = 0$, the recurrence relation from (17) is solved, however, by deriving the generating function, which is the geometric series

$$\begin{aligned} g(c_i) &= kg(c_{i-1}) + \varepsilon \\ &= k^{i-1}\varepsilon + k^{i-2}\varepsilon + \cdots + k\varepsilon + \varepsilon \\ &= \varepsilon \sum_{j=0}^{i-1} k^j = \varepsilon \frac{1-k^i}{1-k}. \end{aligned} \tag{20}$$

The result of the convolution, i.e., n, is mapped back without the use of conditional statements by solving for i, i.e.,

$$\begin{aligned} i &= \left\lfloor \log_k \left[1 - \frac{1-k}{\varepsilon} n \right] \right\rfloor \\ h(n) &= c_i \end{aligned} \tag{21}$$

and can replace (18).

If the robot shape is rectangular, symmetry can be used to reduce the computational complexity, which is shown in the following section.

Algorithm 1 FAMOD

Input: Map M, z kernels $K[]$
Output: Configuration Space Costs $C[]$
1: // *offline*
2: **for** $i \leftarrow 0$ **to** $z-1$ **do**
3: $K_{\mathrm{FFT}}[i] = \mathrm{FFT}(K[i])$
4: **end for**
5: // *online*
6: $T_1 \leftarrow g(M)$
7: $T_2 \leftarrow \mathrm{FFT}(T_1)$
8: **for** $i \leftarrow 0$ **to** $z-1$ **do**
9: $T_3 \leftarrow \mathrm{modulate}(T_2, K_{\mathrm{FFT}}[i])$
10: $T_1 \leftarrow \mathrm{iFFT}(T_3)$
11: $C[i] \leftarrow h(T_1)$
12: **end for**

V. Efficient Exact Grayscale Dilation for Rectangular Robot Footprints (vHGW-360)

Apart from rotation-invariant circular forms, many robots can be well approximated by rectangular shapes, such as wheeled robots such as autonomous vehicles. Here, it is shown how this property is used to efficiently calculate the configuration space costs.

A. Exploiting Symmetry

The symmetry of a rectangular form can be exploited in multiple ways to reduce computations. First, if the anchor of the structuring element is in the center, the dilation of a mask rotated by an angle α is equivalent to the dilation of a mask rotated by $\alpha + 180$, and thus, the number of dilations is reduced to half. If the mask is quadratic, the number of dilations can even be reduced to a quarter.

Second, similar to convolution, dilation is also associative. Hence, for a structuring element S and a map M, if

$$S = S_1 \oplus S_2, \text{then} \tag{22}$$

$$M \oplus S = (M \oplus S_1) \oplus S_2 \tag{23}$$

and thus, the dilation with a 2-D mask is reduced to two dilations with 1-D masks, reducing the number of per-cell operations from wh to $w + h$, if w and h are the width and the height of the structuring element. This property is often used in implementations when dealing with large masks [32]. The decomposition of a 2-D dilation into two 1-D dilations leads, however, to errors when the rectangular mask is nonaxis aligned, as shown in Fig. 4. Certain cells are not included in the dilation, but are in the original rotated mask. Hence, it is not recommended to precompute the rotated masks and to apply them on the grids in the online phase, but the grid itself needs to be rotated, i.e.,

$$M \oplus R_\alpha(S) = R_\alpha \left(R_{-\alpha}(M) \oplus S \right) \tag{24}$$

where R_α denotes a rotation of α degrees.

Looking at (24), it can be seen that every dilation would require two grid rotations. However, the rectangular form is again used to reduce this number. The dilations of rotation α and $\alpha + 90$ are similar with only the width and the height exchanged, and thus, two dilations are computed with one pair of rotations. Although a significant number of grid rotations are needed in order to use structural element decomposition, it is still beneficial as specialized algorithms exist for the dilation with horizontal and vertical structuring elements, which is shown next.

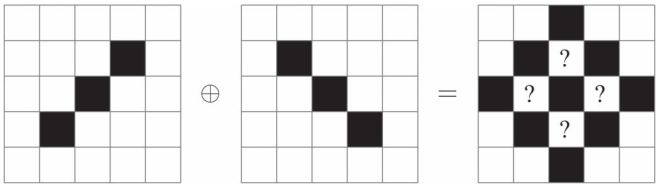


Fig. 4. Structural element decomposition of a 45° rotated quadratic shape of size $3\sqrt{2}$.

B. Speeding Up One-Dimensional Grayscale Dilation

Dilation with an axis-aligned 1-D structuring element is a special case of general grayscale dilation, and multiple algorithms have been developed for its efficient computation, e.g., [33]–[35]. One of the most often implemented algorithms is vHGW [36], [37]. It was shown to perform well on GPUs [38], [39] and is therefore the method of choice in this work.

vHGW is independent of the size of the structuring element and was shown to require $3-(4/w)$ comparisons per pixel, where w is the width of the structuring element and needs to be uneven in the number of cells. This is achieved by partitioning the image into nonoverlapping blocks of size w for which the result is independently calculated. For every block, a larger window of size $2w-1$ is centered around the block, which is used to calculate two cumulative max arrays L and R. Let j denote the center pixel of a particular block of a linearized image or map M; then, $\forall i = 1 : w-1$

$$\begin{aligned} L[i] &= \max\left(L[i-1], M[j-i]\right) \text{ and} \\ R[i] &= \max\left(R[i-1], M[j+i]\right) \text{ with} \\ L[0] &= R[0] = M[j]. \end{aligned} \tag{25}$$

With the use of L and R, the resulting output O is

$$O[k+i] = \max\left(L[w-i], R[i]\right) \tag{26}$$

$\forall i = 0 : w-1$, with k being the starting index of the block. The outline of the entire algorithm is given in the next section.

C. GPU Implementation Details

The implementation of the 1-D vHGW dilation is based upon [39], since the authors showed good performance. The complete algorithm for computing $\mathcal{C}_{\text{costs}}$ is depicted in Algorithm 2. For an efficient GPU algorithm, it is important to consider how the grid resides in memory and how it is accessed, since global memory accesses should be coalesced [40]. Hence, one of the dilations, either the horizontal or the vertical, will be substantially slower. The authors proposed to apply an optimized transposed before and after applying the dilation and to dilate only in the faster direction. In this work, the same strategy is applied, but rotations are used instead of transposes since, by considering their order, it is possible to reduce the total number that is needed. In addition, it was experimentally found that `nppiRotate*`, which is used in the implementation, already performs well in terms of computational time. After the presentation of the two methods for the computation of $\mathcal{C}_{\text{costs}}$, the next section shows how to determine the corresponding cells of a trajectory that need to be checked to determine its cost.

Algorithm 2 vHGW-360

Input: Map M, z kernels $K[]$, robot length l and width w
Output: Configuration Space Costs $C[]$
for $i \leftarrow 0$ **to** $z/2-1$ **do**
  $T_1 \leftarrow \text{rotate}(M, (180i/z))$
  $C[i] \leftarrow \text{vHGW}(T_1, l)$
  $C[(z/2)+i] \leftarrow \text{vHGW}(T_1, w)$
end for
for $i \leftarrow 0$ **to** $z-1$ **do**
  $T_1 \leftarrow \text{rotate}(C[i], 90)$
  if $i < z/2$ **then**
    $T_2 \leftarrow \text{vHGW}(T_1, w)$
    $C[i] \leftarrow \text{rotate}(T_2, -(180i/z)-90)$
  else
    $T_2 \leftarrow \text{vHGW}(T_1, l)$
    $C[i] \leftarrow \text{rotate}(T_2, -(180i/z))$
  end if
end for

VI. Checking Continuous Paths in Discrete Grids

In the previous sections, the problem of determining the cost, which includes the check for collision, of individual configurations is studied in detail. This section looks into how the cells of an entire path or a path segment can be determined in order to calculate the path cost.

A. Calculating Path Costs

As mentioned earlier, given a continuous path or trajectory $\tau \in T$, assuring that a path is collision free requires determining that all of its configurations are collision free, i.e., $\tau : [0, l] \rightarrow \mathcal{C}_{\text{free}} \subseteq \mathcal{C}$. Similarly, calculating the path cost τ_{cost} requires examining the individual costs of all of its continuous configurations. When using the integral as a cost criterion, the path cost can be calculated by

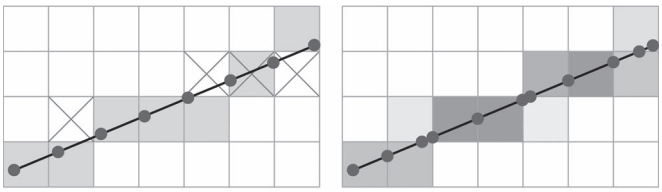
$$\tau_{\text{cost}} : \begin{cases} T \rightarrow \mathbb{R}_0^+ \\ \tau \mapsto \int_0^l \mathcal{C}_{\text{costs}}\left(\tau(s)\right) ds. \end{cases} \tag{27}$$

Practical algorithms need to apply a discrete approximation of the configurations that are checked, and hence, they need to determine discrete values of s to calculate the cost of τ, i.e.,

$$\tau_{\text{cost}} : \begin{cases} T \rightarrow \mathbb{R}_0^+ \\ \tau \mapsto \sum_{k=0}^{n} \mathcal{C}_{\text{costs}}\left(\delta\left(\tau(s_k)\right)\right)\left(s_{k+1} - s_k\right). \end{cases} \tag{28}$$

It is assumed that τ is parametrized by the arc length and that $\delta(q)$ discretizes the configuration q to its corresponding cell in $\mathcal{C}_{\text{costs}}$.

When using grid-based cost maps, the discretization is already inherent in the representation of the map and in the angular resolution of the configuration space costs. Therefore, it



(a) Regular sampling. (b) Weighted intersection sampling.

Fig. 5. Regular sampling leads to missing cells or duplicates marked with X, whereas weighted intersection sampling yields the desired result. (a) Regular sampling. (b) Weighted intersection sampling.

is the natural choice to adopt it. The simplest way of extracting the configurations that ought to be checked is probably by using equidistant sampling. Equidistant sampling is, however, problematic since, even with a high sampling rate, there exists the possibility of missing cells, which is particularly problematic for collisions. In addition, if the sampling rate is even higher, many continuous configurations will fall into the same grid cell causing unnecessary overhead, as shown in Fig. 5(a). The set P of parameters s_k should be such that

$$\forall s \in [0,l] \exists s_k \in P : \delta\left(\tau(s)\right) = \delta\left(\tau(s_k)\right) \wedge \\ \forall s_k, s_j \in P : \delta\left(\tau(s_k)\right) \neq \delta\left(\tau(s_j)\right). \tag{29}$$

Hence, the problem consists of determining all cell positions traversed by the path or the trajectory with the minimum amount of samples needed and in calculating the lengths of the segments in order to weight the cell costs. A solution to this problem is proposed in the following section.

B. Determining Cell Positions for Motion Primitives

In the 2-D case, the problem of determining the cells traversed by a motion primitive is strongly related to the rasterization problem known from the field of computer graphics. It is the process of determining the pixels to color given a continuous representation, e.g., a line or a curve. Several algorithms exist that deal with graphics primitive rasterization (see [41]). When it comes to rasterizing lines, Bresenham's line drawing algorithm is among the most famous, since it is fast and only requires integer operations. It was used in [42] with a slight modification to determine the cell positions for checking collisions on a 2-D grid in Theta* [43].

To incorporate the robot geometry, the algorithm needs to be extended to higher dimensions. In addition, many planners plan in the continuous domain, and hence, integer-only algorithms are not applicable. This is also the reason why it is generally not possible or feasible to precompute the discrete cells traversed by a motion primitive, as the continuous start state influences the traversed cells and their weights. Again, insights from a related field, i.e., volume visualization, can be used. In volume visualization, an essential part is to generate 2-D images from a 3-D volume, which is often done by using ray casting to step through the volume. Similar to the problem of determining the grid cells to check, it is also often desirable not to miss any cell. This is done by sampling the ray at every cell intersection, as shown in Fig. 5(b).

To accomplish this, the parametric equation of a line $\boldsymbol{x}(t) = \boldsymbol{p} + t\boldsymbol{v}$ is separately solved for t, i.e.,

$$t_x(x) = \frac{1}{v_x}x - \frac{p_x}{v_x} \\ t_y(y) = \frac{1}{v_y}y - \frac{p_y}{v_y} \\ t_\theta(\theta) = \frac{1}{v_\theta}\theta - \frac{p_\theta}{v_\theta}. \tag{30}$$

Then, the cell intersections of the next cells are inserted for (x, y, θ), and $\min(t_x, t_y, t_\theta)$ determines the next cell to check in a 3-D configuration space, since it gives the dimension where the first intersection occurs. Moving to the next cell and repeating this test yields all cells traversed by the ray without checking a single cell twice.

Each cell cost extracted from the configuration space costs needs to be weighted by the length of the path segment that traverses the particular cell, which is also shown in Fig. 5(b) and given by the weight $s_{k+1} - s_k$ in (28). If $\|\boldsymbol{v}\| = 1$, it can be directly obtained as $t_{\min}$ without additional calculations.

The incompatibility between angular and Euclidean units is problematic for computation of the arc length, and the robot displacement metric that is sometimes used to overcome that problem might be difficult to calculate, depending on the shape of the robot [3]. Although, for nonholonomic robots such as autonomous vehicles, it is possible to completely discard the orientation θ from the length, since it is not possible to move in the θ direction without moving in the x- and/or y-direction. This makes the discretization of angles used in the configuration space costs independent from the cost map discretization. It is achieved by normalizing $\boldsymbol{v}$ such that $\sqrt{v_x^2 + v_y^2} = 1$. Algorithm 3 shows the complete process.

Often, line segments are not sufficient, and more complex motion primitives are used. Depending on the primitive, it might be feasible to follow the same strategy as shown above, i.e., use the arc length parametrization, and solve for the parameter. Otherwise, it is still possible to sample the primitive function at equidistant parameters and to connect the samples with line segments. In the next section, performance results on the calculation of $\mathcal{C}_{\text{costs}}$ are given.

Algorithm 3 Check Cells Between q_0 and q_1

Require: $\Delta x, \Delta y, \Delta\theta \neq 0$
$(x, y, \theta) \leftarrow \text{round}(x_0, y_0, \theta_0)$
$(x_{\text{end}}, y_{\text{end}}, \theta_{\text{end}}) \leftarrow \text{round}(x_1, y_1, \theta_1)$
$l \leftarrow \sqrt{(\Delta x)^2 + (\Delta y)^2}$
$(v_x, v_y, v_\theta) \leftarrow (1/l)(\Delta x, \Delta y, \Delta\theta)$
$(s_x, s_y, s_\theta) \leftarrow (1/2)(\text{sgn}\Delta x, \text{sgn}\Delta y, \text{sgn}\Delta\theta)$
$(t_x, t_y, t_\theta, t_{\min}) \leftarrow (-1, -1, -1, 0)$
while $(x, y, \theta) \neq (x_{\text{end}}, y_{\text{end}}, \theta_{\text{end}})$ **do**
  if $t_{\min} = t_x$ **then**
    $x \leftarrow x + 2s_x$
  end if
  if $t_{\min} = t_y$ **then**
    $y \leftarrow y + 2s_y$
  end if

14: **if** $t_{\min} = t_\theta$ **then**
15: $\theta \leftarrow \theta + 2s_\theta$
16: **end if**
17: $t_x \leftarrow (x + s_x)/v_x - x_0/v_x$
18: $t_y \leftarrow (y + s_y)/v_y - y_0/v_y$
19: $t_\theta \leftarrow (\theta + s_\theta)/v_\theta - \theta_0/v_\theta$
20: $t_{\text{prev}} \leftarrow t_{\min}$
21: $t_{\min} \leftarrow \min(t_x, t_y, t_\theta)$
22: Check cell (x, y, θ) and weight by $t_{\min} - t_{\text{prev}}$
23: **end while**

VII. Results and Discussion

This section shows the performance and the accuracy of the proposed algorithms for the computation of the configuration space costs. The FAMOD algorithm presented in Section IV can handle arbitrary robot footprints but is an approximate algorithm, whereas the vHGW-360 presented in Section V produces exact results but can only handle rectangular masks. Exact and approximate in this context relates to the computed output for every cell, and it is noted that both algorithms are approximate in the workspace domain as they use grid structures as input.

A. Performance Evaluation

The hardware used throughout the tests consists of a 3.5-GHz Intel Core i7-3770 and an NVIDIA GeForce GTX 660 Ti graphics card. The programming language was C++ compiled with Microsoft Visual Studio 2010, and the graphics card was programmed using NVIDIA CUDA 4.2. The CPU implementation was done without parallelization. All the timing measurements of the GPU include data copying of the result from graphics memory back to host memory. In all implementations, except for FAMOD, where data were provided as float arrays, 8-bit unsigned character arrays were used.

The evaluation only targets the performance and the accuracy of the calculation of $\mathcal{C}_{\text{costs}}$ and does not compare different planning algorithms with its integration. The methods can be integrated in any planner that is able to use a grid-based representation of the obstacles, such as the A* family, lattice planner [29], or sampling-based motion planner such as Rapidly Exploring Random Trees [44]. Since after the computation of $\mathcal{C}_{\text{costs}}$, collision checking and cost evaluation can be done in O(1), there is no benefit in comparing them for this work. However, since some planning algorithms might be efficient in reducing the number of configurations that have to be checked, a comparison between direct configuration checking on the CPU versus the calculation of $\mathcal{C}_{\text{costs}}$ plus the checks are given.

Fig. 6 shows the processing time for a single grayscale dilation with a non-axis-aligned quadratic mask of different sizes using different algorithms. FAMOD and vHGW-360 are compared with NVIDIA's GPU dilation function from the NPP library `nppiDilate_8u_C1R` [45], to MATLAB's `imdilate`, whose structuring element was generated with `strel('arbitrary',...)`, and to a naive CPU implementation. Overall, FAMOD possesses the lowest computing time. It can be observed that the processing time of FAMOD stays constant as expected, whereas the vHGW-360 algorithm, which has constant theoretical complexity per pixel, slightly, and for most cases negligibly, increases with the structuring element size, which is consistent with LTU-CUDA [46]. NVIDIA's `nppiDilate_8u_C1R` seems to implement the naive algorithm on the graphics card and thus shows a strong nonlinear increase with the width of the mask. It is still substantially faster than its CPU versions, i.e., MATLAB, which parallelizes over the available CPU cores and threads, and our serial naive C++ CPU implementation.

In Fig. 7, the total computing time for a varying number of consecutive grayscale dilations, representing different angular resolutions of the configuration space costs, with a 25 $\times$ 11 mask is shown. The mask represents a vehicle of size 5 m $\times$ 2 m at a cell resolution of 0.2 m. Since vHGW-360 requires uneven structuring element sizes, the width was increased by one cell. The number of rotations is equal to the number of dilations, i.e., no symmetry-based reduction was applied. Compared with the performance of a single dilation, when multiple dilations on the same map are computed, in FAMOD, $g(M)$ and its Fourier transform have to be calculated only once for all rotations, as shown in Algorithm 1, and thus, the computational benefits are further increased.

Finally, Fig. 8 demonstrates the total computing time of a varying number of cost checks. It compares a direct CPU implementation without any preprocessing with the time it takes to first compute the FAMOD, copy the result from graphics to host memory, and then perform the checks. Different angular discretizations are shown. In Table I, exact timings for the precomputation t_{ms} and the number of cost checks n_c needed to outperform the direct CPU implementation are shown. FAMOD is the fastest of all tested algorithms. For a rectangular mask, even a 1° resolution, i.e., 180 dilations, by exploiting the symmetry, which will be unnecessarily high in most cases, can be calculated with over 30 Hz. In our applications, a 5° resolution is used [6], [12]. With this angular quantization and a rectangular vehicle shape, it only requires 6000 cost/collision checks to compensate the precomputation time. If a path length of 500 cells is assumed with disjunctive configurations (x, y, θ), this corresponds to the check of as little as 12 paths and shows the potential benefit in a large number of path planning problems. In addition, the table shows the computation time for $\mathcal{C}_{\text{obs}}$ as binary dilation using convolution. The performance difference is under 5%. Break-even points are omitted since a direct CPU implementation would skip evaluating the cells under the mask for a particular configuration after the first collision is detected, and the numbers are thus not comparable.

B. Accuracy of FAMOD

Since a practical FAMOD implementation needs to reduce the number of cost values, for which it is therefore termed approximate grayscale dilation, its accuracy is evaluated here. The algorithm was compared with an exact grayscale dilation on sequences of maps that were built and recorded online in real time during exploration with a vehicle that was equipped with a laser scanner. Two scenarios were used for the evaluation: a parking garage, which represents a typical closed-corridor

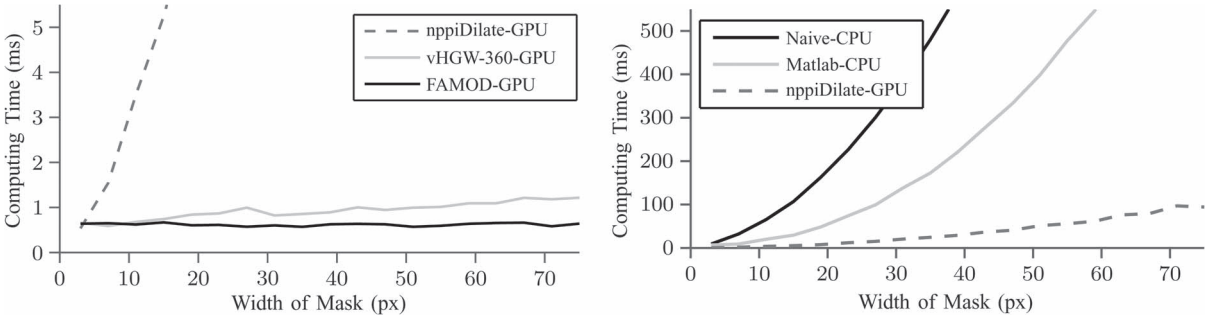


Fig. 6. Required computing time for a single grayscale dilation of a 512 × 512 grid with a 30° rotated quadratic structuring element of varying sizes. The following are shown (from slowest to fastest): a naive serial CPU implementation, MATLAB's imdilate, NVIDIA's nppiDilate, vHGW-360, and FAMOD. NVIDIA's nppiDilate is shown in both plots for comparison.

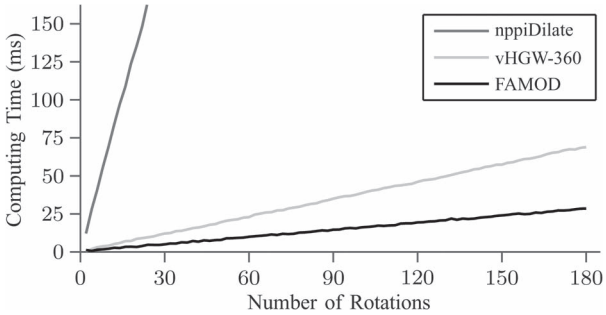


Fig. 7. Required computing time for different numbers of grayscale dilations of a 512 × 512 grid with a rectangular 25 × 11 mask. Shown are NVIDIA's nppiDilate, vHGW-360, and FAMOD.

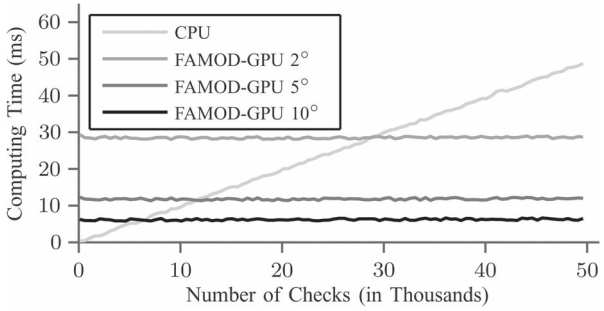


Fig. 8. Total required computing time for different numbers of random checks on a 512 × 512 grid with a randomly chosen 25 × 25 mask. Shown are a direct CPU implementation without preprocessing and FAMOD with different angular resolutions. FAMOD timings include the generation of $\mathcal{C}_{\text{costs}}$ and the memory transfer.

TABLE I
PRECOMPUTATION TIME AND BREAK-EVEN POINTS

	36 dilations		72 dilations		180 dilations	
	t_{ms}	n_c	t_{ms}	n_c	t_{ms}	n_c
FAMOD	6.01	6.1K	11.56	11.8K	28.22	28.7K
vHGW-360	14.19	14.4K	27.87	28.3K	69.03	70K
nppiDilate	245.6	25K	490.7	50K	1226.3	1.25M
BW dil/conv	5.75	–	11.06	–	26.95	–

scenario, and a road construction site, which exhibits noncontinuous road boundaries in the form of traffic cones.

The number of individual cost values used by FAMOD was set to the smallest number that still differentiates it from binary dilation, i.e., 3, to examine worst case accuracy in the given scenarios, although higher values are possible. The functions

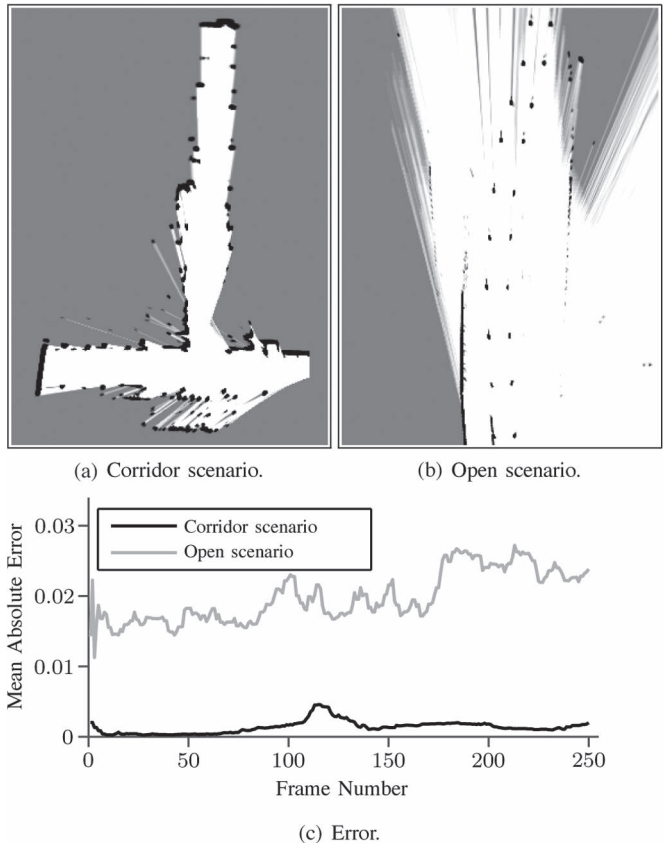


Fig. 9. Mean absolute difference between exact grayscale dilation and FAMOD using three classes. The corridor scenario shows an underground parking lot, and the open scenario shows a road construction site. (a) Corridor scenario. (b) Road construction site. (c) Error.

g and h were designed so that a first class corresponds to the values that are free with a high probability, a second class corresponds to the values that are occupied with high probability, and a third class corresponds to the values in between. Fig. 9 shows the mean absolute error per cell normalized to the range [0, 1]. Due to the sharp borders, the algorithm performs better in the corridor scenario, although, even in the open scenario, the mean absolute error stays low. It is important to notice that, by using g and h as described above, the error only comes from the cells whose occupancy state is uncertain, and hence, collision checking is not affected. Fig. 10 shows exemplary the result of

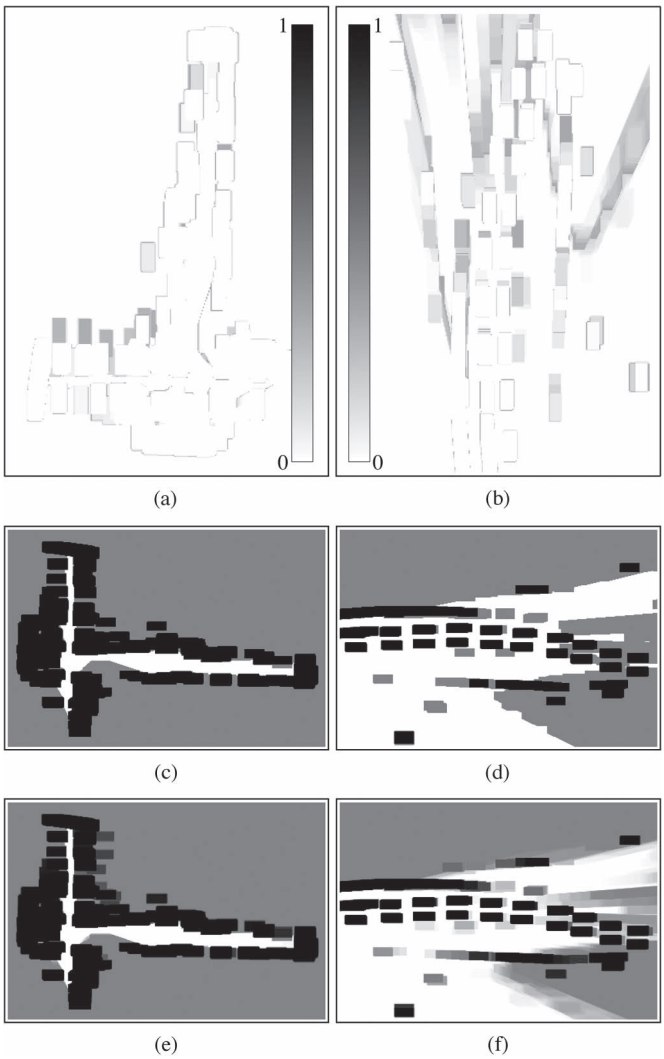


Fig. 10. Exemplary comparison between FAMOD and an exact dilation of the (a), (c), and (e) corridor scenario and the (b), (d), and (f) open scenario with a rectangular vehicle mask. Figures (c)–(f) are rotated clockwise by 90°. (a) Absolute difference corridor. (b) Absolute difference traffic cones. (c) FAMOD three classes corridor. (d) FAMOD three classes traffic cones. (e) Exact dilation corridor. (f) Exact dilation traffic cones.

a dilation from FAMOD in comparison with an exact grayscale dilation from both scenarios.

VIII. Conclusion

This paper has presented how individual configurations can be efficiently checked against cost and collision, which is typically the most computationally expensive part in current path and motion planning algorithms. In order to check a certain configuration, the robot shape needs to be incorporated. Using configuration space costs, which can be seen as generalization of configuration space obstacles, a configuration check is done in constant time.

Two methods to efficiently calculate the configuration space costs were described, i.e., FAMOD and vHGW-360. FAMOD, a fast approximate morphological grayscale dilation, uses convolution to approximate grayscale dilation. Its fast computing times are independent of the shape and the size of the robot mask. Designed for rectangular shapes, vHGW-360 builds upon the popular vHGW dilation algorithm. It produces exact results in the resulting cost values.

Apart from checking individual configurations, this work has also shown how a motion primitive or a whole path can be correctly checked, once configuration space costs have been computed. In addition, this paper underlined the benefits of using graphics hardware for the computation of the configuration space costs as it consists of massively parallel operations. Both algorithms were implemented in CUDA, and their performance proved to be real-time capable on consumer hardware.

References

[1] F. Schwarzer, M. Saha, and J.-C. Latombe, "Exact collision checking of robot paths," in *Algorithmic Foundations of Robotics V*, J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, Eds. Berlin, Germany: Springer-Verlag, 2004, ser. Springer Tracts in Advanced Robotics, pp. 25–42.

[2] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic A*: An anytime, replanning algorithm," in *Proc. Int. Conf. Autom. Planning Sched.*, 2005, pp. 262–271.

[3] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[4] J.-P. Laumond, *Robot Motion Planning and Control*. Secaucus, NJ, USA: Springer-Verlag, 1998.

[5] S. Gehrig and F. Stein, "Collision avoidance for vehicle-following systems," *IEEE Trans. Intell. Transp. Syst.*, vol. 8, no. 2, pp. 233–244, Jun. 2007.

[6] G. Tanzmeister, M. Friedl, A. Lawitzky, D. Wollherr, and M. Buss, "Road course estimation in unknown, structured environments," in *Proc. IEEE Intell. Veh. Symp.*, 2013, pp. 630–635.

[7] M. Ardelt, C. Coester, and N. Kaempchen, "Highly automated driving on freeways in real traffic using a probabilistic framework," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 4, pp. 1576–1585, Dec. 2012.

[8] S. Glaser, B. Vanholme, S. Mammar, D. Gruyer, and L. Nouveliere, "Maneuver-based trajectory planning for highly autonomous vehicles on real road with traffic and driver interaction," *IEEE Trans. Intell. Transp. Syst.*, vol. 11, no. 3, pp. 589–606, Sep. 2010.

[9] K. Chu, M. Lee, and M. Sunwoo, "Local path planning for off-road autonomous driving with avoidance of static obstacles," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 4, pp. 1599–1616, Dec. 2012.

[10] R. Kala and K. Warwick, "Planning autonomous vehicles in the absence of speed lanes using an elastic strip," *IEEE Trans. Intell. Transp. Syst.*, vol. 14, no. 4, pp. 1743–1752, Dec. 2013.

[11] M. Werling, S. Kammel, J. Ziegler, and L. Gröll, "Optimal trajectories for time-critical street scenarios using discretized terminal manifolds," *Int. J. Robot. Res.*, vol. 31, no. 3, pp. 346–359, Mar. 2012.

[12] G. Tanzmeister, M. Friedl, D. Wollherr, and M. Buss, "Path planning on grid maps with unknown goal poses," in *Proc. IEEE Intell. Transp. Syst. Conf.*, 2013, pp. 430–435.

[13] M. Aeberhard, S. Schlichtharle, N. Kaempchen, and T. Bertram, "Track-to-track fusion with asynchronous sensors using information matrix fusion for surround environment perception," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 4, pp. 1717–1726, Dec. 2012.

[14] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, Jun. 1989.

[15] F. Homm, N. Kaempchen, J. Ota, and D. Burschka, "Efficient occupancy grid computation on the GPU with lidar and radar for road boundary detection," in *Proc. IEEE Intell. Veh. Symp.*, 2010, pp. 1006–1013.

[16] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *Int. J. Robot. Res.*, vol. 28, no. 8, pp. 933–945, Aug. 2009.

[17] J. Ziegler and C. Stiller, "Fast collision checking for intelligent vehicle motion planning," in *Proc. IEEE Intell. Veh. Symp.*, 2010, pp. 518–522.

[18] S. Choi and W. Yu, "Any-angle path planning on non-uniform costmaps," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 5615–5621.

[19] L. Jaillet, J. Cortés, and T. Siméon, "Sampling-based path planning on configuration-space costmaps," *IEEE Trans. Robot.*, vol. 26, no. 4, pp. 635–646, Aug. 2010.

[20] J. King and M. Likhachev, "Efficient cost computation in cost map planning for non-circular robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2009, pp. 3924–3930.

[21] R. Szeliski, *Computer Vision: Algorithms and Applications*, 1st ed. New York, NY, USA: Springer-Verlag, 2010.
[22] P. Maragos and R. Schafer, "Morphological filters—Part I: Their set-theoretic analysis and relations to linear shift-invariant filters," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 35, no. 8, pp. 1153–1169, Aug. 1987.
[23] P. K. Ghosh and K. Kumar, "Support function representation of convex bodies, its application in geometric computing, and some related representations," *Comput. Vis. Image Understand.*, vol. 72, no. 3, pp. 379–403, Dec. 1998.
[24] L. Kavraki, "Computation of configuration-space obstacles using the fast Fourier transform," *IEEE Trans. Intell. Transp. Syst.*, vol. 11, no. 3, pp. 408–413, Jun. 1995.
[25] A. Richardson and E. Olson, "Iterative path optimization for practical robot planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 3881–3886.
[26] B. Curto, V. Moreno, and F. Blanco, "A general method for C-space evaluation and its application to articulated robots," *IEEE Trans. Robot. Autom.*, vol. 18, no. 1, pp. 24–31, Feb. 2002.
[27] S. Thrun, S. Thayer, W. Whittaker, C. Baker, W. Burgard, D. Ferguson, D. Hahnel, D. Montemerlo, A. Morris, Z. Omohundro, C. Reverte, and W. Whittaker, "Autonomous exploration and mapping of abandoned mines," *IEEE Robot. Autom. Mag.*, vol. 11, no. 4, pp. 79–91, Dec. 2004.
[28] L. Guibas, L. Ramshaw, and J. Stolfi, "A kinetic framework for computational geometry," in *Proc. 24th Annu. Symp. Found. Comput. Sci.*, 1983, pp. 100–111.
[29] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *J. Field Robot.*, vol. 26, no. 3, pp. 308–333, Mar. 2009.
[30] *CUDA Toolkit Documentation*, NVIDIA Corporation, Santa Clara, CA, USA, Feb. 2013. [Online]. Available: http://docs.nvidia.com/cuda/index.html
[31] *CUDA Fast Fourier Transform Library (cuFFT)*, NVIDIA Corporation, Santa Clara, CA, USA, Feb. 2013. [Online]. Available: http://developer.nvidia.com/cufft
[32] E. R. Davies, *Machine Vision: Theory, Algorithms, Practicalities*, 3rd ed. San Mateo, CA, USA: Morgan Kaufmann, 2005.
[33] M. Van Droogenbroeck and M. J. Buckley, "Morphological erosions and openings: Fast algorithms based on anchors," *J. Math. Imaging Vis.*, vol. 22, no. 2/3, pp. 121–142, May 2005.
[34] D. Lemire, "Faster retrieval with a two-pass dynamic-time-warping lower bound," *Pattern Recognit.*, vol. 42, no. 9, pp. 2169–2180, Sep. 2009.
[35] P. Dokládal and E. Dokládalová, "Computationally efficient, one-pass algorithm for morphological filters," *J. Vis. Commun. Image Represent.*, vol. 22, no. 5, pp. 411–420, Jul. 2011.
[36] M. van Herk, "A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels," *Pattern Recognit. Lett.*, vol. 13, no. 7, pp. 517–521, Jul. 1992.
[37] J. Gil and M. Werman, "Computing 2-D min, median, and max filters," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 15, no. 5, pp. 504–507, May 1993.
[38] L. Domanski, P. Vallotton, and D. Wang, "Parallel van Herk/Gil–Werman image morphology on GPUs using CUDA," NVIDIA GPU Computing Poster Showcase 2009. [Online]. Available: http://www.nvidia.com/content/GTC/posters/14_Domanski_Parallel_vanHerk.pdf
[39] M. Thurley and V. Danell, "Fast morphological image processing open-source extensions for GPU processing with CUDA," *IEEE J. Sel. Topics Signal Process.*, vol. 6, no. 7, pp. 849–855, Nov. 2012.
[40] *CUDA C Best Practices Guide*, NVIDIA Corporation, Santa Clara, CA, USA, 2013. [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUDA_C_Best_Practices_Guide.pdf
[41] D. Hearn and M. P. Baker, *Computer graphics with OpenGL*, 3rd ed. Upper Saddle River, NJ, USA: Pearson Education, 2004.
[42] S. Choi, J.-Y. Lee, and W. Yu, "Fast any-angle path planning on grid maps with non-collision pruning," in *Proc. IEEE Int. Conf. Robot. Biomimetics*, 2010, pp. 1051–1056.
[43] A. Nash, K. Daniel, S. Koenig, and A. Felner, "Theta*: Any-angle path planning on grids," in *Proc. AAAI Conf. Artif. Intell.*, 2007, pp. 1177–1183.
[44] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Comput. Sci. Dept., Iowa State Univ., Ames, IA, USA, Tech. Rep. TR 98-11, 1998.
[45] *NVIDIA Performance Primitives (NPP)*, NVIDIA Corporation, Santa Clara, CA, USA, Feb. 2013. [Online]. Available: https://developer.nvidia.com/npp
[46] M. J. Thurley and V. Danell, LTU-CUDA, Feb. 2013. [Online]. Available: https://github.com/VictorD/LTU-CUDA

Georg Tanzmeister received the BSc degree in computer science and the Dipl.-Ing. degree in computer graphics and computer vision from Technische Universität Wien, Vienna, Austria, in 2009 and 2011, respectively. He studied computer science at Universitat Autònoma de Barcelona, Barcelona, Spain, in 2008 and at The City College of New York, New York City, NY, USA, in 2009. He is currently working toward the Dr.-Ing. degree in electrical engineering at Technische Universität München, München, Germany, in conjunction with BMW Group Research and Technology.

His research interests include autonomous vehicles, path and motion planning, and environment perception and mapping.

Martin Friedl received the Dipl.-Ing. degree in mechanical engineering from Karlsruhe Institute of Technology, Karlsruhe, Germany, in 2007. In 2005, he studied mechanical engineering at Purdue University, West Lafayette, IN, USA.

Since 2007, he has been with the BMW Group, Munich, Germany. His research interests include algorithms for perception, localization, and path planning.

Dirk Wollherr received the Dipl.-Ing. degree in electrical engineering and the Dr.-Ing. degree in electrical engineering from Technische Universität München, München, Germany, in 2000 and 2005, respectively.

Since 2005, he has been a Senior Researcher and a Lecturer with the Institute of Automatic Control Engineering, Department of Electrical Engineering and Information Technology, Technische Universität München, where he has been also a Principal Investigator, an Independent Junior Research Group Leader, and a Research Area Leader within the Deutsche Forschungsgemeinschaft cluster of excellence "Cognition for Technical Systems" since 2005. His current research interests include mobile robotics, human–robot interaction, environment understanding, object recognition, and manipulation of dynamic systems.

Martin Buss received the Dipl.-Ing. degree in electrical engineering from Technische Universität Darmstadt, Darmstadt, Germany, in 1990 and the Dr.-Ing. degree in electrical engineering from the University of Tokyo, Tokyo, Japan, in 1994.

In 1988, he was a Research Student for one year with the Science University of Tokyo, Tokyo. From 1994 to 1995, he was a Postdoctoral Researcher with the Department of Systems Engineering, Australian National University, Canberra, CT, Australia. From 1995 to 2000, he was a Senior Research Assistant and a Lecturer with the Institute of Automatic Control Engineering, Department of Electrical Engineering and Information Technology, Technische Universität München, München, Germany. From 2000 to 2003, he was a Full Professor, the Head of the Control Systems Group, and the Deputy Director of the Institute of Energy and Automation Technology, Faculty IV, Electrical Engineering and Computer Science, Technical University of Berlin, Berlin, Germany. Since 2003, he has been a Full Professor (Chair) with the Institute of Automatic Control Engineering, Faculty of Electrical Engineering and Information Technology, Technische Universität München, where he has been with the Medical Faculty since 2008 and a Coordinator of the Deutsche Forschungsgemeinschaft cluster of excellence "Cognition for Technical Systems" since 2006. His research interests include automatic control, mechatronics, multimodal human–system interfaces, optimization, and nonlinear and hybrid discrete-continuous systems.