3D Camera and Lidar Utilization for Mobile Robot Navigation

Submitted: 20th January 2013; accepted: 4th June 2013

Maciej Stefańczyk, Konrad Banachowicz, Michał Walęcki, Tomasz Winiarski

DOI: 10.14313/JAMRIS_4-2013/28

Abstract:

The article presents a navigation system based on 3D camera and laser scanner capable of detecting a wide range of obstacles in indoor environment. First, existing methods of 3D scene data acquisition are presented. Then the new navigation system gathering data from various sensors (e.g. 3D cameras and laser scanners) is described in a formal way, as well as exemplary applications that verify the approach.

Keywords: *mobile robotics, navigation, RGB-D sensing*

1. Introduction

An autonomous mobile robot navigation requires robust methods of environment analysis and obstacle detection. Typically, to achieve this goal, whole sets of various sensors are used, some of them giving point-like readings (e.g. infrared or ultrasound range finders) [13], some scanning in a plane (e.g. lidars). That kind of sensor configuration allows a safe movement of the robot in an unknown environment, but there is still a broad list of obstacles, that are hard to detect and avoid, most of them being either small (but dangerous) things laying on ground level or things hanging from top, that can hit upper parts of the robot maneuvering beneath them.

Current solutions, based on the aforementioned sets of sensors, have some shortcomings. Sometimes there is necessity of special preparation of the environment to make it reachable for robots (by, for example, clear marking of dangerous zones), restriction of common space used simultaneously by people and robots or making some strong assumptions about the environment (like riding only on smooth, flat surface without bumps or depressions [8]).

In the article a navigation method is presented that, thanks to utilization of data from both lidar and 3D camera, overcomes the most of the above mentioned shortcomings and is capable of detecting a wide range of obstacles in indoor environment. The robot control system (sec. 2, 3) is formally specified for the clearness of presentation and to simplify its further re-usage. On the other hand, presented control system still allows one to apply existing methods of securing robot surrounding, giving also possibility for easy fusion of data from diversity of sensors (which was proven in experiments described in section 4).

2. The method of system specification

Design of robot control systems requires a specification method that would facilitate its subsequent implementation. A multitude of robot controller architectures was proposed in the past, e.g. the subsumption architecture [2], the blackboard architecture [17] or an integrated architecture presented in [1]. In this paper we utilize a method based on earlier works fostering decomposing system into agents and description of the behaviour of each agent in terms of transition functions [19, 21, 22].

An *embodied agent*, being a central tenet of this method, is defined as any device or program having the ability to perceive its surroundings to subsequently influence the environment state, that may communicate with other agents and has an internal imperative to achieve its goal. In recent works [6, 20] the authors facilitated the inner structure of an *embodied agent* a, distinguishing five types of internal subsystems in two groups: agent's corporeal body (com-

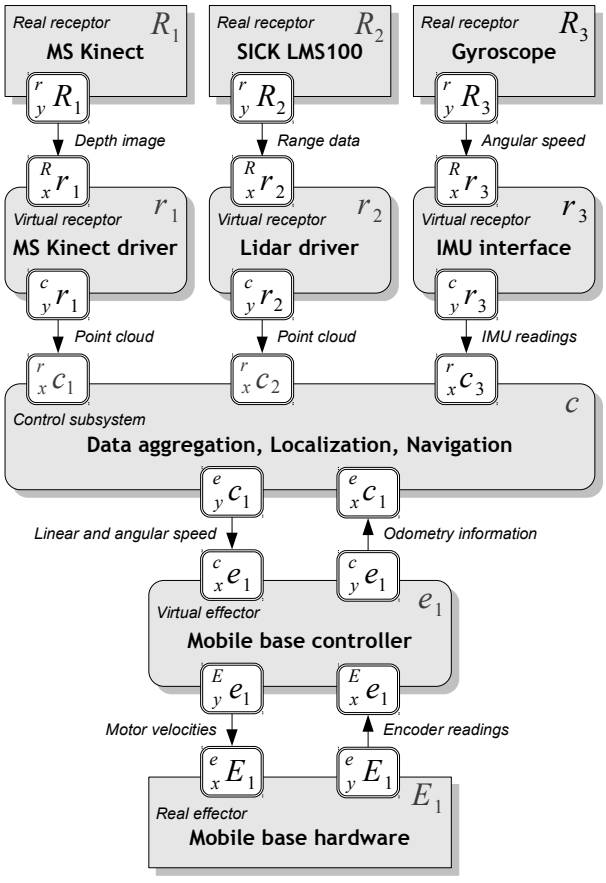

Fig. 1. Control system structure

posed of its effector E and receptor R) and control system (containing virtual effector e, virtual receptor r and a control subsystem c). Virtual effectors e and virtual receptors r form a hardware abstraction layer, presenting to the control subsystem both the state of effectors and receptors readings in a form that is convenient from the point of view of control. They also make it easy to replace effectors or receptors without changes in a control subsystem, as long as new modules are compatible with the created interface. All those subsystems communicate with each other via buffers and, besides that, it is assumed that they possess internal memory. Hence the evolution of the state of each of those subsystems is defined in terms of a transition function, transforming data taken from input buffers and the internal memory into values written to output buffers (and back to the internal memory as well) and sent subsequently to the associated subsystems.

Aside of the briefly described decomposition the authors also introduced data flow diagrams, supplementing the transition functions in the task of description of the behaviour of each of the subsystems constituting the control system. They also introduced a consistent notation, simplifying the description of such systems, as follows. A one-letter symbol located in the center (i.e. E, R, e, r, c) designates a subsystem. To reference its subcomponents or to single out the state of this subsystem at a certain instant of time extra indices around the central symbol are placed. A left superscript designates the referenced buffer of the subsystem or in the case of a function – its type. A right superscript designates the time instant at which the state is being considered. A left subscript tells us whether this is an input (x) or an output (y) buffer. A right subscript refers to the numbers of: a subcomponent (if we have only one agent and one control system, the rest of original symbols can be omitted), an internal memory cell or the ordinal number of a function. In the following we will utilize this method for the description of the designed control system of Elektron [12] mobile robot.

3. General system specification

The control system of Elektron mobile robot was designed in a way that simplifies its usage not only in a variety of applications on this specific robot, but also on different hardware platforms, without many assumptions about hardware structure. Whole system was designed as a single agent (depicted on fig. 1), without any communication buffers with other agents, with a control subsystem c as its vital part. Aside from controlling the system, to adapt the system to a particular robotic platform, one must provide virtual receptors and effectors implementation, that would interface the control system and real hardware.

The analysis of 3D scene acquisition methods [11] was a basis of choice of the adequate set of sensors for this task. Virtual receptors responsible for obstacle detection (in described application r_1 and r_2) gather data from either MS Kinect or SICK lidar and convert those readings into common representation (a cloud of 3D points). The transition functions of both modules are responsible only for changing the representation, so they are not described in detail. In short, in both virtual receptors the data read from input buffer ${}^{R}_{x}r_k$ is transfered to ${}^{c}_{y}r_k$. Virtual receptor r_3 is responsible for communicating with a gyroscope and generating messages of current angular speed. Buffer ${}^{c}_{y}r_3$ contains this speed expressed in SI units (rad/s). On the other side of control system is virtual effector, that translates commands from common format (linear and angular speed in ${}^{c}_{x}e_1$) to velocity values for both motors (${}^{E}_{y}e_1$) and returns current robot position (wheel odometry ${}^{c}_{y}e_1$) calculated from readings from wheel-mounted encoders (${}^{E}_{x}e_1$).

The control subsystem is divided into a set of modules responsible for different tasks: (i) robot localization (sec. 3.1) (local provided by f_1 transfer function and global by f_3), (ii) data filtering (sec. 3.2) (f_2), (iii) obstacle mapping (sec. 3.2) and path planning (sec. 3.3) (f_4). Those parts are independent on the hardware used, so the whole control system is portable between platforms. On the other hand, each of its subparts can be freely replaced with other (conforming to proper interface), so path planning algorithms, data filtering, environment mapping etc. can be either tuned or even replaced by other implementations, more suitable for specific task (without an impact on the whole structure of the agent).

3.1. Robot localization

There are three independent sources producing information about our mobile robot relative localization. Wheel odometry (available through ${}^{e}_{x}c_1$ buffer), based on incremental optical encoders and calculation of the position using matched consecutive laser scans [3], give full information about robot position and orientation (x, y, θ). Gyroscope readings (${}^{r}_{x}c_3$), on the other hand, can be used only to calculate a change in the orientation. All three sources return data with different frequencies, but Extended Kalman Filter [15], which is widely used in robotics, helps to incorporate all available readings into final estimation of local robot position. Transition function f_1 performing corrected local localization is depicted on fig. 2.

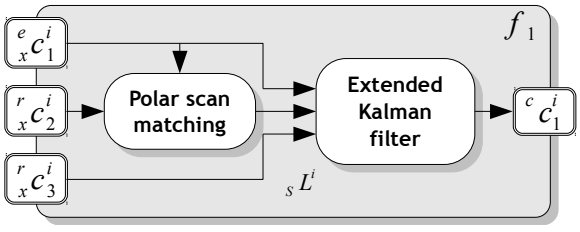

Fig. 2. Local robot localization using Kalman filter

Global robot position (relative to provided map) is calculated using a particle filter with laser scans as input data [16]. Additionally, current estimated position described earlier can be used in order to speed up calculations. It's worth mentioning, that provided map can be only a rough representation of the environment, containing only walls and other, most impor-

tant features, like doors or unmovable obstacles. Additional obstacles, that appear in laser readings, have virtually no impact on localization results, as long as they don't cover most of the robot's field of vision. Fig. 3 shows the structure of transition function f_3 executing global localization.

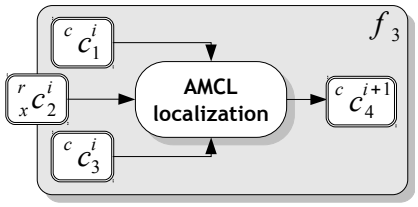

Fig. 3. Transition function of a global localization on provided map

3.2. Mapping of the obstacles

During a robot movement in an environment, which is not fully known (or sometimes even totally unknown), detecting and mapping of a current configuration of obstacles plays a vital role. In a static environment it's sufficient only to mark obstacles on a map after detecting them, but when obstacles can move or disappear, it's clearing obstacles from map what is essential (this part is much harder to do, since it's hard to tell whether obstacle really disappeared or we just can't detect it at the moment). The whole process of obstacle detection and mapping consists of consecutive steps of data aggregation from available sensors, filtering it, marking obstacles in an intermediate, 3D representation and finally the creation of 2D costmap for a motion planner.

3D representation of the environment Out of the mentioned steps, the intermediate 3D representation of the environment had to be considered first, because it affects all the other tasks. Methods based on geometric modeling of surroundings [14] are successfully used in tasks like a creation of 3D plans of buildings or open spaces, additionally giving possibility to filter out moving objects from final result. Octrees [5] are also widely used as a base structure when implementing obstacle maps (in this case the biggest advantage is a memory usage efficiency). Both approaches have a common drawback, which is methods of removing information about obstacles from them. In octrees, when a cell is marked as used, it's hard to perform inverse operation, leading to consistent results. An intermediate solution, containing features from both two- and three-dimensional representations is layered maps. In this case, the whole space is split into horizontal layers, each representing a fixed range of values from a vertical axis (usually z axis). On each layer, an information can be stored either geometrically [10] or in fixed rectangular grid (voxel representation). In the described system the key part is robust obstacle marking and clearing in close robot surrounding and there is less need for accurate mapping of whole corridors (because dynamic character of environment long-range maps with old information about obstacles could have negative impact on global path planners). Taking into account the mentioned facts, a voxel representation (layered map with discrete rectangular grid) has been chosen as a base structure for an obstacle map implementation.

Data aggregation and filtering The first step of processing sensor readings and creating a motion costmap is data aggregation itself. In the presented solution there are two sources giving information about obstacles – SICK LMS100 lidar and Microsoft Kinect. Both sensors produce high amount of data, which can easily lead to overloading the processing unit. Taking into account rather slow movement speed, to detect obstacles only 5 readings per second are used from lidar and 2 per second from Kinect. Data from sensors $^{c}_{x}r_k$ (for $k = 1, 2$) is throttled to desired frequency, giving throttled pointcloud $_{T}P_k$.

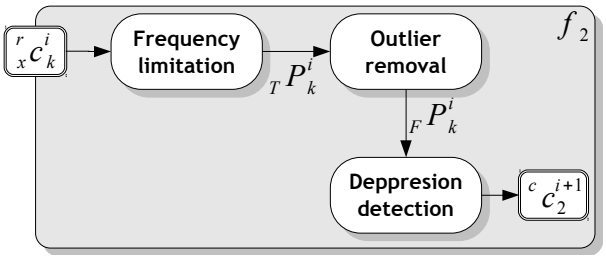

Fig. 4. Filtering readings from available sensors

This pointcloud is then passed through the series of filters (fig. 4), which remove incorrect points and produce new, virtual ones. Incorrect points, being sensor noise, could lead to detecting non-existing obstacles, which sometimes make the goal unreachable. To remove that kind of noise, every data point that has too few neighbours in its surrounding is removed (both sensors produce dense sets of points, so there should be no solitaire ones), producing filtered cloud $_{F}P_k$.

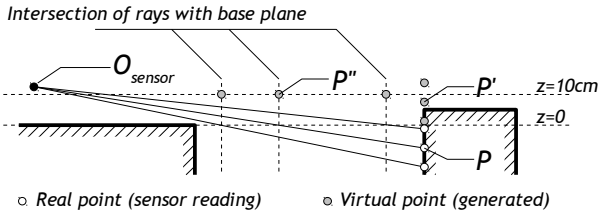

Fig. 5. Detecting dangerous slopes

Afterwards, when false readings are removed, the next filter processes the cloud of points in order to detect possibly dangerous floor slopes (like stairs leading down or holes in the ground). The mechanism is simple, but it gives very good results. All points having z coordinate over set threshold (e.g. $-0.02m$) pass through this filter without any change, the rest is reflected by xy plane, thus point $P = (x, y, z)$ becomes $P' = (x, y, -z)$. This way a „virtual wall" is produced (fig. 5) at the distance of the closest detected depressed obstacle, but it's still possibly far away from

real beginning of dangerous zone, so in next step another set of virtual points is produced. A line, connecting detected, negative point P with sensor origin $O = (o_x, o_y, o_z)$, intersects with ground plane at some coordinates $(x'', y'', 0)$ (1). A point P'' is produced by raising the intersection point 10cm above the ground, so it creates a virtual obstacle for robot. Those virtual points, in result, force path planners to avoid detected, dangerous zones, making the whole system safer. A filtered cloud is then stored in the internal memory of the control subsystem, in ${}^{c}c_2$.

$$P'' = (o_x + (x - o_x) \cdot s, o_y + (y - oy) \cdot s, 0.1) \quad (1)$$

with $s = \frac{o_z}{o_z - z}$.

Voxel map All the points that passed the filtering stage, are put into an occupancy map creating module. Its internal representation is based on voxel grid (three dimensional cuboid grid). The manipulation of grid parameters allows to balance between higher resolution (leading to better mapping and allowing robot to ride through smaller gaps) and lower computational load (for coarse grained maps) leading to faster path computation. A large size of cells, on the other hand, could make some places unreachable, because even small obstacles, like thin table legs, are represented by whole cells being marked as occupied.

When a preprocessed pointcloud is passed to this stage, prior to marking new obstacles, a removal procedure is executed. For every point in current measurement the same algorithm is used – every cell, that intersects with line connecting measured point with sensor origin, is marked as free. Afterwards all cells, in which points from current measurement are placed, are marked as occupied (this task is much simpler than the clearing stage).

The order of those two passes is important – if both clearing and marking were done in a single pass (i.e. for every point clearing and marking executed at sight), a situation would be possible, where just created obstacles are cleared by consecutive processed points from the same pointcloud. A voxel representation and the described algorithm allows an easy incorporation of readings from multitude of sensors, in this case lidar and Kinect are used without any change in algorithm, with one important difference between them – the points from Kinect are used for both clearing and marking, lidar, returning only planar readings, is used only for marking purposes.

Final costmap After marking all obstacles in an intermediate voxel grid, all cells are projected onto a two dimensional costmap. All columns containing occupied cells are marked as occupied, otherwise, the column's value depends on a number of unknown cells. If more than 4 cells are unknown, then the entire column is considered unknown, in every other case it is marked as free to move.

After marking, all the obstacles are inflated by specified value, creating safety zones around objects, which gradually increase the movement cost for cells. Lowering inflation radius makes robot go closer to obstacles, but can possibly lead to a situation, in which robot will ride along the wall so close, that turning is impossible without hitting it. Larger radius, on the other hand, will make some places unreachable.

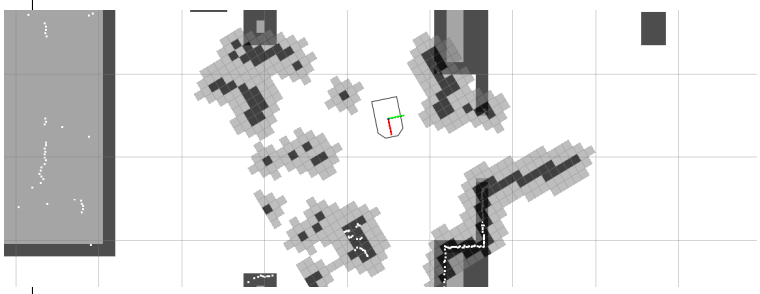

Fig. 6. Local costmap. In the background a global map is visible, darker cells mean occupied cells, safety zones are marked light gray and white cells are empty zones. Additionally, laser scan readings are marked as white points.

A local costmap has a fixed size and its origin is placed always in the origin of the robot's mobile base (fig. 6). In unstructured environments with dynamically changing configuration of obstacles keeping a map of all obstacles may cause some troubles, that were mentioned in earlier sections. A smaller costmap has also much better memory efficiency, because it has a fixed size instead of continuously growing as robot proceeds.

3.3. Path planning

When a goal is commanded to the robot (for example operator sets it manually), a global path planner is executed. This module is exchangeable, so any implementation following the interface rules can be used at this point (so it's one of few places, where future experiments using already created hardware and software platform can be carried on, even concerning such complicated ones as a chaotic space exploration [4]). In test applications Dijkstra algorithm was used as a global planning method, searching for a path in a graph built upon a current global map. Initially, only the global map is used (in the case when no obstacles are detected yet) and if a plan can't be executed (for example a door is closed), currently detected obstacles are included in a global path planning.

When a global plan ${}_{G}R$ is created, its parts are sent to the mobile base controller, where a local path planner (using Dynamic Window Approach) tries to command the robot in a way that follows the given trajectory part the best. The local planner uses only a local costmap around the robot ${}_{L}M$ and current odometry readings ${}^{e}_{x}c_1$. Parameters for the local planner are set so that each move ends as near as possible to the followed trajectory. After experiments with this module and tweaking parameters values, the robot followed the given global path with rather high average speed (approx. 17cm/s, which is 75% of its maximum speed), smoothly avoiding obstacles. General struc-

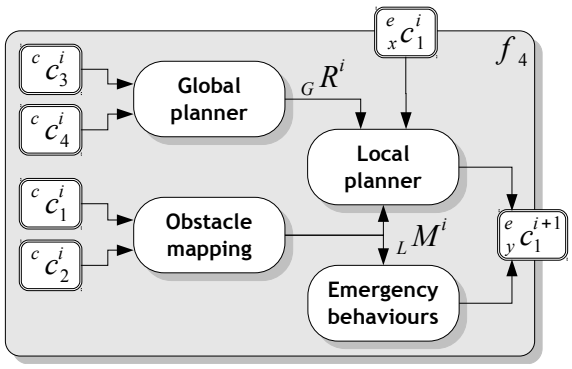

Fig. 7. General structure of navigation subsystem

ture of navigation subsystem is depicted on fig. 7.

3.4. Deadlock detection and solving

When the robot moves through an environment, where obstacles are potentially a-going, even best planning algorithms couldn't avoid being stuck in certain situations (contrary to static environments). When that kind of situation emerges, a series of possible solutions could be applied. First of all, of course, robot stops. Then all obstacles further that set threshold (half a meter) are cleared from the costmap and robot makes a full turn (360°, if possible) to refresh the obstacle configuration. If at this point the robot is still stuck, more aggressive algorithms are used, until all possible solution methods are tried. If at any point the robot can continue following its route, the deadlock is over.

4. Experiments

The experiments were conducted using Elektron mobile robot [12], equipped with control hardware developed according to the methodology presented in [18]. As a set of research oriented controllers, it provides access to all the essential parameters and since the communication protcol is open, it allows to replace any hardware module or plug an additional one. To make the high level software portable, the authors decided to base the control system implementation on ROS software framework [9], which is widely used in robotics. In general, virtual effector e_1 and virtual receptors $r_{1..3}$ are implemented as separate nodes, whilst control subsystem c is divided into multiple nodes, one for each transition function f_i. Operations inside transition functions (i.e. outlier removal or local planner) are implemented as nodelets running inside corresponding transition functions' node. Communication between nodes and nodelets (both transmission buffers between subsystems and internal transmissions) are mapped as ROS topics.

The experiments were conducted in office and workshop rooms, where many objects were constantly carried over the whole place. Those experiments aimed at confirmation of propriety of chosen approach. In some experiments robot moved through the corridor, where students moved chairs and tables from place to place particularly often (fig. 8), doors

Fig. 8. Elektron robot navigating in target environment

were opened and closed, and, of course, like in [7], there were many people walking along (fig. 8). According to the assumptions, robot had to safely drive through avoiding all obstacles and try to reach a given goal.

The basic scenario of each experiment was the same – the robot had to reach a goal (set by operator) autonomously and safely, without any collision with obstacles. Various sensor configurations were tested: only Kinect, only lidar and a fusion of data from both. Each experiment was repeated a number of times to calculate average statistics.

In cases when only one sensor was used, the advantage of lidar over Kinect in terms of overall movement speed was visible. A wide field of view of Sick LMS100 (270°) makes it possible to plan the path faster, without the need for too many robot in-place rotation, which was necessary with narrow angle of 57° Kinect sensor. Those rotations were used in order to find obstacles at the sides of the robot. On the other hand, Kinect detected many obstacles that were invisible for lidar, like office chairs bottoms, which allowed setting much lower critical distance to obstacles during the ride (20cm in case with lidar, 5cm for Kinect). This in turn made much more goals reachable for the robot, which precisely maneuvered through narrow spots (like almost closed doors). The biggest disadvantage of Kinect in terms of obstacle detection was inability to detect narrow table legs, which was potentially dangerous. When both data sources were enabled at the same time, the drawbacks of each of them were eliminated by other sensor capabilities, which improved both overall robot speed and precision of obstacle avoidance. Table 1 presents summarized statistics of conducted experiments.

Tab. 1. Summarized results for 20 repeats of each experiments

Sensors	Avg. speed	Obstacle crit. dist.
Lidar	16cm/s	20cm
Kinect	12cm/s	5cm
Kinect + lidar	17cm/s	5cm

The key issue of the control system of a mobile robot operating in common space with humans is safety. During the tests, there were many people working and walking along, influencing robot actions

(sometimes even on purpose). This led to new obstacles appearing and, in extreme cases when robot was surrounded by them, system stopped movement and, after a few rotations to find a clear path, aborted the goal. In every case, the system properly recognized stairs (when using Kinect) and, if commanded to go to a point unreachable because of them, the system returned an appropriate error message.

5. Conclusions

In an unstructured and dynamic environment, like office and laboratory spaces, three dimensional sensors seem to be the most appealing method for fast and reliable detection of a wide variety of obstacles. Augmenting this data with wide-angle readings from lidar further improves the detection results, making it possible to avoid almost every solid obstacle (e.g. small, laying on ground, hanging from top, or narrow legs). Additional CPU load caused by the processing of three-dimensional point clouds is compensated by an extended set of detectable obstacles, which makes the whole system robust and safe.

The presented system can be used in unstructured and cluttered environments without their prior preparations, in particular there is no need either to remove any obstacles, or mark safe zones. Thanks to this, a robot can be quickly deployed in a new environment and during its work those places can be simultaneously used by people. It makes a range of a new applications possible (e.g. mobile robot assistant of the man) as the future work.

Acknowledgements

The authors gratefully acknowledge the support of this work by The National Centre for Research and Development grant PBS1/A3/8/2012. Tomasz Winiarski has been supported by the European Union in the framework of European Social Fund through the Warsaw University of Technology Development Programme.

AUTHORS

Maciej Stefańczyk* – Institute of Control and Computation Engineering, Warsaw University of Technology, 00–665 Warszawa, ul. Nowowiejska 15/19, e-mail: stefanczyk.maciek@gmail.com.

Konrad Banachowicz – Bionik Robotics Scientific Club, Warsaw University of Technology, 00–665 Warsaw, Nowowiejska 15/19, e-mail: konradb3@gmail.com, www: http://bionik.ia.pw.edu.pl.

Michał Walęcki – Institute of Control and Computation Engineering, Warsaw University of Technology, 00–665 Warsaw, Nowowiejska 15/19, e-mail: mwalecki@elka.pw.edu.pl.

Tomasz Winiarski – Institute of Control and Computation Engineering, Warsaw University of Technology, 00–665 Warsaw, Nowowiejska 15/19, e-mail: tmwiniarski@gmail.com.

*Corresponding author

REFERENCES

[1] R. Alami, R. Chatila, S. Fleury, M. M. Ghallab, and F. Ingrand, "An architecture for autonomy", *Int. J. of Robotics Research*, vol. 17, no. 4, 1998, pp. 315–337.

[2] R. A. Brooks, "A robust layered control system for a mobile robot", *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, 1986, pp. 14–23.

[3] A. Diosi and L. Kleeman, "Laser Scan Matching in Polar Coordinates with Application to SLAM". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[4] A. A. Fahmy, "Implementation of the chaotic mobile robot for the complex missions", *Journal of Automation Mobile Robotics and Intelligent Systems*, vol. 6, no. 2, 2012, pp. 8–12.

[5] D. Jung and K. Gupta, "Octree-based hierarchical distance maps for collision detection". In: *International Conference on Robotics and Automation (ICRA)*, vol. 1, 1996, pp. 454–459.

[6] T. Kornuta and C. Zieliński, "Behavior-based control system of a robot actively recognizing hand postures". In: *15th IEEE International Conference on Advanced Robotics, ICAR*, 2011, pp. 265–270.

[7] B. Kreczmer, "Mobile robot navigation in the presence of moving obstacles (in polish)", *Advances in Robotics. Robot controll with surrounding perception.*, 2005, pp. 177–186.

[8] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment". In: *International Conference on Robotics and Automation (ICRA)*, 2010, pp. 300–307.

[9] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system". In: *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009.

[10] P. Skrzypczynski, "2D and 3D world modelling using optical scanner data", *Intelligent robots: sensing, modeling, and planning*, vol. 27, 1997, p. 211.

[11] M. Stefańczyk and W. Kasprzak, "Multimodal segmentation of dense depth maps and associated color information". In: *Proceedings of the International Conference on Computer Vision and Graphics*, vol. 7594, 2012, pp. 626–632.

[12] W. Szynkiewicz, R. Chojecki, A. Rydzewski, M. Majchrowski, and P. Trojanek. "Modular mobile robot – Elektron (in Polish)". In: K. Tchoń, ed., *Advances in Robotics: Control, Perception and Communication*, pp. 265–274. Transport and Communication Publishers, Warsaw, 2006.

[13] S. Thongchai, S. Suksakulchai, D. Wilkes, and N. Sarkar, "Sonar behavior-based fuzzy control for a mobile robot". In: *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, vol. 5, 2000, pp. 3532–3537.

[14] S. Thrun, W. Burgard, and D. Fox, "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping". In: *International Conference on Robotics and Automation (ICRA)*, vol. 1, 2000, pp. 321–328.

[15] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*, The MIT Press, 2005.

[16] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust monte carlo localization for mobile robots", *Artificial intelligence*, vol. 128, no. 1, 2001, pp. 99–141.

[17] H. Van Brussel, R. Moreas, A. Zaatri, and M. Nuttin, "A behaviour-based blackboard architecture for mobile robots". In: *Industrial Electronics Society, 1998. IECON'98. Proceedings of the 24th Annual Conference of the IEEE*, vol. 4, 1998, pp. 2162–2167.

[18] M. Walęcki, K. Banachowicz, and T. Winiarski, "Research oriented motor controllers for robotic applications". In: K. Kozłowski, ed., *Robot Motion and Control 2011 (LNCiS) Lecture Notes in Control & Information Sciences*, vol. 422, 2012, pp. 193–203.

[19] C. Zieliński and T. Winiarski, "Motion generation in the MRROC++ robot programming framework", *International Journal of Robotics Research*, vol. 29, no. 4, 2010, pp. 386–413.

[20] C. Zieliński, T. Kornuta, and M. Boryń, "Specification of robotic systems on an example of visual servoing". In: *10th International IFAC Symposium on Robot Control (SYROCO 2012)*, vol. 10, no. 1, 2012, pp. 45–50.

[21] C. Zieliński, T. Kornuta, P. Trojanek, and T. Winiarski, "Method of Designing Autonomous Mobile Robot Control Systems. Part 1: Theoretical Introduction (in Polish)", *Pomiary Automatyka Robotyka*, no. 9, 2011, pp. 84–87.

[22] C. Zieliński, T. Kornuta, P. Trojanek, and T. Winiarski, "Method of Designing Autonomous Mobile Robot Control Systems. Part 2: An Example (in Polish)", *Pomiary Automatyka Robotyka*, no. 10, 2011, pp. 84–91.