8

Parallel Computing in Mobile Robotics for RISE

Janusz Będkowski
Institute of Automation and Robotics, Warsaw University of Technology
Industrial Research Institute for Automation and Measurements
Poland

1. Introduction

RISE − Risky Intervention and Surveillance Environment is very demanding task for mobile robot where time is crucial. It can be assumed that on-line task execution is very important, therefore the research in parallel computing applied in mobile robotics is needed. Nowadays many researchers are focused on such approaches that uses GPGPU (General Purpose Graphics Processing Unit) for improvement State of The Art (SoA) algorithms. In this chapter three areas of research are shown such as 3D data registration, robot navigation and 3D cloud of points processing for normal vector computation, all are improved by GPGPU computation. The on − line data registration problem is discussed. The approach based on robust KNN k-nearest neighborhood search applied for improvement of ICP algorithm is shown. The path planning parallel approach based on modified diffusion method is shown. On − line 3D cloud of points' segmentation based on normal vector computation is presented. The set of proposed algorithms where tested on GPGPU NVIDIA CUDA GF 580, the results are satisfying. The improvement of SoA algorithms based on CUDA implementation shows on-line advantages during real task execution.
Robust ICP algorithm is needed in mobile robotics applications where data registration has to be performed on−line. New generation of Time of Flight and RGB-D cameras will offer better accuracy and resolution, therefore GPU accelerated data registration algorithms will improve robot navigation, obstacle avoidance and map building. It can be stated that commercial 3D scanners based on rotated lasers offer data acquisition time < 3 seconds, therefore ICP that works in this time will be enough for on-line map building in stop-scan fashion. For this reason robust data registration algorithm based on 3D space decomposition is proposed and the ICP (Iterative Closest Point) approach is chosen as registration method. Algorithm in current version performs matching of two cloud of points up to $512 \times 512 = 262144$ in 300ms for 30 ICP iterations (NVIDIA GF 580). The proposed solution is efficient since it performs nearest neighbor search using a bucket data structure (sorted using CUDA primitive) and computes the correlation matrix using parallel CUDA all-prefix-sum instruction. The amount of processed points can be increased by implementation on NVIDIA GPU with Compute Capability 2.1.

Robot navigation is very important task that can be improved using parallel computation since robot environment is represented by grid of cells. This report focuses on graphics processing units (GPUs) in particular applied for modified diffusion method. The experiments performed with 512×512 pixels grid maps show an advantage of GPU that results on−line path planning in static environment that can be expanded by dynamic obstacles.
Fast data segmentation technique based on local neighborhood search is implemented. The result of segmentation is colored map where different colors correspond to different objects such as {wall, floor...}. The research is related to the problem of collecting 3D data with DGB-D camera mounted on rotated head and 3D laser scanner, to be used in mobile robot applications. Assumed performance of data registration algorithm is achieved, therefore it can used as on−line. Robust nearest neighbor search procedure is obtaining normal vectors for each range point. Normal vectors are represented as r,g,b values, therefore similar colors correspond to one plane.
The experiments where performed using different type of 3D sensors including PMD Photonic Mixer Devices camera, X-BOX 360 Kinect sensor and rotated LMS SICK 200 (3DLSN Unit). Results are satisfying and it is planned to continue research and expand into another areas from mobile robotics such as 6D SLAM and semantic mapping.
The paper is organized as follows: in *Related work* the State of The Art concerning parallel computing applied in mobile robotics is described. In section $3D$ *data registration* the ICP Iterative Closest Point algorithm improved by parallel KNN (k-nearest neighborhood search) approach is shown. Section *Path planning* is demonstrating parallel approach for robot navigation purpose. In *Cloud of points processing* section the implementation of parallel normal vector computation for 3D cloud of points segmentation is described. In *Experiments* the results are shown. *Conclusion and future work* finalize the paper.

2. Related work

Several researches of 3D mapping are based on the simulation of 3D laser range finder to obtain 3D cloud of points Magnusson et al. (2007). In most cases 3D laser simulator is built on the basis of a rotated 2D range finder. The rotation axis can be horizontal Nüchter et al. (2003), vertical Montemerlo & Thrun (2004) or the rotational axis in the middle of the scanner's field of view Kohlhepp et al. (2004). Another approach of obtaining 3D cloud of points using 2 orthogonal lasers is shown in the work of Thrun Thrun et al. (2000). The applications are related with urban mapping Ortega et al. (2009).
Alignment and merging of two 3D scans, which are obtained from different sensor coordinates, with respect to a reference coordinate system is called 3D registration Huber & Hebert (2003) Fitzgibbon (2001) Magnusson & Duckett (2005). Park Park et al. (2010) proposed a real-time approach for 3D registration using GPU, where the registration technique is based on the Iterative Projection Point (IPP) algorithm. IPP technique is a combination of point-to-plane and point-to-projection registration schemes Park & Subbarao (2003). Processing time for this approach is about $60ms$ for aligning 2 3D data sets of 76800 points during 30 iterations of the IPP algorithm. Fast searching algorithms such as the k-d tree algorithm are usually used to improve the performance of the closest point search Nuchter, Lingemann & Hertzberg (2007) Rusinkiewicz & Levoy (2001). GPU accelerated nearest neighbor search for 3D registration is proposed in work of Qiu Qiu et al. (2009), where the advantage of Arya's priority search algorithm described in Arya & Mount (1993) to fit

NNS in the SIMD (Single Instruction Multiple Data) model was used for GPU acceleration purpose. Purcell suggested that k-d tree and priority queue methods are efficient but difficult to be implemented on GPU Purcell et al. (2003). Garcia proves, that a brute force NNS approach using NVidia Compute Unified Device Architecture (CUDA) is 400 times faster over the CPU k-d tree implementation Garcia et al. (2008). GPU-based NNS with advanced search structures is also used in the context of ray tracing Foley & Sugerman (2005), where NNS procedure builds trees with a different manner from a triangle soup, and takes these triangles as the objects of interest. To convert k-d tree into serialized flat array that can be easily loaded into CUDA device, left-balanced k-d tree was proposed Jensen (2001) Qiu et al. (2009). Another technique for 3D registration using Fast Point Feature Histograms (FPFH) is shown in the work of Rusu Rusu et al. (2009). Rusu also proposed a way of characterizing the local geometry of 3D points, using persistent feature histograms, where the relationships between the neighbors of a point are analyzed and the resulted values are stored in a 16-bin histogram Rusu et al. (2008). The histograms are pose and point cloud density invariant and cope well with noisy datasets. An alternative concept to ICP algorithm which relies on instantaneous kinematics and on the geometry of the squared distance function of a surface is shown in the work of Pottmann Pottmann et al. (2002). The proposed algorithm exhibits faster convergence than ICP, which is supported both by results of a local convergence analysis and by experiments. The ICP algorithm is used in SLAM 6D (Simultaneous Localization and Mapping), where 6 DOF (Degree of freedom) of robot position is computed based on aliment of 3D clouds of points and loop-closing technique Sprickerhof et al. (2009).

3. 3D data registration

Classic Iterative Closest Points algorithm ICP was improved using GPGPU computation to obtain on-line execution. The implementation performs nearest neighbor search using a bucket data structure (sorted using CUDA primitive) and computes the correlation matrix using parallel CUDA all-prefix-sum instruction.

3.1 GPU architecture

NVIDIA GPGPUs are programmable multi core chips built around an array of processors working in parallel. The GPU is composed of an array of streaming multiprocessors (SM), where each of them can launch up to 1024 co-resident concurrent threads. Currently available graphics units are in the range from 1 SM up to 30 SMs for the high end products. Each single SM contains 8 scalar processors (SP) each with 1024 32-bit registers. The total of 64KB of register space is available for each SM. Each SM is also equipped with a 16KB on-chip memory that is characterized by low access latency and high bandwidth. Thread management (creation, scheduling, synchronization) is performed in hardware and the overhead is extremely low. SM works in SIMT scheme (Single Instruction, Multiple Thread), where threads are executed in groups of 32 called *warps*. CUDA programming model defines the *host* and the *device*. *Host* executes CPU sequential procedures while the *device* executes parallel GPU programs - *kernels*. A *kernel* works in a SPMD scheme (Single Program, Multiple Data). CUDA gives an advantage of using massively parallel computation for several applications. Detailed GPU architecture can be found in the original documentation *NVIDIA CUDA C Programming Guide 3.2* (2010). Useful additional programming issues are published in best practices guide CUDA (2010b).

3.2 Improved classic Iterative Closest Point

In this subsection classic Iterative Closest Point algorithm proposed by Besl & McKay (1992) and implementation proposed by Nüchter, Lingemann, Hertzberg & Surmann (2007) improved by GPGPU computing is described. Aligning two-view range images with respect to the reference coordinate system is performed by the ICP (Iterative Closest Points) algorithm. Range images are defined as a model set M and data set D, N_m and N_d denotes the number of the elements in the respective set. The alignment of these two data sets is solved by minimization with respect to **R**,**t** of the following cost function:

$$E(\mathbf{R},\mathbf{t}) = \sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{ij} \left\| \mathbf{m}_i - \left(\mathbf{R}\mathbf{d}_j + \mathbf{t}\right)\right\|^2 \quad (1)$$

w_{ij} is assigned 1 if the i-th point of M correspond to the j-th point in D in the same bucket (or neighbor bucket). Otherwise w_{ij}=0. **R** is a rotation matrix, **t** is a translation matrix. $\mathbf{m}_i$ and $\mathbf{d}_i$ corresponds to the i-th point from model set M and D respectively. The ICP algorithm using CUDA parallel programming is given:

Algorithm 1 ICP - parallel computing approach

allocate the memory
copy data from the host(M_{host}, D_{host}) to the device(M_{device}, D_{device})
for $iter = 0$ to $max\_iterations$ **do**
  select closest points between M_{device} and D_{device}
  calculate (R,t) that minimizes equation 1
  transform D_{device} by (R,t) and put the results into $D_{deviceRt}$
  copy $D_{deviceRt}$ to D_{device}
end for
copy D_{device} to D_{host}
free memory

3.2.1 Calculation of (R,t)

Calculation of the rotation and translation (**R**,**t**) is performed using reduced equation 1:

$$E(\mathbf{R},\mathbf{t}) \propto \frac{1}{N}\sum_{i=1}^{N} \left\| \mathbf{m}_i - (\mathbf{R}\mathbf{d}_i + \mathbf{t})\right\|^2 \quad (2)$$

where

$$N = \sum_{i=1}^{N_m}\sum_{j=1}^{N_d} w_{ij} \quad (3)$$

Rotation **R** is decoupled from computation of translation **t** using the centroids $\mathbf{c}_m$ and $\mathbf{c}_d$ of points:

$$\mathbf{c}_m = \frac{1}{N}\sum_{i=1}^{N} \mathbf{m}_i \quad (4)$$

$$\mathbf{c}_d = \frac{1}{N}\sum_{i=1}^{N}\mathbf{d}_i \tag{5}$$

and modified data sets:

$$\mathbf{M}' = \left\{\mathbf{m}_i{}' = \mathbf{m}_i - \mathbf{c}_m\right\}_{1,\ldots,N} \tag{6}$$

$$\mathbf{D}' = \left\{\mathbf{d}_i{}' = \mathbf{d}_i - \mathbf{c}_d\right\}_{1,\ldots,N} \tag{7}$$

After applying equations 4, 5, 6 and 7 to the mean square error function E(**R**, **t**), the equation 2 takes the following form:

$$E\left(\mathbf{R},\mathbf{t}\right) \propto \frac{1}{N}\sum_{i=1}^{N}\left\|\mathbf{m}_i{}' - \mathbf{R}\mathbf{d}_i{}' - \left(\mathbf{t} - \mathbf{c}_m + \mathbf{R}\mathbf{c}_d\right)\right\|^2 \tag{8}$$

Assuming that:

$$\mathbf{t} - \mathbf{c}_m + \mathbf{R}\mathbf{c}_d = \widetilde{t} \tag{9}$$

equation 8 takes following form:

$$\begin{aligned} E\left(\mathbf{R},\mathbf{t}\right) &\propto \tfrac{1}{N}\textstyle\sum_{i=1}^{N}\left\|\mathbf{m}_i{}' - \mathbf{R}\mathbf{d}_i{}'\right\|^2 - \\ &\tfrac{2}{N}\widetilde{t}\cdot\textstyle\sum_{i=1}^{N}\left(\mathbf{m}_i{}' - \mathbf{R}\mathbf{d}_i{}'\right) + \tfrac{1}{N}\textstyle\sum_{i=1}^{N}\left\|\widetilde{t}\right\|^2 \end{aligned} \tag{10}$$

To minimize 10 the algorithm has to minimize the following term:

$$\sum_{i=1}^{N}\left\|\mathbf{m}_i{}' - \mathbf{R}\mathbf{d}_i{}'\right\|^2 \tag{11}$$

with the assumption:

$$\widetilde{t} = 0 \tag{12}$$

The optimal rotation is calculated by **R**= $\mathbf{V}\mathbf{U}^T$, where matrices **V** and **U** are derived from the Singular Value Decomposition (SVD) described in section 3.2.8 of a correlation matrix **C** = $\mathbf{U}\mathbf{S}\mathbf{V}^T$ given by:

$$\mathbf{C} = \sum_{i=1}^{N}\mathbf{m}_i{}'^T\mathbf{d}_i{}' = \begin{bmatrix} c_{xx} & c_{xy} & c_{xz} \\ c_{yx} & c_{yy} & c_{yz} \\ c_{zx} & c_{zy} & c_{zz} \end{bmatrix} \tag{13}$$

where:

$$c_{xx} = \sum_{i=1}^{N} m_{ix}{}' d_{ix}{}', c_{xy} = \sum_{i=1}^{N} m_{ix}{}' d_{iy}{}', \ldots, c_{zz} = \sum_{i=1}^{N} m_{iz}{}' d_{iz}{}' \tag{14}$$

Correlation matrix elements are computed using optimized parallel reduction described in section 3.2.7. The optimal translation **t** is derived from equation 12 and 9, therefore

$$\mathbf{t} = \mathbf{c}_m - \mathbf{R}\mathbf{c}_d \tag{15}$$

3.2.2 Memory allocation on device and copy from host

Figure 1 and algorithm 2 shows main data used in presented approach. The $table\_of\_found\_buckets$, $table\_of\_sorted\_buckets$, $table\_of\_sorted\_points$ consist of 512×512 integer elements, $table\_of\_amount\_of\_points\_in\_bucket$ and $table\_of\_bucket\_indexes$ consist of $256 \times 256 \times 256$ integer elements. M (reference data) and D (data to be align) data sets are stored in six 512×512 tables consisting float values stored in one dimensional array. For sorting the table of buckets routine described in section 3.2.4 and used in algorithm 2 the CUDA Radix Sort class available in CUDA (2010a) briefly described in Satish et al. (2008) and Satish et al. (2009) is used. The method initialize() called by the constructor of Radix Sort Class allocates temporary storage for the sort and the prefix sum that it uses. Temporary storage is (2*NUMBER_OF_POINTS + 3*8*M/CTA_SIZE) unsigned ints, with a default CTA_SIZE of 256 threads and $NUMBER\_OF\_POINTS = 512 \times 512$. Amount of data is large, therefore the procedure of memory allocation is done initially.

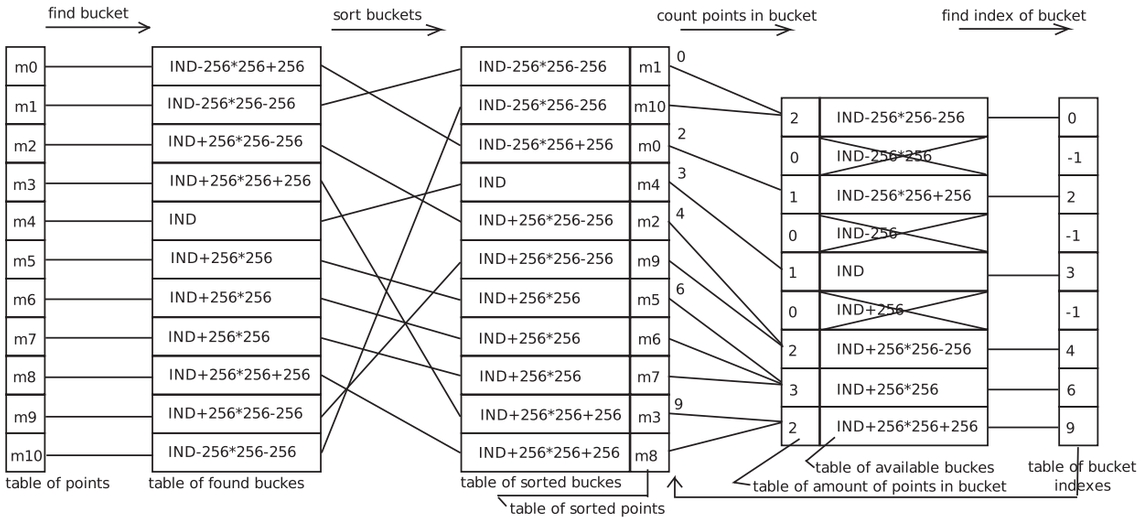


Fig. 1. Initial steps: selection of closest points procedure - example related to figure 4.

3.2.3 Selection of closest points

The distance between two points in Euclidean distance metric for point $p_1 = \{x_1, y_1, z_1\}$ and $p_2 = \{x_2, y_2, z_2\}$ is:

$$distance(p_1, p_2) = \left[(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2\right]^{\frac{1}{2}} \tag{16}$$

To find pairs of closest points between model set M and data set D, the decomposition of XYZ space, where x,y,z $\in < -1, 1 >$, into 2^8x2^8x2^8 buckets is proposed. The idea of the decomposition will be discussed for 2^2x2^2x2^2 case. Figure 2 shows decision tree that decomposes XYZ space into 64 buckets. Each node of the decision tree includes boundary decision, therefore points are categorized into left or right branch. Nodes that do not have branches assign buckets. Each bucket has unique index and is related to cubic subspace with length,width,height = $2/2^2$,$2/2^2$,$2/2^2$. Each bucket that does not belong to border has 26 neighbors. The 27 neighboring cubic subspaces are shown in figure 3 where also the way of

indexing is given. Figure 4 demonstrates the idea of nearest neighbor (NN) search technique on 2 dimensional example. Assuming that we are looking for nearest neighbor that satisfies condition R $< 2/2^8$ and $circle_{R=2/2^8} \subset bucket_{3R}$ NN will be found in the same bucket or in neighboring bucket (in this example NN of point d is m5). Algorithm 2 describes the procedure of selection of closest points. For better explanation figure 1 shows initial steps of this algorithm where set M of 10 points from figure 4 is used for NN search. The details of the algorithm will be discussed in next subsections.

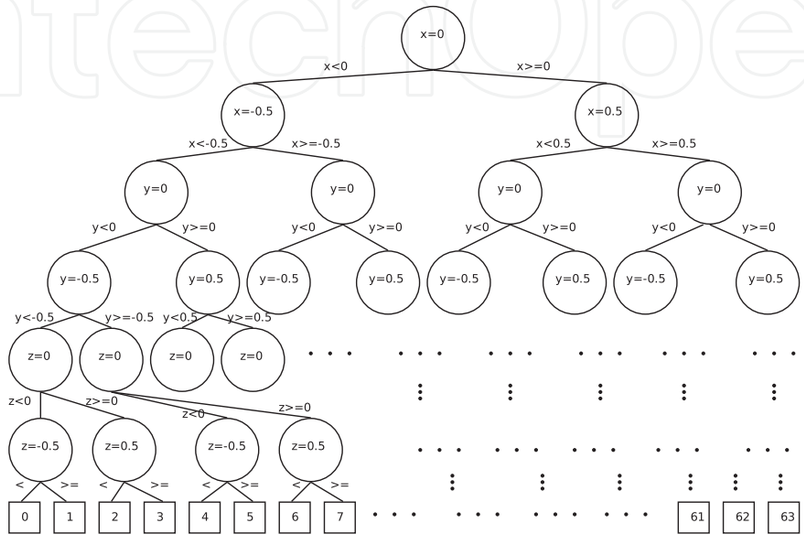


Fig. 2. Tree structure used for XYZ space decomposition into 64 buckets. From root to the leaf/bucket chosen left or right brunch depends on the current separation line.

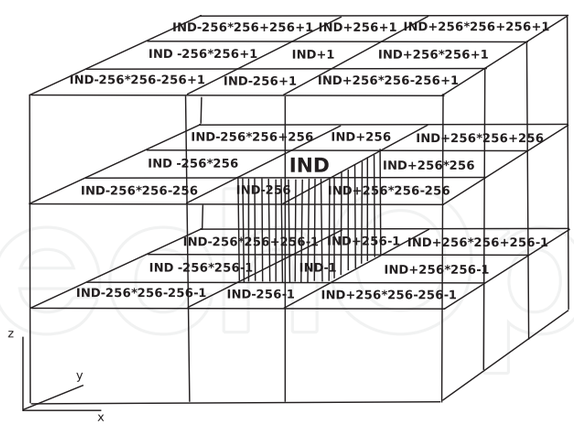


Fig. 3. Cubic subspaces - neighboring buckets, the way of indexing is explained in section 3.2.6.

3.2.4 Sort buckets

Radix sort class CUDA (2010a) is used to sort unsigned integer key-value pairs. Keys correspond to the elements of table of buckets and value corresponds to elements from table of points. Procedure outputs sorted table of buckets. Figure 1 shows an example of sorting

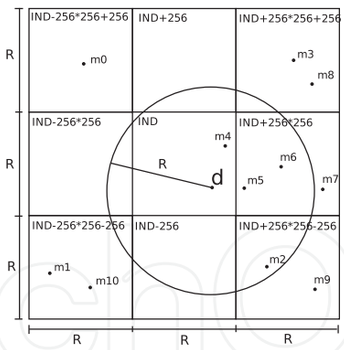


Fig. 4. 2 dimensional example of NN search in neighboring buckets.

Algorithm 2 Selection of closest points

for all points m_{xyz} in parallel **do**
  find $bucket_m$
  update $table\_of\_found\_buckets$
end for
in parallel sort $table\_of\_found\_buckets$ {radix sort}
in parallel count points in each bucket
for all points d_{xyz} in parallel **do**
  find $bucket_d$
  for all neighbors of $bucket_d$ **do**
    find NN for d_{xyz} {nearest neighbor is one from m_{xyz}}
  end for
end for

result. Radix sort is a well known sorting algorithm, very efficient on sequential machines for sorting small keys. It assumes that the keys are d-digit numbers and sorts on one digit of the keys at a time, starting from least and finishing on most significant. The complexity of sorting n keys will be O(n). The details of GPU based radix sort implementation can be found in Satish et al. (2008) Satish et al. (2009). The implementation of GPU based radix sort is robust, therefore it can be used for on-line computation.

3.2.5 Count points in bucket and find index of bucket

In the procedure of counting points that belong to the same bucket the counting is based on $table\_of\_sorted\_buckets$ (see figure 1). It is important to notice, that also the index of the found bucket is computed. This index, along with the information concerning an amount of points in the bucket, will be used for searching nearest neighbor in algorithm 2.

3.2.6 Find bucket

Figure 2 shows tree structure used for indexing of 2^2x2^2x2^2 buckets. The concept of finding bucket index for point m_{xyz} is shown on the scheme 5, where x corresponds to border for current level in the tree and 0, 1, 2, 3, ... 14, ... correspond to actual bucket index during its computation. The bucket indexing procedure is executed in parallel, where each CUDA kernel computes bucket index for different $point_{xyz}$.

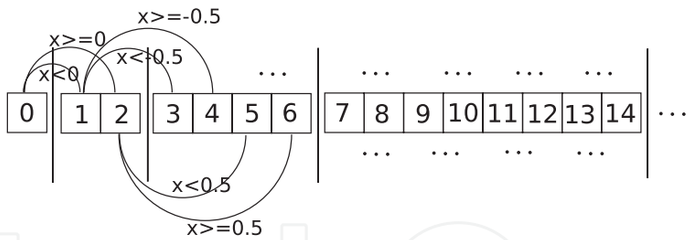


Fig. 5. The scheme of bucket indexing procedure.

3.2.7 Correlation matrix elements computation using optimized parallel reduction

For correlation matrix (equation 13) parallel prefix sum Harris et al. (2007) available in CUDA (2010a) is used. The all-prefix-sums operations take a binary associate operator $\oplus$ with identity I, and an array of n elements

$$[a_0, a_1, ..., a_{n-1}] \tag{17}$$

and returns the array

$$[I, a_0, (a_0 \oplus a_1), ..., (a_0 \oplus a_1 \oplus ... \oplus a_{n-2})] \tag{18}$$

All-prefix-sums operations on array of data is commonly known as scan. The parallel implementation uses multiple thread blocks for processing an array of 512 $\times$ 512 data points stored in one dimensional array. The strategy is to keep all multiprocessors on the GPU busy to increase the performance. An assumption is that each thread block reduces a portion of the array. To avoid problem of global synchronization the computation is decomposed into multi kernel invocations. Optimized kernel available in CUDA (2010a) is used in parallel computation.

3.2.8 Singular Value Decomposition (SVD)

The equation for singular value decomposition of A 3 $\times$ 3 matrix is the following:

$$A = U\Sigma V^T \tag{19}$$

where U is an 3 $\times$ 3 matrix, Σ is an 3 $\times$ 3 diagonal matrix, and V^T is also an 3 $\times$ 3 matrix. The columns of U are called the left singular vectors, { u_k }, and form an orthonormal basis. The rows of V^T contain the elements of the right singular vectors, { v_k }. The elements of Σ are only nonzero on the diagonal, and are called the singular values. Thus, Σ = diag($\sigma_1, ..., \sigma_n$). Furthermore, $\sigma_k > 0$ for $1 \leq k \leq r$, and $\sigma_i = 0$ for (r+1) $\leq k \leq n$. The ordering of the singular vectors is determined by high-to-low sorting of singular values, with the highest singular value in the upper left index of the Σ matrix. In this particular application we need to compute the SVD of a 3x3 matrix. For such a small matrix, generalized SVD algorithms from libraries like LAPACK (Linear Algebra PACKage) LAPACK (2011) are not beneficial especially when we have to implement it on GPGPU. Our implementation computes the singular values by solving for the roots of a cubic polynomial and then eigenvectors of A^TA for V, then it uses A and V to compute U. The algorithm is executed in 5 steps.
1. Compute A^T and A^TA.
2. Determine the eigenvalues of A^TA (by solving for the roots of a cubic polynomial) and sort these in descending order. Compute square roots to obtain singular values of A.

3. Construct diagonal matrix Σ by placing singular values in descending order along its diagonal. Compute Σ^{-1}.
4. Use the ordered eigenvalues from step 2 and compute the eigenvectors of $A^T A$. Place these eigenvectors along the columns of V and compute V^T.
5. Compute $U = AV\Sigma^{-1}$

4. Path planning

Motion planning is very important task for mobile robot working in unknown environment. Assuming that we have grid map describing robot environment, a trajectory of the robot can be computed using the modification of diffusion method described in Siemiatkowska (2008) improved by GPU computation. GPGPU implementation is using two 2 dimensional grids of 512x512 cells. One grid is used for initiation, where each cell can be free, occupied by the obstacle, occupied by a robot or occupied by a goal. Second grid is used for diffusion computation. The idea of usage GPU is to perform computation for each cell in parallel till diffusion reach robot position. To obtain robot trajectory we start from robot position and iteratively by finding local maximum in neighboring cells we are approaching the goal.

5. Cloud of points processing

Main idea is to decompose 3 dimensional space into grid of buckets. For each bucket local approximation plane is computed (if more than 3 points belong to bucket) and for each point in bucket normal vector is assigned. The advantage of proposed method is satisfactory result obtained with noisy data set. The procedure of normal vectors computation for registered range images is given in algorithm 3, it uses CUDA for robust nearest neighbors search briefly described in previous sections. The parameter *maxnumberofplanes* in algorithm 3 is assigned experimentally as 10. This value guarantee robust procedure execution with satisfying heuristic of random planes generation.

Algorithm 3 Compute normal vectors (r,g,b)

```
for all range points (x,y,z) in parallel do
  bucket = findbucket(x,y,z)
  for allneighboringbuckets do
    add points from bucket to listofpoints
  end for
  for i = 0 to maxnumberofplanes do
    compute plane based on 3 random points
    sum_i = 0
    for all points in listofpoints do
      sum_i += distance(point,plane)
    end for
  end for
  normalvector = plane for min(sum_i)
end for
```

6. Experiments

All experiments were performed using NVIDIA GF 580 GPGPU. Following subsections demonstrate results in area of accelerated ICP, path planning and normal vector computation.

6.1 GPU accelerated ICP

Experiments where performed using different type of sensors: Time of Flight camera (see figure 6), Kinect sensor (see figure 8) and 3DLSN unit (see figure 10). Data registration result with low cost Time of Flight camera IFM O3D201 is not satisfactory because of the low resolution of sensor (figure 7). New RGB-D Kinect camera (figure 8) offers high resolution with acceptable level of noise, therefore the registration result is acceptable (figure 9). Unfortunately Kinect sensor is not designed for robotics application especially working in outdoor environments, therefore usage is limited. The most optimistic result is obtained

Fig. 6. Time of Flight camera type IFM O3D201.

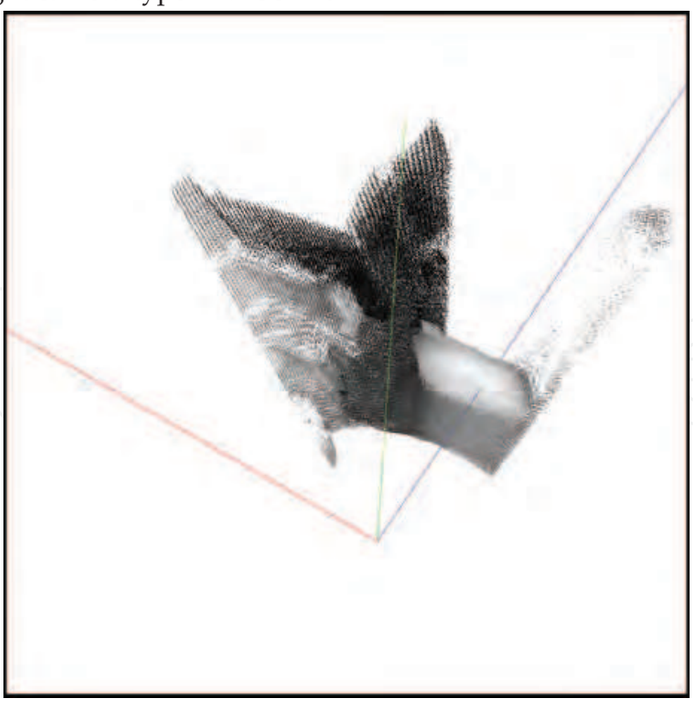


Fig. 7. Data registration using Time of Flight camera type IFM O3D201. The result is not satisfactory because of low resolution of sensor.

Fig. 8. Robot PIONEER 3AT equipped with Kinect sensor. Photo taken during Land Robot Trial CELROB 2011.

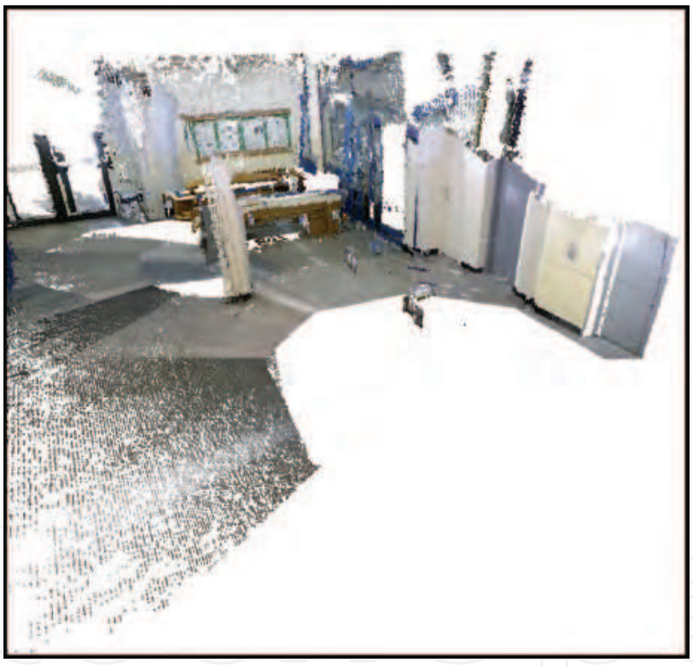


Fig. 9. Data registration using Kinect sensor. Result is satisfactory because of high resolution of Kinect sensor with acceptable level of noise.

for commercially available 3DLSN unit (figure 10) composed of laser measurement system LMS SICK 200 offering accurate data acquisition working in INDOOR and OUTDOOR environments. The disadvantage is the limitation of data acquisition in stop-scan fashion. Data where acquired in INDOOR and OUTOOR environments shown in figures 11 and 14. Results of ICP algorithm for 30 and 100 iterations are shown in figures 12 and 13. Average time of 30 iterations of GPGPU based ICP is less than 300ms and 100 iterations is less than 1 second. Empirical evaluation shows that 30 iteration enough for accurate data

registration, therefore it is assumed that ICP takes 300ms in average for alignment of two data sets containing 512 x 512 data points.

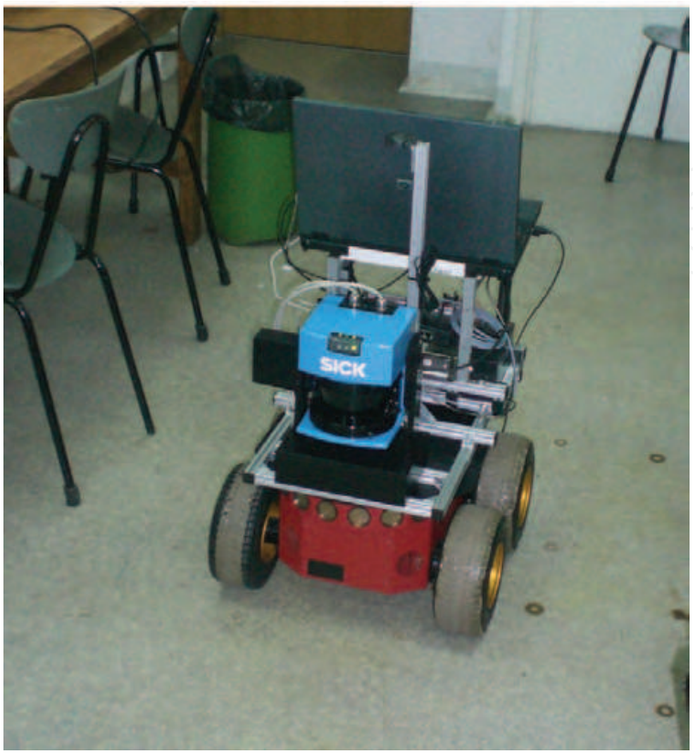


Fig. 10. Robot PIONEER 3AT equipped with 3DLSN sensor.

Fig. 11. Data acquisition using robot equipped with 3DLSN unit in INDOOR and OUTDOOR environments (Faculty of Mechatronics, Warsaw University of Technology).

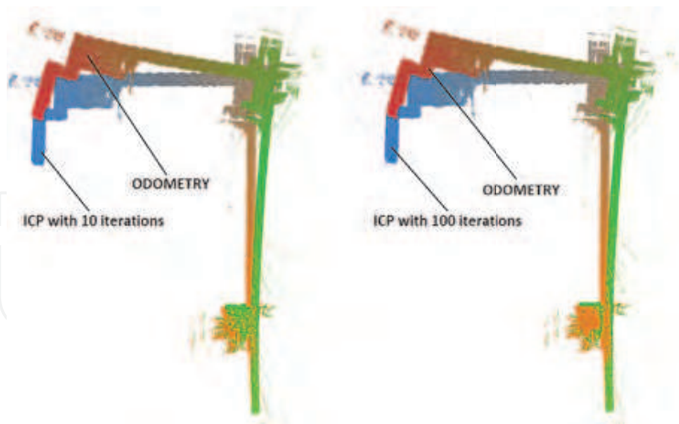


Fig. 12. Result of data registration in INDOOR environment from figure 11. It is shown a small difference of the result of ICP with 30 and 100 iterations.

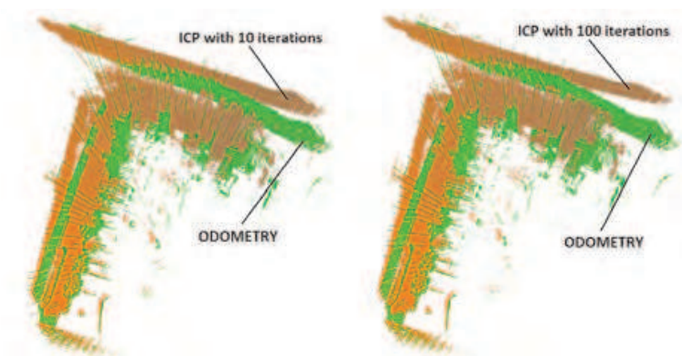


Fig. 13. Result of data registration in INDOOR environment from figure 11. It is shown a small difference of the result of ICP with 30 and 100 iterations.

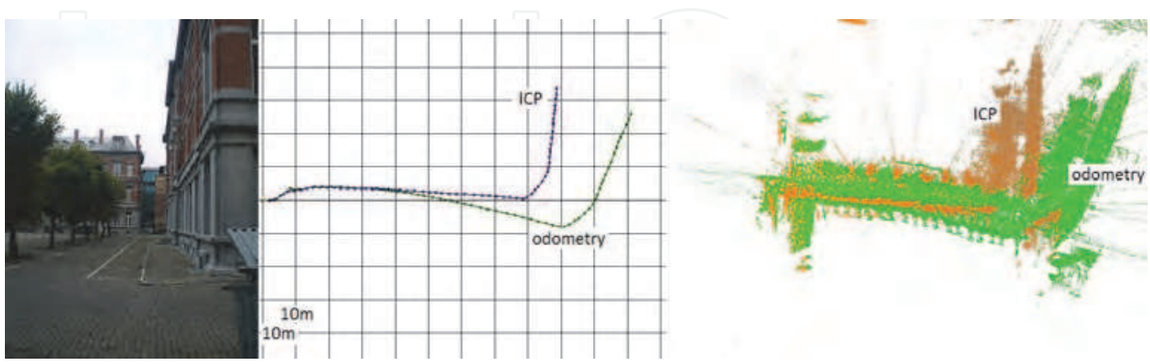


Fig. 14. Data registration result (Royal Military Academy Campus, Brussels, Belgium).

6.2 Path planing simulation results

Path planning was tested using simulated environments with different amount of random obstacles (figure 15, 16, 17, 18). Robot environment is represented by two dimensional grid of cells (512 x 512). Goal is located on the left and robot position is located on the right, path is visualized as black and white line. Diffusion process is visualized in gray scale, where white pixels correspond to max values and black pixels correspond to min values. The average time of CUDA kernel execution of elementary diffusion process takes 0.06 ms, and total diffusion

Fig. 15. Path planning result (goal- circle on the left, robot position - circle on the right). Environment with no obstacles.

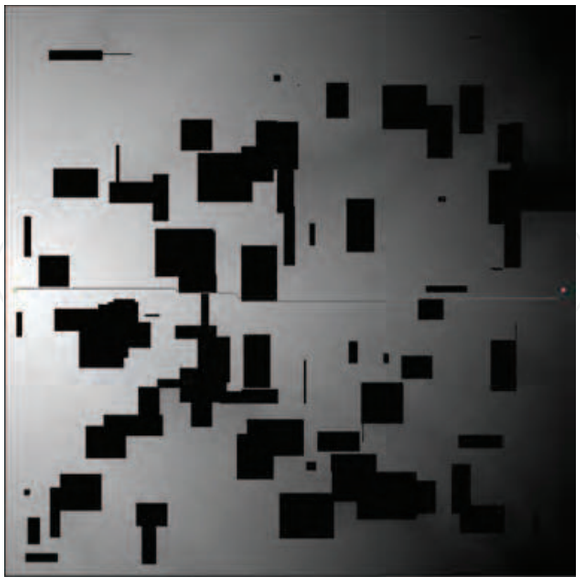


Fig. 16. Path planning result (goal- circle on the left, robot position - circle on the right). Environment with 100 random obstacles.

process for 512 iterations takes in average 30ms. The result is satisfactory because it is showing on-line capability of proposed implementation.

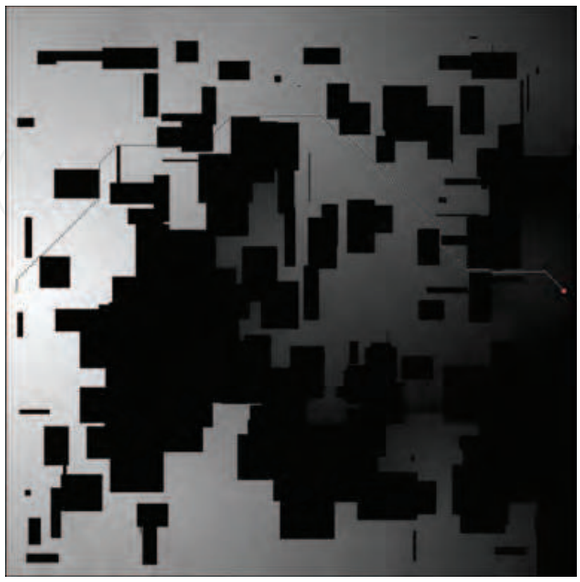


Fig. 17. Path planning result (goal- circle on the left, robot position - circle on the right). Environment with 200 random obstacles.

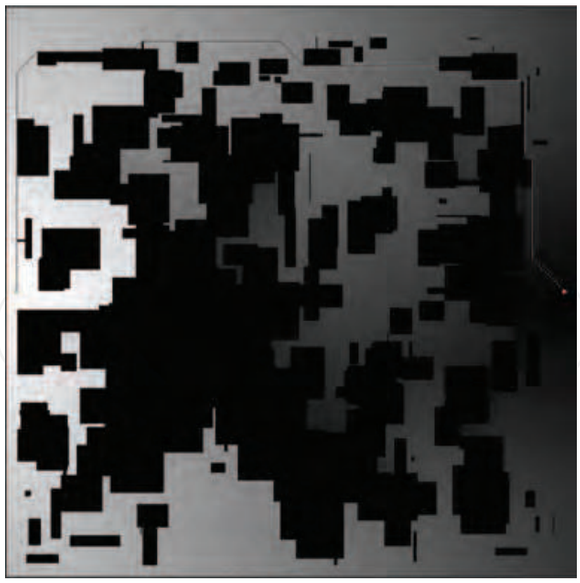


Fig. 18. Path planning result (goal- circle on the left, robot position - circle on the right). Environment with 300 random obstacles.

6.3 Normal vector computation

GPGPU accelerated normal vector computation was tested for data acquired using Kinect sensor (figure 19) and 3DLSN unit (figure 20). The average time of normal vector computation is less than 100ms for data set of 512 x 512 data points. The implementation is robust for noisy data delivered by Kinect sensor.

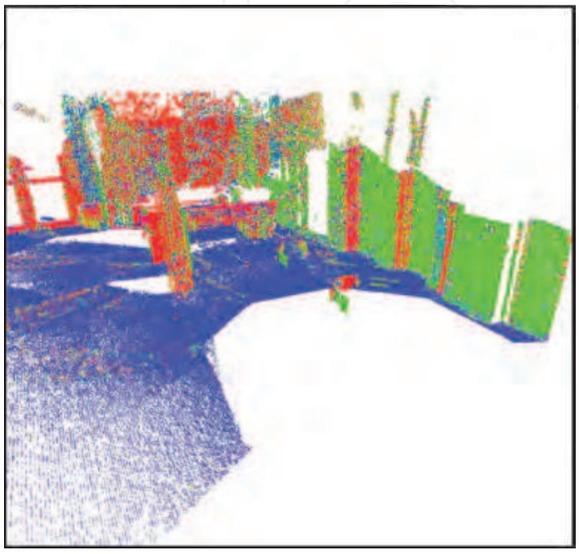


Fig. 19. Normal vector computation for data acquired by Kinect sensor (see also figure 9). Vectors (x,y,z) are represented by colors (r,g,b).

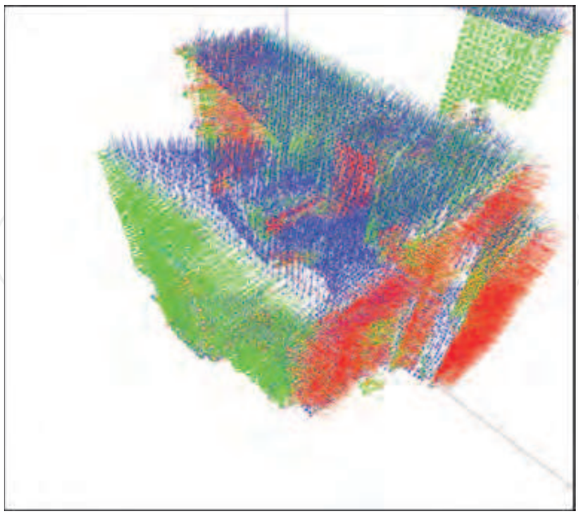


Fig. 20. Normal vector computation for data acquired by 3DLSN unit. Vectors (x,y,z) are represented by colors (r,g,b).

7. Conclusion and future work

In this chapter three areas of research where shown such as 3D data registration, robot navigation and 3D cloud of points processing. All approaches are improved using GPGPU parallel computation. The on − line data registration problem was discussed. New approach based on robust KNN nearest neighborhood search applied for improvement of ICP algorithm where shown. The implementation is using Radix Sort for bucket sorting. Data registration implementation was tested for data sets delivered by different sensors in different environments. The average ICP time computation for 30 iteration is less than 300ms.
The path planning parallel approach based on modified diffusion method was shown. It was tested using simulated environment compound from different amount of random obstacles. The result is satisfactory because 30 ms of computation guarantee on-line execution in practical application.
On − line 3D cloud of points' segmentation based on normal vector computation was presented. The result is satisfactory even for noisy data obtained by Kinect sensors. 100ms average time is optimistic to use this implementation in practical application.
The set of proposed algorithms was tested on GPGPU NVIDIA CUDA GF 580. The improvement of SoA algorithms based on CUDA implementation shows the possibility to use in real RISE applications because of decreased time of execution. Future work will be related to development of 6DSLAM using GPU-ICP as odometry correction and normal vector computation for obtaining semantic images for Loop Closing procedure.

8. References

Arya, S. & Mount, D. M. (1993). Algorithms for fast vector quantization, *Proc. of DCC '93: Data Compression Conference*, IEEE Press, pp. 381–390.

Besl, P. J. & McKay, N. D. (1992). A method for registration of 3-d shapes, *IEEE Trans. Pattern Anal. Mach. Intell.* 14: 239–256.
URL: *http://portal.acm.org/citation.cfm?id=132013.132022*

CUDA (2010a). http://www.nvidia.com/cuda.

CUDA (2010b). CUDA C Best Practices Guide 3.2, http://www.nvidia.com/cuda.

Fitzgibbon, A. W. (2001). Robust registration of 2d and 3d point sets, *In British Machine Vision Conference*, pp. 411–420.

Foley, T. & Sugerman, J. (2005). Kd-tree acceleration structures for a gpu raytracer, *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, HWWS '05, ACM, New York, NY, USA, pp. 15–22.

Garcia, V., Debreuve, E. & Barlaud, M. (2008). Fast k nearest neighbor search using gpu, *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1–6.

Harris, M., Sengupta, S. & Owens, J. D. (2007). *GPU Gems 3, Parallel Prefix Sum (Scan) with CUDA*, Addison-Wesley, chapter 39, pp. 851–876.

Huber, D. & Hebert, M. (2003). Fully automatic registration of multiple 3d data sets, *Image and Vision Computing* 21(1): 637–650.

Jensen, H. W. (2001). *Realistic image synthesis using photon mapping*, A. K. Peters, Ltd., Natick, MA, USA.

Kohlhepp, P., Pozzo, P., Walther, M. & Dillmann, R. (2004). Sequential 3d-slam for mobile action planning, *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai, Japan*, pp. 722–729.

LAPACK (2011). http://www.netlib.org/lapack.

Magnusson, M. & Duckett, T. (2005). A comparison of 3d registration algorithms for autonomous underground mining vehicles, *In Proc. ECMR*, pp. 86–91.

Magnusson, M., Duckett, T. & Lilienthal, A. J. (2007). 3d scan registration for autonomous mining vehicles, *Journal of Field Robotics* 24(10): 803–827.

Montemerlo, M. & Thrun, S. (2004). A multi-resolution pyramid for outdoor robot terrain perception, *AAAI'04: Proceedings of the 19th national conference on Artifical intelligence*, AAAI Press, pp. 464–469.

Nuchter, A., Lingemann, K. & Hertzberg, J. (2007). Cached k-d tree search for icp algorithms, *Proceedings of the Sixth International Conference on 3-D Digital Imaging and Modeling*, IEEE Computer Society, Washington, DC, USA, pp. 419–426.

Nüchter, A., Lingemann, K., Hertzberg, J. & Surmann, H. (2007). 6d slam for 3d mapping outdoor environments, *Journal of Field Robotics (JFR), Special Issue on Quantitative Performance Evaluation of Robotic and Intelligent Systems* 24: 699–722.

Nüchter, A., Surmann, H. & Hertzberg, J. (2003). Automatic model refinement for 3D reconstruction with mobile robots, *Fourth International Conference on 3-D Digital Imaging and Modeling 3DIM 03*, p. 394.

NVIDIA CUDA C Programming Guide 3.2 (2010). http://www.nvidia.com/cuda.

Ortega, A., Haddad, I. & Andrade-Cetto, J. (2009). Graph-based segmentation of range data with applications to 3d urban mapping, *4th European Conference on Mobile Robots ECMR 09, September 23-25, 2009, Mlini Dubrovnik, Croatia*, pp. 193–198.

Park, S.-Y., Choi, S.-I., Kim, J. & Chae, J. (2010). Real-time 3d registration using gpu, *Machine Vision and Applications* pp. 1–14. 10.1007/s00138-010-0282-z.

Park, S.-Y. & Subbarao, M. (2003). An accurate and fast point-to-plane registration technique, *Pattern Recogn. Lett.* 24: 2967–2976.

Pottmann, H., Leopoldseder, S. & Hofer, M. (2002). Registration without icp, *Computer Vision and Image Understanding* 95: 54–71.

Purcell, T. J., Donner, C., Cammarano, M., Jensen, H. W. & Hanrahan, P. (2003). Photon mapping on programmable graphics hardware, *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, Eurographics Association, pp. 41–50.

Qiu, D., May, S. & Nuchter, A. (2009). Gpu-accelerated nearest neighbor search for 3d registration, *Proceedings of the 7th International Conference on Computer Vision Systems: Computer Vision Systems*, ICVS '09, Springer-Verlag, Berlin, Heidelberg, pp. 194–203.

Rusinkiewicz, S. & Levoy, M. (2001). Efficient variants of the ICP algorithm, *Third International Conference on 3D Digital Imaging and Modeling (3DIM)*.

Rusu, R. B., Blodow, N. & Beetz, M. (2009). Fast point feature histograms (fpfh) for 3d registration, *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, ICRA'09, IEEE Press, Piscataway, NJ, USA, pp. 1848–1853.

Rusu, R. B., Marton, Z. C., Blodow, N. & Beetz, M. (2008). Persistent Point Feature Histograms for 3D Point Clouds, *Proceedings of the 10th International Conference on Intelligent Autonomous Systems (IAS-10), Baden-Baden, Germany*.

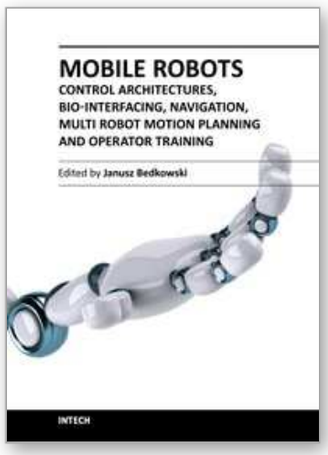
Satish, N., Harris, M. & Garland, M. (2008). Designing efficient sorting algorithms for manycore gpus, *NVIDIA Technical Report NVR-2008-001*, NVIDIA Corporation.

Satish, N., Harris, M. & Garland, M. (2009). Designing efficient sorting algorithms for manycore gpus, *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, IEEE Computer Society, Washington, DC, USA, pp. 1–10.

Siemiatkowska, B. (2008). Coordinating the motion of mobile robots using cellular neural network, *Journal of Automation, Mobile Robotics and Intelligent Systems* 2(2): 65–69.

Sprickerhof, J., Nüchter, A., Lingemann, K. & Hertzberg, J. (2009). An explicit loop closing technique for 6d slam, *4th European Conference on Mobile Robots ECMR 09, September 23-25, 2009, Mlini/Dubrovnik, Croatia*, pp. 229–234.

Thrun, S., Burgard, W. & Fo, D. (2000). A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping, *ICRA*, pp. 321–328.



Mobile Robots - Control Architectures, Bio-Interfacing, Navigation, Multi Robot Motion Planning and Operator Training

Edited by Dr. Janusz Będkowski

ISBN 978-953-307-842-7
Hard cover, 390 pages
Publisher InTech
Published online 02, December, 2011
Published in print edition December, 2011

The objective of this book is to cover advances of mobile robotics and related technologies applied for multi robot systems' design and development. Design of control system is a complex issue, requiring the application of information technologies to link the robots into a single network. Human robot interface becomes a demanding task, especially when we try to use sophisticated methods for brain signal processing. Generated electrophysiological signals can be used to command different devices, such as cars, wheelchair or even video games. A number of developments in navigation and path planning, including parallel programming, can be observed. Cooperative path planning, formation control of multi robotic agents, communication and distance measurement between agents are shown. Training of the mobile robot operators is very difficult task also because of several factors related to different task execution. The presented improvement is related to environment model generation based on autonomous mobile robot observations.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Janusz Będkowski (2011). Parallel Computing in Mobile Robotics for RISE, Mobile Robots - Control Architectures, Bio-Interfacing, Navigation, Multi Robot Motion Planning and Operator Training, Dr. Janusz Będkowski (Ed.), ISBN: 978-953-307-842-7, InTech, Available from: http://www.intechopen.com/books/mobile-robots-control-architectures-bio-interfacing-navigation-multi-robot-motion-planning-and-operator-training/parallel-computing-in-mobile-robotics-for-rise

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen