2010 IEEE Intelligent Vehicles Symposium
University of California, San Diego, CA, USA
June 21-24, 2010

Fast Collision Checking for Intelligent Vehicle Motion Planning

Julius Ziegler and Christoph Stiller

Department of Measurement and Control (MRT)
Karlsruhe Institute of Technology (KIT)
76131 Karlsruhe, Germany
{ziegler|stiller}@kit.edu

***Abstract*— We present a method for fast collision checking that is suitable for application in motion planning for intelligent vehicles. One of the difficulties that arises in this domain is the fact that typical, car-like autonomous vehicles cannot easily be approximated by a rotationally invariant disk shape. Instead, the orientation of the vehicle must be accounted for explicitly. Our proposal is to decompose the vehicle shape into several disk shaped primitives, so that the task of collision checking can be broken down into few very simple collision tests. We also propose a highly optimised method to perform these primitive collision tests that requires a minimum of arithmetic operations. We show by experiments that our method bears significant performance benefits over conventional methods.**

I. Introduction

When solving motion planning problems, collision checking is widely considered ([1], [2], [3]) as one of the major run time bottlenecks. This is true especially for search based methods ([4], [5], [3], [6]), where, in the worst case, the workspace must be searched exhaustively and all possible system states must be tested for collision. For this reason, one often chooses to calculate the subset of the robots state space that leads to collision with the environment up front, yielding the set of so called configuration space obstacles. While this precalculation can be a comparatively expensive operation, it can reduce the cost for a subsequent collision check drastically and, for large scale problems that typically require millions of checks, easily pays off in terms of overall run time.

The geometric footprint of many experimental robot platforms is approximately as long as it is wide, so that it can be reasonably approximated by a disk shape. This simplifies calculation of the configuration space obstacles enormously, since a disk is invariant to rotation. Consequently, the orientation of the robot does not have to be considered. Obviously, this is not true for autonomous vehicles that are based on production cars. These typically have an aspect ratio of approximately 5 to 2, and approximation by a bounding disk would be far too conservative, at least in scenarios where good utilisation of the free space is necessary. Such scenarios include parking manoeuvres or driving on narrow roads.

Assuming the simple model of a rigid robot that is restricted to translation (or has a rotationally invariant shape), the configuration space obstacles can be expressed in terms of the Minkowski sum of the robot and obstacle shapes. The Minkowski sum of two subsets A and B of a vector space

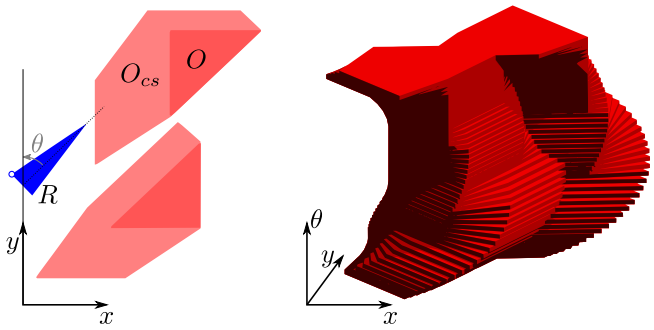


Fig. 1. Configuration space obstacles for a triangular robot. The reference point of the robot is marked. (a): Robot fixed at specific orientation. (b): Robot is allowed to rotate. Rotation angle θ has been discretised.

is defined as the result of adding all elements of A to all elements of B:

$$A \oplus B = \{a + b | a \in A, b \in B\}.$$

Figure 1a exemplary shows the Minkowski sum of a triangular robot R and a set of polygonal obstacle shapes, O. The robot is not allowed to rotate, but fixed to the displayed orientation. Note how the Minkowski sum $O_{CS} = R \oplus O$ exactly represents the shape of the configuration space obstacles, *i.e.* placing the robots reference point within the shaded region leads to collision, while placing it outside of it is always feasible.

If both robot shape and obstacles are represented as polygons, a geometric convolution algorithm [7] can be employed to compute the Minkowski sum.

If we drop the restriction of a purely translational motion model and allow the robot to rotate, explicit calculation of the now 3 dimensional configuration space obstacles becomes difficult [8]. A common approach is to discretise the rotation angle θ and to "stack" multiple 2 dimensional slices of O_{CS} to form the 3 dimensional configuration space. Figure 1b illustrates this for the triangular robot.

Polygonal or polyhedral representation of robot and environment has been at the core of most early motion planning research [9], and many important theoretical properties of the problem have been derived on this basis [10]. It is still widespread in motion planning for articulated, industrial robots operating in a well known, static environment, where

978-1-4244-7868-2/10/$26.00 ©2010 IEEE

exact polyhedral models of the workspace are available from CAD data. However, in most contemporary mobile robotic applications, polyhedral representation of the environment has been replaced by discrete occupancy grids ([11], [12], [13]).

Occupancy grids can approximate arbitrary obstacle shapes up to their resolution limit. This representation - helped by the rapidly increasing capacity of computer memory - has established itself as the more practical one, since it can be derived naturally from most sensor systems (such as LIDAR and stereo sensors), and even fusion of different sensors can easily be accomplished when using a discrete grid as the target data structure [14]. Moreover, even higher level representations of the environment (such as object lists) can easily be lowered to a grid representation.

In [3], collision checking of a vehicle shape against a discrete occupancy grid was accelerated using a hierarchical approach. First, a minimum bounding circle that completely contains the vehicle shape, and a maximum inscribing circle that is itself contained by the vehicle shape, are checked for collision. If the minimum bounding circle is collision free, the vehicle must be collision free, and if the inscribing circle is in collision, the vehicle is guaranteed to be in collision. Only in cases where the first check fails, but the latter succeeds, a more expensive test for the exact vehicle shape must be carried out.

Our approach for collision checking a vehicle shape against a discrete obstacle grid consists of a stepwise decomposition of the vehicle shape into geometric primitives which can quickly be checked for collision. As a first step, the vehicle shape is approximated by a set of overlapping, circular disks. Subsequently, circles are decomposed into axis aligned rectangles, for which collision checking can be done extremely fast using a simple precomputation. Circular decomposition has been used to accelerate collision checking before [15], however, both obstacle and vehicle shapes were decomposed into circles. This does not work well for general obstacle shapes as can be represented in a discrete obstacle map.

The rest of this article is structured as follows: In section II, we will derive a relationship between the discrete convolution operation and both shape decomposition and collision checking, that can be exploited to yield a more efficient approach to calculating configuration space obstacles. In section III, we will focus on both the geometric aspects of shape decomposition and efficient methods to check the elementary shape primitives themselves. In section IV we show how our methodology can meet the requirements of different classes of motion planning algorithms. Finally, we present and discuss results of the described method in section V.

II. COLLISION CHECKING AND CONVOLUTIONB

Let the shape and position of the obstacles be given as a two dimensional discrete map $o : \mathbb{Z}^2 \to \{0, 1\}$ that assigns 1 to every point in the integer plane that is occupied by an obstacle, and 0 to every free point:

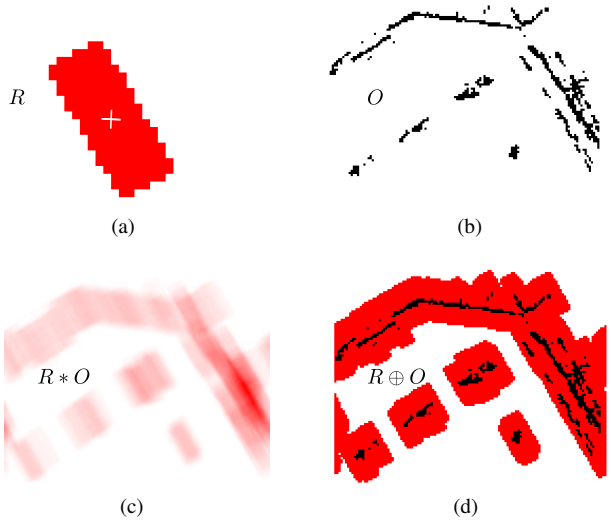


Fig. 2. Minkowski sum and convolution. (a): A vehicle shape in discrete grid representation. (b): discrete occupancy grid. (c) convolution of the former two. (d) Minkowski sum with original obstacles overlaid.

$$o(u, v) = \begin{cases} 1 & (u, v) \in O \\ 0 & \text{else.} \end{cases} \quad (1)$$

Let the robots shape R be defined analogously by the function r. Let c be the discrete, two dimensional convolution of r and o:

$$\begin{aligned} c(u, v) &= r(u, v) * o(u, v) \\ &= \sum_{\tau_u} \sum_{\tau_v} r(\tau_u, \tau_v) o(x - \tau_u, y - \tau_v) \end{aligned}$$

The Minkowski sum of R and O are all cells for which the convolution is greater than zero:

$$R \oplus O = \{(u, v) \in \mathbb{Z}^2 : c(u, v) > 0\}.$$

Figures 2a-d illustrate the relation of convolution and Minkowski sum by example. In [16], this property was exploited to implement a fast way of calculating configuration space obstacles by carrying out the convolution using the fast Fourier transform (FFT).

The intuition that additional performance can be gained by decomposing the vehicle shape into disks follows from the associativity of the convolution operation, that propagates to the Minkowski sum. The decomposition of the vehicle shape into disks can be described by a convolution itself, as illustrated by figure 3a-c. Figure 3a shows a disk shaped convolution kernel D. The kernel I in figure 3b consists of only three non-zero elements (impulses). Figure 3c shows the convolution of $D * I$, which is a good approximation to the original vehicle shape R from figure 2a.

The overall Minkowski sum $R \oplus O$ from figure 2d can now be approximated by $O \oplus D \oplus I$ as illustrated in figure 3d-e. The Minkowski sum is an associative operation, so that

$$(O \oplus D) \oplus I = O \oplus (D \oplus I). \quad (2)$$

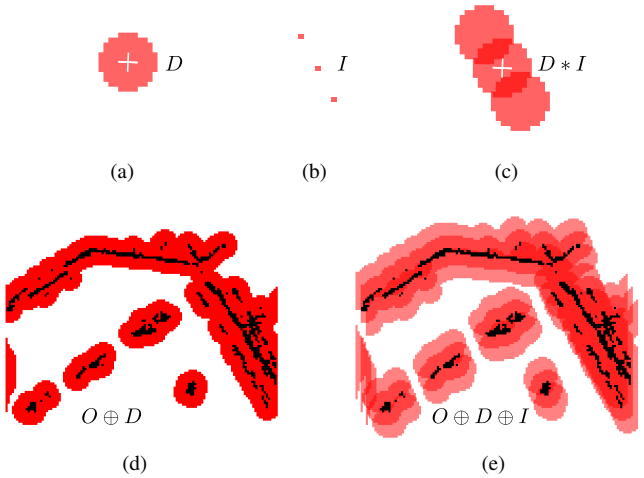


Fig. 3. Decomposition and convolution. (a): a disk shaped convolution kernel. (b): a kernel comprised of three impulses. (c) convolution of the previous two. (d) Minkowski sum of disk shaped kernel with obstacles. (e) convolution of the previous with impulse kernel, original obstacles overlaid.

The left hand side of 2 suggests a processing order that bears significant benefits for runtime performance. The reason for this is that the term $O \oplus D$ is independent from the vehicle orientation. The vehicle orientation only affects the impulse kernel I, which is sparse and allows for quick, direct convolution in position space.

III. Decomposition of vehicle shape into rotation invariant primitives

This section will detail on the geometric aspects of shape decomposition, and present efficient methods for collision checking the final shape primitives, namely axis aligned rectangles, themselves.

A. Decomposition of vehicle shape into disks

A car-like vehicle is typically of approximately rectangular shape, but it can be reasonably approximated by a set of overlapping circular disks. The problem of covering rectangles [17] or arbitrary polygons [18] with equal sized disks in an optimal way has been studied in the literature and was shown to be NP-hard by [19]. However, for the case of a rectangle, good ad-hoc solutions are easy to find, as demonstrated by figures 4a-b. For example, a rectangle of length l and width w can be covered by n circles of radius

$$r = \sqrt{\frac{l^2}{n^2} + \frac{w^2}{4}}$$

placed at distance

$$d = 2\sqrt{r^2 - \frac{w^2}{4}},$$

as can be shown by simple geometric reasoning, which is illustrated in figure 4a, for the exemplary parameters $n = 3$, $l = 5$ and $w = 2$. If multiple radii are admissible, a rectangle can be covered even more efficiently (figure 4c).

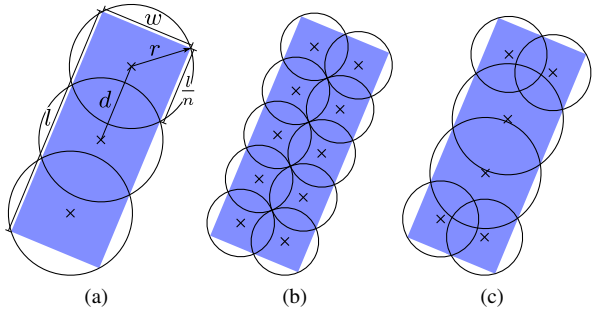


Fig. 4. Disk approximations of vehicle shape. (a): 3 disk approximation. (b): 10 disk approximation. (c): 6 disk approximation with two different radii.

B. Decomposition of disks into axis aligned rectangles

Figure 5a highlights the pixels that must be identified to cover a disk with rectangles. This is done by plotting a step function approximation to the boundary of the disk. A pixel is identified as a rectangle corner whenever a unit change in the x-coordinate happens. By exploiting the 8-way symmetry of the disk, each of these corner points (x, y) can be used to define the two rectangles $[-x, -y] \times [x, y]$ and $[-y, -x] \times [y, x]$. As can be easily seen, scanning one octant of the disk boundary is sufficient to cover the whole disk (see figure 5b).

The described step function can be generated by a modification of Bresenham's midpoint algorithm [20]. The midpoint algorithm is based on the parametric disk equation $F(x, y) < 0$ with

$$F(x, y) = x^2 + y^2 - r^2. \tag{3}$$

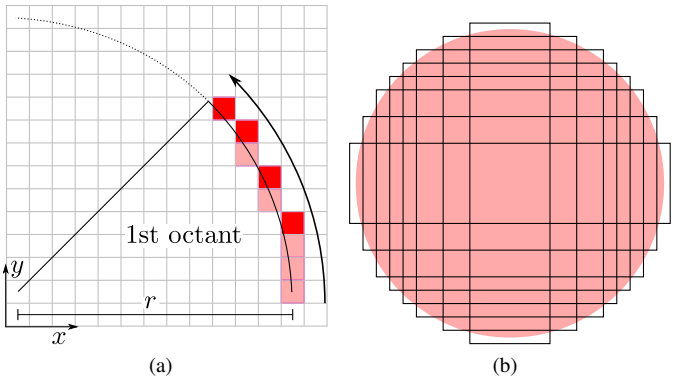
It scans the first octant of the circle upwards in the direction indicated in figure 5a. While the y coordinate must be incremented in every step, it must be decided at every iteration if the x coordinate must be decremented or not. This is done by inserting the midpoint between the two possible successor pixels $P_{\text{up}} = (x, y + 1)$ and $P_{\text{up-left}} = (x - 1, y + 1)$ into the circle equation 3

$$\begin{aligned} F_{\text{mid}} &= F(x - \frac{1}{2}, y + 1) \\ &= y^2 + 2y + x^2 - x - r^2 + \frac{5}{4}. \end{aligned}$$

If this term yields a positive value, the midpoint lies outside of the circle, and x is decremented by one. This method can be turned into a fast, incremental algorithm by analysing the change of F_{mid} with respect to moving up or up-left:

$$\begin{aligned} \Delta_{\text{up}} &= F(x - \frac{1}{2}, y + 2) - F_{\text{mid}} \\ &= 2y + 3 \\ \Delta_{\text{up-left}} &= F(x - \frac{3}{2}, y + 2) - F_{\text{mid}} \\ &= 2y - 2x + 5 \\ &= \Delta_{\text{up}} - 2x + 2. \end{aligned}$$

As can be seen, Δ_{up} increments by 2 with every increment of y, and $\Delta_{\text{up-left}}$ increments by 2 with every decrement



(a) (b)

Fig. 5. (a): Scanning for border pixels that are used as rectangle corners. (b): The rasterisation of a disk of radius 12 can be covered by 8 overlapping rectangles.

```
def draw_circle(radius):
  F = 1-radius
  x = radius
  y = 0
  delta_up_left = -2*radius
  delta_up = 1
  while y >= x:
    if F >= 0:
      draw_box(-x, -y, x, y)
      draw_box(-y, -x, y, x)
      x -= 1
      delta_up_left += 2
      F += delta_up_left
    y += 1
    delta_up += 2
    F += delta_up
```

Code Listing 1. Extension of the midpoint algorithm to cover a disk with a set of axis aligned rectangles. The disk is centered at the origin.

of x. Code listing 1 shows Python code for drawing an overlapping rectangle representation of a disk using this incremental variant of the method. It only requires integer additions and subtractions.

C. Fast collision checking for axis aligned rectangles

In this section, we stick with the representation of the obstacles O by the discrete mapping o, as defined in equation 1. An integral image or summed area table is a data structure for efficiently generating the sum over a rectangular, axis aligned subsets of a grid. In each cell, it contains the sum of all cells in the original grid, o, above and to the left of the cell:

$$I_o(X,Y) = \sum_{x \le X, y \le Y} o(x,y).$$

It can be precomputed by a single run over the occupancy grid. Once it is computed, the sum over a rectangular region $A = \{x, y | x_0 < x < x_1, y_0 < y < y_1\}$ in o can be queried by just four array references:

$$\begin{aligned} \sum_{x,y \in A} o(x,y) \quad &= \quad I_o(x_0,y_0) + I_o(x_1,y_1) \\ &\quad -I_o(x_0,y_1) - I_o(x_1,y_0). \end{aligned}$$

precomputations	precomputation time	time per 1000000 random collision checks
full, 72 discrete angles. [using FFT]	62 ms [972 ms]	110 ms
full, 144 discrete angles. [using FFT]	115 ms [1940 ms]	110 ms
translational only [using FFT]	7.7 ms [13.5 ms]	120 ms
integral image only	1.2 ms	410 ms

TABLE I

RUNTIME PROFILES FOR DIFFERENT AMOUNTS OF PRECOMPUTATION.

Now, the Minkowski sum of a rectangle and a discrete obstacle map can be calculated easily, since

$$(x,y) \in O \oplus A \Leftrightarrow \sum_{x,y \in A} o(x,y) > 0.$$

IV. DENSE AND SPARSE COLLISION CHECKING

The described method can be used to precalculate collisions densely for the whole three dimensional configuration space, in the way illustrated by figure 1b. This requires to discretise the orientation θ of the robot, in addition to discretising the x and y-directions. While the discretisation of x and y is predetermined by the occupancy grid resolution, the resolution of the discretisation of θ is a design parameter that must be chosen, trading of runtime against precision. This dense precalculation of the configuration space obstacles is advisable for search based motion planning methods, since these typically require millions of checks, each of which can be reduced to a single table lookup through the precomputation.

Alternatively, precalculation can be reduced, retaining a part of the computation and calculating it on demand, *i.e.* once per single collision check. This procedure is suited for motion planning methods that do not search the configuration space exhaustively. This applies, *e.g.*, to reactive planners ([21], [22]) that choose a solution from a relatively small set of trajectories, where the whole set typically only covers a small fraction of the workspace. If $e.g.$, precomputation is reduced to calculating the discrete Minkowski sum of the obstacles and a disk shape, for every collision check the centres of the disks that form the vehicle shape must be calculated, followed by as many table array references as there are disks in the decomposition.

For motion planners that need to analyse the workspace only very sparsely, precomputation can even further be reduced. If at least the integral image is precomputed, a single collision check for a disk can still be evaluated fairly quickly through the rectangle decomposition proposed in section III-B.

V. RESULTS AND CONCLUSIONS

Table I presents timing profiles for different levels of precomputation. The experiments where performed on a 512 by 512 cell occupancy grid covered at 25% with obstacles. Resolution was 0.1 m per cell. The decomposition chosen

was equivalent to that in figure 4a, consisting of three circles of radius 1 m. A reference value was determined without use of decompositions. Instead, the most highly optimized implementation of the fast fourier transform available todate [23] was used to calculate convolutions.

One observation in the timing data is the negligible benefit of precalculating the full configuration space. The reason for this unexpected results are cache effects: The CPU architecture of the test system (INTEL CORE2 DUO) has approximately 4 MB of very fast cache memory. This is sufficient to store the two dimensional configuration space for one disk ($512 \cdot 512 \approx 262$K), but can not hold the full, three dimensional configuration space ($512 \cdot 512 \cdot 72 \approx 19$M, resp. $512 \cdot 512 \cdot 144 \approx 38$M). By aggregating the respective numbers for preprocessing and subsequent collision checks, one can derive that, if less then 22400 collision checks are expected, calculating the translational configuration space of a disk kernel in advance does not pay of. On the other hand, due to the cache effects observed, the break-even point for calculating the full configuration space in advance comes only at $5 \cdot 10^6$ (72 discrete angles) or even $10 \cdot 10^6$ (144 discrete angles) total collision checks.

VI. ACKNOWLEDGEMENTS

We gratefully acknowledge support of this work by the German research foundation (DFG) within the transregional collaborative research centre 28 "cognitive automobiles".

REFERENCES

[1] F. Schwarzer, M. Saha, and J. C. Latombe, "Exact collision checking of robot paths," *Algorithmic Foundations of Robotics*, pp. 25–41, 2004.

[2] P. Jimanez, F. Thomas, and C. Torras, "Collision detection algorithms for motion planning," in *Lecture Notes in Control and Information Sciences, 229*, pp. 305–343, Springer, 1998.

[3] M. Likhachev and D. Ferguson, "Planning long dynamically-feasible maneuvers for autonomous vehicles," in *Proceedings of Robotics: Science and Systems IV*, 2008.

[4] J. Ziegler, M. Werling, and J. Schröder, "Navigating car-like vehicles in unstructured environment," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2008.

[5] M. Pitvoraiko and A. Kelly, "Efficient constrained path planning via search in state lattices," in *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, 2005.

[6] M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," in *International Conference on Robotics and Automation*, 1999.

[7] L. J. Guibas, L. Ramshaw, and J. Stolfi, "A kinetic framework for computational geometry," in *FOCS*, pp. 100–111, IEEE, 1983.

[8] J.-C. Latombe, *Robot Motion Planning*. Kluwer, 1991.

[9] Y. K. Hwang and N. Ahuja, "Gross motion planning - a survey," *ACM Computing Surveys*, vol. 24, no. 3, pp. 219–291, 1992.

[10] J. F. Canny, *The complexity of robot motion planning*. Cambridge, MA, USA: MIT Press, 1988.

[11] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, pp. 46–57, 1989.

[12] K. Konolige, "Improved occupancy grids for map building," *Auton. Robots*, vol. 4, no. 4, pp. 351–367, 1997.

[13] S. Thrun, "Learning occupancy grid maps with forward sensor models," *Auton. Robots*, vol. 15, no. 2, pp. 111–127, 2003.

[14] H. Moravec, "Certainty grids for sensor fusion in mobile robots," *Sensor Devices and Systems for Robotics*, pp. 243–276, 1989.

[15] I. Papadimitriou and M. Tomizuka, "Fast lane changing computations using polynomials," in *American Control Conference*, 2003.

[16] L. Kavraki, "Computation of configuration-space obstacles using the fast fourier transform," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 3, pp. 408–413, 1995.

[17] J. B. M. Melissen and P. C. Schuur, "Covering a rectangle with six and seven circles," *Discrete Applied Mathematics*, vol. 99, no. 1-3, pp. 149 – 156, 2000.

[18] G. K. Das, S. Das, S. C. Nandy, and B. P. Sinha, "Efficient algorithm for placing a given number of base stations to cover a convex region," *Journal of Parallel and Distributed Computing*, vol. 66, no. 11, pp. 1353 – 1358, 2006.

[19] S. Masuyama, T. Ibaraki, and H. Hasegawa, "The computational complexity of the m-center problems on the plane," *Transactions IECE Japan*, vol. E64, pp. 57–64, 1981.

[20] J. Bresenham, "A linear algorithm for incremental digital display of circular arcs," *Commun. ACM*, vol. 20, no. 2, pp. 100–106, 1977.

[21] T. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *International Journal of Robotics Research*, vol. 26, pp. 141–166, 2007.

[22] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a frenet frame," in *IEEE International Conference on Robotics and Automation*, 2010. *(accepted for publication)*.

[23] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".