Parallelized Energy Minimization for Real-Time Markov Random Field Terrain Classification in Natural Environments

Marcel Häselich, Simon Eggert and Dietrich Paulus
Active Vision Group
University of Koblenz-Landau
Universitätsstr. 1, 56070 Koblenz, Germany
{mhaeselich,george,paulus}@uni-koblenz.de

Abstract— **Regarding collision avoidance, precise track following, a robust and fast obstacle detection, and surface information extraction, terrain classification in unstructured natural environments is a complex and challenging task. In our scenario, an autonomous ground robot has to explore rough terrain without any knowledge of underlying road networks, as they could be available in form of a given map. Therefore, the robot has to detect surface conditions and obstacles by itself in real-time. Our robot uses the data of a 3D Laser range finder to perceive the environment.**

Due to the movement of the robot over rough terrain or sensor noise, the perceived information may be disturbed or erroneous. Markov random fields allow an incorporation of neighborhood information to improve robustness against such influences. Our previously published terrain classification algorithm achieves recall ratios of about 90 % for detecting obstacles and drivable areas. The system operates in real-time at distances of up to 20 m. In order to further increase the processible area and therefore enable safe navigation at higher speeds, we seek to optimize the system performance using multicore CPU and GPGPU hardware.

In this paper, we extend our previous terrain classification approach and present runtime optimizations for our Markov random field energy minimization via Gibbs sampling. Besides memory optimizations, we present a solution to parallelize the terrain energy minimization using multiple CPU threads. Further, we introduce a parallelization performed on the graphics card. While maintaining the aforementioned high recall ratios, we are able to accelerate energy minimization by a factor 10 which allows us to process a 2.5 times larger area in real-time.

I. INTRODUCTION

In natural environments, classification of the terrain surface is essential for autonomous robots in order to plan optimal and safe routes. Modern 3D laser range finders (LRFs) provide a detailed and thorough representation of the scenery. This representation allows mobile systems to extract precise terrain surface information as a basis for path planning. Our approach uses the data of a *Velodyne HDL-64E S2* LRF for this task. The head of this 3D LRF consists of 64 lasers which permanently gather data of the environment as the head spins at a frequency of up to 15 Hz around the upright axis. In doing so, the sensor gathers a rich dataset of up to 1.8 million distance measurements per second. The data of one full rotation are accumulated into one point cloud. Besides the LRF, our robot has several other sensors at its disposal, which are omitted in this work since they are dispensable.

In order to deal with sensor noise and percussions caused by the robot moving over rough terrain, we use a Markov random field (MRF) for the terrain classification task. The MRF integrates the information of adjacent regions into the classification process. As a result, our MRF approach achieves recall ratios of about 90 % for detecting obstacles and drivable areas (cf. [7]).

For each point cloud, our MRF needs around 63 ms (max. 82 ms) for a $40 \times 40\,\mathrm{m}^2$ grid. Most of this time is consumed by the optimization step in which the optimal labeling of the terrain regions is ascertained. The Velodyne HDL-64E S2 has a range of up to 120 m and within a distance of 50 m around the robot, sufficient distance measurements for terrain classification are gathered. Considering these properties, a grid size of $100 \times 100\,\mathrm{m}^2$ is desirable. Its frequency of up to 15 Hz corresponds to a minimal update time of 67 ms for each point cloud gathered in a full rotation of the sensor.

Therefore, we need to accelerate the energy minimization to increase the size of the terrain and to ensure a terrain classification with a MRF in real-time. A longer range results in a more foresightful behavior of a path planning algorithm and an improved runtime makes collision avoidance faster and therefore more secure at higher speeds. This paper describes our implementations of different runtime optimizations to achieve this specific goal.

The paper is organized as follows. First, we present related work in Sec. II. After an brief overview of our MRF terrain classification approach in Sec. III, we focus on the energy minimization and its runtime optimization in Sec. IV. Experimental results are presented in Sec. V and the paper closes with a conclusion in Sec. VI.

II. RELATED WORK

Terrain classification in unstructured environments is an active research topic. Besides camera-based approaches ([1], [9]), the usage of LRFs has become quite popular. Modern LRFs gather rich point clouds of the surrounding areal which are well suited for surface structure analysis and obstacle detection.

The 3D distance measurements of a moving 2D scanning mount on a mobile platform are classified into the three different terrain classes *surface*, *clutter*, or *wires* by Vandapel et al. [14]. Their laser gathers 100.000 measurements per second which are classified every 3 seconds or after a traversed distance of 7 meters.

Wolf et al. [16] present a terrain classification approach where the data of pitched down 2D LRF are classified into *flat areas*, *grass*, *gravel*, and *obstacles*. The approach is able to process the 14.000 data points of their LRF in real-time. Another pitch down 2D LRF with a similar resolution is used by Ye and Borenstein [17]. Their robot creates elevation maps while driving in indoor scenarios.

A color stereo camera and a 2D LRF are used by Manduchi et al. [10] to detect obstacles in real-time. The authors present a color-based classification system and an algorithm that analyses the laser data to discriminate between grass and obstacles. Furthermore, Vernaza et al. [15] present a camera based terrain classification approach for the DARPA LAGR program. Their approach uses a MRF that classifies image data of a stereo system into obstacles or ground regions for an autonomous robot in real-time.

The energy minimization step of our MRF terrain classification approach consumes the most runtime. There exist approaches to reduce the computational load of this step in the literature. Considering an acceleration of the MRF energy minimization, Gonzales et al. [5] propose two methods to construct parallel Gibbs samplers. One method uses graph coloring to construct a direct parallelization of the sequential Gibbs sampler and the other method is a complementary strategy which can be used when variables are tightly coupled.

Martens and Sutskever [11] introduce a Markov chain transition operator that updates all the variables of a pairwise MRF in parallel by using auxiliary Gaussian variables. Their experiments show that the results are comparable to the sequential Gibbs sampler in terms of mixing speed, while being simple to implement and to parallelize.

Pock et al. [13] propose a convex relaxation approach for the Potts model. Their approach uses an efficient primal dual projected gradient algorithm which also allows a fast implementation on parallel hardware but does not guarantee to find global minimizers.

Kato et al. [8] present a multiscale MRF model that consists of a label pyramid in which the optimization is first performed at higher scales by a parallel relaxation algorithm. Then, the next lower scale is initialized by a projection of the result. On the one hand, the authors introduce an interaction scheme between two neighbor grids in this label pyramid and show that these connections allow to propagate local interactions more efficiently, yielding faster convergence in some cases. On the other hand, these interactions make the model more complex, demanding computationally more expensive algorithms.

The energy minimization of a MRF can be done with a variety of algorithms with different advantages and disadvantages. Unlike image segmentation, it is essential in our domain to be able to maintain certain terrain cells even if they are an antipode to their whole neighborhood. Typical examples for such terrain cells in a natural environment are single small trees, separate tall but thin rocks, or poles on a field. Approaching these obstacles, it is essential for our robot that these single terrain cells remain obstacles during the energy minimization in order to guarantee a collision-free navigation. For this reason we choose a Markov chain Monte Carlo method, the Gibbs sampler, to minimize the energy of our MRF. Our goal is to process the large point clouds of 120.000 3D measurements in less than 67 ms to fulfill a hard real-time criterion defined by our sensor update frequency. In the following section, we briefly introduce our terrain classification approach before we present the proposed acceleration strategies of our sampling algorithm.

III. Markov Random Field Terrain Classification Approach

The large 3D point clouds of the Velodyne are unfeasible to process in real-time without an efficient data representation. Therefore, we use a 2D equidistant grid centered around the origin of our LRF to represent the data. Our grid has a variable size from $40 \times 40\,\mathrm{m}^2$ up to $100 \times 100\,\mathrm{m}^2$. Within the grid, each cell C has a size of approximately $50 \times 50\,\mathrm{cm}^2$. The following section introduces our previously published MRF terrain classification as basis for our optimization in the subsequent section. For a more detailed description on the MRF model and classification, we refer the reader to [7].

For easier computation and modeling, MRFs are often used as Gibbs random fields. As proven by the *Hammersley-Clifford Theorem* ([2], [6]), MRFs and Gibbs random fields (GRFs) can be seen equal. In a GRF, the distribution of the random states corresponds to the Gibbs distribution, which can be optimized by finding an optimal configuration that minimizes the energy of the GRF. In our previous publication, we presented a MRF that works directly on these terrain cells. We chose a MRF model introduced for image segmentation tasks by Deng and Clausi [3]. The neighborhood of each cell consists of the eight adjacent cells (except for border cells) and the neighborhood relations are modeled according to the Potts model. Cell features are computed from the data of the Velodyne. We introduced the cell classes *road* and *rough* to determine between easy or hard to pass cells. The class *obstacle* represents impassable areas for our robot. Regions without measurements are also relevant, since they could contain potholes or descents and are represented by the class *unknown*.

Let i, j index a cell position in the terrain grid. The neighborhood energy of a cell computed according to the Potts model is denoted by $\mathrm{E}_{\mathcal{N}_{i,j}}$ and the feature energy is denoted by $\mathrm{E}_{f_{i,j}}$. Then, the complete energy $\mathrm{E}_{i,j}$ of the cell at position i, j is calculated as

$$\mathrm{E}_{i,j} = \mathrm{E}_{\mathcal{N}_{i,j}} + \alpha \cdot \mathrm{E}_{f_{i,j}}, \tag{1}$$

where α is a weighting constant used to control the influence of the different energy types. Note that α is fixed to 0.8 for all experiments conducted in the context of this work. The neighborhood of a terrain cell C at position i, j for the first-order neighborhood system is given by

$$\begin{aligned} \mathcal{N}_{i,j} = \{&C_{i-1,j-1}, C_{i-1,j}, C_{i-1,j+1}, C_{i,j-1}, \\ &C_{i,j+1}, C_{i+1,j-1}, C_{i+1,)}, C_{i+1,j+1}\} \end{aligned} \tag{2}$$

with respect to border cells. The neighborhood energy is computed as

$$\mathrm{E}_{\mathcal{N}_{i,j}} = \sum_{C_{m,n} \in \mathcal{N}_{i,j}} \delta(C_{i,j}, C_{m,n}), \tag{3}$$

where the function $\delta(C_{i,j}, C_{m,n})$ returns a value of -1 if the classes of the two cells are equal, else $+1$, which is in compliance with the Potts model.

The feature energy $\mathrm{E}_{f_{i,j}}$ of a cell is based on our assumption that the features are Gauss distributed. Its computation is defined as

$$\mathrm{E}_{f_{i,j}} = \sum_k \left(\frac{(f_{i,j_k} - \mu_{i,j_k})^2}{2\sigma_{i,j_k}^2} + \log\left(\sqrt{2\pi}\sigma_{i,j_k}\right) \right), \tag{4}$$

where f_{i,j_k} is the k-th feature of $C_{i,j}$ and μ_{i,j_k} and σ_{i,j_k} are the mean respectively the standard deviation of the k-th feature of the class assigned to $C_{i,j}$. Mean and standard deviation of all features need to be computed in advance for all classes which we solved by an initial learning. For example, we use the local height disturbance [12] of the laser measurements as a feature.

For classification, the sum of all computed energies $\mathrm{E}_{i,j}$ needs to be minimized, which leads to a maximization of the a posterior probability of the labeling of the terrain. These energies are minimized by finding a label for each cell which fits best for the computed features *and* the labels of the neighbor cells. We apply the Gibbs sampler described by Geman and Geman [4] to solve this task. An example result is shown in Fig. 1.

Once the model is defined, the task of the Gibbs sampler is to assign class labels to each cell so that the energy over the complete grid is minimized. In order to accelerate this cost-intensive step, we describe our runtime optimizations for energy minimization of our MRF in the next section.

IV. Optimization

The energy minimization of our MRF published in [7] uses a straightforward implementation of the classical Gibbs sampler. Thereby, it is able to classify a terrain of $40 \times 40\,\mathrm{m}^2$ in real-time. Since we want to increases the range of the terrain grid to $100 \times 100\,\mathrm{m}^2$, our goal is to accelerate the energy minimization so that the algorithm is able to meet a hard real-time criterion for the extended grid.

At this point, two different approaches arise to improve the Gibbs sampler. The first idea is a modification of the sampling algorithm itself, as proposed by other authors (see [11], [13]). On the other side, our terrain grid representation allows a subdivision into smaller terrain pieces, which can than be processed by an unmodified Gibbs sampler. Though each of the sub-grids will suffer the burn-in phase of the sampling process, the parallelization opportunity especially on the graphics card makes this approach very interesting. Further, this method has the advantages that the ergodicity of the original algorithm is obtained without any interventions and the implementation is easy to realize.

In the following section, we first describe the process of our initial straightforward implementation Gibbs sampler followed by the three new contributions in separate subsections.

The original algorithm takes the initial grid and a temperature as input values and runs in a loop until convergence. It iterates over the width and height of the terrain given by the number of terrain cells in each direction. Within the loop, it computes the neighborhood and feature energies according to Eq. 3 and Eq. 4, and computes the energy of a possible label change for each class. This computation uses the exponential function of the negative energy (cf. [3]) divided by the current temperature value, which is decreased at the end of each iteration. Afterwards, random numbers are used to sample from the target distributions and a new class is chosen for each cell with respect to its energy value in case it yields to a minimization of the energy. This approach is used in [7] and will be evaluated as *Basic*-algorithm in Sec. V.

A. Feature Computation Optimization

Our first optimization concerns the energy computation step on the part of the feature energy. The energy is computed as sum of the neighborhood and the feature energy according to Eq. 1 and computes both energy values of a cell during runtime. In contrast to the neighbors, the features extracted from the 3D points contained in a cell won't change within the loop respectively within one sensor point cloud. Hence, it is more efficient to compute the features in advance and to access the result during runtime. This improvement will be evaluated as *Feature*-algorithm in Sec. V. For example, for a terrain grid of $100 \times 100\,\mathrm{m}^2$ and a cell size of around $50 \times 50\,\mathrm{cm}^2$, this results in a memory usage of 40.000 floating point values. In addition, this acceleration is independent of available hardware and will be used by both parallelization strategies in Sec. IV-B and Sec. IV-C.

B. CPU Parallelization

In order to take advantage of multiple CPUs, we split the terrain into slices. Hence, we create a number of slices corresponding to the number of terrain rows divided by the number of available threads. Each thread handles the amount of terrain cells in each row assigned to it. For the terrain of $100 \times 100\,\mathrm{m}^2$ and a CPU with 8 cores for example, we create 8 threads each handling 25 terrain rows with 200 cells of $50 \times 50\,\mathrm{cm}^2$ in each row. An exemplary illustration is given in Fig. 2. This way, the whole terrain

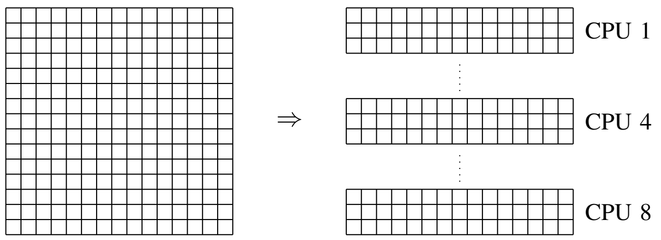

Fig. 2. The image shows how the terrain grid is divided for the multi-core CPU. Each thread handles an equal amount of full rows.

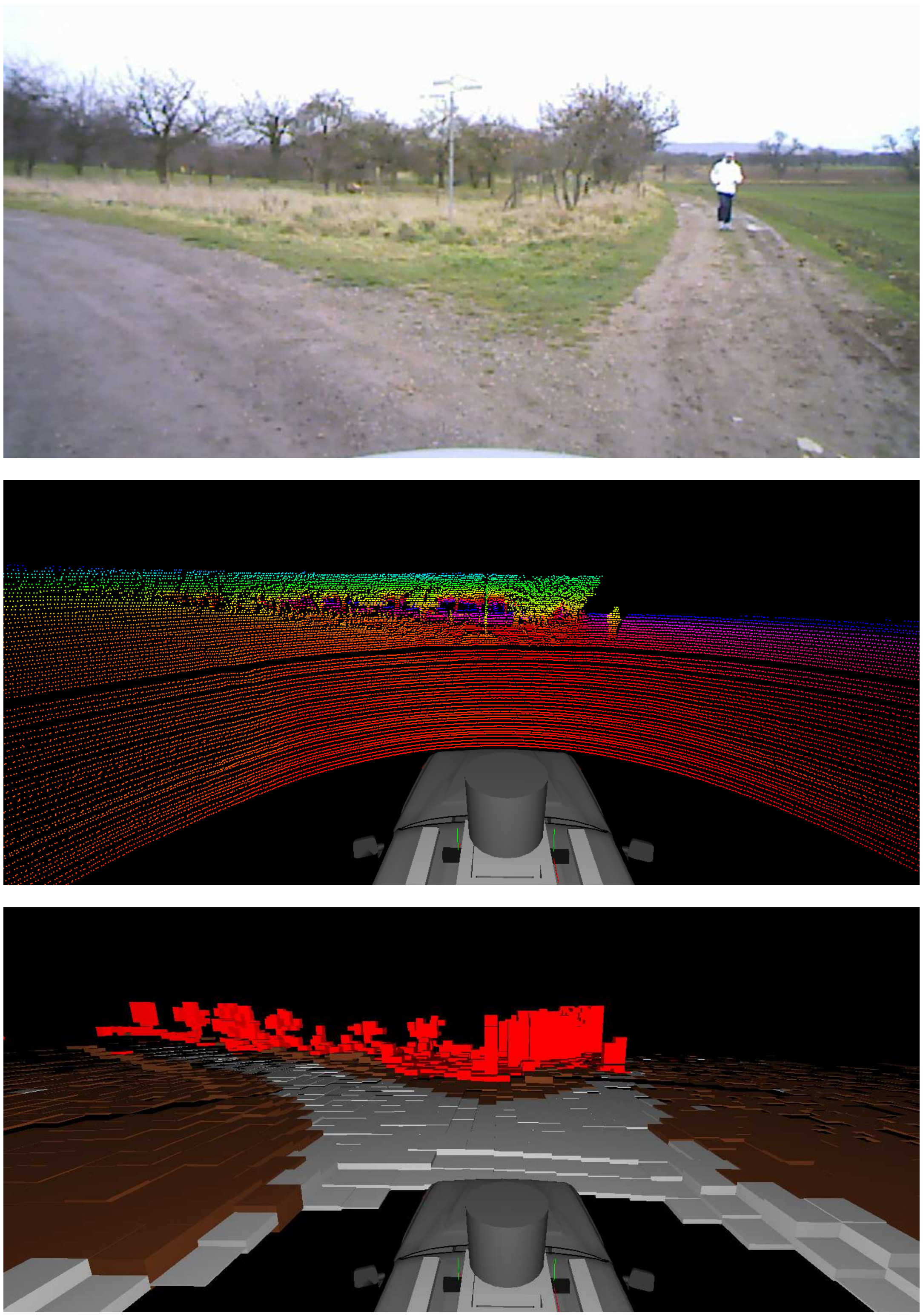

Fig. 1. Example result of the terrain classification. A camera image taken from the perspective of the robot is shown in the upper part. The image in the middle shows the corresponding 3D point cloud delivered by the Velodyne HDL-64E S2, where the displayed *points* are colored according to their height. In the lower image, the assigned terrain classes are visualized. Cells displayed in *red* represent the class *obstacle*, *gray* cells correspond to the class *street* and *brown* cells stand for the class *rough*. Regions without measurements are shown in *black* and represent the class *unknown*.

grid is processed at once by the multi-core processor. In avoidance of accessing neighbors of different instants of time, the terrain is copied and updated at the beginning of the while loop and each thread retrieves the neighborhood information from the current copy. For the implementation, we used the boost library [1]. Our platform is able to handle up to 8 threads simultaneously and the CPU parallelization will be evaluated as *Multicore*-algorithm in Sec. V.

C. GPU Parallelization

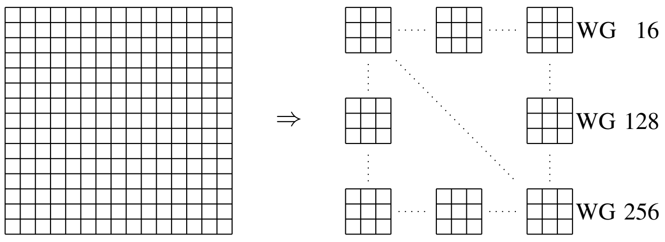

Fig. 3. Similar to the CPU approach visualized in Fig. 2, the cells of the terrain grid are partitioned into equal parts. In contrast to multiple CPUs, the hardware-limited amount of work-items requires the 64 GPUs to use 4 clock cycles before all 256 work-groups (WGs) are processed.

Graphics processing units (GPUs) are high-performance many-core processors that can be used to efficiently accelerate applications. In contrast to CPU parallelization, various properties of the graphics card play an important role and need to be considered carefully for an optimal implementation. For our implementation, we used the OpenCL 1.1 framework. To take advantage of our GPU, we divide the terrain grid into squared work-groups. The work-groups are executed in clock cycles, one work-group per cycle per GPU core. A large work-group size is able to hide the memory latency while a small one decreases the amount of active threads. The amount of work-items within one work-group is restricted for each graphics card. Therefore, the optimal work-group size needs to be adjusted for each GPU individually to achieve the best performance. The Gibbs sampler is implemented as a GPU kernel that is executed for every work-item with the exception that the loop and the steering of the temperature remains on the CPU. Fig. 3 shows how the work groups are arranged to process the terrain. Since our graphics card limits the maximum number of work-items, the GPUs will not be able to process the complete terrain in one cycle. In fact, we need a total of 4 cycles to process all terrain cells which results in an interesting comparison, since the *Multicore*-algorithm is able to process the whole terrain at once. Besides the splitting of the terrain into quadratic work-groups, we use the local memory of each GPU core and copy relevant terrain cells to this fast GPU memory. This copying is done in parallel and the local memory of the GPU allows faster computations during runtime. For a terrain of $100 \times 100\,\text{m}^2$ with our nVidia (R) Quadro (R) FX 880M for example, we use 256 work-groups each with 144 work-items that are processed by 48 CUDA (TM) cores. This general-purpose computing on the graphics processing unit will be evaluated as *GPGPU*-algorithm in Sec. V.

[1]http://www.boost.org

[2]http://www.khronos.org/opencl

TABLE I

RUNTIME RESULTS OF THE DIFFERENT OPTIMIZATIONS. MEAN VALUES THAT MEET OUR SOFT REAL-TIME CRITERION AND MAXIMA THAT MEET OUR HARD REAL-TIME CRITERION ARE DENOTED BY AN ASTERIX.

Algorithm	Grid size [m×m]	Mean [ms]	Std. dev. [ms]	Min [ms]	Max [ms]
Basic	20×20	15.1*	1.0	11.3	25.8*
	40×40	51.2*	2.1	40.2	116.9
	60×60	102.8	2.4	82.8	162.3
	80×80	165.2	3.0	140.5	282.0
	100×100	248.3	3.4	219.3	387.7
Feature	20×20	8.1*	0.7	5.8	14.4*
	40×40	32.6*	1.4	26.8	57.1*
	60×60	75.6	1.9	66.1	123.5
	80×80	125.1	2.2	108.6	161.2
	100×100	206.2	3.1	185.1	307.8
Multicore (8 CPU cores)	20×20	3.4*	0.3	1.9	8.9*
	40×40	13.6*	0.8	8.4	30.6*
	60×60	28.1*	1.2	18.3	46.2*
	80×80	43.8*	1.4	31.6	80.8
	100×100	67.4	1.6	54.2	124.2
GPGPU (48 CUDA (TM) cores)	20×20	4.1*	0.2	3.5	12.0*
	40×40	6.2*	0.3	5.1	14.5*
	60×60	11.0*	0.3	9.7	27.7*
	80×80	16.1*	0.6	14.1	39.5*
	100×100	23.6*	0.9	20.5	43.0*

V. EXPERIMENTAL RESULTS

The evaluation was performed on a laptop with an Intel(R) Core(TM) i7 QM with 1.73 GHz and 8 GB RAM and an nVidia (R) Quadro (R) FX 880M. The MRF classification was performed on real sensor data of urban, rural, and forest scenarios. During the experiments, the cell size and all parameters were fixed. The performance of the implementations is displayed in Table I. In the table, mean values that meet our soft real-time criterion and maximum runtimes that meet our hard real-time criterion, both at 67 ms, are denoted by an asterix.

A graphical comparison of the mean runtimes is given in Fig. 4. The orange dashed line represents the measured mean update frequency of our Velodyne and hence can be used guideline for a soft real-time criterion. In contrast to that, a hard real-time defined by the maximum frequency of 67 ms can only be achieved by the GPGPU-algorithm with a maximum runtime of 43.0 ms. Though the *Feature*-algorithm yields the weakest acceleration, it has the advantage of hardware-independence and is used by the two parallelization approaches.

At this point it is observable that even with a non-optimal configuration of work-items and terrain grid, the standard laptop GPU outperforms the multi-core approach considerably. Compared to other results, where the Gibbs sampler itself or other energy minimizing algorithms are parallelized, our approach yields a sufficient acceleration, too. Reservations for the terrain segmentation approach

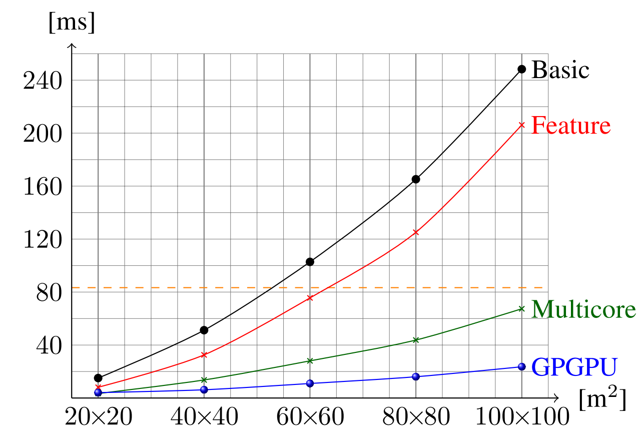

Fig. 4. Representation of the mean runtimes as curves. The *orange dashed line* indicates that both parallelizations (Multicore and GPGPU) are fast enough to process the data of the Velodyne HDL-64E S2 (mean update frequency of 12 hz) for all grid sizes

where that each thread, either on the CPU or the GPU, has to deal with the burn-in phase of the sampler. Thus, it is interesting to see, that a parallelization without a manipulation of the original Gibbs sampler, is able to process all data in real-time.

VI. Conclusions

In this paper, we presented runtime optimizations for our MRF energy minimization via Gibbs sampling. The energy minimization was parallelized both on the CPU and the GPU. Our evaluation yielded runtime optimizations of up to factor 10 for different terrain dimensions. The approach is now able to process a 2.5 times larger terrain in real-time which significantly increases the reach of vision for a more foresightful behavior. The implementation on the graphics card is fast enough to meet a hard real-time criterion given by the maximum update frequency of our LRF. Thereby, we are able to process 1.8 million data points per second and provide our path planning algorithm with a terrain grid of $100 \times 100\,\mathrm{m}^2$ containing obstacles and terrain surface conditions in real-time. Further, by transferring obstacle detection and terrain surface analysis to the GPU, the CPU is unburdened and other tasks profit from this acceleration as well.

Considering future work, we will work on an integration of time in the MRF. Accessing sensor data from previous points in time can yield improvements of the classification, especially in regions with few sensor data caused by newly emerged occlusions. Another quite promising extension is a self-supervised learning of the feature parameters. Thus, the MRF can adapt to various situations and is able to improve the classification during runtime if the features are relearned under certain circumstances.

VII. Acknowledgments

This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG) under research contract PA 599/11-1.

References

[1] J. Alberts, D. Edwards, T. Soule, M. Anderson, and M. O'Rourke. Autonomous Navigation of an Unmanned Ground Vehicle in Unstructured Forest Terrain. In *Proceedings of the ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems*, pages 103–108, Edinburgh, Scotland, 2008.

[2] J. Besag. Spatial Interaction and the Statistical Analysis of Lattice Systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):192–236, 1974.

[3] H. Deng and D. A. Clausi. Unsupervised Image Segmentation Using a Simple MRF Model With a New Implementation Scheme. *Pattern Recognition*, 37(12):2323–2335, 2004.

[4] S. Geman and D. Geman. Stochastic Relaxation, Gibbs Distribution and Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984.

[5] J. Gonzalez, Y. Low, A. Gretton, and C. Guestrin. Parallel Gibbs Sampling: From Colored Fields to Thin Junction Trees. *Journal of Machine Learning Research - Proceedings Track*, 15(1):324–332, 2011.

[6] J. Hammersley and P. Clifford. Markov Fields on Finite Graphs and Lattices. unpublished, 1971.

[7] M. Häselich, M. Arends, D. Lang, and D. Paulus. Terrain Classification with Markov Random Fields on fused Camera and 3D Laser Range Data. In *Proceedings of the 5th European Conference on Mobile Robotics*, pages 153–158, Örebro, Schweden, 2011.

[8] Z. Kato, M. Berthod, and J. Zerubia. A Hierarchical Markov Random Field Model and Multi-Temperature Annealing for Parallel Image Classification. *Computer Vision, Graphics and Image Processing: Graphical Models and Image Processing*, 58(1):18–37, 1996.

[9] K. Konolige, M. Agrawal, R. C. Bolles, C. Cowan, M. Fischler, and B. Gerkey. Outdoor Mapping and Navigation Using Stereo Vision. In *Proceedings of the International Symposium on Experimental Robotics*, pages 179–190, Rio de Janeiro, Brazil, 2006.

[10] R. Manduchi, A. Castano, A. Talukder, and L. Matthies. Obstacle Detection and Terrain Classification for Autonomous Off-road Navigation. *Autonomous Robots*, 18(1):81–102, 2004.

[11] J. Martens and I. Sutskever. Parallelizable Sampling of Markov Random Fields. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pages 517–524, Sardinia, Italy, 2010.

[12] F. Neuhaus, D. Dillenberger, J. Pellenz, and D. Paulus. Terrain Drivability Analysis in 3D Laser Range Data for Autonomous Robot Navigation in Unstructured Environments. In *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation*, pages 1686–1689, Catalonia, Spain, 2009.

[13] T. Pock, A. Chambolle, H. Bischof, and D. Cremers. A Convex Relaxation Approach for Computing Minimal Partitions. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 810–817, Miami, Florida, 2009.

[14] N. Vandapel, D. Huber, A. Kapuria, and M. Hebert. Natural Terrain Classification Using 3-D Ladar Data. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 5117–5122, New Orleans, USA, 2004.

[15] P. Vernaza, B. Taskar, and D. Lee. Online, Self-supervised Terrain Classification via Discriminatively trained Submodular Markov Random Fields. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2750–2757, Pasadena, USA, 2008.

[16] D. F. Wolf, G. Sukhatme, D. Fox, and W. Burgard. Autonomous Terrain Mapping and Classification Using Hidden Markov Models. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2026–2031, Barcelona, Spain, 2005.

[17] C. Ye and J. Borenstein. A New Terrain Mapping Method for Mobile Robots Obstacle Negotiation. In *Proceedings of the UGV Technology Conference at the SPIE AeroSense Symposium*, pages 21–25, Orlando, Florida, 2003.