

Efficient Occupancy Grid Computation on the GPU with Lidar and Radar for Road Boundary Detection

Florian Homm, Nico Kaempchen, Jeff Ota and Darius Burschka

Abstract—Accurate maps of the static environment are essential for many advanced driver-assistance systems. In this paper a new method for the fast computation of occupancy grid maps with laser range-finders and radar sensors is proposed. The approach utilizes the Graphics Processing Unit to overcome the limitations of classical occupancy grid computation in automotive environments. It is possible to generate highly accurate grid maps in just a few milliseconds without the loss of sensor precision. Moreover, in the case of a lower resolution radar sensor it is shown that it is suitable to apply super-resolution algorithms to achieve the accuracy of a higher resolution laser-scanner. Finally, a novel histogram based approach for road boundary detection with lidar and radar sensors is presented.

I. INTRODUCTION

Comprehensive and accurate models of the environment are essential for advanced driver-assistance systems (ADAS). With an increasing amount of autonomy in ADAS it is crucial not only to have detailed information about other road users (dynamic objects) but also about stationary objects. These are of high importance to determine the drivable corridor (path planning and obstacle avoidance) or to estimate the position with respect to some reference frame (landmark based localization). Classical tracking methods of dynamic objects are the first choice if there are a small number of targets of which the geometry can be specified beforehand. However, they are less applicable to track arbitrary static objects. The geometry of static objects in outdoor environments can be quite complex and hard to describe by geometric primitives like lines, boxes or circles.

One of the most popular methods to overcome this problem is the occupancy grid framework [1]. Occupancy grids are a *location based* representation of the environment. They divide the environment into an evenly spaced grid of cells and estimate the probability of each cell for being occupied based on the sensor readings. They have been successfully applied to sensor data fusion [2], path planning [3], simultaneous localization and mapping (SLAM) [4] and target tracking [5]. While the dominant sensor for occupancy grid mapping are laser range-finders, some authors have built grids with stereo vision [6] and radar sensors [7].

Arguably, occupancy grid maps are one of the most successful environment representations today. However, while

theoretically elegant, their implementation in applications for automotive environments suffers from two major problems: occupancy grids are computational intensive and there is a trade-off between the size and the resolution of the grid. The field of view is therefore strongly limited when choosing a cell-size that is acceptable to represent the resolution of modern automotive sensor systems. This can cause problems if performing path planning or collision avoidance at high velocities. Even if the computations can be carried out in realtime for a single sensor, the performance rapidly decreases when fusing multiple sensors. Furthermore, to fulfill the realtime requirements it is common practice to use a quite simplified *inverse sensor model* that maps the measurements to the grid. This results in a loss of sensor precision and causes the so called Moiré effect [8] at far distances. While the loss of sensor precision limits its application to high precision ego localization, the Moiré effect badly affects any kind of path planning algorithm that needs to look far ahead.

In this paper an approach for occupancy grid mapping on the Graphic Processing Unit (GPU) is introduced. A detailed description for the fast computation of accurate occupancy grids with laser range-finders on the GPU using the CUDA framework is given. It is shown that this approach is suitable to compute highly accurate grid maps in just a few milliseconds on standard consumer hardware and a brief extension to radar sensors is presented. Finally, a novel histogram based approach for road boundary detection is introduced.

II. GENERAL PURPOSE PROGRAMMING ON THE GPU WITH CUDA

In the past few years the commodity Graphic Processing Unit has emerged as an economical and fast co-processor. Driven by the consumer market for high-definition 3D graphics, the GPU has evolved into a highly parallel multicore processor with tremendous computational power. While today CUDA-capable GPUs are only available in parallel clusters for scientific computing or consumer products such as phones and netbooks, they will become available for automotive platforms in the future. Their key advantage over specialized hardware devices such as FPGAs lies in the standard unified programmable interface and their low production costs.

At the hardware level, a GPU is a collection of multiprocessors with multiple processing units in each processor. It is well-suited to address problems that can be expressed as data-parallel computations with high arithmetic intensity [9]. However, the GPU has had quite a complex programming model, which made it difficult to use a GPU for non graphic operations. To overcome this problem, nVidia introduced

F. Homm is with BMW Group, Research and Technology, D-80788 Munich, Germany florian.homm@bmw.de

N. Kaempchen is with BMW Group, Research and Technology, D-80788 Munich, Germany nico.kaempchen@bmw.de

J. Ota is with BMW Group, Research and Technology, Palo Alto, CA 94301, USA jeff.ota@bmw.de

D. Burschka is with Technische Universität München, Department of Computer Science, D-85748 Garching, Germany burschka@cs.tum.edu

the CUDA framework, a general purpose parallel computing architecture. Since its first release in November 2006 the CUDA framework has attracted immense attention. Many algorithms, such as object segmentation [10] and object tracking [11], [12], have been successfully ported to the CUDA framework for nVidia GPUs.

In general, CUDA comes with a programming interface to use the parallel GPU architecture for general purpose computations. At its core are three key abstractions: a hierarchy of grouped threads, different types of memory transfer and barrier synchronization functions. CUDA manages a large number of programming *threads* and organizes them into a set of logic *blocks*. All *blocks* are clustered to a logical unit called a *grid* (Fig. 1). Each block is assigned to one

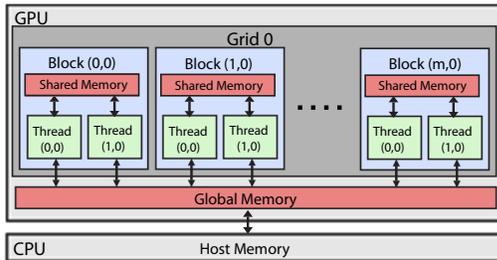


Fig. 1. CUDA programming model. Logic division into the units: Grid, Blocks and Threads.

multiprocessor and the threads of a block execute concurrently on one multiprocessor in a SIMT (Single Instruction Multiple Thread) fashion. For a more detailed description, the interested reader is referred to the CUDA Programming Manual [9]. The following section focuses only on the issues which are of high importance for the computation of occupancy grids on the GPU.

As mentioned before, a multiprocessor executes threads in a SIMT fashion which is akin to SIMD (Single Instruction Multiple Data). It schedules and executes threads in groups of 32 parallel threads called *warps*. A single warp executes one common instruction at a time, so maximum efficiency is achieved when all threads of a warp agree on their execution path. If they diverge via data-dependent flow control instructions (for instance 'if' statements), the execution paths have to be serialized. In other words, the best performance is achieved if all threads of a warp execute the same command. Depending on the number of diverging threads, this can significantly impact the performance of computations on the GPU. Furthermore, it is necessary to minimize the memory transfers between the CPU and the GPU. Due to the communication bus bottleneck, transfers are rather slow and can rapidly deteriorate the performance advantage of parallel computing on the GPU. These issues are addressed in the next sections, where it will be shown how to efficiently build Occupancy Grids without violating these policies.

III. OCCUPANCY GRID MAPPING

In this section, a brief introduction to the problem statement of classical occupancy grid mapping is first given.

Then, the computation of a temporary Polar measurement grid considering actual sensor readings and how to efficiently map it into Cartesian space using the CUDA framework is discussed. Finally, the consecutive temporary grids from different vehicle poses (positions and orientations) are combined to an occupancy grid.

A. Introduction

Occupancy grids represent the environment by fine-grained metric grids of cells and estimate the probability of any cell for being "occupied" depending on the sensor readings. Unlike target tracking approaches, they do not only contain information about the presence of objects, but also about the absence (free-space). Let m be a planar occupancy grid. A single grid cell is denoted as $m_{x,y}$ and z_1, \dots, z_t denotes all of the measurements up to time t along with the vehicle poses x_1, \dots, x_t . Each grid cell corresponds to a binary value which specifies whether a cell is occupied or not. The notation $p(m_{x,y} = 1)$ is used to denote an occupied cell. Furthermore, it is assumed that x_1, \dots, x_t are known which is called *mapping with known poses* [1]. The problem addressed by occupancy grid mapping is to determine the posterior

$$p(m|z_1, \dots, z_t, x_1, \dots, x_t) \quad (1)$$

given all measurements and poses. However, grid maps are defined over high-dimensional space and this posterior cannot be computed easily. The classical occupancy grid algorithm breaks down the problem into many one-dimensional estimation problems, which are treated independently of each other. The problem is reduced to the estimation of the posterior for each $m_{x,y}$ of the grid:

$$p(m_{x,y}|z_1, \dots, z_t, x_1, \dots, x_t) \quad (2)$$

The formulation (2) is fundamental for occupancy grid mapping on the GPU, since it allows us to process the cells in parallel order. For computational reasons, it is common practice to represent the probability

$$p(m_{x,y}|z_1, \dots, z_t) = \frac{p(m_{x,y}|z_1, \dots, z_t)}{1 - p(m_{x,y}|z_1, \dots, z_t)} \quad (3)$$

in the so called odds form. After applying Bayes Theorem and removing some hard-to-compute terms we finally get:

$$\frac{p(m_{x,y}|z_1, \dots, z_t)}{1 - p(m_{x,y}|z_1, \dots, z_t)} = \frac{p(m_{x,y}|z_t)}{1 - p(m_{x,y}|z_t)} \cdot \frac{p(m_{x,y}|z_1, \dots, z_{t-1})}{1 - p(m_{x,y}|z_1, \dots, z_{t-1})} \quad (4)$$

The probability of each cell $p(m_{x,y})$ at time step t is easily obtained by multiplying $p(m_{x,y}|z_t)$, that contains the probability of being occupied with respect to the actual measurement update, with the grid-cell $p(m_{x,y}|z_1, \dots, z_{t-1})$ containing all previous measurements. Assuming a prior probability for each cell of being occupied $p(m_{x,y} = 0.5)$, the estimation problem simply becomes to update the probability $p(m_{x,y}^t)$ of each cell in a temporary grid m^t , given the actual measurement z_t .

This update is usually performed by an inverse sensor model, which maps the measurement vector z_t to the grid m^t . To fulfill the realtime requirements, it is common practice to use a simple ray casting model based on line drawing to determine the cells that are affected by the measurement update [1], [13], [14]. Even for laser range-finders with their relative small angular perceptual field such a model is not appropriate. It assumes a constant beam width which is set according to the previously chosen cell-size of the grid. This is an incorrect assumption, since beams overlap multiple cells depending on the sensor characteristics and the cell-size of the grid. In regions close to the sensor origin, the number of beams that contribute to the probability of a single cell is quite large. In contrast, far from the origin a single beam overlaps multiple cells. The latter one is even worse and causes the aforementioned undesired Moiré effect [8]. Additionally, it is also important to have an appropriate inverse sensor model to represent the measurement uncertainty of z_t and the cells that are directly affected by this to avoid biased grid maps. An accurate model should compute the probability of each cell depending on the position in the map, the beam width and the distance to the beam center.

B. Mapping on the GPU

Each sensor that operates by the measurement principle of time-of-flight, like laser range-finders, records detection events in a polar coordinate system due to the polar geometry of wave propagation. In order to overcome the problem that the number of cells affected by a single beam directly depends on the measured distance and the specific beam width, a temporary measurement grid in polar coordinate space is first defined. This temporary grid at time step t is referred to as m' . Unlike m^t in Cartesian space, z_t can be easily mapped to m' regardless of the sensor intrinsics. The discrete angle steps of the laser range-finder are assumed to be known and that adjacent beams do not overlap. Each beam then only affects a 1-D linear strip (row) in the grid m' (Fig. 2). One single linear strip is denoted as m'_k . The dimension of the grid in the ϕ direction is given by number of discrete angle steps, which is equivalent to the size of z_t , and the dimension in r direction by the field of view along the beams divided by the desired depth sample rate. We will use z_t^k to refer to an individual range measurement (in cells) of the vector z_t .

$$z_t = z_t^1, \dots, z_t^K \quad (5)$$

If the sensor returns no target for a discrete angle step, z_t^k is considered to be infinite. An inverse sensor model which only depends on the measurement vector z_t can now be defined.

1) *Inverse Sensor Model:* The objective of the inverse sensor model is to describe the formation process by which measurements are generated in the physical world and how they affect the grid cells. A laser range-finder actively emits a signal and records its echo. The returned echo depends on a variety of properties, such as the distance to the target, the surface material and the angle between the surface normal and the beam. The measurement error is modeled by a 1-D narrow Gaussian with mean z_t^k and standard deviation

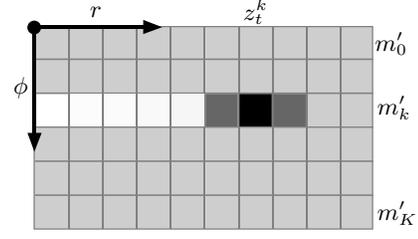


Fig. 2. Example for a polar grid m'_k with a single target z_t^k in the 2th row. Lower probabilities are encoded in white, higher probabilities in black.

$\delta(z_t^k)$. This follows the assumption that z_t^k is perturbed by local noise and is close to the true position z_t^{k*} [15]. The function δ describes the deviation in dependence of the range. It is useful to use a function instead of a 'fixed' deviation to account for errors in the vehicle pose x_t . Unlike [1] other errors, those induced by random noise or specular reflections are not incorporated for two reasons. First, errors induced by specular reflections rarely happen in outdoor scenarios. Second, errors such as random noise are filtered out by the probabilistic fusion process of the occupancy grid methodology. The probability for each cell $m'_{k,i}$ for being occupied is then given by:

$$p_{occ}(m'_{k,i}|z_t^k) = \frac{\alpha(z_t^k)}{\sqrt{2\pi\delta(z_t^k)}} e^{-\frac{(m'_{k,i}-z_t^k)^2}{2\delta^2(z_t^k)}} \quad (6)$$

The parameter α serves as an attenuation or amplification factor. It reflects the amount of trust associated with a certain measurement with respect to the distance measured and is usually inversely proportional to z_t^k . As mentioned before, occupancy grids not only provide information about occupied space, but also about free space. Therefore it is necessary to define the cells that are considered to be free given the measurement z_t^k . Cells that are in front of the measurement z_t^k should have a substantially lower posterior. These are referred to as *freespace hypothesis areas*. For the probability distribution in the freespace area, the definition from [13] is adopted. They proposed a linear function which sets the probability with respect to the measured distance. This follows the assumption that small obstacles that might have been missed by the sensor are more likely to appear in far distances. We denote this function as $p_{free}(m'_{k,i}|z_t^k)$. Cells that are behind the measurement are considered to be unknown $p(m_{x,y} = 0.5)$. Putting this all together the final probability for each cell $m'_{k,i}$ is given by:

$$p(m'_{k,i}|z_t^k) = \begin{cases} \max(p_{free}, p_{occ}) & i < z_t^k \\ \max(0.5, p_{occ}) & i > z_t^k \end{cases} \quad (7)$$

The maximum operator is used to combine the probabilities in the transition region around z_t^k . For Bayesian probabilities, which are non-negative by definition, this operator can be formulated without any conditional branches (if statements).

$$\max(a, b) = \frac{a + b + |a - b|}{2} \quad (8)$$

As an example, Fig. 2 depicts the probability distribution for a single target in the 2nd row. Lower probabilities are

encoded in white, higher probabilities in black.

2) *GPU Computation of the Polar Measurement Grid:*

Given the final probability value for each cell $m'_{k,i}$ by (7), it can now be defined how to compute the polar grid m' on the GPU. With respect to the performance guidelines in Section II, each m'_k is assigned to a single thread block k . Since $m'_1 \dots m'_N$ are independent, they can be easily computed in parallel without having any performance loss by diverging threads. Moreover, the only information that has to be transferred from the CPU to the global memory of the GPU is the measurement vector z_t . This reduces the overhead to a minimum. Fig. 3 gives an example for a polar grid computed on the GPU with simulated measurements.



Fig. 3. Polar grid computed on the GPU with 70 simulated distance measurements and a depth resolution of 256 cells.

The proposed method can be applied to any kind of laser range-finders. For instance, modern lidar sensors for automotive environments often have the capability to detect multiple targets within a single beam. Furthermore, to compensate for pitch motion some have multiple vertical detection layers [16]. To account for these characteristics one can easily extend m' to m'^i and fuse them according to the Bayesian filter rule given in (4).

3) *Switch from Polar to Cartesian Space:* In order to combine the consecutive maps m' , gathered at different poses x_1, \dots, x_t , we need to map them into Cartesian space first.

$$m' \mapsto m^t \quad (9)$$

This is of course is not trivial and implies problems. The number of cells in m' that correspond to a single cell in m^t and vice versa directly depends on the position in the grid and the sensor intrinsics. Cells in m' with low r (close to the polar origin) map to just a few cells in m^t . For example, cells with $r = 0$ map to a single cell in m^t . In case of small obstacles extreme care has to be taken so that they are not canceled out by undersampling effects. Far from the polar origin a single cell of m' spreads over multiple cells in m^t . In the first case, specific cells of m' have to be merged in order to obtain the probabilities for the cells in m^t . In the second case, the cells in m' have to be interpolated to determine the correct values for m^t .

The authors of [8] were the first to investigate this subject. They considered the mapping problem as a geometric transformation problem. Namely the problem of mapping a geometric mesh in Polar space to its corresponding mesh in Cartesian Space. They compared different types of mapping algorithms: the *exact* algorithm which is an adaption of the Guibas and Seidel algorithm, the *sampling* approach and the *texture mapping* approach. The most interesting outcome of this analysis is the fact that the *texture mapping*

approach in comparison to the *exact* algorithm, which is not realtime capable, leads to quite accurate results with an acceptable amount of errors. The texture mapping approach is beneficial, since it is one of the basic operations that GPUs are designed for and accelerated by hardware. For an introduction to texture mapping the reader is referred to [17] or any other literature about computer graphics. Simplified, texture mapping describes the process by which an image is mapped to a target polygon. The polygon can have any shape and the texture is rotated, stretched and compressed so that it fits to it.

Let T be a texture that corresponds to the grid m' . Each cell $m'_{k,i}$ then maps to one texture element v_i . The target polygon P is simply given by the sensor intrinsics (perceptual field-of-view). Fig. 4 outlines a Polar grid in texture representation and its Cartesian equivalent as a polygon for a beam width of 10° . The fashion in which the resulting

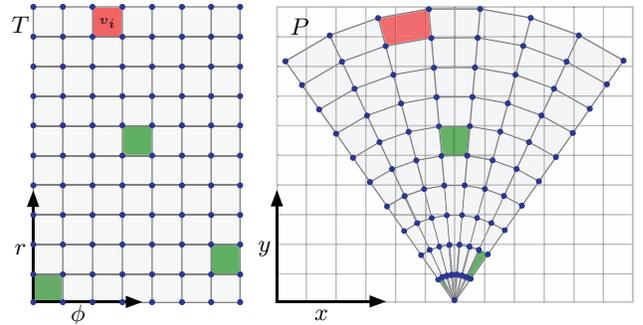


Fig. 4. Texture T in Polar space and its target polygon P in Cartesian space. The beam width is considered to be 10° .

pixels are calculated from the input image pixels is governed by so called texture filtering. This filtering is exactly what is required compute the final probabilities in m^t . A so called trilinear filter is used, which is a bilinear filter algorithm with mip-mapping [17]. In regions close to the Polar origin, where a single cell in m^t corresponds to multiple cells in m' , it avoids problems caused by undersampling. The trilinear filter fetches the mip-map at the right resolution and maps it to the target polygon. Fig. 5 shows the polar grid from Fig. 3 mapped to Cartesian space using texture mapping with trilinear filtering. Once the temporary Cartesian grid m^t has been computed, we have to combine it with the the previous measurements $p(m_{x,y}|z_1, \dots, z_{t-1})$. The definition from [13] is adopted, where the grid m is defined with respect to a global coordinate reference frame. This avoids discretization errors that occur when the grid is rotated with respect to a relative vehicle coordinate system. To account for translational ego movement, the grid m is shifted in a *sliding window* fashion. As previously mentioned, the texture mapping methodology simply allows to rotate the input texture with respect to the actual heading angle θ . To obtain the final occupancy up to timestep t , each overlapping cell of the grid m^t is fused with m using the binary Bayes filter from (4). This kind of cell-wise fusion is highly suitable for parallel processing and can be carried out on the GPU. Due to the definition in (2) each cell is independent of its

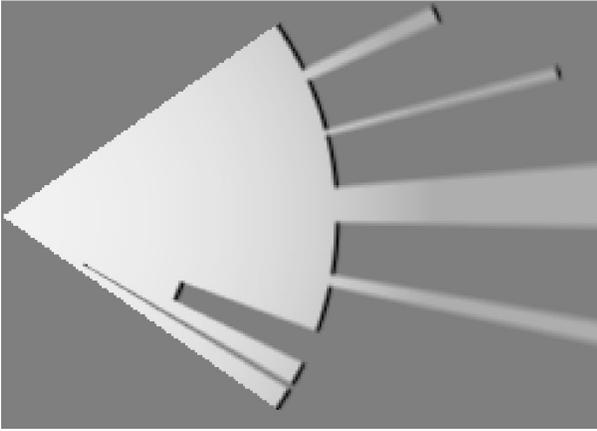


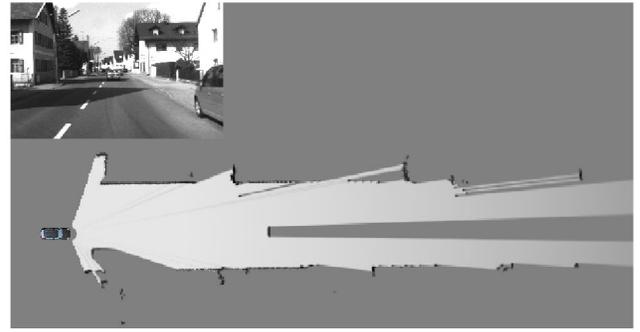
Fig. 5. Polar grid m' from Fig.3 mapped to Cartesian space. Each m'_k maps to a beam with a width of 1° .

neighboring cells. Both grids can be divided arbitrarily and then assigned to CUDA blocks. For performance reasons this division should not contain less than 256 *threads*. Thus a 16×16 or 1×256 assignment is convenient. However, when using the binary Bayes filter from (4) care has to be taken when $p(m_{x,y} \approx 1)$. To avoid numerical pitfalls, it is common practice to represent the probability $p(m_{x,y})$ in the so called *log odds* form [1]. This formulation, however, has a major drawback. It allows $p(m_{x,y})$ to take probabilities which are almost 0 or 1. For instance, if one cell was updated with freespace several times and is close to 0, then it takes as many counter updates with occupied to converge back to 0.5. In particular in large outdoor environments it can happen that some obstacles were missed by the sensor because their relative reflection angle was rather small. Later in time when the reflection angle changes they abruptly appear. Of course it is not desirable to miss these obstacles and the states that each cell can take have to be limited. In [13] a simple threshold test is performed. If the probability is lower or higher than a certain threshold the probability is set to ϵ . Unfortunately, this formulation is inconvenient on the GPU since it would cause the threads to diverge in their execution path. Therefore a different kind of ϵ test which works with a nested formulation of the min / max operator is defined.

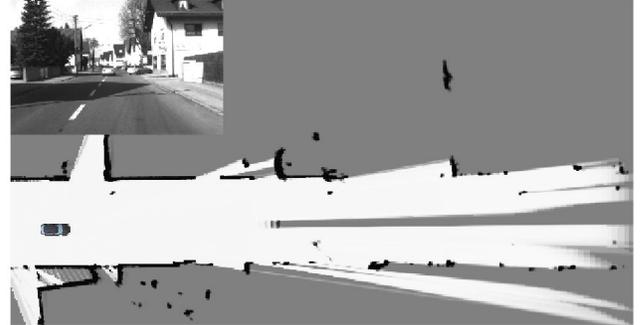
$$m'_{x,y} = \max(\epsilon, \min(1 - \epsilon, m_{x,y})) \quad (10)$$

C. Experimental Results

Fig. 6 gives as example a grid map computed with laser range-finder measurements in an urban scenario at a constant velocity of 12 m/s. The used device was a 4 layer automotive laserscanner with an opening angle of 140° . Fig. 6(a) shows the grid map after the first measurement update with the corresponding video image as reference in the upper left corner. The grid has a size of 512×512 cells with a resolution of 0.2 m. Fig. 6(b) shows the map after 30 measurement updates. The reader may have noticed the artifacts behind the vehicle in Fig. 6(b). They do not belong to any *real* objects and are caused by the violation of the static-world assumption in occupancy grid mapping. These artifacts can be removed or at least marked by the approach proposed



(a)



(b)

Fig. 6. Image (a), occupancy grid after the first measurement update. Image (b), after 30 updates at a constant velocity of 12 m/s.

in [14]. Table I depicts the achieved performance on two different GPUs, a Nvidia Quadro FX 1700M and a GeForce 285GTX. The first device has a total number of 4 multiprocessors and the second device 32. Since the complexity is independent of the probabilistic states in m , synthetic data has been used to evaluate the proposed algorithm.

Grid-size [cells]	$ z^t $	Layers	Avg. Time FX 1700 M [ms]	Avg. Time GTX 285 [ms]
256×256	200	1	4.13	3.21
256×256	400	2	4.58	3.43
512×512	200	1	7.22	5.39
512×512	400	2	7.75	5.8

TABLE I

ACHIEVED PERFORMANCE ON TWO DIFFERENT GPUS, A NVIDIA QUADRO FX 1700M AND A GEFORCE 285GTX.

D. Extension to Radar Sensors

Like laser range-finders, radar sensors measure distance and bearing. They usually have a lower horizontal field of view, angular resolution and distance accuracy than lidar sensors. However, due to the Doppler effect they do not only provide information about the distance but also about the relative speed of objects. This can be helpful in identifying and removing dynamic objects in the static map.

The proposed algorithm was tested with a radar sensor which is similar to the one used in [18]. This scanning device covers a horizontal field of view of 17° with a maximum distance of 200 m. The radar delivers the distance and

Doppler information in two Polar images (Fig. 7). The first

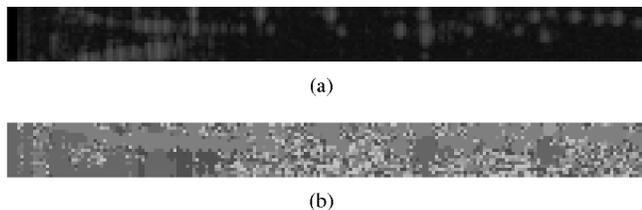


Fig. 7. Example for an amplitude (a) and frequency image (b) of the radar sensor. Both images have a size of 200×17 pixels.

image (Fig. 7(a)) contains information about the presence of objects. The brighter the value, the more likely it is that this region contains an object. As in [7] the signal-to-noise ratio can be used to calculate the probabilistic weights for the measurement update. The second image (Fig. 7(b)) contains information about the relative speed of each cell in the first image. All cells which do not correspond to our negative ego velocity can be ignored, since they are considered to belong to dynamic objects. Furthermore, it is assumed that the noise in both images is Gaussian. Putting all of this together, a grid which is similar to the Polar grid in Fig. 3 is obtained. This grid can be simply mapped to Cartesian space using the *texture mapping* method. Fig.8 shows as example an occupancy grid in a road construction area.

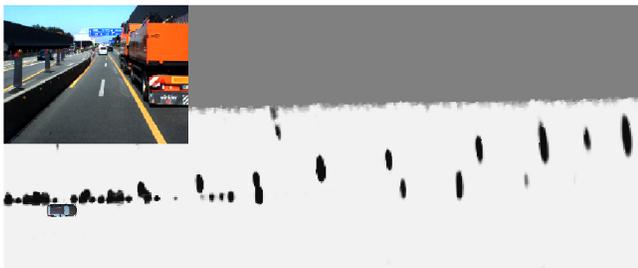


Fig. 8. Example for an occupancy grid computed with a radar sensor in a road construction area. It is clearly visible that the lateral uncertainty, induced by the low angular resolution, has been taken into account.

IV. HISTOGRAM-BASED ROAD BOUNDARY DETECTION USING OCCUPANCY GRIDS

The information about the position of the vehicle itself on the road is a basis of several functionalities. One of the most famous is the general lane/road keeping task. The key challenge of this task is the robust and accurate estimation of the ego position with respect to the road. A popular approach is the detection of lane markings. Once they have been extracted it is assumed that they define the drivable path on the road. In the past decades, extensive research has been done in the field of lane detection. Most authors use vision to extract the lane markings [19], some others use laserscanners [20]. In the case of white border lines these approaches achieve good results. However, inconspicuous or missing lane markings and complex driving environments (road construction areas) make the road keeping task using lane markings alone extremely difficult. Beside lane

markings, another suitable feature to determine the driving corridor are continuous road boundaries like guard rails, jersey barriers or curbs. If the distance of the road boundaries to the lanes is known in advance, like for instance from a digital map [21], they can be used to determine the exact position on the road.

A. Road Boundary Model

Similar to lane detection, one is mainly interested in an abstract representation of the boundary that can be directly used as input for a controller or in a fusion system. Like in [20] a circular model to describe the road (Fig. 9) is used. This model comes with the minimum set of parameters to approximate a road which usually consists of lines, clothoids and circles. Each road boundary can be uniquely described by the state vector:

$$x_i = [c \ \gamma \ b \ y]^T \quad (11)$$

where c is the curvature, b the width of the road boundary, y the offset and γ the heading angle with respect to the vehicle coordinate system.

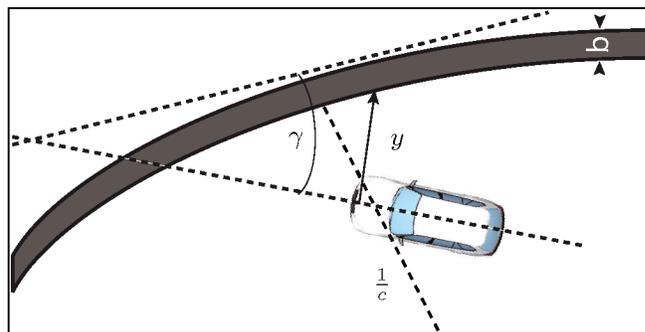


Fig. 9. Road boundary model with curvature c , offset y , heading γ and width b .

B. Histogram-Based Parameter Extraction

The goal of the histogram-based parameter extraction is to find all relevant¹ road boundaries in the occupancy grid. Let

$$\mathbb{P}_R = [c_j \ \gamma_k \ b_l \ y_m]^T \quad (12)$$

be the set of all possible road boundaries. There exists a subset $X \subset \mathbb{P}_R$ that includes all relevant road boundaries in m . The key idea of the proposed algorithm is to represent \mathbb{P}_R as a corresponding set of histograms over m where each variation of c_j and γ_k defines a single histogram H_i . Each bin $k_x(H_i)$ contains the normalized accumulated sum over the probabilities in m along a circular path given by the curvature c_j and heading γ_k . For instance, if $c_j = \gamma_k = 0$ the circular path is equivalent to a row or column wise histogram binning. The position of a bin k_x in the histogram H_i is defined by the discrete offset y_m . Fig. 11(a) shows the corresponding histogram to the occupancy grid in Fig. 10. For convenience, a case was chosen where the curvature and heading for all relevant boundaries in m is equivalent. The

¹Boundaries that contain information about the path of the road

depicted histogram H_0 in Fig. 11(a) is denoted as *optimal histogram*, if there exists no other parameter set in P_R , that gives a better approximation of the road boundaries in m . In the given example $H_{opt} = H_0$ is true, but usually neither the curvature of the boundary nor the heading to the boundary is 0. Hence, the histogram H_0 would not be the optimum. Fig. 11(b) and 11(c) outline what happens to the histogram H_{opt} if a small error is added to the curvature c .



Fig. 10. Example for an occupancy grid in a freeway scenario. The left and right road boundaries are clearly visible.

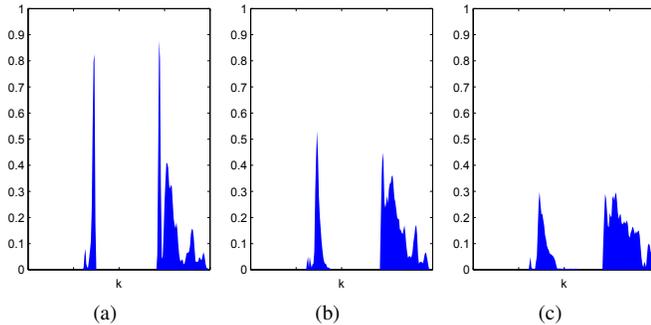


Fig. 11. In (a), the optimal histogram H_{opt} with $c_j = 0$. In (b), an additive curvature error of $0.003m$. In (c), $0.008m$

With an increasing error in the curvature of the circular path the histogram H_{opt} degenerates to a flat uniform distribution. This is because the road boundaries in m now distribute among multiple bins of H_i instead of falling into a specific bin k_x . Similarly for γ , each deviation of the optimal heading parameter causes a degeneration of the histogram H_{opt} . This fact can be exploited to define a cost function $q(H_i)$ which rates each histogram according to its distribution.

$$q(H_i) = \sum_{x=-d}^d k^2(x, c_j, \gamma_k) \quad (13)$$

The sum over the squares of k_x is used to give a higher weight to histograms which have all probabilities distributed in a minimum set of bins. Fig. 12 shows the results of the cost function (13) for different variations of curvature and heading. It is clearly visible that there exists a maximum $q(H_{opt})$

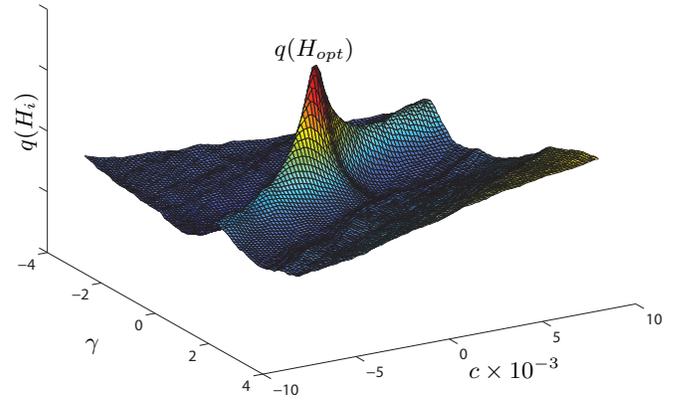


Fig. 12. Results of the cost function $q(H_i)$ for $\gamma = -4^\circ \dots 4^\circ$ and $c = -0.0001m \dots 0.0001m$

which represents the optimal parameter combination of c and γ . Thus, the problem to find the heading and curvature of the road boundaries becomes a parameter optimization problem. Due to the computational burden it is not advisable to evaluate the cost function q over all possible combinations of c_j and γ_k . A modified version of the iterative Nelder-Mead-Simplex optimization algorithm [22] is used to obtain the optimal histogram H_{opt} . At each iteration the algorithm computes a set of 3-4 histograms in a Simplex order and rates them according to the cost function (13). After a sufficient number of iterations the Simplex algorithm converges to the maximum and remains stable. The Nelder-Mead-Simplex algorithm is fast, deterministic and robust to local minima. Once the algorithm has converged to the maximum, a total number of 5-8 iterations at each timestep is sufficient to keep track of the curvature and heading of the road boundary. In order to account for different curvatures and headings of the left and right boundary the grid m can be separated into two parts. For each part a histogram optimization can be carried out to obtain the most likely boundaries for the left and right side.

Once the curvature and heading of the dominant road boundary has been found, the distance y and width b can be extracted from H_{opt} . To do so, one has to define a criteria which bin k_x belongs to a valid boundary. As in [22] the signal-to-noise ratio in a predefined vicinity of k_x is used to rate each bin with respect to its adjacent bins. This is equivalent to an adaptive threshold and can be used to decide whether k_x is a valid road boundary or not. One of the advantages of the histogram based approach is that it becomes possible to estimate the exact width b of the boundary. This is of high importance to distinguish between road borders that are suitable for high-precision ego localization and those that are not. Furthermore, once the road boundaries have been identified we can interpolate between k_{x-1} , k_x and k_{x+1} to obtain an accuracy for y which is higher than the previously chosen cell resolution of m . Since the accumulated measurements in m are Gaussian, H_{opt} is Gaussian too and a performed *sub-pixel* computation between adjacent bins (circular paths) can improve the accuracy by a factor of 2-3.

C. Experimental Results

Fig. 13 shows the histogram-based detection of the left road boundary in a road construction area. The used occupancy grid has a size of 512×512 with a front view of 468 and back view of 48 cells. The cell resolution was set to 0.25 m. Fig. 14 outlines the results for the

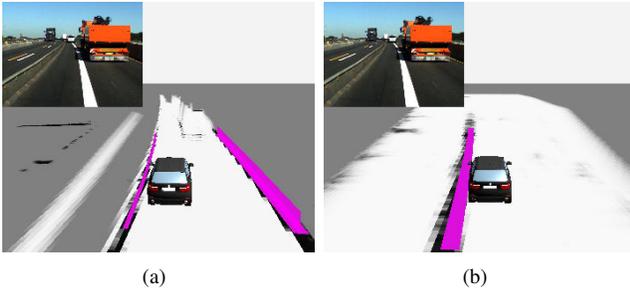


Fig. 13. Histogram-based detection of the left road boundary with a laserscanner in (a). In (b), the same scenario with a radar sensor.

histogram based extraction of the offset y in comparison to raw measurements of a reference laserscanner with a distance accuracy of ± 5 cm. Even in the case of the lower resolution radar sensor, where the road boundary spreads over a width of approximately 0.75 – 1.25 m, the achieved accuracy is comparable to the accuracy of the raw measurements.

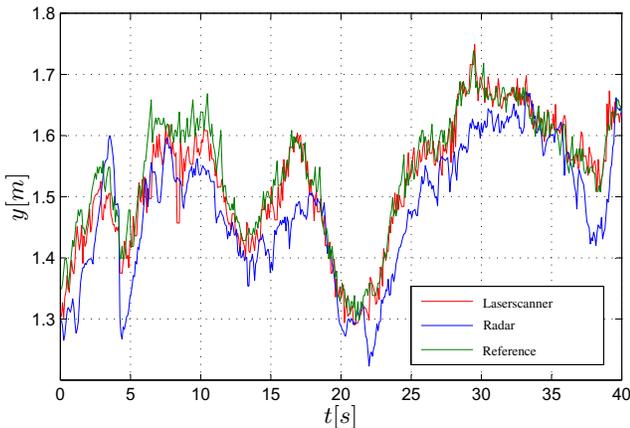


Fig. 14. Achieved accuracy of the proposed algorithm for the calculated offset to the left road boundary in the road construction scenario from Fig. 13 in comparison to raw measurements of a laserscanner as reference.

V. CONCLUSION

A method for the fast computation of occupancy grid maps on the GPU was introduced. It has been shown that the GPU based algorithm can generate grid maps in large outdoor environments in just a few milliseconds without using any specialized hardware devices such as FPGA. Additionally, a novel approach for the detection of continuous road boundaries was developed. In the case of a laser-scanner it was shown that in comparison to raw measurements, the proposed occupancy grid comes without any loss in sensor precision. Furthermore, it has been proven that in the case of a lower resolution radar sensor the approach is suitable to

apply super-resolution algorithms to achieve the accuracy of a higher resolution laser-scanner.

REFERENCES

- [1] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.
- [2] P. Stepan, M. Kulich, and L. Preucil. Robust data fusion with occupancy grid. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 35(1):106–115, 2005.
- [3] Michael Himmelsbach, Felix von Hundelshausen, and Hans-Joachim Wünsche. LIDAR-Based Perception for Offroad Navigation. In *Proceedings of FAS 2008, Fahrerassistenzsysteme Workshop 2008*, Waltham, Germany, April 2008. C. Stiller and M. Maurer.
- [4] G. Grisetti, C. Stachniss, and W. Burgard. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Proc. IEEE International Conference on Robotics and Automation ICRA 2005*, pages 2432–2437, 18–22 April 2005.
- [5] C. Chen, C. Tay, C. Laugier, and K. Mekhnacha. Dynamic environment modeling with gridmap: A multiple-object tracking application. In *Proc. 9th International Conference on Control, Automation, Robotics and Vision ICARCV '06*, pages 1–6, 5–8 Dec. 2006.
- [6] Uwe Franke Hernan Badino and Rudolf Mester. Free space computation using stochastic occupancy grids and dynamic programming. In *Dynamic Vision Workshop for ICCV*, 2007.
- [7] S. Lüke M. Darms, M. Komar. Gridbasierte strassenrandschätzung für ein spurhaltesystem. In *Proceedings of FAS 2009, Fahrerassistenzsysteme Workshop 2009*, 2009.
- [8] M. Yguel, O. Aycard, and C. Laugier. Efficient gpu-based construction of occupancy grids using several laser range-finders. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 105–110, Oct. 2006.
- [9] NVIDIA Corporation. *CUDA Programming Guide 2.2*, <http://www.nvidia.com>, 2009.
- [10] V. Vineet and P.J. Narayanan. Cuda cuts: Fast graph cuts on the gpu. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops CVPR Workshops 2008*, pages 1–8, 23–28 June 2008.
- [11] Y. Mizukami and K. Tadamura. Optical flow computation on compute unified device architecture. In *Proc. 14th International Conference on Image Analysis and Processing ICIAP 2007*, pages 179–184, 2007.
- [12] Jing Huang, Sean P. Ponce, Seung In Park, Yong Cao, and Francis Quek. Gpu-accelerated computation for robust motion tracking using the cuda framework. In *Proc. 5th International Conference on Visual Information Engineering VIE 2008*, pages 437–442, July 29 2008–Aug. 1 2008.
- [13] T. Weiss, B. Schiele, and K. Dietmayer. Robust driving path detection in urban and highway scenarios using a laser scanner and online occupancy grids. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 184–189, 2007.
- [14] Trung-Dung Vu, O. Aycard, and N. Appenrodt. Online localization and mapping with moving object tracking in dynamic outdoor environments. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 190–195, 13–15 June 2007.
- [15] Kurt Konolige. Improved occupancy grids for map building. *Auton. Robots*, 4(4):351–367, 1997.
- [16] Dietmayer K. Fürstenberg, K. Fahrzeugumfelderfassung mit mehrzeiligen laserscannern. In *Journal Technisches Messen 71*, 2004.
- [17] Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008.
- [18] Chris Urmson. Tartan racing: A multi-modal approach to the darpa urban challenge. Technical report, Carnegie Mellon University, 2007.
- [19] R. Risack, P. Klausmann, W. Krüger, and W. Enkelmann. Robust lane recognition embedded in a real-time driver assistance system. In *Proc. IEEE IV*, pages 35–40, 1998.
- [20] A. Reyher. *Lidarbasierte Fahrstreifenanzuordnung von Objekten für eine Abstandsregelung im Stop&Go-Verkehr*. PhD thesis, Technischen Universität Darmstadt, 2006.
- [21] Kaempchen N. Homm F. Waldmann P. Ardel M. Umfelderkennung für den nothalteassistenten - ein system zum automatischen anhalten bei plötzlich reduzierter fahrfähigkeit des fahrers. In *11. Braunschweiger Symposium, AAET 2010*, 2010.
- [22] Homm F. Duda A. Kämpchen N. Waldmann P. Ardel M. Lidar based lane detection with occupancy grids for lane keeping and lane change assist systems. In *4. Tagung Sicherheit durch Fahrerassistenz*, 2010.