Accelerated DNA-SLAM for RGB-D images

Mina Ameli[1], Oliver Wasenmüller[2], Mohammad Reza Soheili[1], Jamshid Shanbehzadeh[1] and Didier Stricker[2]

[1]Computer and Electrical Engineering Department,
Kharazmi University of Tehran, I.R. Iran.
+98 (21) 88830891

{std_ameli, soheili, jamshid}@khu.ac.ir

[2]Augmented Vision Department, DFKI GmbH, German Research
Center for Artificial Intelligence, Kaiserslautern, Germany.
+49 (631) 205-75 3500

{oliver.wasenmueller, didier.stricker}@dfki.de

ABSTRACT

In the highly active research field of Simultaneous Localization And Mapping (SLAM), RGB-D images have been a major interest to use. Real-time SLAM for RGB-D images is of great importance since dense methods using all the depth and intensity values showed superior performance in the past. Due to development of GPU and CPU technologies, the real-time implementation of the mentioned algorithms is no longer an impassable problem. In this paper, we present an acceleration approach for the DNA-SLAM algorithm. We argue some possible challenges while converting the CPU implemented algorithm to the GPU. Finally, runtime evaluation and improvements are shown on the public CoRBS dataset.

CCS Concepts

• **Computing methodologies ~Motion capture**
• **Computing methodologies ~Vision for robotics**
• **Computing methodologies ~Parallel programming languages**

Keywords

RGB-D images; Dense SLAM; Real-time; ToF Camera; CUDA; DNA-SLAM; GPU-accelerated.

1. INTRODUCTION

Many Robotics and Computer Vision applications include navigation and mapping, which needs to be performed in real-time. Visual Simultaneous Localization And Mapping (SLAM) is the problem of finding the location of the camera and simultaneously creating a map of the environment, using only the information of the captured images.

Proposed approaches for this purpose can be dichotomized into a sparse and dense category. The former methods rely on the extraction and matching of sparse visual feature points while the latter category are dense, performing pixel-wise minimization of photometric and/or geometric constraints for all intensity and depth pixels.

While real-time sparse SLAM (as in [27]) and offline dense SLAM (like in [30]) are quite mature, recently real-time depth mapping (e.g. the works in [12], [13] and [24]) and visual odometry, the problem of tracking the pose of the camera/robot (e.g. proposed methods in [4], [9] and [28]) have become conceivable. There are two important enabler and impetuses. The first is the opening up of graphics processing units for general purpose computing and the second is the advent of affordable RGB-D sensors.

One of the parallel computing platforms, which are often utilized for dense SLAM methods are General-Purpose computing on Graphics Processing Units (GPGPUs). In order to perform high speed and often real-time processing on full-resolution images for every frame, they actuate the enormous parallelism in graphic cards. GPU-based programming, as an affordable and attainable technology, has indisputable empowering role in recent dense SLAM research [22].

Another above mentioned catalyst is the release of cheap RGB-D sensors. They are capable of capturing RGB-D images containing a synchronous color image and depth image. These cameras have been widely applied and their release has resulted in great progress in odometry and dense mapping in recent years [23].

The two common approaches for measuring depth data are Pattern Projection and Time-of-Flight (ToF). Cameras with Pattern Projection, such as Microsoft Kinect v1, Asus Xtion Pro, project a known pattern into the scene and estimate the depth out of the distortion of the pattern. Recently, ToF cameras, such as Microsoft Kinect v2 [14] or Google Tango [15], resolve distance by estimating the time emitted light takes from the camera to the subject and back for each pixel in the image.

Indeed, the new Kinect v2 device is used more in many recent and future research since they claim a higher accuracy in general and there are publicly available datasets like CoRBS [2] using this device. In addition, a rigorous evaluation and comparison of the depth images of Kinect v1 and Kinect v2 is available in [3], providing the basis for modeling the errors of the mentioned devices. ToF cameras have a noise characteristic [3], which sources are dark and glossy scenes, colors, large scene distances, pixels close to the image boundaries, flying pixels close to depth discontinuities, etc. In some experiments with ToF cameras by [1], it was detected that the geometric consistency assumption for dense motion estimation is often violated due to the sensor noise, leading

ICIGP'18, February 24–26, 2018, Kowloon city, Hong Kong.
 ISBN: 978-1-4503-6367-9…$15.00.
DOI: http://dx.doi.org/10.1145/12345.67890

to inaccurate trajectories. Thus, Wasenmüller *et al.* proposed a Dense Noise Aware SLAM (DNA-SLAM) to address this problem by a sophisticated weighting scheme.

In this paper, we present a GPU-based extension for DNA-SLAM to accelerate this algorithm. First, we review number of publications that have been released over the last few years for real-time RGB-D SLAM. Then, after explaining DNA-SLAM approach, we argue some possible challenges while converting the CPU implemented program to the GPU. Finally, runtime evaluation on the public CoRBS dataset using Kinect v2 (ToF) is provided showing improved results.

2. RELATED WORK

In this section, we review numerous works on real-time RGB-D SLAM, according to the categories mentioned in the introduction. First, we will discuss CPU-based and then GPU-based implementations.

There are publications on SLAM that only take the advantages of RGB-D images to be executed in real-time and use CPU implementation. Among sparse methods, the approach of Engelhard *et al.* [17] can be mentioned, in which they use SURF features for point-wise correspondence and in next step, estimate the position by RANSAC and finally use ICP and pose graph solver respectively for refinement and optimization of the pose.

Regarding dense methods, two main primitive works are Steinbrücker *et al.* [4] and Audras *et al.* [18] which minimize the photometric error between consecutive RGB-D frames and perform in real-time. Following, Kerl *et al*. [6] extend their approach by weighting photometric errors according to the t-distribution and published as the well-known DVO algorithm. Wasenmüller *et al.* [1] extended this concept to the noise characteristic of ToF cameras. Klose *et al.* [19] present a motion estimation approach based on second-order minimization that performs in real-time. Furthermore, evaluation and comparison over the algorithms, which minimize photometric errors, is available in [19] and provides a categorization of them. Ma *et al.* [25] as a dense method, used frame-to-plane alignment beside frame-to-keyframe approach for tracking, in order to reduce drifts and still performs in real-time.

On the other hand, some proposed works use both catalysts, i.e. they are GPU-based RGB-D SLAM. Lee *et al.* [16] present a sparse solution that achieves odometry by feature extraction and RANSAC and subsequently optimize the estimation with ICP. In order to accelerate the execution time, they compute feature extraction and ICP step in parallel.

As one of the basic work on dense methods can refer to KinectFusion by Newcombe et al. [13], which is extended in numerous publications. Kintinuous by Whelan et al. [11], developed KinectFusion using camera odometry estimation method and also creating a high-quality map. Then, in [10] they proposed a GPU-based implementation of visual odometry algorithm that provides real-time operation of their earlier work Kintinuous. Roth et al. [29] also expand KinectFusion, in order to enabling camera to roam freely in mobile robotics and similar applications. As more example of GPU-based dense RGB-D SLAM, we can mention ElasticFusion by Whelan et al. [31]. They used dense frame-to-model for odometry and windowed surfel-based fusion with model refinement.

To sum up, in the ideal case, the solutions should perform in real-time with high accuracy but there is almost always a tradeoff between accuracy and execution time. In contrast to all related works, there is still lack of a real-time method to deal with noise characteristic of Time-of-Flight cameras with high accuracy. To fulfill this purpose, in this paper, we accelerate DNA-SLAM [1], the noise aware approach which especially designed for ToF RGB-D cameras. Details and the results are discussed in the following.

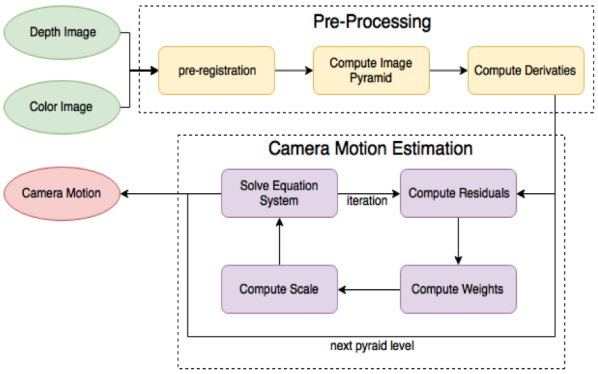

Figure 1. Overview of DNA-SLAM algorithm [1]. After preprocessing step, iterative motion estimation is performed on RGB-D pairs. Detail is described in the DNA-SLAM algorithm section.

3. BACKGROUND

In this section, we explain two specified components of this paper including the original DNA-SLAM algorithm [1] and CUDA programming.

3.1 DNA-SLAM

Dense Noise Aware SLAM (DNA-SLAM) [1], performs a weighting approach to specifically address noise characteristics of ToF RGB-D cameras. The basic idea behind this work is computing an individual weight for each single pixel based on its reliability in dense motion estimation. The camera motion is estimated according to the photometric and geometric consistency assumptions, following the state-of-the-art works [5], [10] and [21].

By experiments on ToF RGB-D images, Wasenmüller *et al.* [1] found a violation on geometry consistency assumption for many pixels in ToF cameras leading to inaccuracies in motion estimation. In their experiments they detected that the local depth derivative is a good indicator for the location and magnitude of the violation. Thus, they transform the derivative together with the photometric and geometric residual into a sophisticated weighting function. In the same concept with [4], [6] and [21] papers, the equation for estimating camera motion ξ, using minimization of the residuals r_i for entire n pixels is

$$\xi = \arg\min_{\xi} \sum_{i}^{n} w(r_i)(r_i(\xi))^2 , \qquad (1)$$

where w is the mentioned weight that is assumed as t-distribution of the derivative residuals in computation.

A detailed discussion about preliminaries and how to solve this minimization equation is provided in original paper [1]. An overview of the algorithm is depicted in Figure 1: after pre-processing step on RGB-D input images, camera motion estimation

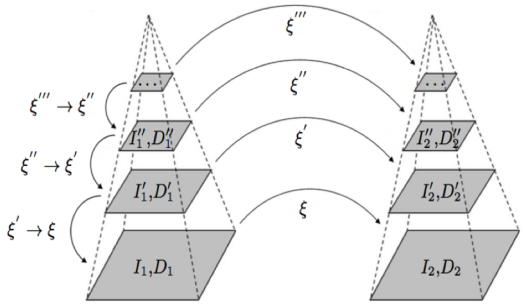

Figure 2. Coarse-to-fine strategy in DNA-SLAM [1] in order to ensure small motion estimation between two consecutive images. Each image tuple (intensity and depth) is represented as a pyramid with four levels.

Table 1. Average execution time of different reduction methods

Methods	Average [1]	Total mean time(s)
Pure CPU	0.150	2.161
atomicAdd	0.230	3.445
Pure CUDA	0.120	1.845
Thrust reduction [7]	0.310	4.447
Thrust reduction by key [7]	0.270	3.969

[1] Average execution time of each 50 frames in seconds on exemplary selected E1 image sequence of CoRBS dataset [2].

is computed with four serial tasks, consisting of compute residuals, compute weight, compute scale and solve equation system respectively, in two loops.

The external loop contains building pyramid representation of the image in order to utilize coarse-to-fine strategy for ensuring small camera motions. As demonstrated in Figure 2., in this process they used four levels of pyramid, which RGB and depth image pairs (I_i, D_i) are subsampled by halving the RGB-D pairs resolution. The camera motion estimation starts in the coarsest level first and is then used as an initialization to finer levels.

The inter-loop is composed of the above mentioned tasks to perform iterative re-weighted least square in order to solve the basic motion estimation formula in Equation 1. They worked in the same concept of the state-of-the-art algorithms [6], [25] and [26] and minimize the photometric and geometric error, which is defined in the residuals. Residuals are used for both weighting and motion estimation purpose. Detailed information on residual definition can be referred to the original paper.

Compared to state-of-the-art algorithms, while DNA-SLAM is applicable with Kinect v1 datasets, shows superior accuracy on ToF cameras like Kinect v2, drifts are reduced and results are closer to ground truth. But, in order to gain around 100ms execution time for performing motion estimation in real-time, requires an acceleration.

3.2 CUDA Programming

Compute Unified Device Architecture (CUDA) is a parallel computing technology and API model, released in 2006 by NVIDIA. This platform provides ease of programming on Graphical Processing Units (GPUs), and numerous CUDA accelerated libraries are available for wide range of Computer Vision applications.

Every CUDA program is performed in a general procedure. After allocating space in device memory, first, required data should be transformed from Host memory to GPU global memory. Next, CPU instructs the process to GPU. Then, GPU execute parallel in each core. And finally, results should be transformed back from GPU memory to Host memory and free the allocated memory space. So all GPU-based applications, also use the CPU for performing subsidiary tasks include initializations, launching the kernels and post-processing.

Fundamental building block of a parallel program is consisting of threads. We usually need thousands of concurrent threads to gain the best possible performance on a device, so, for better data association, threads are grouped into blocks. Aggregation of blocks form grids, which same blocks in a grid contain the same number of threads. To prevent poor memory management, the suitable number of threads per block that determine the number of blocks and grids, is of great importance based on length of data [9].

Other important preliminary aspect in CUDA programming is to be familiar with different memory systems within the GPU, because memory throughput can generally dominate the program performance. In fact, perception above three class of storage consist of registers, global memory and shared memory, result in maximum utilization of each type [9].

Since the GPU has its own challenges, the optimum GPU-based algorithm, is not necessarily the best in the CPU and vice versa. So, there are different standard patterns for parallel algorithms, which imply on the access pattern of reading and writing from/on memory locations. The patterns include Map, Reduce, Gather, Scatter, Scan, Search and Sort.

We introduce three related patterns here. Map convey the scheme of pattern, in which a function should be applied on a data array. This pattern is straightforward in implementation; each thread operates on an indexed data of array with no collision in parallel. Reading multiple data items to a single location of the memory, is the procedure of Gather pattern. In comparison, writing a single data item to multiple locations, is Scatter pattern. Reduce, point to the pattern, in which a binary associative operator should be applied on a list of linear values and result would be written on a single location of the memory. Common operators are summation, multiply, max and min. We will briefly discuss about the effect of these three patterns on gaining better performance, in the program pattern subsection.

4. ACCELERATION APPROACH

The DNA-SLM [1], as a sophisticated dense noise aware approach, requires an acceleration to execute around 100ms. Since having dense data and working on list of linear values in each step of this algorithm, in order to accelerate the original DNA-SLAM, we implement motion estimation function by CUDA programming. According to Figure 1., we have two nested loops with inter-loop dependencies of the tasks: Compute Residuals, Compute Weights, Compute Scale and likelihood part of Solve Equation System blocks, which perform on intensity and depth images and also list of linear values. We implement these tasks with CUDA

Table 2. Rotational drifts in deg/s on exemplary selected E1 sequence of CoRBS dataset [2].

Errors	CPU	GPU
RMSE	1.426480	1.430815
Mean	1.287833	1.289820
Median	0.021025	0.020910
STD	0.613457	0.619353
Min	0.043994	0.043237
Max	3.331796	3.399846

Table 3. Translational drifts in m/s on exemplary selected E1 sequence of CoRBS dataset [2].

Errors	CPU	GPU
RMSE	0.034953	0.034825
Mean	0.031289	0.031152
Median	0.030043	0.029902
STD	0.015580	0.015525
Min	0.000938	0.000975
Max	0.064148	0.064503

programming. We confront several challenges that are likely to occur in other similar applications. First, we express them as the general issues and then explain them in the concept of DNA-SLAM.

4.1 Challenges

In this part, we briefly mention several challenges that might occur in some applications and some of them may prevent gaining best results. Also some guidelines which should be followed in order to achieve optimum performance are described.

4.1.1 Program Pattern

Acceleration of GPU-based implementation in contrast with CPU code is highly related to the pattern of the program. Both Gather and Scatter patterns, in case of locality and repeated access, will perform faster. Reduction pattern can almost be referred as the most time consuming one, because of the atomic characteristic of its operations. Atomic operations are those where hardware performs a barrier point at the entry of it and only can be guaranteed the completion of the single operation without any other thread interruption. In fact, it occurs as the race condition, in which sequence of execution for the threads makes a difference in the results without no barrier. The updated generation GPUs support faster atomic operations [9], different approaches have been proposed to address faster solution. Also a nicely discussed approaches issued this matter for reduction pattern is presented in [20], which explores the algorithms trade off and compare their execution time.

4.1.2 Serial vs. Parallel

One of the possible paralleling approaches beyond the performing concurrent threads in GPU computing, is executing functions in parallel like in [16], which they compute feature extraction and ICP step in parallel. In fact, we can use task-based parallelism, beside data-based parallelism. The concept that can relatively be used, is dynamic parallelism, a CUDA extension, which enables a kernels to perform and synchronize nested tasks that provides easier paralleling approach for task-based parallelism purpose. Regarding to carrying out the execution of processes simultaneously, is taking into consideration the dependency between input and output of each function, is especially significant. Dependent tasks must be performed in serial and there is no chance for serialization.

4.1.3 Shared Memory

Data reuse requires the use of shared memory into consideration. To ensure coalesced access to the global memory, i.e. avoid redundant access to global memory, leading to decrease of the wasted global memory bandwidth. Shared memory is held in common for the threads of a block. Besides limitation of this resource, that can not be sophisticated for applications with high demand level of data reuse, some sort of programs, do not have any reusable data to take advantages of this facility and for every single task, data would be updated entirely.

4.1.4 Floating Point Precision

Floating point precision performance is of great importance for particular utilizations in order to achieve the eligible accuracy for the numerical results. In experiments regardless of hardware and compiler differences on preserving semantics of floating points, there are different floating point standards in CPUs and GPUs. On the GPU devices with compute capability 2.0 and above, the single precision is 32-bit and double precision is 64-bit. On the other hand, some CPUs support 80-bit extended precision. Consequently, during calculations on floating points you would likely to get small differences even if using double precision. Fully-descripted information on issued related to floating point is available in [8].

4.1.5 Data Transfer

The last but not the least important factor is data transfer. Data transfer between CPU and GPU is a really time critical action, specially by increasing amount of data in scale of million above, spent time on transferring data is processing time lost. So, one of the most important factor to gain optimum performance is minimizing the transfers of data between CPU and GPU.

4.2 Accelerated DNA-SLAM

Given the mentioned contents in above sections, we implement GPU-based motion estimation function of DNA-SLAM [1] by CUDA programming. As presented in Figure 1. Compute Residuals, Compute Weights, Compute Scale and likelihood part of Solve Equation System blocks, are the main components of this conversion. We will concern each of the challenges respectively:

All functions have Reduction pattern and we had examined four different reduction methods contain of atomic defined function of CUDA for summation (atomicAdd), faster parallel reduction of Kepler GPUs and two reduction methods of well-known Thrust library [7], leading to different execution times. In comparison, faster parallel reduction of Kepler GPUs, using Kepler's shuffle instructions, gained the best performance. The average execution time of each approach is available in Table 1. Pure CUDA implies for Kepler shuffle method.

We had two nested loops with inter-loop dependencies of the tasks, that input of each step is related to output of the prior task, so we were not able to take advantage of task-based parallelism and all the methods are using data-based parallelism. Consequently, we had limitation in using dynamic parallelism since having serial dependent functions.

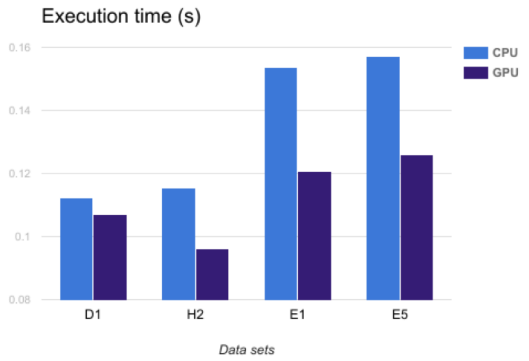

Figure 3. Comparison of CPU and GPU implementation execution time in second, on selected sequence of the CoRBS dataset [2].

In addition, except constant variables, we had bounded data reuse to gain better performance above shared memory. Also time critical action of data transferring for solving small equation system in the CPU is the other restriction for our acceleration approach. Since operations on small size data would yield poor performance in GPU rather than CPU, and using equation solver libraries in CUDA for the short equation system is more time consuming, we had data transformation in this part of the iteration. As the final matter, having only once memory allocation for iterative loop, had the great impact on performance.

5. EXPERIMENTS

In this section, comparison of results between the CPU and GPU implementation is discussed. First, we introduce the used dataset and then, results are presented. All the experiments were implemented on an Intel Xeon CPU W3520 with 2.67GHz and the NVIDIA GeForce GTX 780 Ti graphic card supporting GPU programming with CUDA platform and computational capability of 3.5.

5.1 Dataset

We have evaluated the implemented algorithm on the public Comprehensive RGB-D Benchmark for SLAM (CoRBS) dataset [2], the only available dataset using Microsoft Kinect v2 [14]. Twenty image sequences of CoRBS benchmark is consist of four different scenes, which are *Human, Desk*, *Electrical Cabinet* and *Racing Car*. Different characteristic is concerned for each scenes trajectory, so it can be applied for diverse scenarios and applications. The *Electrical Cabinet* scene cover the most challenging geometric characteristic. The ground truth trajectories are acquired by an external precise motion capture system.

5.2 Results

Evaluations are based on accuracy and speed of performed implementation. Accuracy measurement is determined by the rotational and translational drifts which are presented in Table 2. and Table 3. respectively in deg/s and m/s with six different measures (RMSE, Mean, Median, STD, Min and Max). There is a small difference between CPU and GPU errors amount, because of different floating point precision, that discussed heretofore.

Table 4. contains evaluation of execution time in second that we prepare a visual chart in Figure 3. for better demonstration. There

Table 4. Comparison of execution time in second, on selected sequence of the CoRBS dataset [2].

Dataset	CPU	GPU	Improvement
D1	0.11216	**0.10704**	1.04x
H2	0.11534	**0.09872**	1.16x
E1	0.15355	**0.12087**	1.27x
E5	0.15736	**0.12606**	1.24x

is different improvement for each image sequences. In result of analyzing, we discovered, there is a relation between length of data and acceleration improvement. In E1 and E5 sequences, valid pixels of the images are more than D1 and H2 sequences. As CUDA programming is guided for massive parallelism, better performance is acquired for more massive data. We gain almost 20% acceleration for these datasets.

As depicted in DNA-SLAM algorithm overview in Figure 1. motion estimation building block consist of two loops within four serial dependent modules. As mentioned in challenges section, we confront the situation that could not parallel these modules to gain better results. Also, the patterns of our program, limited us in using shared memory in achieving foremost performance.

Applying the more powerful GPU and everyday fast progress of GPGPUs technology and proponing new facilitated methods for CUDA programming, definitely will lead to better result for this work and similar researches.

6. CONCLUSION

This paper presented an accelerated implementation of DNA-SLAM algorithm, a Dense Noise Aware SLAM approach, which specifically addresses noise characteristics of ToF RGB-D cameras. For acceleration purpose, we use CUDA programming approach and review some possible challenges that occur during converting CPU code into GPU implemented one. That includes different kinds of program patterns, the capability of parallelization of the tasks, limitation of shared memory usage, different floating point precision and data transfer bottleneck.

We evaluate the results by using public CoRBS dataset using Kinect v2 device. The results of execution time show almost 20% improvement for datasets with more valid pixels, compared to the original DNA-SLAM algorithm [1]. Also, there is a small difference between rotational and translational errors due to different floating point standard precision of CPU and GPU.

Summarized, applying the more powerful GPU and also the necessities along with the fast progress of GPGPUs technology, which is supporting new facilitated methods for CUDA programming definitely will consequence in better result for this work and other researches in the same context.

7. ACKNOWLEDGMENTS

This work was carried out in the context of a research cooperation between the Augmented Vision department of DFKI, Germany, and Kharazmi University of Tehran, Iran. We would like to give a special gratitude to Stephan Krauß for his suggestions.

8. REFERENCES

[1] Wasenmüller, O., Ansari, M. D. and Stricker, D. 2016. DNA-SLAM: dense noise aware SLAM for ToF RGB-D cameras. In *Asian Conference on Computer Vision Workshop (ACCV workshop).* (Springer, 2016).

[2] Wasenmüller, O., Meyer M., and Stricker, D. 2016. CoRBS : Comprehensive RGB-D benchmark for SLAM using Kinect v2. In *Winter Conference on Applications of Computer Vision (WACV)*. http://corbs.dfki.uni-kl.de/.

[3] Wasenmüller, O. and Stricker, D. 2016. Comparison of kinect v1 and v2 depth images in terms of accuracy and precision. In *Asian Conference on Computer Vision Vision Workshop (ACCV workshop)*. (Springer, 2016).

[4] Steinbrücker, F., Sturm, J. and Cremers, D. *2011.* Real-time visual odometry from dense RGB-D images. In *IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*.

[5] Kerl, C., Sturm, J. and Cremers, D. 2013. Robust odometry estimation for RGB-D cameras. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[6] Kerl, C., Sturm, J. and Cremers, D. 2013. Dense visual SLAM for RGB-D cameras. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

[7] Thrust: (CUDA toolkit) www.docs.nvidia.com/cuda/thrust/.

[8] Whitehead, N. and Fit-Florea, A. 2011. Precision & performance: Floating point and IEEE 754 compliance for NVIDIA GPUs. *rn (A+ B)*, *21*(1).

[9] Cook, S. 2012. CUDA programming: a developer's guide to parallel computing with GPUs. Newnes.

[10] Whelan, T., Johannsson, H., Kaess, M., Leonard, J. J. and McDonald, J. 2013. Robust real-time visual odometry for dense RGB-D mapping. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[11] Whelan, T., Kaess, M. Fallon, M. F., Johannsson, H., Leonard, J. J. and McDonald, J. 2012. Kintinuous: Spatially extended kinectfusion. In *Advanced Reasoning with Depth Cameras RSS Workshop on RGB-D*.

[12] Wasenmüller, O., Meyer M., and Stricker, D. 2016. Augmented Reality 3D discrepancy check in industrial applications. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*.

[13] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J. Kohi, P., Shotton, J., Hodges, S. and Fitzgibbon, A. 2011. KinectFusion: Real-time dense surface mapping and tracking. In *IEEE international symposium on Mixed and augmented reality (ISMAR).*

[14] Microsoft: (Kinect v2) www.microsoft.com/enus/kinectforwindows/.

[15] Google: (Tango) www.google.com/atap/project-tango/.

[16] Lee, D., Kim, H. and Myung, H. 2012. Gpu-based real-time rgb-d 3d slam. *9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*.

[17] Engelhard, N., Endres, F., Hess, J., Sturm, J. and Burgard W. .2011. Real-time 3D visual SLAM with a hand-held RGB-D camera. In *Proc. of RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum.*

[18] Audras, C., Comport, A., Meilland, M. and Rives, P. 2011. Real-time dense appearance-based SLAM for RGB-D sensors. In *Australasian Conf. on Robotics and Automation*.

[19] Klose, S., Heise, P. and Knoll, A. 2013. Efficient compositional approaches for real-time robust direct visual odometry from RGB-D data. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

[20] van den Braak, G.J., Nugteren, C., Mesman, B. and Corporaal, H., 2011, August. Fast hough transform on GPUs: Exploration of algorithm trade-offs. In *International Conference on Advanced Concepts for Intelligent Vision Systems*. Springer, Berlin, Heidelberg.

[21] Gutierrez-Gomez, D., Mayol-Cuevas, W. and Guerrero, J.J. 2016. Dense RGB-D visual odometry using inverse depth. In *Robotics and Autonomous Systems*, *75*.

[22] Whelan, T., Kaess, M., Johannsson, H., Fallon, M., Leonard, J.J. and McDonald, J. 2015. Real-time large-scale dense RGB-D SLAM with volumetric fusion. In *The International Journal of Robotics Research*, *34*(4-5),

[23] Yoshida, T., Wasenmüller, O. and Stricker, D. 2017. Time-of-Flight Sensor Depth Enhancement for Automotive Exhaust Gas. *IEEE International Conference on Image Processing (ICIP)*.

[24] Newcombe, R. A., Lovegrove, S. J., and Davison, A. J. 2011. DTAM: Dense tracking and mapping in real-time. In *IEEE International Conference on Computer Vision (ICCV).*

[25] Ma, L., Kerl, C., Stückler, J. and Cremers, D. 2016. May. Cpa-slam: Consistent plane-model alignment for direct rgb-d slam. In *IEEE International Conference on Robotics and Automation (ICRA).*

[26] Meilland, M. and Comport, A.I., 2013, November. On unifying key-frame and voxel-based dense visual SLAM at large scales. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* 3677-3683.

[27] Endres, F., Hess, J., Engelhard, N., Sturm, J., Cremers, D. and Burgard, W. 2012. An evaluation of the RGB-D SLAM system. In *International Conference on Robotics and Automation (ICRA).*

[28] Meilland, M., Comport, A.I. and Rives, P., 2011. Dense visual mapping of large scale environments for real-time localisation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

[29] Roth, H. and Vona, M., 2012, September. Moving Volume KinectFusion. In *BMVC*. Vol. 20, No. 2, 1-11.

[30] Huang, A.S., Bachrach, A., Henry, P., Krainin, M., Maturana, D., Fox, D. and Roy, N., 2017. Visual odometry and mapping for autonomous flight using an RGB-D camera. In *Robotics Research.* Springer International Publishing.

[31] Whelan, T., Leutenegger, S., Salas-Moreno, R., Glocker, B. and Davison, A. 2015. ElasticFusion: Dense SLAM without a pose graph. Robotics: Science and System.