Scalability of GPU-Processed 3D Distance Maps for Industrial Environments

Atle Aalerud[1], Joacim Dybedal[1], and Geir Hovland[1].

[1]Mechatronics Group, Faculty of Engineering and Science, Department of Engineering Science, University of Agder, N-4898 Grimstad, Norway. `atle.aalerud@uia.no`

***Abstract* – This paper contains a benchmark analysis of the open source library GPU-Voxels together with the Robot Operating System (ROS) in large-scale industrial robotics environment. Six sensor nodes with embedded computing generate real-time point cloud data as ROS topics. The overall data from all sensor nodes is processed by a combination of CPU and GPU on a central ROS node. Experimental results demonstrate that the system is able to handle frame rates of 10 and 20 Hz with voxel sizes of 4, 6, 8 and 12 cm without saturation of the CPU or the GPU used by the GPU-Voxels library. The results in this paper show that ROS, in combination with GPU-Voxels, can be used as a viable solution for real-time 3D collision detection and avoidance applications in relatively large-scale industrial environments.**

B.1 Introduction

GPU-Voxels is an open source library for CUDA based Graphics Processing Units (GPU) which allows massively parallel computation of collision checking in 3D environments. In [1] it was found that the solution offers constant runtime regardless of the occupancy density in the environment. Here, a deterministic voxel map is used, meaning each voxel holds a bit vector identifying the occupancy status of the voxel. In later versions of the library, the voxel map is also able to store and calculate a distance and a vector to the nearest obstacle.

When multiple sensor nodes are used for mapping a large volume in real-time, a scalable framework is needed to handle the map data in an efficient manner. OctoMap, [2], is a popular efficient framework based on octrees easily implemented in the Robot Operating System (ROS) [3]. However, as OctoMap currently is based on using the Central Processing Unit (CPU) only, it is likely to yield poor real-time performance when scaled for large industrial applications. A solution assumed to be better for large-scale implementation should also utilize the vast parallelization that can be done in a GPU. In [4], A. Hermann et al. gave an overview of the GPU-Voxels framework that unifies different data structures and algorithms that are optimized for GPU architectures.

In this paper, the authors will verify and benchmark if the desirable behavior of constant runtime holds for the later versions of the GPU-Voxels library regardless of the occupancy density in the environment. That is, using the Distance Voxel Map which includes distance and vector calculation to the nearest obstacle in each voxel. It will also be tested if GPU-Voxels and ROS can be used efficiently together for real-time collision avoidance in a large industrial environment including both robots and humans.

There are not many publications available where CPU and GPU utilization and performance in collision detection applications have been evaluated. One recent example is [5] where pedestrian detection is benchmarked in an Advanced Driving Assistance System (ADAS) based on the Viola-Jones image recognition algorithm. The optimization strategies were presented and the implementation evaluated on a multiprocessor system featuring multiple GPUs, showing an overall 88.5x speedup over the sequential version. The main difference between the work in [5] and our work is the use of computer vision images vs. 3D point clouds. The performance, about 20 frames per second, is similar in the two papers.

The paper is organized as follows: Paper B.2 gives an overview of the industrial environment and the necessary map dimensions. Further, both hardware and software are listed before describing how the experiments were conducted. The results of these experiments, including execution time and utilization of CPU and GPU, are given in Paper F.5. Thereafter, the observations made in the results are discussed in Paper E.7. Here, the implications on the hardware limitations of the system and lessons learned are discussed further. Finally, a conclusion is made regarding scalability before future work is evaluated.

B.2 Experimental Setup

This paper will use the same experimental setup as described in [6]. However, key information is included here for completeness.

A relatively large robotics lab at the University of Agder is the test case used for evaluating the scalability of GPU-Voxels for industrial safety applications. The lab, depicted in Figure C.1, consists of two rail-mounted ABB IRB4400 robots, one ABB IRB2400 robot mounted on a GÜDEL gantry and a processing facility. As the size of the lab is approximately $10\,\mathrm{m} \times 15\,\mathrm{m} \times 5\,\mathrm{m}$, GPU-Voxel maps used in benchmarking are selected to cover at least this volume.

GPU-Voxels requires that the voxel dimensions of the map are divisible by 64 and that two of the sides are equal. Thus, combinations of voxel- and map sizes are selected so that metric dimensions remain the same in X- and Y direction (15.36 m) of the map. Ideally, Z direction should also remain constant to maintain a constant volume, but two different heights are necessary to fulfill the mentioned requirements (5.12 m and 7.68 m). Combination of these dimensions yields six possible map resolutions. Given by the voxel side length, these are twelve, eight, six, four, three and two centimeters. However, the latter two are deemed non-relevant due to relatively low sensor accuracy and high computational cost. Benchmarking the remaining four map resolutions using two sensor frequencies, ten and twenty frames per second, yields a total of eight experiments as illustrated in Table B.1.

The numbered experiments in Table B.1 relate to the different map sizes and resolutions. Here, f is the frequency that is used to obtain sensor data. V_L is the resolution shown as voxel side length. Multiplying this by the map dimensions (Dim.) yields the metric dimensions of the map. The total amount of voxels in the map is shown as Map in the

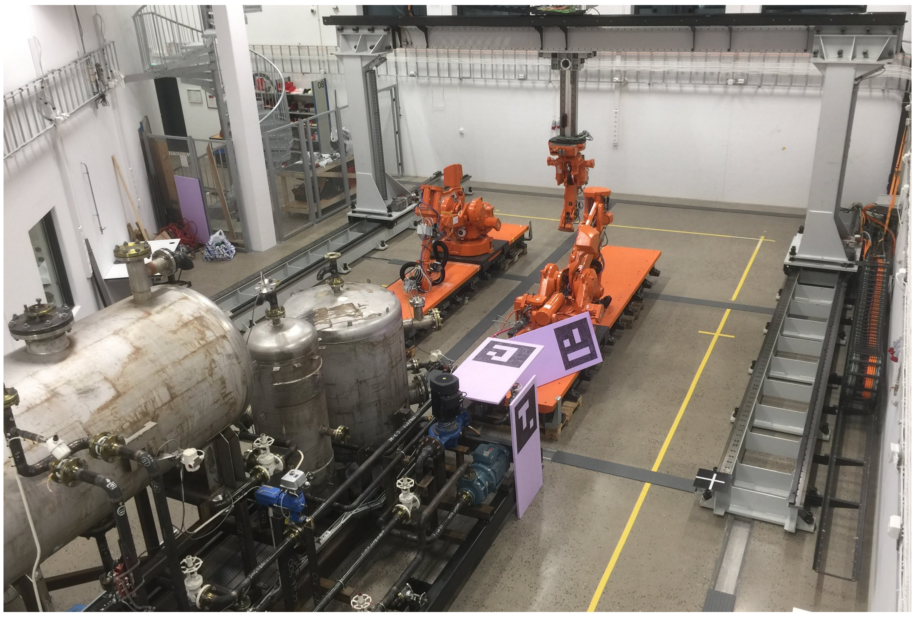

Figure B.1: A Robotics Lab at the University of Agder is used as the example case of an industrial environment.

last column.

Table B.1: List of experiments comprising four map resolutions and two frequencies

	f [Hz]	V_L [cm]	Dim. [vox]	Dim. [m]	Map [vox]
1	10	12	128, 128, 64	15.36, 15.36, 7.68	1 048 576
2	10	8	192, 192, 64	15.36, 15.36, 5.12	2 359 296
3	10	6	256, 256, 128	15.36, 15.36, 7.68	8 388 608
4	10	4	384, 384, 128	15.36, 15.36, 5.12	18 874 368
5	20	12	128, 128, 64	15.36, 15.36, 7.68	1 048 576
6	20	8	192, 192, 64	15.36, 15.36, 5.12	2 359 296
7	20	6	256, 256, 128	15.36, 15.36, 7.68	8 388 608
8	20	4	384, 384, 128	15.36, 15.36, 5.12	18 874 368

Mapping of the described environment is done by using six of the Microsoft Kinect for Xbox One (Kinect V2) which contains an infrared time-of-flight depth cameras capable of processing 30 frames per second. Each camera is connected to an NVIDIA Jetson TX2 Development board for local point cloud processing. The processed data is sent to a central PC comprising an Intel Core i5-2500 Central Processing Unit (CPU) running four cores at 3.3 GHz. Further, the PC has 8 GiB system memory and an NVIDIA GeForce GTX TITAN Graphics Processing Unit (GPU), which in turn has 6 GiB memory and 2688 CUDA Cores capable of parallel execution of 86016 threads. This is the same GPU as used by the author of GPU-Voxels in [1, 4] and [7].

Communication between the sensor nodes and the central PC is handled by ROS (version Kinetic Kame) running on Ubuntu 16.04. Each of the sensor nodes sends a ROS topic containing a compressed, filtered and transformed point cloud (see [8]). This is reconstructed by a local ROS node before it is inserted into GPU-Voxels. As there are multiple sensor nodes transmitting data, ROS callback handling is done using multiple CPU-cores. This is necessary as serial processing of the sensor data received in ROS callbacks is not deemed to be a scalable solution.

Benchmarking the performance is done by logging the utilization of the CPU and GPU over three minutes at one-second intervals. The first minute is only used to determine what the idling performance of the system is. Then, the second minute runs both the decompressor and GPU-Voxels calculating the described collision distances. Lastly, throughout the last minute, the GPU-Voxels visualizer is also active.

According to [1], collision checks should be strictly dependent on the size of the Voxel Map, but independent of the number of measurement points and degree of occupancy. Thus, it is desired to see if this is valid also for the Distance Voxel Map which is assumed to be more computationally expensive due to the distance calculation included in every voxel. This evaluation is done by repeating experiment number four (see Table B.1) multiple times while using different degrees of occupancy whilst the map size and frequency remain constant. Increasing the occupancy, while still using real sensor data, is achieved by reducing the intensity threshold in the decompressor and changing the calibration of sensor poses to misalign the input point clouds. This experiment is logged in the same manner as previously described.

B.3 Experimental Results

After the point clouds are combined in GPU-Voxels, they can be visualized using the GPU-Voxels visualizer. In Figure B.2, the voxel representation of a person is shown using the different map resolutions. Figure B.3 shows the entire Distance Voxel Map where the

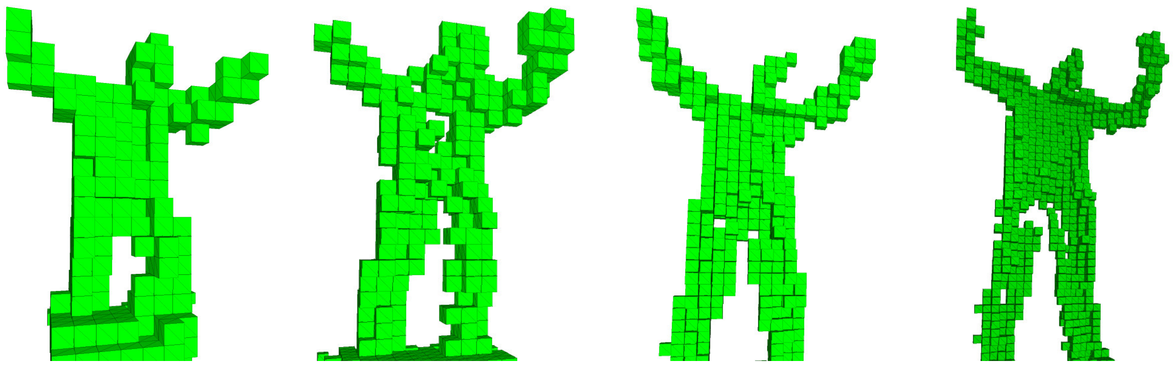
(**a**) 12 cm (**b**) 8 cm (**c**) 6 cm (**d**) 4 cm

Figure B.2: A person visualized with the different voxel sizes that are used in the experiments.

decompression node used to receive the sensor data is configured to disregard all voxels that contain fewer than four measurements. Note that the floor and walls are removed. This is done as part of the compression on each sensor node. The map dimensions correspond to experiment 1 and 5 from Table B.1.

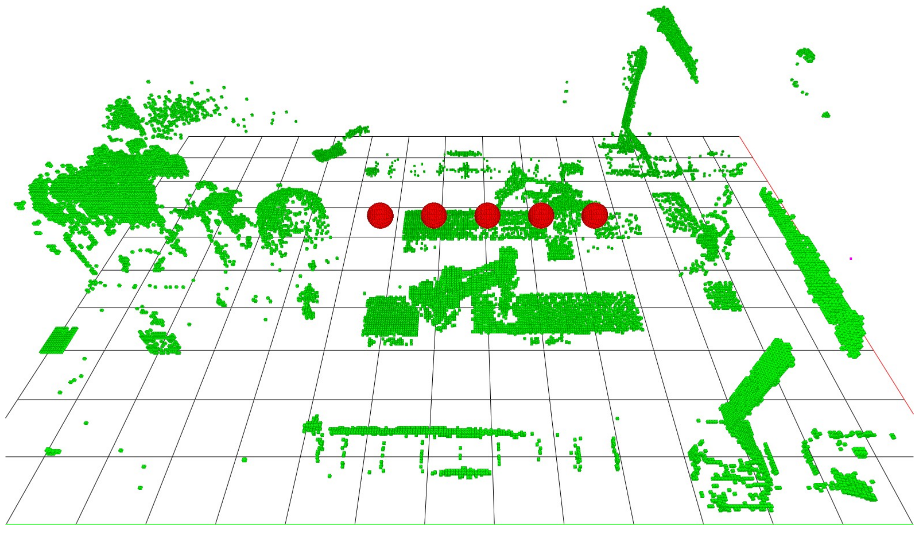

Figure B.3: Occupied voxels visualized by the GPU-Voxels visualizer. Resolution is 4 cm and the red spheres are inserted as obstacles used for distance calculation.

In the results presented in Tables B.3 and E.1, idling use of CPU and GPU are omitted for clarity. The idling CPU and GPU utilization during the experiments were 13.6 % and 8.0 % respectively. Further, the GPU-Voxel visualizer used on average 7.7 % of the GPU while activated. Thus, to evaluate the total utilization, these should be included.

Table E.1 shows the results from the experiments listed in Table B.1. Here, $\boldsymbol{D}$ is the number of occupied voxels after inserting filtered point clouds from all sensor nodes. Further, the load of the CPU and GPU are listed in percent. $\boldsymbol{t_D}$ and $\boldsymbol{\sigma_D}$ are the time and standard deviation of the distance calculation for the occupied voxels. $\boldsymbol{t_C}$ and $\boldsymbol{\sigma_C}$ are the total ROS cycle time (including t_D) and corresponding standard deviation. New measurement data is received every one hundred milliseconds for experiment 1 to 4, and every fifty milliseconds for experiments 5 to 8. Thus, all calculations are processed in time as the slowest cycle time is 39.7 ms.

Table B.2: Results of the numbered experiments from Table B.1. In the first four columns map size, number of occupied voxels and hardware utilization are shown. Further, in the last four columns, execution times and standard deviations are shown in milliseconds.

	Map [vox]	$\boldsymbol{D}$ [vox]	**CPU**	**GPU**	$\boldsymbol{t_D}$	$\boldsymbol{\sigma_D}$	$\boldsymbol{t_C}$	$\boldsymbol{\sigma_C}$
1	1 048 576	3 900	21.1 %	7.1 %	3.4	1.0	14.8	1.4
2	2 359 296	6 900	21.0 %	11.8 %	5.8	1.5	17.3	1.8
3	8 388 608	9 600	21.6 %	22.0 %	13.6	1.2	24.9	1.3
4	18 874 368	16 300	28.3 %	34.2 %	28.1	1.8	39.7	2.0
5	1 048 576	3 900	44.1 %	22.1 %	4.1	1.6	15.9	2.1
6	2 359 296	6 900	41.4 %	22.8 %	6.1	1.6	17.8	2.1
7	8 388 608	9 600	47.5 %	42.4 %	15.6	2.4	27.3	2.7
8	18 874 368	16 300	56.1 %	65.4 %	26.1	2.3	37.9	2.7

The results shown in Table E.1 are illustrated in Figures B.4 to B.6. In Figures B.4 and B.5, it can be seen that increasing the size of the map has a minor effect on the CPU utilization, but a large impact on the GPU utilization. This can also be seen in Figure B.6 as t_D increases whilst t_C remains relatively constant offset. On the other hand, doubling the frequency also doubles the CPU and GPU utilization whilst execution times remain relatively unchanged.

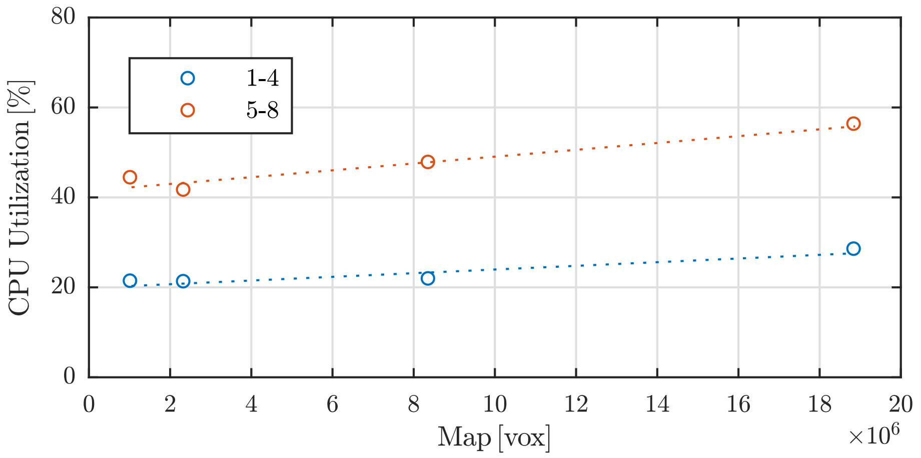

Figure B.4: Average CPU utilization shown as a function of Map size in voxels. Idle load has been subtracted for clarity.

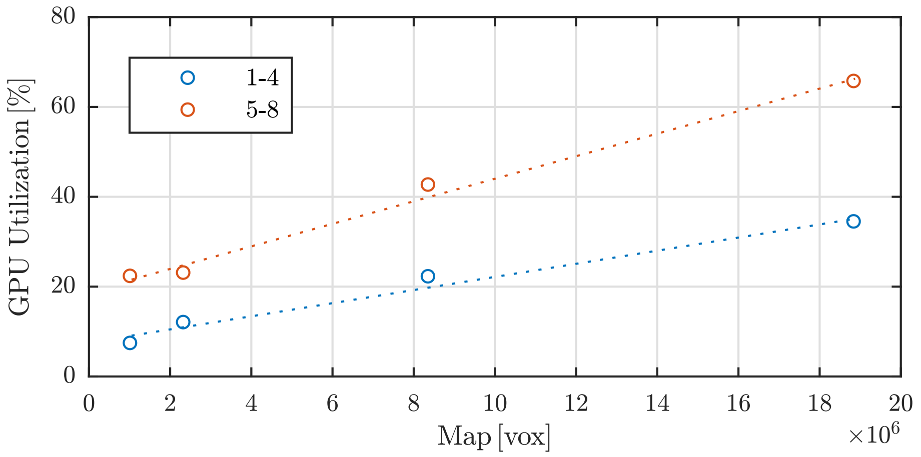

Figure B.5: Average GPU utilization shown as a function of Map size in voxels. Idle load has been subtracted for clarity.

The results of increasing the occupancy is displayed in Table B.3. The table contains similar information as described for Table E.1. However, in this case, the number of occupied voxels increased throughout nine experiments without changing map size or frequency. Thus, the changes in utilization and execution time is isolated to occupancy.

Figures B.7 to B.9 illustrate the results shown in Table B.3. Here, it is clear that increasing the occupancy increases the CPU utilization. Further, it appears that the GPU utilization remains relatively unchanged from the fifth measuring point. The execution time of t_D also saturates into a near horizontal trend from the fifth measuring point. t_C has a relatively constant offset similar to Figure B.6.

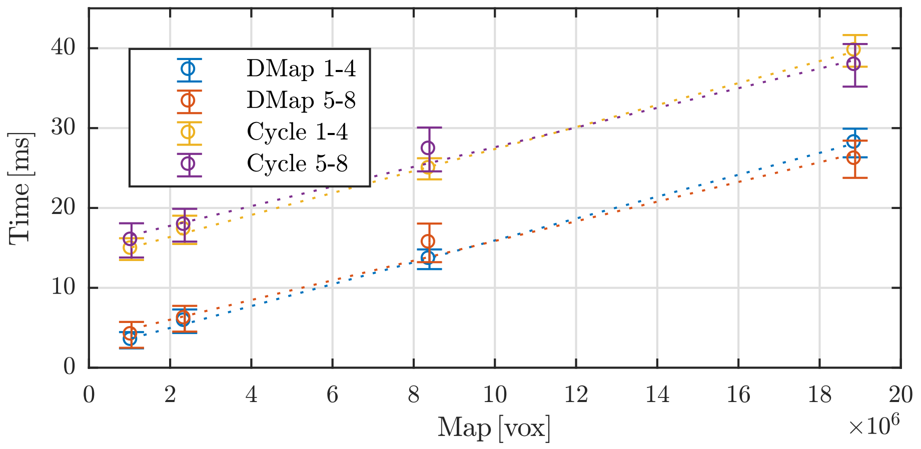

Figure B.6: Distance calculation and total cycle time for experiment 1-8.

Table B.3: Experiment number four repeated with increasing amount of inserted voxels. CPU and GPU utilization in percent and runtime shown in milliseconds.

	D [vox]	CPU	GPU	t_D	σ_D	t_C	σ_C
4a	8 050	30.3 %	37.5 %	26.7	1.7	38.5	1.9
4b	16 450	28.2 %	37.0 %	27.4	1.7	39.2	1.9
4c	24 950	31.4 %	34.8 %	28.8	1.7	40.7	1.9
4d	40 000	28.8 %	35.5 %	29.4	1.8	41.2	1.9
4e	78 300	31.8 %	41.5 %	33.0	1.7	44.7	1.8
4f	111 250	29.9 %	38.3 %	34.2	1.8	45.7	1.9
4g	158 100	41.4 %	41.7 %	32.0	2.5	44.0	2.9
4h	212 500	52.4 %	41.6 %	34.0	2.4	46.0	2.7
4i	220 300	50.0 %	39.2 %	35.3	4.3	47.2	4.9

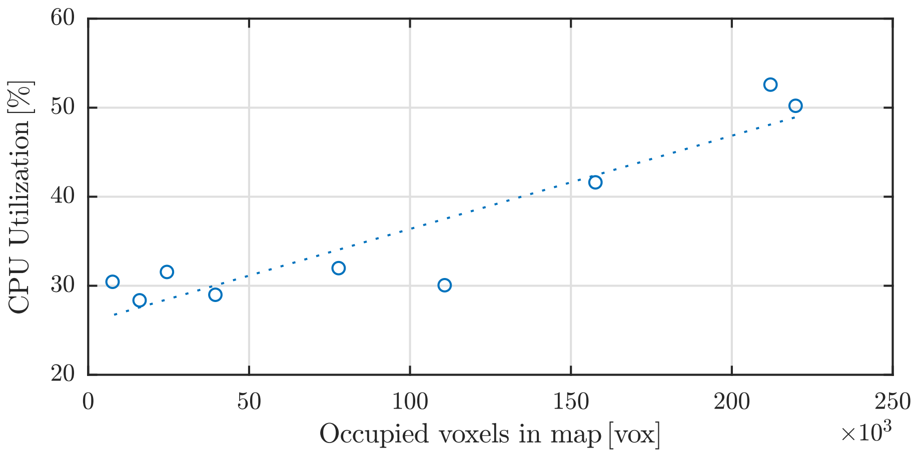

Figure B.7: Average CPU utilization shown with the number of occupied voxels that are inserted in distance map. Map dimensions and frequency are kept constant according to test number 4. Idle CPU load has been subtracted for clarity.

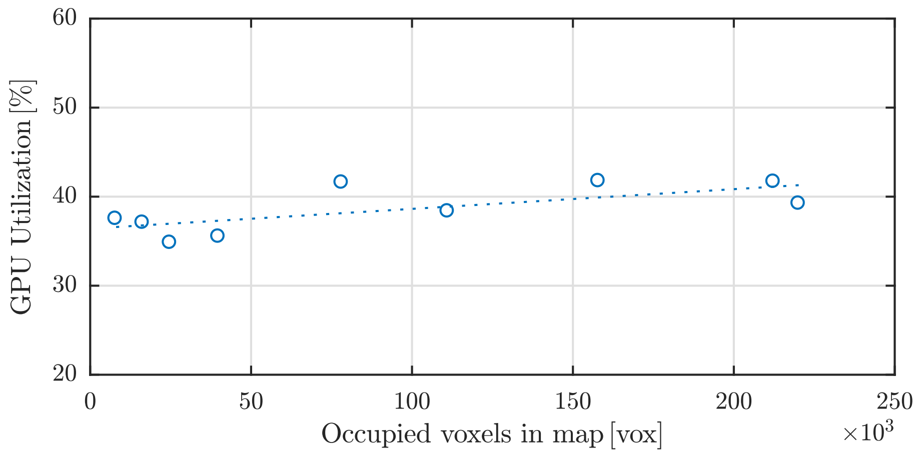

Figure B.8: Average GPU utilization shown with the number of occupied voxels that are inserted in distance map. Map dimensions and frequency are kept constant according to test number 4. Idle GPU load has been subtracted for clarity.

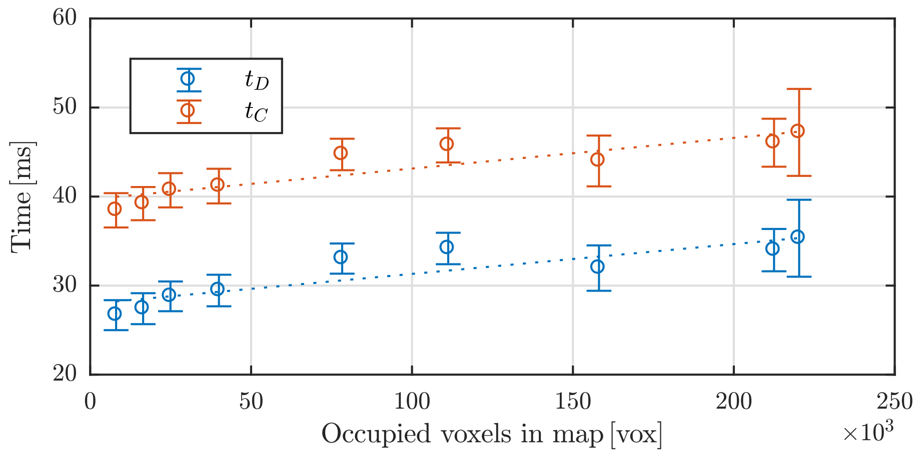

Figure B.9: Distance calculation and total cycle time for experiment 4a-4i.

As an observation, the frequency of the sensor nodes from experiment eight cannot be increased to run faster than the GPU can handle. That is, approximately 30 ms or 33 Hz. Suppose around 90 % of the GPU may be utilized and that this utilization increases with 31.2 % per 10 Hz as indicated in Table E.1. Then, the GPU is limited to receive new maps at approximately 27.9 Hz. However, the CPU uses around 40 ms, which limits the performance for this map size to 25 Hz.

B.4 Discussion and Conclusion

As ROS callbacks are handled by the CPU, it is obvious that an increased callback frequency also leads to increased use of the CPU. Only the actual measuring points are processed, not empty space. Thus, the size of the map is less relevant for the CPU. Hence, the slightly increased CPU usage observed in Figure B.4 is more likely caused by the increased number of measurements, $\boldsymbol{D}$, than the map size directly. This relation is further clarified as Figure B.7 shows that the amount of measurement points in the map has a significant influence on the CPU usage. The ROS callbacks are programmed using multithreading. Thus the ROS cycle time is less affected by the number of measurement points received than if data from all the nodes are received in a serial manner.

The GPU appears to be quite unaffected by an increase in map occupancy. Both execution time and utilization are near constant as the calculations are performed in a highly parallel manner. The calculation time increase with map size which is reflected in GPU utilization. GPU utilization is also affected by the frequency whilst the execution time is not. Nevertheless, execution time will limit the possible sensor frequency.

The current hardware will not be able to run the largest map at higher frequencies than approximately 25 Hz. However, this is possible using smaller maps. So inevitably, there is a trade-off between speed and resolution. This choice highly depends on the application. For instance, if the map is used for collaboration with lightweight robots that move slowly in the proximity of obstacles, then a high-resolution map is needed for the interaction.However, if the map is used in safety applications for heavy industrial machines moving at high velocities, the speed is more crucial than the resolution as close interaction is less important than time needed to stop the machine safely. As seen in Figure B.2, all the tested resolutions are capable of representing a human worker sufficiently for such safety applications.

The conducted experiments use only the Distance Voxel Map. However, it is not necessary to calculate distances for known and static environments, only unknown objects. Thus, the actual amount of distance calculations may be much less than the experimental maps. One person, as seen in Figure B.2(**d**), can typically be mapped using approximately 1100 voxels of 4 cm resolution. Comparing with the number of occupied voxels in experiment four and eight (from Table E.1), shows that the experiment corresponds to mapping distances from approximately fifteen people.

The used compression scheme (see [8]) reduces the bandwidth required to transfer data from the sensor nodes. However, this comes at a cost to the CPU that needs to decompress the data for all sensor nodes. The current CPU has only four cores, and as there are currently six sensor nodes, some serialization must be expected when receiving the sensor data. Hence, adding more than two additional sensor nodes would cause even further serialization which in turn could increase the ROS cycle time. Ideally, there should be only one sensor node per core to avoid serialization completely. Thus, increasing the number of sensor nodes to more than the double compared to CPU cores should not be done without reducing the frequency or further optimization of the cycle time.

Future work will focus on map filtration, classification and segmentation into different map types according to function such as static environment, machines with known kinematics, unknown obstacles, and personnel. These will be used in applications such as collision detection and avoidance. However, for these applications, overall latency of the system should also be evaluated. Thus, ROS2 will be considered for real-time communication.

Acknowledgment

The research presented in this paper has received funding from the Norwegian Research Council, SFI Offshore Mechatronics, project number 237896.

References

[1] A. Hermann, S. Klemm, Z. Xue, A. Roennau, and R. R. Dillmann, "GPU-based real-time collision detection for motion execution in mobile manipulation planning," in *2013 16th Intl. Conf. on Advanced Robotics (ICAR)*. IEEE, 11 2013. doi: 10.1109/ICAR.2013.6766473. ISBN 978-1-4799-2722-7 pp. 1–7.

[2] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 4 2013. doi: 10.1007/s10514-012-9321-0

[3] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, vol. 3, no. 3.5, Kobe, May 2009, p. 5.

[4] A. Hermann, F. Drews, J. Bauer, S. Klemm, A. Roennau, and R. Dillmann, "Unified GPU voxel collision detection for mobile manipulation planning," *2014 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, no. Iros, pp. 4154–4160, 9 2014. doi: 10.1109/IROS.2014.6943148

[5] M. M. Trompouki, L. Kosmidis, and N. Navarro, "An open benchmark implementation for multi-CPU multi-GPU pedestrian detection in automotive systems," in *2017 IEEE/ACM Intl. Conf. on Computer-Aided Design (ICCAD)*. IEEE, 11 2017. doi: 10.1109/ICCAD.2017.8203793. ISBN 978-1-5386-3093-8 pp. 305–312.

[6] A. Aalerud, J. Dybedal, E. Ujkani, and G. Hovland, "Industrial Environment Mapping Using Distributed Static 3D Sensor Nodes," in *2018 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*

(MESA). Oulu: IEEE, Jul. 2018. doi: 10.1109/MESA.2018.8449203. ISBN 978-1-5386-4643-4 pp. 1–6.

[7] A. Hermann, F. Mauch, K. Fischnaller, S. Klemm, A. Roennau, and R. Dillmann, "Anticipate your surroundings: Predictive collision detection between dynamic obstacles and planned robot trajectories on the GPU," in *2015 European Conf. on Mobile Robots (ECMR)*. IEEE, 9 2015. doi: 10.1109/ECMR.2015.7324047. ISBN 978-1-4673-9163-4 pp. 1–8.

[8] J. Dybedal, A. Aalerud, and G. Hovland, "Embedded Processing and Compression of 3D Sensor Data for Large Scale Industrial Environments," *Sensors*, vol. 19, no. 3, p. 636, Feb. 2019. doi: 10.3390/s19030636