International Journal on Electrical Engineering and Informatics - Volume 4, Number 1, March 2012

GPGPU computation in mobile robot applications

Janusz Będkowski[1] , Andrzej Masłowski[2.3]

[1.2]Institute of Automation and Robotics, Warsaw University of Technology
ul. Św. A. Boboli 8, 02-525, Warsaw, Poland [2.3]Institute of Mathematical Machines
ul. Ludwika Krzywickiego 34, Warsaw, Poland
[1]januszbedkowski@gmail.com
[2]a.maslowski@mchtr.pw.edu.pl
[3]a.maslowski@imm.org.pl

Abstract: The paper concerns the results related with GPGPU computing applied for mobile robotics applications. The scalable implementation of the point to point and point to plane 3D data registration methods with an improvement based on regular grid decomposition is shown. 3D data is delivered by mobile robot equipped with 3D laser measurement system for INDOOR environments. Presented empirical analysis of the implementation shows the On-Line computation capability using modern graphic processor unit NVIDIA GF 580. In the paper the discussion concerning the comparison between these two methods is given. It will be shown why the point to plain ICP implementation can achieve better performance than the point to point approach. We show parallel vector computation that is used for semantic objects identifications and for loop closing detection.

Keywords: Data registration, parallel computing, point to point, point to plane, mobile robot.

1. Introduction

Several researches of 3D mapping are based on the so called simulation of 3D laser range finder to obtain 3D cloud of points [1]. In most cases 3D laser simulator is built on the basis of a rotated 2D range finder. The rotation axis can be horizontal [2], vertical [3] or the rotational axis in the middle of the scanner's field of view [4]. Another approach of obtaining 3D cloud of points using two orthogonal lasers is shown in [5]. The applications are related with urban mapping [6].

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 [7] [8] [9]. Park [10] 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 [11]. Processing time for this approach is about 60ms for aligning two 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 [12] [13]. GPU accelerated nearest neighbor search for 3D registration is proposed in [14], where the advantage of Arya's priority search algorithm described in [15] 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 [16]. 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 [17]. GPU-based NNS with advanced search structures is also used in the context of ray tracing [18], 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

Received: December 14th, 2011. Accepted: January 9th, 2012

CUDA device, left-balanced k-d tree was proposed [19] [20]. Another technique for 3D registration using Fast Point Feature Histograms (FPFH) is shown in the work of Rusu [21]. 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 [22]. 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 [23]. 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 [24].

The Iterative Closest Point algorithm with its variations point to point and point to plane is a well known method since it appeared in [25]. It is already proven that the ICP algorithm needs a good prediction to achieve an accurate matching. The fastest implementation that can be found in literature needs 60 ms to align two point clouds, each of 320 x 240 data points [10], but the authors unfortunately did not report about scalability of proposed method. In this paper, we will discuss new implementations of ICP (point to point, point to plane) that are designed especially for high performance with an assumption of scalability for future GPGPU devices. At the current stage of the implementation, we can process two data sets of $2^{10}*2^{10}$ data points in parallel. The main goal was to decrease the bottlenecks, therefore they are limited to copying data from/to host (CPU) to/from device (GPU). All computation related to neighbor search and to calculating the correlation matrix is performed by the GPU. We will show why the point to plain ICP implementation can achieve better performance than the classic point to point approach. In the same time, we demonstrate the weaknesses of this method in contrast to the advantages reported in many articles.

The paper shows an improved implementation and empirical validation of GPGPU ICP point to point and point to plane algorithms and the practical applications for normal vector parallel computation. This algorithms offer online computation. The main difference compared to State of The Art approaches is NNS procedure, where 3D space was divided into regular grid of buckets, therefore there is no need to build complex data structures such as k-d tree, and the time of ICP is decreased. We are optimistic that proposed methods will be used in future real-task applications using for example FPGA or embedded computers with GPUs.

2. Classic point to point iterative closest point algorithm

First classic ICP point to point algorithm will be described. Iterative Closest Points algorithm is aligning two-view range images with respect to the reference coordinate system. 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:

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

w_{ij} is assigned 1 if the i-th point of M correspond to the j-th point in D. Otherwise w_{ij}=0. **R** is a rotation matrix, **t** is a translation matrix. m_i and d_i corresponds to the i-th point from model set M and D respectively. Calculation of the rotation and translation (**R,t**) is performed using reduced equation 1:

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

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

Rotation **R** is decoupled from computation of translation **t** using the centroids c_m and c_d of points:

$$\mathbf{c}_m = \frac{1}{N}\sum_{i=1}^{N}\mathbf{m}_i , \qquad \mathbf{c}_d = \frac{1}{N}\sum_{i=1}^{N}\mathbf{d}_i , \tag{4}$$

and modified data sets:

$$\mathbf{M}' = \{\mathbf{m}_i' = \mathbf{m}_i - \mathbf{c}_m\}_{1,\dots,N} , \tag{5}$$

$$\mathbf{D}' = \{\mathbf{d}_i' = \mathbf{d}_i - \mathbf{c}_d\}_{1,\dots,N} . \tag{6}$$

After applying equations 3-6 to the mean square error function E(R, t), the equation 1 takes the following form:

$$E(\mathbf{R},\mathbf{t}) \propto \frac{1}{N}\sum_{i=1}^{N}\left\|\mathbf{m}_i' - \mathbf{R}\mathbf{d}_i' - (\mathbf{t} - \mathbf{c}_m + \mathbf{R}\mathbf{c}_d)\right\|^2 \tag{7}$$

Assuming that:

$$\mathbf{t} - \mathbf{c}_m + \mathbf{R}\mathbf{c}_d = \tilde{\mathbf{t}} , \tag{8}$$

Equation 1 takes following form:

$$E(\mathbf{R},\mathbf{t}) \propto \frac{1}{N}\sum_{i=1}^{N}\left\|\mathbf{m}_i' - \mathbf{R}\mathbf{d}_i'\right\|^2 - \frac{2}{N}\tilde{\mathbf{t}}\sum_{i=1}^{N}(\mathbf{m}_i' - \mathbf{R}\mathbf{d}_i') + \frac{1}{N}\sum_{i=1}^{N}\left\|\tilde{\mathbf{t}}\right\|^2 , \tag{9}$$

To minimize 9 the algorithm has to minimize only term:

$$\sum_{i=1}^{N}\left\|\mathbf{m}_i' - \mathbf{R}\mathbf{d}_i'\right\|^2 , \tag{10}$$

With an assumption:

$$\tilde{\mathbf{t}} = 0 , \tag{11}$$

The optimal rotation is calculated by $\mathbf{R} = \mathbf{V}\mathbf{U}^T$, where matrices $\mathbf{V}$ and $\mathbf{U}$ are derived from the singular value decomposition of a correlation matrix $\mathbf{C} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ given by:

$$\mathbf{C} = \sum_{i=1}^{N}\mathbf{m}_i'^T\,\mathbf{d}_i' = \begin{bmatrix} c_{xx} & c_{xy} & c_{xz} \\ c_{yx} & c_{yy} & c_{yz} \\ c_{zx} & c_{zy} & c_{zz} \end{bmatrix} , \tag{12}$$

where:

$$c_{xx} = \sum_{i=1}^{N} m_{ix}' d_{ix}' , c_{xy} = \sum_{i=1}^{N} m_{ix}' d_{iy}' , \dots , c_{zz} = \sum_{i=1}^{N} m_{iz}' d_{iz}' , \tag{13}$$

The optimal translation **t** is derived from equation 11 and 8, therefore

$$\mathbf{t} = \mathbf{c}_m - \mathbf{R}\mathbf{c}_d , \tag{17}$$

Listing 1 shows a classic ICP algorithm.

Algorithm 1 Classic ICP

```
INPUT: Two point clouds A = $\{a_i\}$, B= $\{b_i\}$, an initial transformation $T_0$
OUTPUT: The correct transformation T, which aligns A and B
$T \leftarrow T_0$
for $iter \leftarrow 0$ to $maxIterations$ do
  for $i \leftarrow 0$ to $N$ do
    $m_i \leftarrow$ FindClosestPointInA$(T \cdot b_i)$
    if $\|m_i - T \cdot b_i\| \leq d_{max}$ then
      $w_i \leftarrow 1$
    else
      $w_i \leftarrow 0$
    end if
  end for
  $T \leftarrow \underset{T}{argmin} \left\{ \sum_i w_i \|T \cdot b_i - m_i\|^2 \right\}$
end for
```

3. GPGPU based point to point iterative closest point algorithm

The algorithm of point to point method using GPU is shown on listing 2.

Algorithm 2 ICP - parallel computing approach

```
INPUT: Two point clouds $M = \{m_i\}$, $D = \{d_i\}$, an initial transformation $T_0$
OUTPUT: The correct transformation $T$, which aligns $M$ and $D$
$M_{device} \leftarrow M$
$D_{device} \leftarrow D$
$T_{device} \leftarrow T_0$
for $iter \leftarrow 0$ to $maxIterations$ do
  for $i \leftarrow 0$ to $N$ {in parallel} do
    $m_i \leftarrow$ FindClosestPointInM $(T_{device} \cdot d_i)$ {using regular grid decomposition}
    if $foundClosestPointInNeighboringBuckets$ then
      $w_i \leftarrow 1$
    else
      $w_i \leftarrow 0$
    end if
  end for
  $T_{device} \leftarrow \underset{T_{device}}{argmin} \left\{ \sum_i w_i \|T \cdot d_i - m_i\|^2 \right\}$ {calculation T←R,t with SVD}
end for
$M \leftarrow M_{device}$
$D \leftarrow D_{device}$
$T \leftarrow T_{device}$
```

The main idea of using the GPU is to decompose the 3D space into a regular grid of k*k*k buckets, where $k = 2^n$, n=5,6,7,8. Because we are violating the assumption of full overlap, we are forced to add a maximum matching threshold d_{max} related to the dimension of single bucket. This threshold accounts for the fact that some points will not have any correspondence in the second scan. In most implementations of ICP, the choice of d_{max} represents a trade of between convergence and accuracy. A low value results in bad convergence, a large value causes incorrect correspondences to pull the final alignment away from the correct value. In our implementation the choice of d_{max} is done by a normalization point cloud, which has XYZ coordinates from the interval < 1; 1 >. We improved the state of the art algorithm described in

[12] by replacing the complex k-d tree data structure to improve the performance of the closest point search by performing all computation using only GPU, we obtained TRUE nearest neighbor search and after some approximation real-time ICP with minimal lost of accuracy. NVIDIA GPGPUs are fully programmable multi core chips built around an array of processors working in parallel. Details about the GPU architecture can be found in [26] and useful additional programming issues are published in [27]. The GPU is composed of an array of SM multiprocessors, where each of them can launch up to 1024 co-resident concurrent threads. It should be noticed that available graphics units are in the range from 1 SM up to 30 SMs in 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. It is important to realize that all thread management (creation, scheduling, synchronization) is performed in hardware (SM), and overhead is extremely low. The SM multiprocessors work in SIMT scheme (Single Instruction, Multiple Thread), where threads are executed in groups of 32 called warps. The CUDA programming model defines the host and the device. The Host executes CPU sequential procedures, whereas the device executes parallel programs - kernels. A kernel works according to a SPMD scheme (Single Program, Multiple Data). CUDA gives an advantage of using massively parallel computation for several applications.

4. Correlation matrix elements computation using optimized parallel reduction

For the computation of the correlation matrix, the parallel prefix sum [28] is used. The all-refix-sums operations take a binary associate operator $\oplus$ with identity I, and an array of n elements

$$[a_0; a_1; \dots ; a_{n-1}] \tag{18}$$

and returns the array

$$[I; a_0; (a_0 \oplus a_1) ; \dots; (a_0 \oplus a_1 \oplus \dots \oplus a_{n-2})] \tag{19}$$

All-prefix-sums operations on an array of data is commonly known as a scan. The parallel implementation uses multiple thread blocks for processing an array up to 1024*1024 data points stored in a 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. An optimized kernel available in CUDA SDK [26][27] is used in parallel computation.

5. Singular Value Decomposition (SVD)

The equation for the singular value decomposition of a 3 x 3 matrix A is the following:

$$A = U\Sigma V^T \tag{20}$$

where U is an 3x3 matrix, Σ is an 3x3 diagonal matrix, and V^T is also an 3x3 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(ϕ_1;…; ϕ_n). Furthermore, $\phi_k > 0$ for $1 \le k \le r$, and $\phi_i = 0$ for $(r+1) \le k \le n$.

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) [29] are not beneficial especially when we have to implement them 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 the roots of a cubic polynomial) and sort these in descending order. Compute square roots to obtain singular values of A.
3. Construct the diagonal matrix $\sum$ by placing singular values in descending order along its diagonal. Compute $\sum^{-1}$.
4. Use the ordered eigenvalues from step 2 and compute the eigenvectors of A^TA. Place these eigenvectors along the columns of V and compute V^T .
5. Compute $U = AV\sum^{-1}$

6. Point to plain ICP

The alignment of the M and D data sets for point to plane ICP is solved by minimization of the following cost function:

$$E = (\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{N_m^p} \sum_{j=1}^{N_d} w_{ij} \left\| \mathbf{m}^p{}_i - (\mathbf{Rd}_j + \mathbf{t}) \right\|^2 \tag{21}$$

where $m^p{}_i$ corresponds to projected point from data set D onto the approximation plane of data set M. w_{ij} is assigned 1 if the i_{th} point of M correspond to the j_{th} point in D. Otherwise w_{ij}=0. R is the rotation matrix, t is the translation matrix, d_j corresponds to points from data set D. We also assume that the INDOOR dataset is locally planar. Since we are sampling from two different perspectives, we will not in general sample the exact same point. For this reason, in theory the point to plane method is more accurate than point to point (see Figures 1 and 2). Unfortunately in practical application there are many exceptions where this method is less accurate than classic point to point method. We observed that in some cases it is difficult to find automatically an accurate plane approximation and a correct point to plane correspondence (see Figure 3). Of course everything is determined by the need for high performance using GPU. Therefore, the implementation based on decomposition into a regular grid of buckets may be affected by this problem. The advantage of the point to plane method over point to point is the decreased time of computation, as there is no need to search best correspondence for each bucket. This is also related to the global synchronization of threads busy with executing the same kernel for each bucket. Instead of searching the best correspondence for each bucket, the point to plane method computes a projected point, and all kernels are executed in almost the same time. Therefore, the synchronization of threads does not affect global execution time. In the consequence point to plane method can be 3 times faster than point to point. However, we would like to emphasize the fact that the bottleneck related to the large amount of processed points affects the global performance of the method.

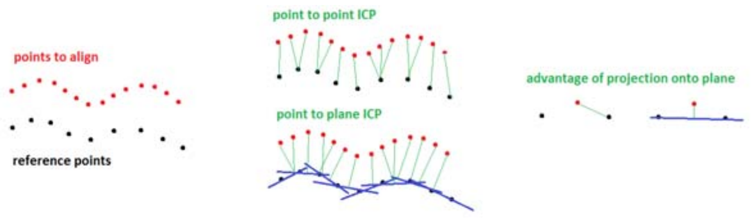

Figure 1. An advantage point to plane over point to point method.

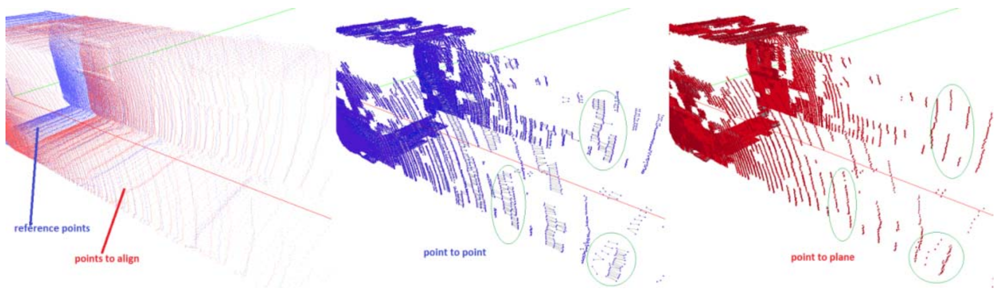

Figure 2. Comparison between point to point and point to plane methods. In the central image correspond points are connected via line segments. When we are sampling from two different perspectives, we will not in general sample the exact same point. Point to plane method eliminates this problem (the interesting regions are marked as green circles).

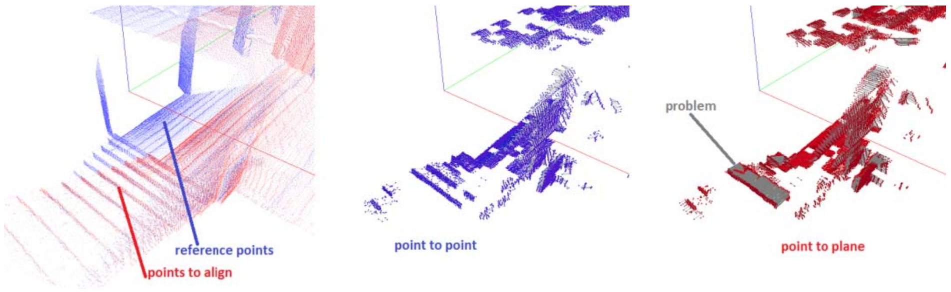

Figure 3. An example of disadvantage of point to plane method related to wrong correspondence between point and local plane approximation.

7. Using parallel vector computation for loop closing detection

Loop closing occurs when robot is visiting the same place a second time. It is very important to realize that the robot is not able to perform exactly the same path, therefore a displacement between loop closing robot positions occurs. In our opinion, this displacement should be minimized to increase the efficiency of the loop closing method. This can be done using the strategy of robot motion, especially when it traverses narrow paths. Even when the loop closing method is guarantying heading and displacement invariance, the compromise has to be taken into the consideration. For this reason we define loop closing as a location of the robot that it visited twice with the similar region of observation. To improve loop closing in INDOOR environment we propose new method based on semantic images comparison based on entropy. This method is faster than classic approach and performed experiments are showing satisfying results. For our purpose the semantic image is represented as a color image (RGB) where different colors correspond to different semantic objects. To obtain semantic image from 3D laser scan normal vectors for each point are computed. Normal vectors (X,Y,Z) are visualized as colors (R, G, B). As a result we expect that points belong to floor, ceiling or walls will have the same color. To find entropy between semantic image A and semantic image B for each semantic object from image A following entropy has to be computed using following equation:

$$E_{d^A}\left(P_{d^A}\right)=\sum_{d^B \in C_B} -\frac{\left|P_{d^B \in P_{d^A}}\right|}{\left|P_{d^A}\right|}\log\frac{\left|P_{d^B \in P_{d^A}}\right|}{\left|P_{d^A}\right|} \tag{22}$$

where:

P_{d^A} - region of pixels in image A of category d,

d - category/label,

C_B -set of categories in image B,

$\left|P_{d^B \in P_{d^A}}\right|$ - amount of pixels in image B of category d^B from region of category d^A in image A,

$\left|P_{d^A}\right|$ - amount of pixels in region of category d^A in image A.

Final entropy is given by following equation:

$$E(P_A) = \sum_{d^A \in C_A} \frac{\left|P_{d^A}\right|}{P_A} E_{d^A}\left(P_{d^A}\right) \tag{23}$$

where:

P_A -all pixels of image A,

d -category/label,

C_A -set of categories in image A,

$\left|P_{d^A}\right|$ -amount of pixels of category d^A,

$\left|P_A\right|$ - amount of pixels in image A,

$E_{d^A}\left(P_{d^A}\right)$ - entropy computed using equation 22.

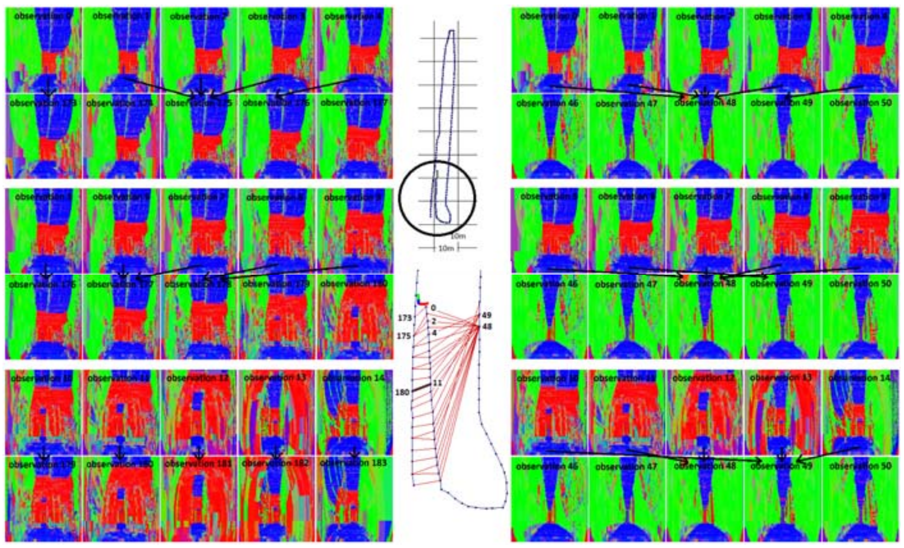

Figure 4. Loop closing with semantic images comparison based on entropy. Corresponding semantic images are connected via black arrows. Corresponding robot position are marked by red lines. Loop closing was found between robot observation 11 and 180, what is related with the minimum value of ICP error and maximum amount of corresponding points between 3D scans. Positive examples are shown on the left, negative example are shown on the right.

The entropy of the same image equals 0, therefore to find corresponding semantic images first we should minimize entropy. To demonstrate an approach figure 4 shows the loop closing procedure performed for data set shown on figure 7. The minimization of entropies is very fast (100ms per matching) but it does not guarantee loop closing, therefore further computation is needed. It can be used as a fast method for finding the prerequisites of loop closing. To validate correct loop closing it is necessary to perform ICP matching.

8. Using parallel vector computation and HSV image representation for vegetation detection

To find vegetation in outdoor environments we use RGB-D data obtained by the fusion of RGB data and 3DLSN (3D Laser Measurement System 3D) unit 3D data. The prerequisites of vegetation is determined by "noisy" normal vectors in 3D cloud of points. We assumed that vegetation is related to green color (in most cases), therefore we transformed RGB to HSV (Hue – Saturation –Value) image and assigned color using threshold obtained experimentally. From 3D cloud of points' image (noise) and HSV image (color) we obtain regions of vegetation. Figure 5 demonstrates the results of proposed method.

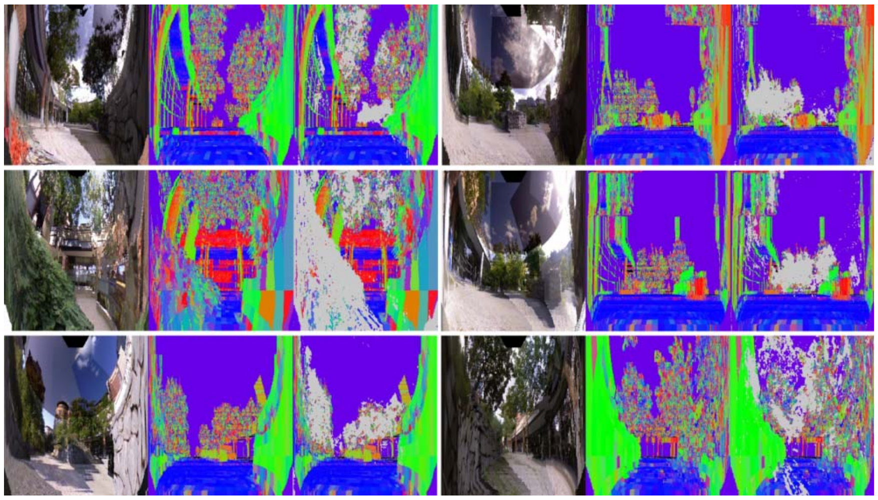
Figure 5. Vegetation detection. 9 examples of triple images are shown (RGB, semantic, semantic augmented with vegetation regions marked by gray color).

9. Empirical evaluation

The Proposed Iterative Closest Point variations were compared during real-task experiment, which was performed in an INDOOR environment shown on figure 6. The robot was acquiring observations in a stop-scan fashion with one meter step. The goal was to align iteratively all scans, therefore the odometry error was decreased. The data set is composed of 142 scans of 361*498 3D data points. Measurement was done using commercially available robotic platform PIONEER 3AT equipped with 3DLSN unit (rotated SICK LMS 200). We set as a benchmark for obtaining a satisfying result of odometry correction, performed with different variants of ICP. The main observation is that all ICP variants did correction of robot path derived from odometry with gyroscopic correction system. It seems that classic ICP gave the most accurate result, which was determined by the result of plane-approximation of complex shapes in the point to plane variant. However, we want to emphasize the fact that the accuracy of ICP variants strongly depends on various of factors. The accuracy of ICP point to point depends more on the amount of points in the bucket (it can be tuned by normalization of the point cloud) rather than number of iterations. The average amount of points in buckets should be more than 10 and less than 100 to obtain accurate alignment with on-line computation (300ms for 30 iterations). We observed that 30 iterations guarantee satisfying result. The accuracy of ICP point to plane strongly depends on the complexity of shapes in the cloud of points and an amount of projected points. The point to plane method is the fastest because there is only one computation of point projection instead of many computations related to nearest neighbor search in point to point method.

To conclude the discussion concerning the Iterative Closest Point method used for odometry correction, we show a result of an additional experiment on figure 7. The idea was to collect 3D data sets by the same robot (PIONEER 3AT equipped with 3DLSN unit and gyroscope odometry correction) two times in the same environment using different types of wheels (PIONEER 3AT indoor wheels and PIONEER 3AT outdoor wheels). The wheels have different radius, size and friction. The result is a longer odometry path for the indoor wheels (figure 7 C). We observed two important aspects of the applied ICP point to point method. First, the odometry error was decreased satisfactory in both cases. Second, there are some situations where the ICP accuracy can be decreased drastically, especially during rotations. Therefore, a proper strategy of data acquisition during robot motion has to be applied to avoid occlusions in the scan.

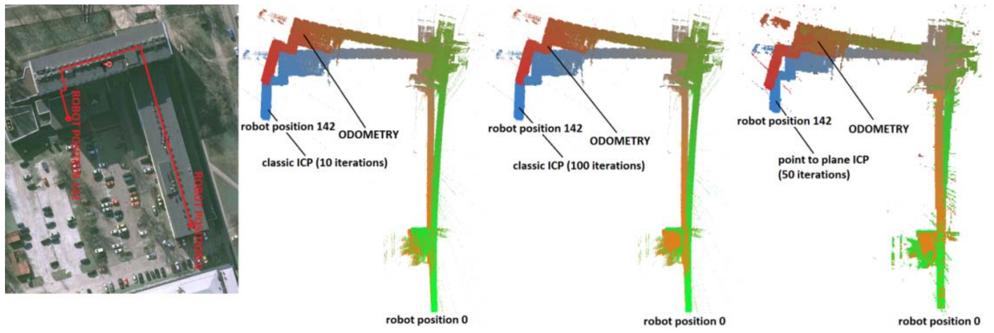

Figure 6. Comparison of ICP variants results. From left – indoor environment (Warsaw University of Technology, Faculty of Mechatronics), ICP point to point 10 iterations, ICP point to point 100 iterations, ICP point to plane 50 iterations.

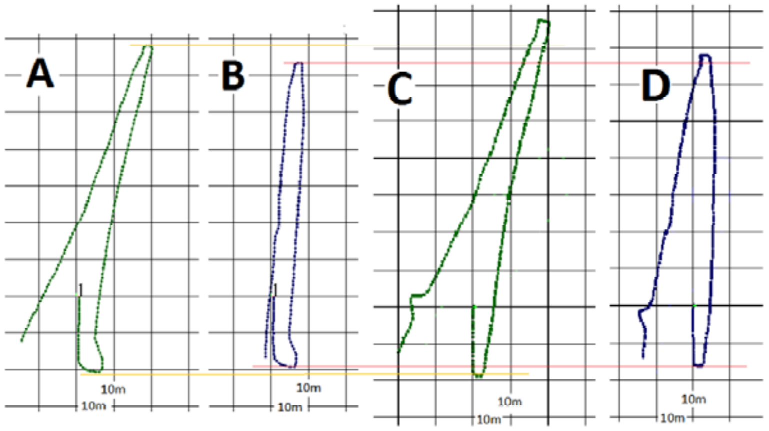

Figure 7. Comparison of ICP point to point (30 iterations) 10dometry correction for robot PIONEER 3AT equipped with 3DLSN unit and two different types of wheels (AB-OUTDOOR, CD-indoor). A-odometry path using OUTDOOR wheels, B-ICP result for A, C-odometry path using indoor wheels, D-ICP result for C. Experiment was performed in MECA laboratory in Royal Military Academy, Brussels, Belgium.

Conclusion

Compared to a state of the art method [10] where 60ms are needed to align two data sets of 320 x 240 data points, our implementation can process 361*498 data sets in 130ms for 30 iterations. Our main contribution was to propose a scalable method with an assumption of satisfying performance. Based on our best knowledge it will be very difficult to improve the performance while increasing amount of processed points because of the bottlenecks. For practical application it is very beneficial to process $2^{10} * 2^{10}$ points because we can align not only iteratively neighboring scans, but also building meta models containing more scans. It will improve the accuracy of the method and it can be done with the newest GPUs with the FERMI architecture. We demonstrated the problem of point to plane method's accuracy and compared with ICP point to point method based on various data sets. We have shown the applications of the loop closing and the vegetation detection based on normal vector computation. Future work will be related with GPGPU based ICP integration in 6DSLAM algorithm for robot localization and map building purpose.

References

[1] Martin Magnusson, Tom Duckett, and Achim J. Lilienthal. 3d scan registration for autonomous mining vehicles. Journal of Field Robotics, 24(10):803–827, Oct 24 2007.

[2] Andreas Nuchter, Hartmut Surmann, and Joachim Hertzberg. Automatic model refinement for 3D reconstruction with mobile robots. In Fourth International Conference on 3-D Digital Imaging and Modeling 3DIM 03, page 394, 2003.

[3] Michael Montemerlo and Sebastian Thrun. A multi-resolution pyramid for outdoor robot terrain perception. In AAAI'04: Proceedings of the 19th national conference on Artificial intelligence, pages 464–469. AAAI Press, 2004.

[4] Peter Kohlhepp, Paola Pozzo, Marcus Walther, and Rdiger Dillmann. Sequential 3d-slam for mobile action planning. In Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai, Japan, pages 722–729, September 28 - October 2 2004.

[5] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping. In ICRA, pages 321–328, 2000.

[6] Agustin Ortega, Ismael Haddad, and Juan Andrade-Cetto. Graph-based segmentation of range data with applications to 3d urban mapping. In 4th European Conference on Mobile Robots ECMR09, September 23- 25, 2009, Mlini/Dubrovnik, Croatia, pages 193–198.

[7] Daniel Huber and Martial Hebert. Fully automatic registration of multiple 3d data sets. Image and Vision Computing, 21(1):637–650, July 2003.

[8] Andrew W. Fitzgibbon. Robust registration of 2d and 3d point sets. In British Machine Vision Conference, pages 411–420, 2001.

[9] Martin Magnusson and Tom Duckett. A comparison of 3d registration algorithms for autonomous underground mining vehicles. In In Proc. ECMR, pages 86–91, 2005.

[10] Soon-Yong Park, Sung-In Choi, Jun Kim, and Jeong Chae. Real-time 3d registration using gpu. Machine Vision and Applications, pages 1–14, 2010. 10.1007/s00138-010-0282-z.

[11] Soon-Yong Park and Murali Subbarao. An accurate and fast point-to-plane registration technique. Pattern Recogn. Lett., 24:2967–2976, December 2003.

[12] Andreas Nuchter, Kai Lingemann, and Joachim Hertzberg. Cached k-d tree search for icp algorithms. In Proceedings of the Sixth International Conference on 3-D Digital Imaging and Modeling, pages 419–426, Washington, DC, USA, 2007. IEEE Computer Society.

[13] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the ICP algorithm. In Third International Conference on 3D Digital Imaging and Modeling (3DIM), June 2001.

[14] Deyuan Qiu, Stefan May, and Andreas Nuchter. Gpu-accelerated nearest neighbor search for 3d registration. In Proceedings of the 7th International Conference on Computer Vision Systems: Computer Vision Systems, ICVS '09, pages 194–203, Berlin, Heidelberg, 2009. Springer-Verlag.

[15] Sunil Arya and David M. Mount. Algorithms for fast vector quantization. In Proc. of DCC '93: Data Compression Conference, pages 381–390. IEEE Press, 1993.

[16] Timothy J. Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, and Pat Hanrahan. Photon mapping on programmable graphics hardware. In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS, 2003.

[17] Vincent Garcia, Eric Debreuve, and Michel Barlaud. Fast k nearest neighbor search using gpu. In 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, pages 1–6, 2008.
[18] Tim Foley and Jeremy Sugerman. Kd-tree acceleration structures for a gpu raytracer. In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, HWWS'05, pages 15–22, New York, NY, USA, 2005. ACM.
[19] Henrik Wann Jensen. Realistic image synthesis using photon mapping. A. K. Peters, Ltd., Natick, MA, USA, 2001.
[20] Deyuan Qiu, Stefan May, and Andreas Nuchter. Gpu-accelerated nearest neighbor search for 3d registration. In Proceedings of the 7th International Conference on Computer Vision Systems: Computer Vision Systems, ICVS '09, pages 194–203, Berlin, Heidelberg, 2009. Springer-Verlag.
[21] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In Proceedings of the 2009 IEEE international conference on Robotics and Automation, ICRA'09, pages 1848–1853, Piscataway, NJ, USA, 2009. IEEE Press.
[22] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Persistent Point Feature Histograms for 3D Point Clouds. In Proceedings of the 10th International Conference on Intelligent Autonomous Systems (IAS-10), Baden-Baden, Germany, 2008.
[23] Helmut Pottmann, Stefan Leopoldseder, and Michael Hofer. Registration without icp. Computer Vision and Image Understanding, 95:54–71, 2002.
[24] Jochen Sprickerhof, Andreas Nuchter, Kai Lingemann, and Joachim Hertzberg. An explicit loop closing technique for 6d slam. In 4th European Conference on Mobile Robots ECMR09, September 23-25, 2009, Mlini/Dubrovnik, Croatia, pages 229–234.
[25] P. J. Besl, H. D. Mckay, A method for registration of 3-d shapes, Pattern Analysis and Machine Intelligence, IEEE Transactions on 14 (2) (1992) 239-256. doi:10.1109/34.121791
[26] NVIDIA CUDA C Programming Guide 3.2. http://www.nvidia.com/cuda, 10 2010.
[27] CUDA C Best Practices Guide 3.2. http://www.nvidia.com/cuda, 8 2010.
[28] Mark Harris, Shubhabrata Sengupta, and John D. Owens. GPU Gems 3, Parallel Prefix Sum (Scan) with CUDA, chapter 39, pages 851–876. Addison-Wesley, 2007.
[29] http://www.netlib.org/lapack (2011).

Janusz Bedkowski, PhD in Automation and Robotics, adjunct in Industrial Research Institute for Automation and Measurements, Institute of Automation and Robotics- Warsaw University of Technology, Institute of Mathematical Machines, Warsaw, Poland. The scope of research: Mobile Assistive Intelligence, inspection intervention robot systems, semantic mapping, virtual training with AR techniques, GPU computing.

Andrzej Maslowski, full professor in Automation and Robotics Institute of Automation and Robotics - Warsaw University of Technology, Institute of Mathematical Machines Warsaw, Poland. The scope of research: Intelligent mobile systems, multi robot systems for RISE applications, e-Training systems for advanced mobile robotics. Member of IFIP, IFAC, IMACS, IMEKO TC-17 Measurement and Control in Robotics. Since 2006 representation of Poland in Joint Coordinating Forum International Advanced Robotics Programme.