A Mobile Robot Navigation with Use of CUDA Parallel Architecture

Barbara Siemiątkowska, Jacek Szklarski, Michał Gnatowski, Adam Borkowski, Piotr Węclewski

Abstract:

In this article we present a navigation system of a mobile robot based on parallel calculations. It is assumed that the robot is equipped with a 3D laser range scanner. The system is essentially based on a dual grid-object, where labels are attached to detected objects (such maps can be used in navigation based on semantic information). We use a classical SMPA (Sense - Model - Plan - Act) architecture for navigation, however, some steps concerning object detection, planning and localization are parallelized in order to speed up the entire process. The CUDA (Compute Unified Device Architecture) technology allows us to execute our algorithms on many processing units with use of a inexpensive graphics card which makes it possible to apply the proposed navigation system in a real time.

Keywords: *navigation, neural network parallel computing.*

1. Introduction

As robots move away from laboratory and act in complex real-world scenarios, both the control architectures and perception must become more powerful. For example, a service robot collaborating with a human needs to classify three-dimensional objects, know their functionalities and relationships between them. It also has to plan a path, avoid collisions and calculate low level control. All operations have to be done in a real time. The oldest classical control architecture is called SMPA. In this approach the navigation system is decomposed into a series of units (Fig. 1). It consists of following, repeated steps:

- Perception - the robot senses its environment
- Modeling - the map of the environment is built
- Planning - actions of the robot are planned
- Task execution - the robot performs the planned action

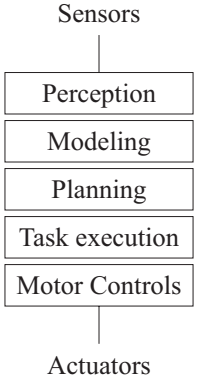

Fig. 1. A decomposition of a mobile robot control system.

This architecture is natural and easy to implement, however, one has to take care about sensor fusion, world modeling and path planning before executing an action. Traditionally an algorithm is implemented as a serial stream of instructions, and long time of reaction to environmental changes can become an issue [1,2].

In 1986 Rodney Brooks introduced subsumption architecture [3,4,5]. It consists of layered behaviors, with simple interfaces between them - units can take the inputs and/or outputs of other units. Each behavior has its own control program that is capable to react to environmental changes in real time. This architecture supports parallel computation and avoids centralized control. When this approach is used it is difficult to obtain smooth behavior of the robot. The method fails in case of more complex tasks.

The hybrid architecture [6,7] integrates the advantages of SMPA and subsumption and avoids the disadvantages. The whole framework of the decision-making system is based on the planning (SMPA architecture) and behavioral models are applied in the dynamic situation. This third approach is efficient but it can still fail when the robot acts in complex real-world environment.

In our approach we propose to use hybrid architecture, however, main modules: mapping, localization and collision-free path planning are divided into independent units so that each processing element can execute its part of the algorithm simultaneously with the other. Parallel programming allows multiple processes to be executed concurrently using separate threads. It can help reduce runtimes while still producing the same results as if it were run in serial. There are some restrictions with using parallelism and not every algorithm can be done in parallel.

Parallel computing [8] can by classified according the level at which the hardware supports parallelism. We can distinguish single multi-processor computers and clusters and grids of computers.

Recently platforms using graphics processing units (GPU) have become very popular. In our approach we propose to use Cellular Neural Network implemented using CUDA technology. The article consists of the following parts: in section 2 the cellular neural network paradigm and the implementation of CNN using CUDA technology are presented. In section 3 the architecture of the system is described. In section 4 the experimental results are shown. The article finishes with the conclusions.

2. CNN Implementation using CUDA Technology

Cellular Neural Network (CNN) also known as Nonlinear Neural Network was invented by Leon O. Chua and Lin Young in 1990 [9]. CNN is an array of analog dynamic

processors called cells. A standard CNN architecture consists of an $N \times M$ cells which are denoted $C(i, j)$. A typical example of CNN is presented in Fig. 2.

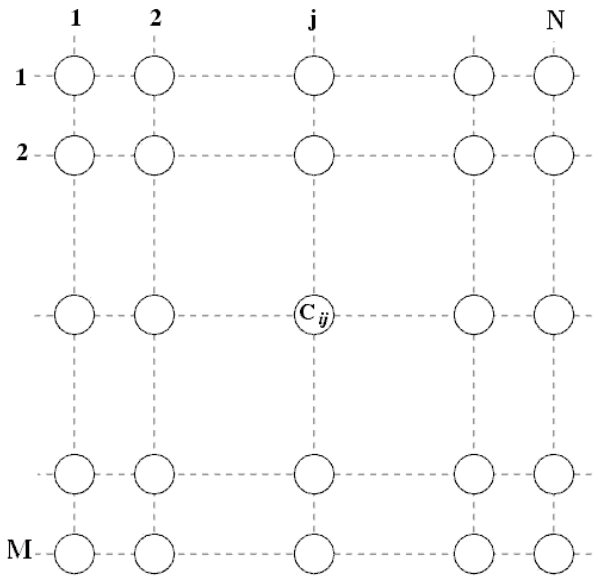

Fig. 2. The CNN architecture.

All CNNs are characterized by the fact that a cell's state $x_{i,j}$ is a function of its input $y_{i,j}$ and output of its neighbors only (and is discrete in the time domain). In other words:

$$x_{ij}^{t+1} = f(x_{i,j}^{t}, N^{t}), \quad (1)$$

where N^t denotes states at time t of neighborhood of $x_{i,j}$ (exact definition of neighborhood depends on the particular implementation).

The structure of CNN is similar to a human retina and the CNNs are also widely used in image processing. Using graphics processing units is a natural way of CNN implementation.

The most popular architectures are CUDA (*Compute Unified Device Architecture*) and Fire Stream. The former is a system dedicated for NVidia cards while the latter for ATI-AMD cards. It is worth to mention about a project of OpenCL libraries which is developed by the above companies and IBM (cluster computing technology) and others. Merging the simplest technologies of parallel computing emulations in operating systems (threads), computing on multi-core computers, interfaces to operations on graphic cards and computing the cluster systems give versatile and scalable solution but is more demanding to a programmer.

In our experiments we decided to use NVidia CUDA, due to its popularity, wide range of useful tools, documentation and examples.

While writing a sequential programme, a programmer does not need to know the hardware he or she works with. In applications where the key point lies on the short execution time, in particular implementing parallel computing, the structure and hardware often become important.

The graphic card GPU (*Graphics Processing Unit*) consists of multiprocessors (SM - stream multiprocessors) and a steering processor. Every SM device in each cycle processes threads grouped in so called "wraps". The number of threads in each wrap (wrap size) is different in each graphic card. SM may also be classified as a SIMT (*Single Instruction, Multiple Thread*) due to computing by a number of threads at the same time. Graphic card is also equipped with fast RAM named "Global". Additionally every SM has a local memory.

In computer games this technology is used in graphics rendering, physics calculations like fire, smoke or water diffusion. CUDA technology is not only used in graphical applications, but also in computational biology, cryptography and others. An example of CUDA technology is BOINIC which is a non-commercial system for grid computing. It became useful as a platform of distributed applications in mathematics, medicine, molecular biology, astrophysics and climatology.

CUDA provides simple access to GPGPU (*General-Purpose Computation on Graphics Processing Units*) and allows us parallel computing on its own GPU's. GPU's have numerous cores that operate in parallel to run time-consuming graphics operations. They have much more processing power than CPU's.

The flowchart of CUDA program consists of following stages:

- Blocks and threads are configurated
- Global memory is allocated
- The data is copied to the GPU
- The GPU kernel is called

Output data is copied back to the CPU. Fig. 3 presents the GPU architecture.

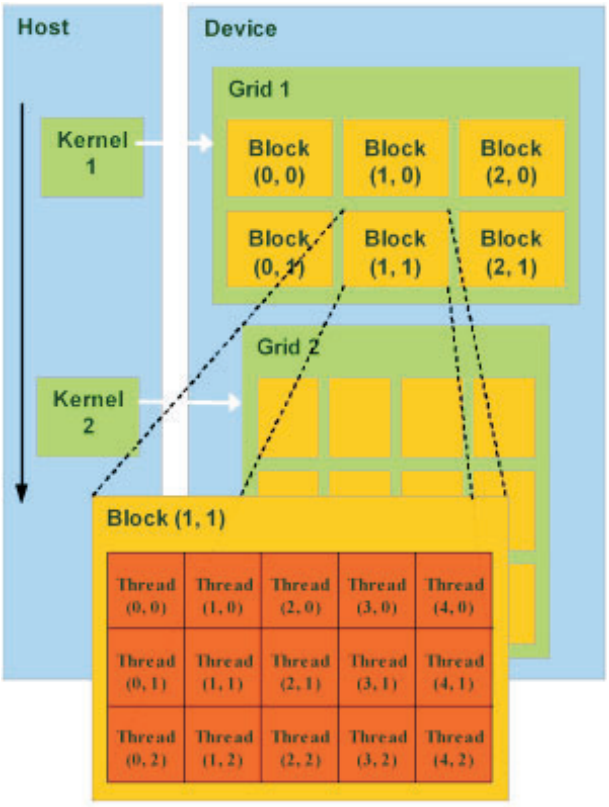

Fig. 3. GPU architecture (from NVidia.com).

3. The System Architecture

The architecture of the navigation system is presented in Fig. 4. Gray units present time consuming processes which we try to parallelize. In this section steps of the algorithm are described.

3.1. Sensors

The decision making system is essentially based on observations of distance to the nearest obstacles. At the present stage of the research, the mobile robot Elektron [11] is equipped with a 2D laser range finder Sick LMS 200, which measures this distance in a 2D plane for $-90° \leq \theta \leq 90°$ with the resolution 0.5° (for $\theta = 0$ the laser ray points forward $\theta = -90°$ towards robot's left, etc.). Moreover, the laser is mounted on a support which rotates around the horizontal axis. Therefore, the range finder can move up and down with the angle φ, and finally it provides a matrix *M* with numbers representing distance to an obstacle. Such matrix can be depicted as a gray-scale image. Fig. 6 presents the matrix *M* obtained for point cloud shown in Fig. 5.

The robot is also equipped with odometry which gives information about robot's displacement.

3.2. Processing

The matrix *M* is used for:

1) Localization
2) Constructing a map of the environment used for navigation.

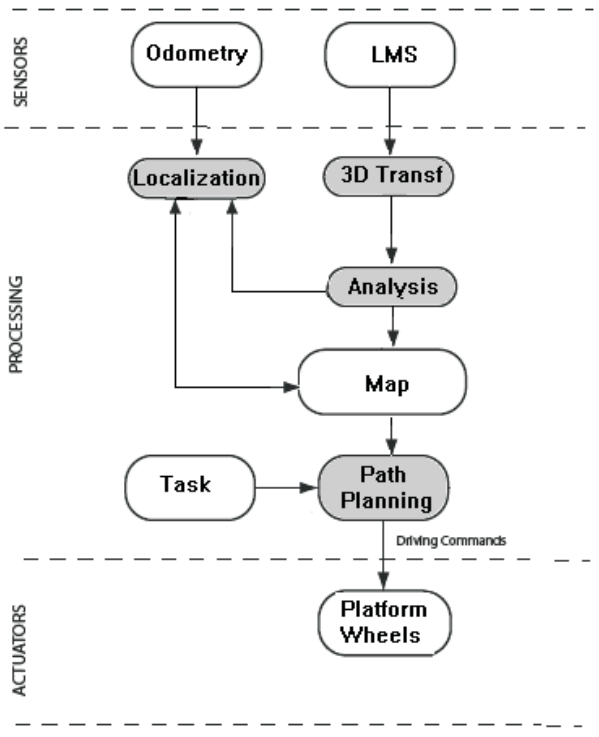

Fig. 4. The architecture of navigation system.

3) Recognition of objects of certain classes. The recognized objects can be placed onto the map of environment enabling *the semantic navigation*. Such navigation makes it possible to give to the robot commands in a human-understandable form, e.g., "find a nearest wastebasket".

Localization problem is solved using particles filter. The probability density of the estimated robot position is represented by a set of "particles", each encoding a single possible state (pose) of the vehicle. The particles are iteratively propagated using control input (motion model). On the basis of the measurement model a weight is attached to each particle. The number of particles depends on the uncertainty of odometry.

Tab. 1 presents the dependency between number of particles and the accuracy of determining robot displacement (the accuracy of odometry is assumed to be 10%). The experiments were performed with the use of robot Elektron1 in a real office environment.

Tab. 1. Dependency between the accuracy of displacement and the number of particles.

particles	the accuracy of displacement [%]
1000	7
10000	2
100000	1
1000000	0.2

For more than 1000000 particles the accuracy does not change. Due to its natures the particles filter algorithm can easily be parallelized.

The problem of object recognition is a much more complicated task, and there are no general solutions.

Our approach to this subject is based on the idea that the matrix *M* can be represented as an RGB image which possesses useful geometric information. Each row and column of *M* corresponds to certain θ and φ angles, so each entry $M_{i,j}$ describes a point in a 3D (θ, φ, *R*) space (*R* is the distance to obstacles). The robot is at the center of this coordinate system. Let $M_{k,l}$ is a vector defined by points $M_{i,j}$ and $M_{i+k,j+l}$. By taking the sum of the cross products:

$$V = N_{10} \times N_{01} + N_{01} \times N_{0_i-1} + N_{0_i-1} \times N_{0_i-1} + N_{0_i-1} \times N_{10} \quad (2)$$

one obtains the vector $\mathbf{V}=[n_x, n_y, n_z]$ which is (approximately) normal to the surface spanned on $M_{i,j}$ and its neighbors. **V** is normalized:

$$|\mathbf{V}|=1 \quad (3)$$

$$|n_x| \leq 1, |n_y| \leq 1, |n_z| \leq 1 \quad (4)$$

The final image is constructed by assigning red/green/blue (RGB) color values according to the *x,y,z* components of **V**.

$$\begin{aligned} R&=255*|n_x| \\ G&=255*|n_y| \\ B&=255*|n_z| \end{aligned} \quad (5)$$

Such images have interesting properties, which makes them useful as an input to an object recognition algorithm [12-14].

The process of converting matrix *M* into the RGB image can be efficiently solved via a typical SIMD (Single Instruction Multiple Data) architecture. Therefore it is a perfect task which can be parallelized, so the time-cost can be

significantly reduced. In particular, the relatively inexpensive CUDA technology fits very well for that purpose.

The object recognition procedure is based on Haar-like features and utilizes the concept of integral images. It is very fast and even for sequential single-processor, takes only short amount of time (approx. 0.1 s for typical parameters). However, for each Haar classifier only an object of a *single certain* class can be detected. Therefore, parallel approach is necessary to achieve short times for large databases of objects. Although at present we do not use any parallel programs to achieve this goal, it will be done in the future research.

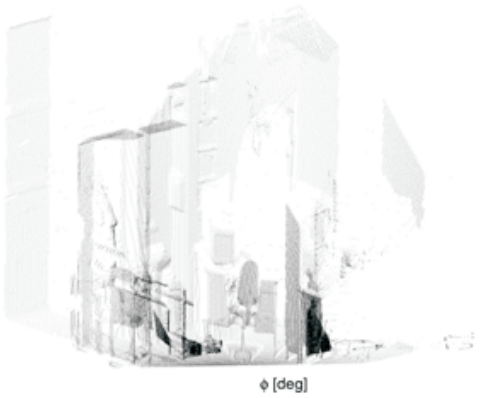

Fig. 5. Point cloud.

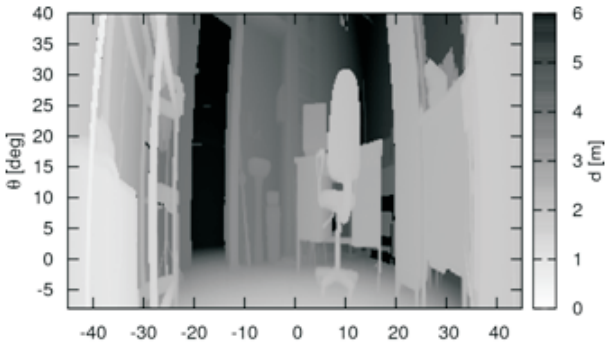

Fig. 6. Matrix M, representing distance to obstacles in meters, depicted as a gray-scale image.

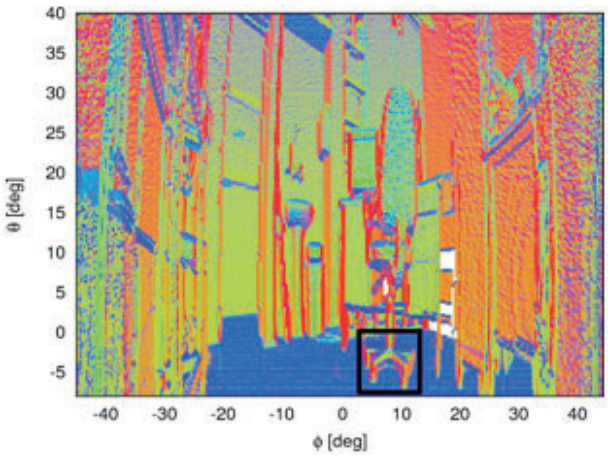

Fig. 7. The RGB image constructed from M using the procedure described in text. The black rectangle denotes the region which has been recognized as "base of an office chair" by the Haar classifier.

3.3 Path planning

Having a map of the environment, together with some detected objects, the robot must find its way. The path planning algorithm is realized with use of CNNs. For our CNN planning (described in details in, e.g., [14]), each $x_{i,j}$ corresponds to a cell in the grid-based map. Then, after initialization and the appropriate diffusion process, a steady state is achieved. The robot then, being at a place represented by a cell $x_{k,l}$, moves towards increasing gradient of x, and achieves its goal where maximum of x have been found. Due to its natures, such CNN, can easily be parallelized, also with use of the CUDA technology. This algorithm has following advantages: the time of collision-free path planning is comparable to reactive behavior, the goal can be given in natural language so we can ask the robot to go to specify object for example a chair. We can also ask the robot to go to the object far from other object or to avoid some places. In comparison to well known potential field method it does not suffer from local minima problem.

4. Experiments

The main goal of the experiments was to compare serial and parallel implementations of selected units. We implemented the mobile robot localization method using particle filters and the algorithm which computes normal vector. The experiments were performed using Intel Centrino (2x2.0GHz) processor (serial implementation) and NVidia GeForce 9300MG (parallel implementation)

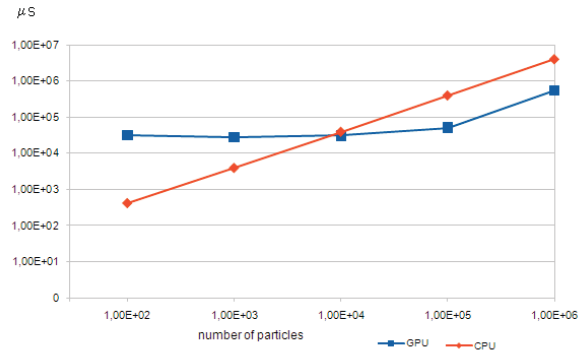

Fig. 8. Timing results (particles filter).

Fig. 8 presents a log-log plot of the overall execution time for the CPU and GPU implementations for different numbers of particles. The speedup factors (CPU_time / GPU_time) increase with problem size. We can notice that a speedup factor is greater than 1 if the number of particles exceeds 10^4.

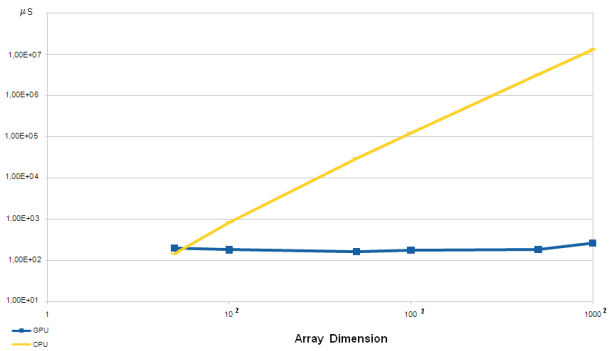

Fig. 9 Timing results (normal vector).

Fig. 9 presents the time of computing normal vectors for 361×215 point clouds using GPU and CPU. The speed-up factors (CPU_time / GPU_time) increase with problem size. Speedup factor is greater than 1 if the array size exceeds 10×10.

Fig. 10 presents the pseudo-code of the algorithm implemented using GPU.

Finally, we must stress that at the present stage of our research, the slowest part is still the process of gathering data about the distance to obstacles. However, in the next stage we shall start experiments with hardware stereoscopic cameras giving such matrices at the rate of 20 FPS. In this case, parallel calculations will lead to significant performance improvement.

Fig. 9. A pseudo-code showing how to calculate the matrix with normal vectors from the distance matrix *M*. The function NormVect is executed on GPU multiprocessors using nBlocks blocks, each running blockSize threads.

```
__global__ NormVect(Matrix<Point3D> in) {
index ← blockDim.x * blockIdx.x + threadIdx.x
 col ← index % width(in)
 row ← index / width(in)
    /* calculate sum of cross products of vectors
    from in[col][row] and its neighbors: */
out[col][row] ←vector(from in[col][row] to
in[col+1][row]) ×
      vector(from in[col][row] to in[col][row-1]) +
      vector(from in[col][row] to in[col][row+1]) ×
      vector(from in[col][row] to in[col-1][row]) + ...
 return out;
}

CalcPointCloud(Matrix M, Vector φ, Vector θ) {
Matrix<Point3D> r;
for each element i,j in M:
   r[i][j].x ← M[i][j] * sin θ[j] * cos φ[i]
   r[i][j].y ← M[i][j] * sin φ[i]
   r[i][j].z ← M[i][j] * cos θ[j] * sin φ[i]
return r
}


CUDACalcNormalVectors(Matrix M, Vector φ,
Vector θ) {
    /* M[i][j] is the distance for
    angles φ[i] and θ[j] */
 Matrix<Point3D> cloud ← CalcPointCloud(M, φ, θ)

copyToCUDADevice(cloud)
allocateOnCUDADevice(Matrix<Vector> out)
blockSize ← 16;
N ← width(M) * height(M)
nBlocks ← N/blockSize + (N % block_size == 0 ? 0:1)
NormVect <<< nBlocks, blockSize >>>
result ← copyFromCUDADevice(out) }
```

Fig. 10. The Pseudo-code for calculating normal vectors with use of CUDA.

5. Conclusions

In this article we presented a navigation system of a mobile robot equipped with a 3D laser range finder. Considerable amount of processing time can be saved by parallelizing some stages of the navigation system. This is accomplished by breaking problems into small independent parts which are executed simultaneously. The main goal of the experiments was to compare serial and parallel implementations of selected units of navigation system. The project is on an early stage of development but the results presented in Fig. 8 and 9 are promising and show that for the tested algorithms and typical data size, CUDA allows us to perform the task significantly faster than on a regular CPU. For example, as can be seen in Fig. 9, a typical image (with size 300 x 200) based on normal vectors is created in less than 0.001 s with use of CUDA, and about 1s otherwise. This is a crucial difference for a system working in real.

AUTHORS

Barbara Siemiątkowska* - Warsaw University of Technology, Department of Mechatronics, Institute of Fundamental Technological Research PAS, Warsaw, Poland.
E-mail bsiem@ippt.gov.pl

Jacek Szklarski - Institute of Fundamental Technological Research PAS, Warsaw, Poland.
E-mail jszklar@ippt.gov.pl

Michał Gnatowski - Institute of Fundamental Technological Research PAS, Warsaw, Poland.
E-mail mignat@ippt.gov.pl

Adam Borkowski - Institute of Fundamental Technological Research PAS, Warsaw, Poland.
E-mail abork@ippt.gov.pl

Piotr Węclewski - Warsaw University of Technology, Department of Mechatronics, Warsaw, Poland.
E-mail: piotr.weclewski@stud.mchtr.pw.edu.pl

* Corresponding author

References

[1] J. Nilsson, "Artificial Intelligence: A New Synthesis", Machine Industry Press, 1999.

[2] G. Saridis. "Toward the realization of intelligent controls", *Proceedings of the IEEE*. vol. 67, 1979, pp. 1115-1133.

[3] R.A. Brooks, "A robust layered control system for a mobile robot", *IEEE Journal of Robotics and Automation*, vol. 2 no. 11, 1986, pp. 14-23.

[4] R.A. Brooks. "New approaches to robotics". *Science,* vol. 253, 1991, pp. 1227-1232.

[5] M.J. Mataric., "Integration of representation into goal-driven behavior-based robots", *IEEE Trans. on Robotics and Automation*, 8(3), 1992, pp. 304-312.

[6] K.H. Low, W.K. Leow, and M.H. Ang, Jr., "A hybrid mobile robot architecture with integrated planning and control". In: *Proc. 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS-02)*, 2002, pp. 219–226.

[7] N.J. Nilsson. "Teleo-reactive programs for agent control", *Journal of Artificial Intelligence Research*, vol. 1, 1994, pp. 139–158.

[8] S.H. Seyed, "*Parallel processing and parallel algorithms: theory and computation"*, Springer, 2000.

[9] L. Chua, L. Young, "Cellular Neural Network", *IEEE Transaction on Circuit System*, 1990, pp. 500-505.

[10] Harris, Mark, "*Optimizing Parallel Reduction in CUDA*", *NVIDIA Developer Technology*. http://developer.download.nvidia.com

[11] Chojecki. R, Olszewski M., "A Mobile Robot for Laboratory Purposes and Its Applications", *PAK*, no. 3, 2009, 55, pp. 190-193.

[12] B. Siemiątkowska, J. Szklarski, M. Gnatowski, A. Zychewicz, "Budowa hybrydowej semantyczno-rastrowej reprezentacji otoczenia robota mobilnego na podstawie wskazań dalmierza laserowego 3D", *PAK*, no. 3, 2010, pp. 278-282. (in Polish)

[13] M. Gnatowski, B. Siemiątkowska, J. Szklarski, "Extraction of semantic information from 3D laser range Finder", In: *18th Symposium on Robot Design, Dynamics, and Control*, Springer, 2010, pp. 383-389.

[14] A. Borkowski, B. Siemiątkowska, J. Szklarski, "Towards Semantic Navigation In Mobile Robotics". In: G. Engles, C. Lewerenz, W. Schafer, A. Schurr, B. Westfechtel (Eds.): *Graph Transformations and Model Driven Engi-neering - Essays Dedicated to Manfred Nafle*, LNCS 5765, Springer.