Modeling, Identification and Control, Vol. 33, No. 4, 2012, pp. 123–130, ISSN 1890–1328

3D Sensor-Based Obstacle Detection Comparing Octrees and Point clouds Using CUDA

K.B. Kaldestad [1] G. Hovland [1] D.A. Anisi [2]

[1] *Faculty of Technology and Science, Department of Engineering, University of Agder, N-4898 Grimstad, Norway. E-mail:* {knut.b.kaldestad,geir.hovland}@uia.no

[2] *ABB AS, Strategic R&D for Oil, Gas and Petrochemicals, N-0666 Oslo, Norway. E-mail:* david.anisi@no.abb.com

Abstract

This paper presents adaptable methods for achieving fast collision detection using the GPU and Nvidia CUDA together with Octrees. Earlier related work have focused on serial methods, while this paper presents a parallel solution which shows that there is a great increase in time if the number of operations is large. Two different models of the environment and the industrial robot are presented, the first is Octrees at different resolutions, the second is a point cloud representation. The relative merits of the two different world model representations are shown. In particular, the experimental results show the potential of adapting the resolution of the robot and environment models to the task at hand.

Keywords: Collision Detection, Industrial Robot, Hidden Markov Model, Expert System

1 INTRODUCTION

Traditionally, the main industrial impact of robot technology has lied in domains containing repetitive routine tasks, in particular; manufacturing and automotive industries. In such environments, objects inside the workspace of the robot are most often either fixed at a known location or moving according to *a priori* determined and known patterns. Hence, the robot programs can be written such that these objects are avoided. Recently, there have been several initiatives to use industrial robots in un-, or semi-structured outdoor environments. As an illustrative example, consider the case of robot automation in the Oil and Gas (O&G) industry, Anisi et al. (2010, 2011). As a general trend within that industry, most of the easy accessible oil and gas fields have already been exploited, leaving the more remote and geo- politically challenging reserves for future exploration. Given the importance and focus of the oil and gas industry related to safety, environmental impact, cost efficiency and increased production, the potential for more extensive use of automation in general, and robot technology in particular, is evident.

Within the oil and gas industry, the use of robotics has so far been limited to special functions such as sub-sea and pipeline intervention and inspections using Remotely Operated Vehicles (ROVs), automation of drilling operations and well tractors. The degree of autonomy in these applications has however been very low, meaning that the robots are either remotely controlled, or that the robot follows *a priori* determined routes and hence has no ability to cope with environmental variations and uncertainty. The non-determinism mainly is due to mismatch between existing world model and the exact location of process components in the physical world. As presence of unexpected obstacles can not be ruled out in semi-structured environments typically found in the oil and gas industry, collision-free route cannot be guaranteed solely by offline programming. To avoid collisions in such settings, online, sensor-based collision detection and avoidance methods must be adopted. Collision detection considers the problem of indicating the presence of unmodelled obstacles while collision avoidance

doi:10.4173/mic.2012.4.1

aims at finding an alternative path around discovered obstacles leading to the target. To this end, use of external sensors, *e.g.*, laser-, ultrasound- and vision sensors is necessary and the robot programs must be able to process these data and adjust accordingly online.

There are not many papers describing mapping of the environment in combination with collision detection in an industrial robotic setup. The work presented in Henrich et al. (1998) shows the concept of modeling the robot and the environment, and finding the shortest distance to the obstacles. However, the approach in Henrich et al. (1998) is based on simple models of both the robot and of the environment, and the models are created offline. In contrast, the work presented in this paper uses a more detailed offline generated model of the robot, which in real life applications will not change during the time of the operation, but as apposed to Henrich et al. (1998) it presents an online mapping of the environment.

Ramisa uses the Microsoft Kinect together with a robotic manipulator for grasping wrinkled cloth Ramisa et al. (2012). As with this paper, the Kinect is used to gather depth data. As apposed to this paper, the Kinect is mounted on a structure external to the robot and not on one of the robot axes.

Dziegielewski's work on accurate mapping, von Dziegielewski et al. (2012), shows that combining mapping and CUDA is not a new teqnique, even though the use of the CUDA cores are different in this paper. The mapping presented in this paper is different from von Dziegielewski et al. (2012) in that the presented approach uses point clouds rather than a mesh based approach. The approach based on a point cloud is faster, but in some cases less informative.

This paper does not discuss different techniques for fast nearest neighbour search (NNS) such as NNS via k-d trees Elseberg et al. (2012), because all the points are a part of the calculation, such methods will not improve the calculation time. The focus for this paper is to present how the calculation time in some cases could be decreased in collision detection applications by using the GPU, the result is based on our own algorithms developed for CUDA and CPU together with with open source CPU-based algorithms. CPU-based algorithms distance calculation which are performed on single core would perform better on multiple cores, this is not considered in this paper.

The remaining of this paper is organized as follows. Section 2 provides the problem formulation. Following that, the details of the system setup are given in Section 3. The experimental study conducted in this work is presented in Section 4 while Section 5 gives thorough discussions of the results. Finally, Section 6 provides concluding remarks and discussions on future work.

2 PROBLEM FORMULATION

For industrial robots executing pre-planned tasks within a workcell, the concept of introducing new objects to the work cell without explicitly reprogramming the robot controller is a research topic deserving further attention and development. The common thing to do if a new object is added to the robot work cell is to stop the production. If the new object does not occlude the robot path while it manipulates any of the objects in the environment, the process could be started again without implications. On the other hand, if the new objects obstructs the path, a new path must be created by the operator. In the near future, it could be desirable to operate the robot in an industrial location with a reduced amount of protective fences. In such setups, it is important that the robot system is capable of detecting new objects and their locations. Hence, the robot must be part of a more adaptive system, with the possibility to dynamically introduce new objects to the scene. This raises the question about how the information about the objects should be made available to the robot controller. As described in previous work Kaldestad et al. (2012b) and Kaldestad et al. (2012a), one of these techniques could be manual input of a map. Depending on the application, the approach of manual input could have significant drawbacks. In addition, there are at least two drawbacks with respect to time. A manual human input of a map takes time and reduces performance. The second issue is the time aspect related to the resources of the human, which may be better utilised with other tasks. Because of these drawbacks, autonomous map generation using an external sensor is advocated. Next, the placement of the sensor is not trivial, and since the industrial robots to be used should potentially be capable to move on linear tracks along the floor or the ceiling, it is often desirable to mount the sensor on the robot.

To avoid excessive amounts of 3D sensor data, it is desirable to compress or reduce the data to increase performance. One approach that ensures that the data in the map does not exceed predefined space limits is the Octree approach.

Using robot manipulators in combination with Octrees has been done before Faverjon (1984), Hayward (1986) and there already exist methods for collision detection, based on overlapping Octrees. In most cases however, it is not desirable to wait until an actual collision before giving a warning, therefore using another approach such as representing the robot by an oversized Octree solves the problem with having the warning at time of impact, but it does not address the issue of generating a message when the robot approaches an object.

One of the advantages of using Octrees, is that the 3D position information is equally distributed in the Cartesian space. This approach has of course both its advantages and drawbacks, and for that reason we have also compared it to using point clouds directly. One of the main advantages of using an Octree representation of both the robot and the environment, is especially when a large number of points are present within a small volume, and often these points may not give additional valuable information to act on. In such settings, all of these points which belong to their respective place in the Octree, will be reduced to one cube.

The trend is to explore more and more complex methods, but it is a necessity to simplify in order to stay within reasonable time limits. Still, a few years back, in late 2005 the computer frequency of the Intel and, a bit later, the AMD CPUs' stalled Ross (2008). Due to the heat dissipated and the power consumed by increasing the clock, it was found more reasonable to increase the number of processing areas on one CPU. This was followed by Nvidia's hardware and software architecture Compute Unified Device Architecture CUD (2006) (CUDA) in 2006. Because the programmer would benefit by Moore's law, a doubling of the transistor count every two years, which resulted in a doubling of the CPU frequency every second year up to 2005, today, other technologies could be used to continue improving application execution time. The CUDA architecture is one excellent technology to use when there are large parallel computational problems, such as calculating distances between points.

In this paper two main questions are addressed. First, how well will an Octree structure behave in terms of computational efficiency, compared to a pure point cloud. Second, what will be the fastest way to detect a collision or near collision, if all the known location on the robot and in the environment are compared, given the previously mentioned representations, using the CPU or CUDA (GPU).

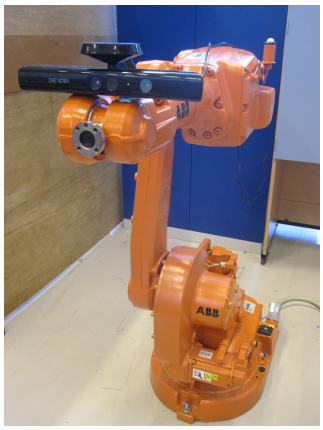

Figure 1: *ABB IRB1600-1.45 robot with Microsoft Kinect sensor mounted on the 4th link.*

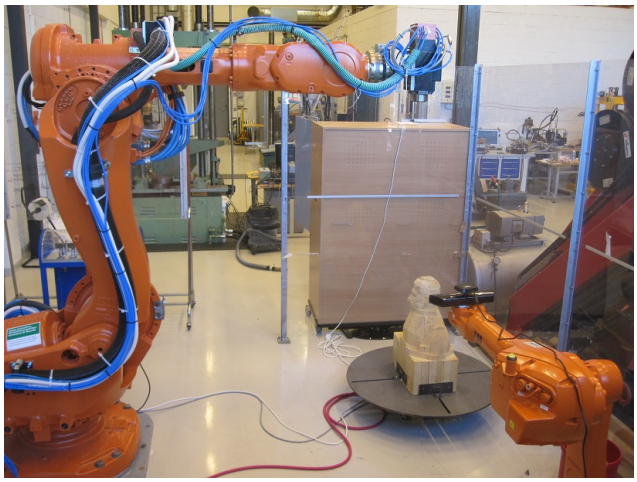

Figure 2: *ABB IRB1600-1.45 robot and surrounding environment.*

3 SYSTEM SETUP

The setup depicted in Fig. 3 consists of an ABB IRB1600-1.45 robot, an IRC5 industrial controller and a Microsoft Kinect sensor mounted on the robot's 4^{th} link, see Fig. 1. The larger environment Fig. 2, contains a second robot, an object on a rotary axis and protective walls inside the reachable workspace of the IRB1600 robot. Except the IRB1600 and the Kinect, all the other objects are static and part of the environment. Communication with the robot is done in a network connection with a separate computer (PC_{rob}). A second computer in the network (PC_{obs}) with Nvidia 280 GTX CUDA-Enabled graphics card, was used for managing 3D-point data, which includes different models of the robot and the environment. The environment map is updated by gathering the Kinect distance data and storing it to the map. Further, robot motor angles are received from PC_{rob}, and used to transform the environment Kinect data in addition to transforming a point model of the robot. Before the Kinect-data is inserted to an Octree or point cloud, it is filtered using a distance filter, and it is also run through a statistical outliers removal to remove noisy depth readings. The transformed environment map is then used to calculate the distance from the robot to the environment.

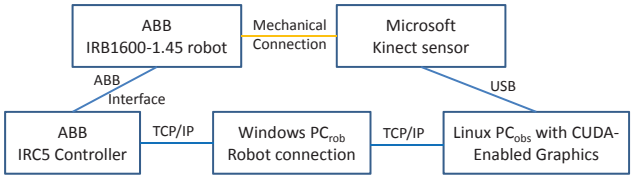

Figure 3: *Overview of system components.*

4 EXPERIMENTS

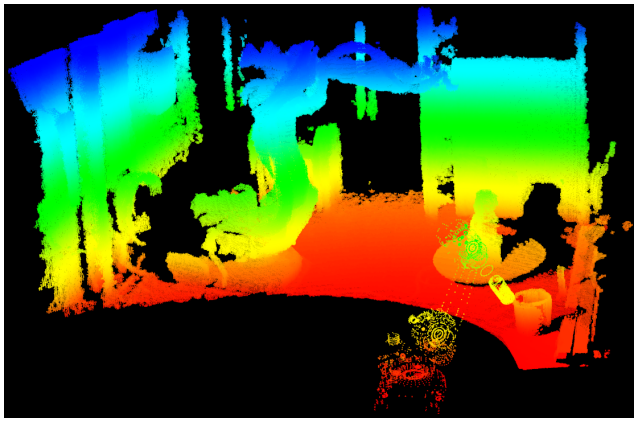

Figure 4: *Example of concatenated point cloud of environment generated with Kinect sensor.*

Given one or several concatenated point clouds of the environment as shown for example in Fig. 4, an Octree representation can be generated. In this paper the Octree class in the Point PCL library Rusu and Cousins (2011) (PCL) is used. Six different Octree implementations have been compared with an approach working directly on a point cloud. The approaches were compared with respect to computational time and the number of Octree elements. Different sidelengths of the Octree cubes were used (1mm, 10mm and 100mm). In addition, the representation of the IRB1600 robot and the environment could be with different cube sidelengths. Fig. 5 shows a representation where 100mm cube sidelengths were used for the IRB1600 robot and 1mm sidelengths were used to represent the environment. Notice that only the centre points of the cubes are shown in the figure. The representation of the robot was generated from a CAD file, while the representation of the environment was generated using the Kinect sensor. Fig. 6 shows another representation where 10mm side lengths were used for both the robot and the environment, here the cubes are shown.

The different approaches were compared based on

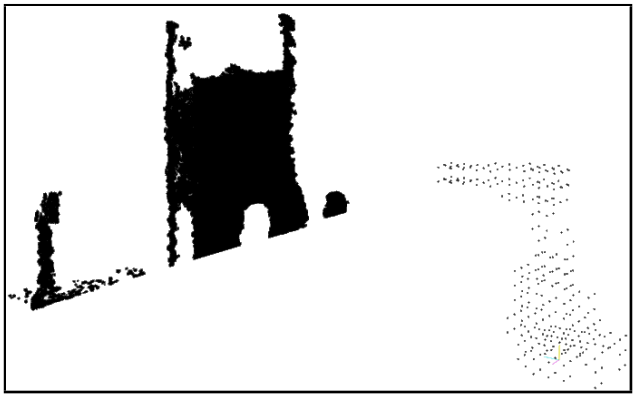

Figure 5: *Illustration of centre points in Octree. 100mm sidelengths were used for the robot and 1mm sidelengths for the environment.*

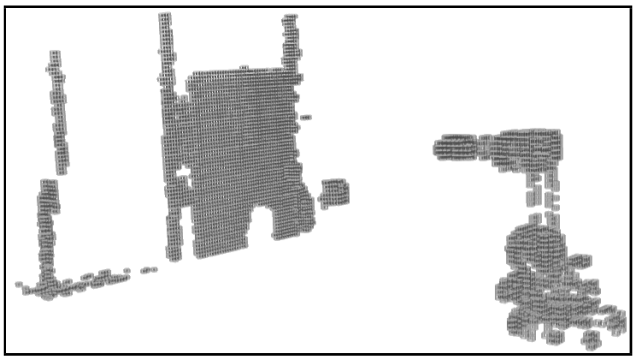

Figure 6: *Illustration of both centre points and cubes in Octree with 10mm sidelengths for both the robot and the environment.*

computational time, which was found when calculating the smallest distance L_{min} between all the robot points R_i and all the environment points E_j, ie.

$$L_{min} = \min_{i,j} \|R_i - E_j\| \tag{1}$$

When using a point cloud, all the robot points generated from the CAD file were compared with all the points generated by the Kinect sensor. When using an Octree, the centre points in all the robot cubes were compared with the centre points in all the environment cubes.

The minimum collision distance for an Octree representation of both the environment and the manipulator is given by

$$L_{min} < \|\mathbf{c}_s\| \tag{2}$$

while for an uncompressed and un-approximated point

cloud it is given by

$$L_{min} = 0 \tag{3}$$

where $\mathbf{c}_s$ is the vector pointing from the cube centre to one of its corners.

Table 1 shows the experimental results for a relatively small point cloud. Eq. (1) was parallelised and run on a graphics card (GPU) with 240 cores using the CUDA library. Specifically the CUDA time includes the transfer of data to the GPU memory (two float arrays), the execution of two kernels; Where the first calculates the distance between tow coordinates in $\mathbb{R}^3$ and the second kernel finds the minimum value of the previous distances. Finally the minimum value is returned to the CPU. In addition, the computational time of Eq. (1) was tested on the CPU (Intel Core-i5 3.3 GHz) without parallelisation. As expected, the computational time depends on the number of point-to-point distance calculations. When using the largest Octree representation (100mm side lengths for both robot and environment) the required calculation time is 0.070 and 0.008 seconds respectively for the GPU and the CPU. It is interesting to notice that for this coarse representation the CPU calculation is actually faster than the GPU calculations, most likely caused by the initial setup time required for the GPU calculations. For all the other representations (smaller Octree cubes and directly on point cloud), the CUDA-based GPU calculations are faster than the CPU calculations. For example for the Octree 10+1mm representation in Table 1 the GPU calculation takes 4.647 seconds compared to the CPU calculation of 18.276 seconds which makes the GPU calculation approximately 4 times faster. Table 2 shows the same result for a point cloud approximately 5 times larger. The same speed comparison for the Octree 10+1mm representation in this case shows that the GPU calculation is more than 20 times faster than the CPU calculation. Fig. 7 illustrate the computational time vs. the number of point-to-point calculations for both the GPU and the CPU. The number of point-to-point calculations is given by the multiplication of robot and environment points. The most time consuming operation was when applying eq. (1) to the point cloud. For the smallest point cloud set, Table 1 which requires 3.7 billion operations, the GPU was approximately 20 times faster than the CPU. The difference in speed increased even more for the largest point cloud in Table 2, consisting of 18 billion operations, in this case the CUDA approach was 32 times faster than the CPU. Fig. 7 summarises the number of elements with respect to time. The table shows that for a small number of operations, the CPU calculation time is lower than the CUDA time, but somewhere between 250,000 to 1,700,000 operations, the GPU is faster than the CPU.

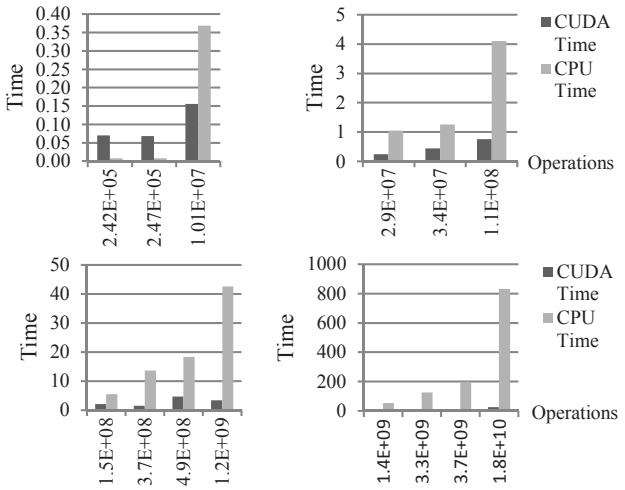

Figure 7: *Computational time vs. number of point-to-point operations.*

5 DISCUSSION

From the results in Table 1 and 2 it can be seen that for a smaller number of operations with 303 robot elements and 800 environment elements totaling $303 \cdot 800 = 242400$ operations, the CPU is 8.75 times faster than the GPU. Here, one operation is defined as one distance calculation. On the other hand, for a bit finer resolution of the Octree, 100mm for the robot and 10mm for the environment, there are 10,073,235 operations, but in this case the graphics card is 2.37 times faster than the CPU. The results presented in this paper, shows that for a smaller number of operations, it would be faster to let the CPU handle the calculations. The tipping point comes at around 10 million operations, when it is faster to let the GPU handle the calculations.

The graphics card used in this research, the Nvidia GTX 280 with 240 cores, was a card introduced in 2008. In comparison, state of the art today, a Nvidia GTX 690 introduced in 2012 has core count 12.8 times larger, with 3072 cores total. Based on the number of cores, not taken the difference in clock speed into consideration, the Nvidia GTX 690 should perform several factors better than the GTX 280. But this speed-up would not just depend on the number of cores directly. As discussed in Gregg and Hazelwood (2011), for cases where huge amounts of data have to be copied to the graphics cards memory, which in this case is the number of elements in the point cloud / Octree, the total time could be increased drastically.

When comparing the pros and cons of the point cloud and the Octree as two different means to present the environment, it was found that that the Octree is ideal for setups where the exact coordinate location

Method	CUDA Time	CPU Time	Robot Elements	Environment Elements
Point cloud	9.694	197.455	19 221	192 953
Octree 1+1mm	3.369	42.567	10 809	107 213
Octree 10+1mm	4.647	18.276	4 397	111 395
Octree 100+1mm	0.443	1.254	303	112 639
Octree 10+10mm	2.049	5.494	4 397	33 574
Octree 100+10mm	0.156	0.369	303	33 245
Octree 100+100mm	0.070	0.008	303	800

Table 1: *Experimental results with a relatively small point cloud of environment. Octree X +Y mm means cubes with X mm sidelength for the robot and Y mm sidelength for the environment.*

Method	CUDA Time	CPU Time	Robot Elements	Environment Elements
Point cloud	25.268	833.228	19 221	958 021
Octree 1+1mm	5.29	124.213	10 809	308 069
Octree 10+1mm	2.450	52.897	4 397	322 638
Octree 100+1mm	0.754	4.101	303	370 194
Octree 10+10mm	1.459	13.639	4 397	85 277
Octree 100+10mm	0.246	1.053	303	94 670
Octree 100+100mm	0.068	0.008	303	814

Table 2: *Experimental results with a relatively large point cloud of environment.*

is not needed. If the exact measurement information is needed the uncompressed and un-approximate point cloud is the best approach. It is notable that by increasing the resolution of the Octree to the floating point representation, one may match the accuracy of the point cloud representation. However, this special case may be giving the system much more total overhead than using the point cloud directly. Another positive thing with the Octree is the inherent compression of the sensor data, which is distributed by steps of one cube side. Varying the cube size, gives a density and hence different compression, which in turn reduces the number of elements and therefore also the calculation time.

By expanding the collision detection algorithms, it is possible to tell on which link, and at which location there is soon to be a collision. This information could be very valuable to an adaptive system, because the robot representation could be changed on the fly to change the resolution of the robot model and there are two main reasons to do so. First, by having a lower resolution of the robot when it is far away from other objects will decrease the calculation time, which in some cases have the desirable outcome that the robot could follow a path at higher velocity. Second, if the robot is approaching to do a part manipulation, the resolution of e.g. the wrist could be increased such that no collision is reported by passing objects on a very short range, an example of this is shown in 8. To be able to choose when the links should change resolution will require a modification of (3)

$$L_{min} - RC_{dist} < \|\mathbf{c}_s\|$$
$$RC_{dist} < L_{min} + \|\mathbf{c}_s\|$$

where RC_{dist} is the user defined resolution change parameter such that PC_{obs} increases the resolution of a particular joint or a collection of joints if the corresponding part is greater. For a new resolution a new value for RC_{dist} has to be chosen. Of course the values of RC_{dist} would be chosen before robot task program execution, to make the system more independent from human input at execution time.

In this paper, all the points in the scene were checked, but another approach could be to check only the points which the robot could reach in the next step (plus a safety margin). This could be done by doing a quick conservative calculation of where each robot joint could be positioned at the next time step and make the calculations only in the surrounding volume.

A challenge when the robot should adapt to the environment is to differentiate between the physical changes that are expected and should happen, and the changes that not are supposed to happen. One example of such expected changes could be when the robot

manipulates one or more objects in the environment. As this is a part of the normal operation, it should not raise a collision detection, which again means that the robot should be able to interact, or put in another words, "collide" with the object. To address this, one may associate an attribute with each object. As a simplistic example, let these attributes have the following values: "green", "orange" and "red". The elements of the static part of the scene, i.e., the areas that should not change during the time of operation and the robot is not allowed to interact with it, are marked "orange". Objects that the robot should interact with, should be marked with "green". Finally, objects introduced which are new to the scene should be marked as "red". Further, it should also be associated a distance to each of the three attributes. The new unknown objects in the room, marked with "red", are expected to have higher uncertainty associated with them, compared the original environment, and hence, the associated collision detection distance should therefore be larger than that for the static environment. The static environment would then warn about collision at closer distance than for the new objects. Finally, the objects to be manipulated would have zero distance or no distance, which means that interaction with these objects should not result in a collision detection.

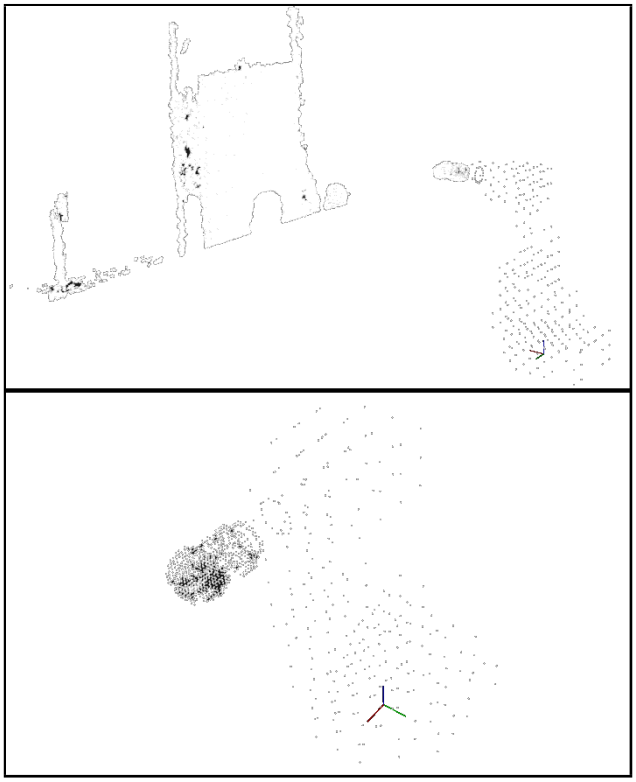

Figure 8: *Example of different Octree resolutions on the robot.*

6 CONCLUSIONS AND FUTURE WORK

This paper shows that the traditional approach with intersecting Octree cubes could be challenged by the emerging massively parallel GPU technology, bringing additional information such as near collision. CUDA could reduce the calculation time significantly for a large number of operations, on the other hand, for a smaller number of calculations, the CPU is faster. In addition, this paper shows that it is beneficial to use an Octree representation of the environment if the computational time should be kept low. In addition, it is proposed to be using different resolution for the robot and the environment models, which will yield higher performance for an adaptable system. It has been shown that the developed CUDA algorithms in many cases outperforms the proposed CPU implementation and that GPU-based algorithms could be a favorable choice in real-time industrial robot collision detection applications.

Future work will focus on using and expanding the methods which have been developed in this paper. The next steps will focus on collision avoidance, which requires an accurate mapped environment and fast calculations to the nearest objects. In addition to this, developing methods for including newly discovered objects that have entered into the environment during operation, to the world map needs further attention.

Acknowledgments

The work is funded by ABB and the Norwegian Research Council through project number 193411/S60.

References

NVIDIA CUDA C - Programming Guide, 2006.

Anisi, D., Gunnar, J., Lillehagen, T., and Skourup, C. Robot Automation in Oil and Gas Facilities: Indoor and Onsite Demonstrations. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. Taipei, Taiwan, pages 4729–4734, 2010. doi:10.1109/IROS.2010.5649281.

Anisi, D., Persson, E., Heyer, C., and Skourup, C. Real-World Demonstration of Sensor-Based Robotic Automation in Oil & Gas Facilities. In

IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS). San Francisco, California, 2011. doi:10.1109/IROS.2011.6094440.

von Dziegielewski, A., Hemmer, M., and Schomer, E. High quality conservative surface mesh generation for swept volumes. In *Robotics and Automation (ICRA), IEEE Intl. Conf.* pages 764 –769, 2012. doi:10.1109/ICRA.2012.6224921.

Elseberg, J., Magnenat, S., Siegwart, R., and Nüchter, A. Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration. *Journal of Software Engineering for Robotics (JOSER)*, 2012. 3(1):2–12.

Faverjon, B. Obstacle avoidance using an octree in the configuration space of a manipulator. In *Robotics and Automation. Proc. IEEE Intl. Conf. on*, volume 1. pages 504 – 512, 1984. doi:10.1109/ROBOT.1984.1087218.

Gregg, C. and Hazelwood, K. Where is the data? why you cannot debate cpu vs. gpu performance without the answer. In *Performance Analysis of Systems and Software (ISPASS), IEEE Intl. Symposium on*. pages 134 –144, 2011. doi:10.1109/ISPASS.2011.5762730.

Hayward, V. Fast collision detection scheme by recursive decomposition of a manipulator workspace. In *Robotics and Automation. Proc. 1986 IEEE Intl. Conf. on*, volume 3. pages 1044 – 1049, 1986. doi:10.1109/ROBOT.1986.1087620.

Henrich, D., Wurll, C., and Wörn, H. 6 dof path planning in dynamic environments-a parallel online approach. In *Robotics and Automation, Proc. IEEE Intl. Conf.*, volume 1. pages 330 –335 vol.1, 1998. doi:10.1109/ROBOT.1998.676417.

Kaldestad, K., Hovland, G., and Anisi, D. CAD-Based Training of an Expert System and a Hidden Markov Model for Obstacle Detection in an Industrial Robot Environment. In *IFAC Intl. Conf. on Automatic Control in Offshore Oil and Gas Production*. Trondheim, Norway, 2012a. doi:10.3182/20120531-2-NO-4020.00036.

Kaldestad, K., Hovland, G., and Anisi, D. Obstacle Detection in an Unstructured Industrial Robotic System: Comparison of Hidden Markov Model and Expert System. In *10th IFAC Intl. Symposiums on Robot Control*. Dubrovnik, Croatia, 2012b. doi:10.3182/20120905-3-HR-2030.00059.

Ramisa, A., Alenya, G., Moreno-Noguer, F., and Torras, C. Using depth and appearance features for informed robot grasping of highly wrinkled clothes. In *Robotics and Automation (ICRA), IEEE Intl. Conf. on*. pages 1703 –1708, 2012. doi:10.1109/ICRA.2012.6225045.

Ross, P. Why cpu frequency stalled. *Spectrum, IEEE*, 2008. 45(4):72. doi:10.1109/MSPEC.2008.4476447.

Rusu, R. B. and Cousins, S. 3D is here: Point Cloud Library (PCL). In *IEEE Intel. Conf. on Robotics and Automation (ICRA)*. Shanghai, China, 2011. doi:10.1109/ICRA.2011.5980567.