Design of Control Algorithms for Mobile Robots in an Environment with Static and Dynamic Obstacles

Submitted: 23rd April 2019; accepted: 20th December 2019

Robert Piotrowski, Bartosz Maciąg, Wojciech Makohoń, Krzysztof Milewski

DOI: 10.14313/JAMRIS/4-2019/34

Abstract: *This article proposes the construction of autonomous mobile robots and designing of obstacle avoidance algorithms for them. Nowadays, mobile robots are gaining more and more popularity on the customer as well as industrial market, for example as automatic vacuum cleaners or lawnmowers. Obstacle avoidance algorithms play an important role in performance of this types of robots. The proposed algorithms were designed for builds with rather not expensive electronic components, especially sensors with limited precision and dynamics. The project began with the selection of needed parts and building materials as well as designing of the PCB and assembling the whole construction. The project included also designing and developing the software responsible for, among others, implementation of obstacle avoidance algorithms. After the project's completion, a series of tests in a closed environment was conducted to verify the quality of vehicles' performance. Results of tests were positive.*

Keywords: *obstacle avoiding, control system, software development, mobile robot, mechatronics*

1. Introduction

Autonomous vehicles find frequent use in various fields of human life as aid or substitution of a laborer in dangerous, hard or wearisome work conditions. Besides that, they are also used in the industry, military or consumer markets, e.g. package delivery, surveillance by unmanned aerial vehicles (UAV) or toys. Primary features of autonomous vehicles are: independence in decision-making, collecting and processing data about surrounding environment and influence on that environment [1].

Necessity of work in unknown or dynamically changing environment as well as the purpose of eliminating human's participation in work, have made it necessary to develop effective obstacle avoidance algorithms [2]. Robots equipped with these systems can easily pass by obstacles, both static and dynamic, situated on their path.

Popular nowadays robotic vacuums are an example of this kind of vehicles [3]. Due to their small dimensions they are able to reach areas that are difficult to access. Variety of models are available to use, starting with simple ones containing only the vacuum cleaner function, up to multifunctional cleaning robots. Many of them use advanced systems like creating a virtual map of the room, optimizing energy drain needed to cover the work area as well as communication systems allowing the supervisory control of the vehicle by mobile application. Obviously it is necessary for those robots to be able to avoid obstacles not to damage household appliances, furniture and own construction.

Another example of use of autonomous vehicles is terrain exploration. The Curiosity Rover made by NASA has been sent to Mars to survey its surface to analyze soil and rock composition and to search for water and minerals [4]. It is also a scientific research station measuring and processing collected data. Avoidance of obstacles and moving in hard, uneven and rocky environment is a key aspect of its proper functioning.

Autonomous vehicles also find use in the industry. For example, Amazon branches are using mobile robots to aid workers [5]. Machines independently search for specific packages and then carry them to the demanded place using special lifts. They are able to transport weights up to 340 kg and significantly accelerate the work, freeing laborers of need for searching and carrying heavy packages. With this in mind, it is essential for robots to avoid obstacles and other machines so that they can work effectively.

The paper is organized as follows. In the second section, the realization of mobile robots and board is described. The design of control algorithms is presented in the third section. The fourth section proposes software development. Verification tests results are given in Section 5. The last section presents the conclusions.

2. Realization of Mobile Robots and the Board

To begin with, it was necessary to set goals for construction of mobile robots. Requirements for them were as follows: small dimensions, agility, low power consumption and ability to gather data from the environment. Moreover, the goal was to design effective control methods and obstacle avoidance algorithms

which would be implemented in digital hardware with small costs.

All electric schematics were made using the Computer Aided Design (CAD) software – Eagle. The project of the machine was divided into a few parts containing elements responsible for various features. The central processing unit (CPU) is an 8-bit microcontroller from ATmega series. The main part consists of previously mentioned CPU as well as all necessary components such as quartz resonator, passive voltage filters and reset pull-up resistor. Memory capacity of CPU is 32 kilobytes, which is enough to sustain software features. It allows processing of gathered data and realization of control algorithms. The part of the electric scheme presenting CPU and all its connections is shown in Figure 1. Connections labeled as IA1, IA2, IB1 and IB2 are Pulse-Width Modulation (PWM) channels controlling motors' speeds and direction.

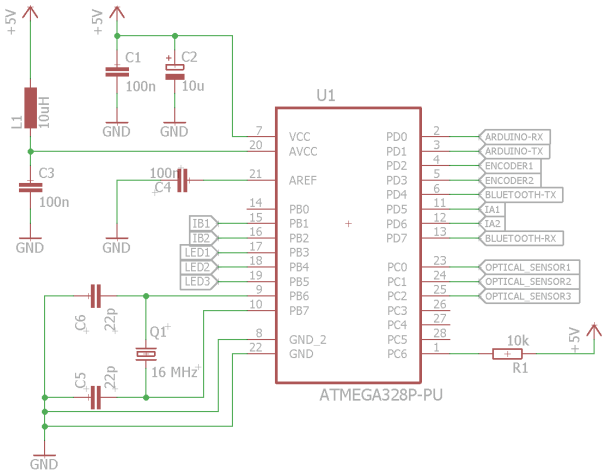

Fig. 1. Part of the scheme with CPU

The second part of project scheme is the power supply. It contains a voltage regulator, a Li-Pol (Lithium-polymer) battery pack and a LED (Light Emitting Diode) informing about power supply set on. The used accumulator provides high current efficiency and a long lifespan level. This part of the scheme is presented in figure 2.

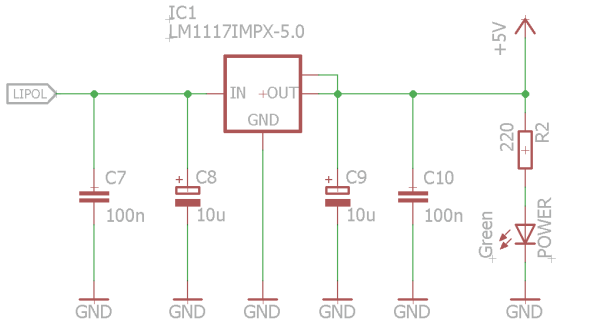

Fig. 2. Part of the scheme with voltage regulation

The next part are motors, their controller and encoders. Two DC motors settled at the back of construction are controlled with a dedicated module. Encoders count spins of motors shafts and convert them into electric impulses, further sent to the microprocessor. Due to CPUs limitations, only information about quantity of rotations is gathered and data about its direction is overlooked. The direction is calculated programmatically. Resolution of outputs is also limited to 6 impulses per rotation. Connections of motor controller have been presented in Figure 3. Pins labeled as M1+, M1-, M2+ and M2- are directly connected to motors.

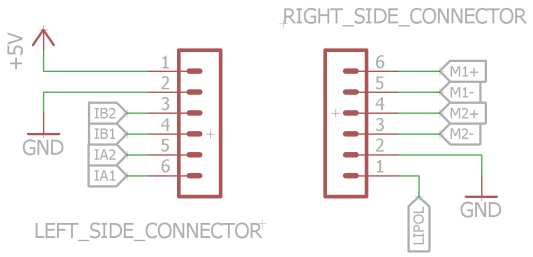

Fig. 3. Part of the scheme with motors' controller

The following part is measuring equipment. Three optical sensors were chosen. They work in range of 4 – 30 centimeters. One of them is settled in the middle of the front edge of the PCB and two on both sides of it, rotated by 15 degrees. Each of the sensors has been connected in a manner shown in Figure 4.

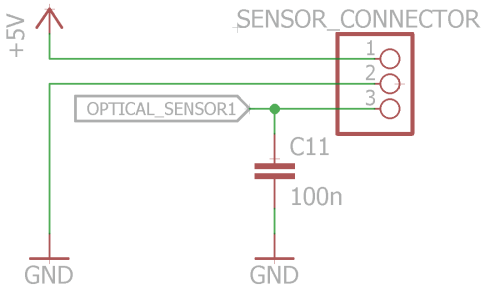

Fig. 4. Part of the scheme with connections of optical sensor

The last part is communication. The connection between robots and other devices is provided by the UART module or Bluetooth module. Also, LED diodes were mounted, which can be programmed to inform the user about the current machine state. Due to a mismatch in voltage standards (3.3 V for Bluetooth module and 5 V for ATmega microcontroller), the signal coming from CPU has to be passed through a voltage divider, to lower the voltage to roughly 3.3 V. There is no need to boost the voltage of a signal coming from module to CPU, since 3.3 V is already interpreted as a high state in 5 V standard logic. The connections of the Bluetooth module are shown in Figure 5.

Next stage was the PCB design project. The board is the chassis of the robot, all electrical and mechanical parts are mounted on it. The dimensions are 90.15 mm by 88.57 mm (length, width). Projects of upper and bottom layer were prepared using the EAGLE software (see Figures 6-7).

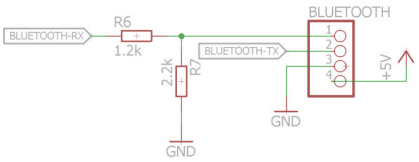

Fig. 5. Part of the scheme with connections of the Bluetooth module

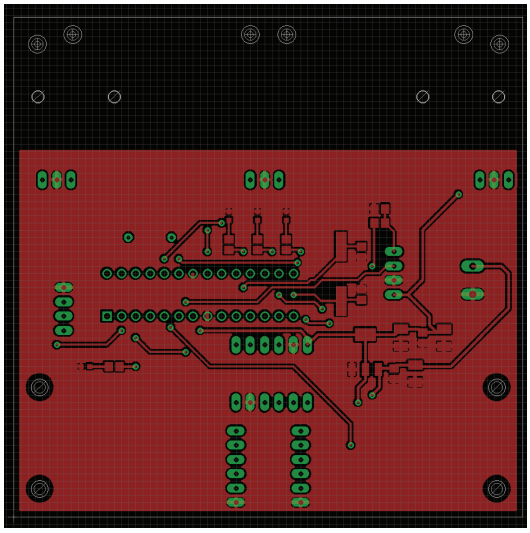

Fig. 6. Upper layer of the board

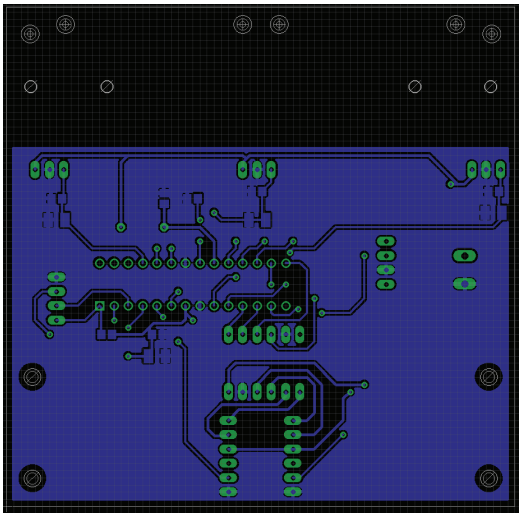

Fig. 7. Bottom layer of the board

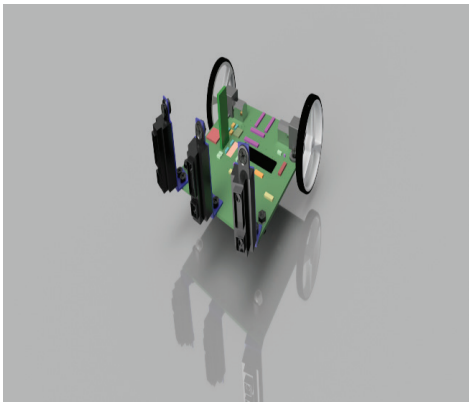

Fig. 8. Render of the robot model

After that part of design has been completed, a simplified 3D model of the robot was made using Autodesk Inventor environment. Imported files of PCB were the base to be supplemented by other elements, such as motors, wheels and optical sensors. Figure 8 shows a render of the robot model.

After the design stage was completed, the constructions of robots were prepared. The final effect of the assembled robot is shown in Figures 9-10.

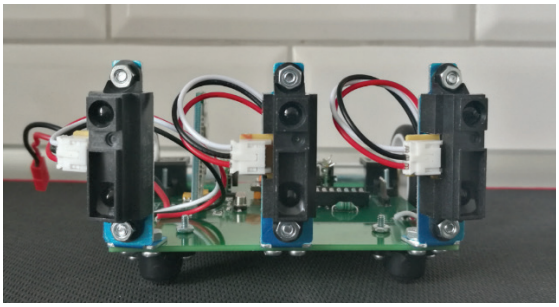

Fig. 9. Front view of the mobile robot

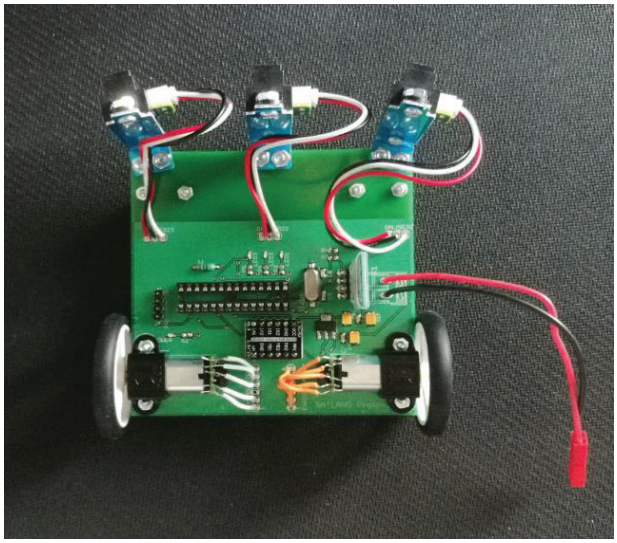

Fig. 10. Top view of the mobile robot

The working area is limited to provide an easier control of their work and eliminates part of external factors such as smoothness and surface texture which may actually affect performance. The working area also had to be big enough to freely test the behavior of robots controlled by implemented avoidance algorithms.

Static obstacles were spatial elements of different shape and size, settled on the surface of the robots working area. Design of the board was also prepared with CAD software – Autodesk Inventor. It is a square of 1.25 m side length, which gives 1.5625 m^2 of place to work with. The height of boarding walls is 15 cm. After designing the board, a 3D model was prepared (see Figure 11).

Final construction was made with laminated chipboard, which is a universal, cheap material easy to process further. The base is separated into two parts. The walls are made with single parts. Everything is assembled with screws. Figure 12 presents the assembled board.

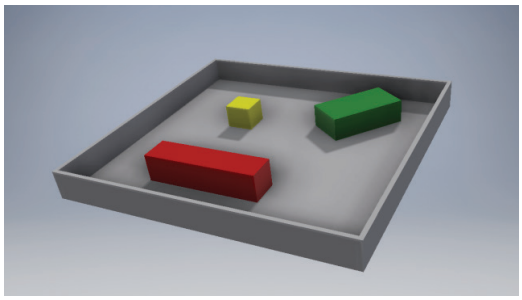

Fig. 11. Render of the board

Fig. 12. Assembled board

3. Design of Control Algorithms

The purpose of each obstacle avoidance algorithm is to analyze data coming from sensors and, based on that, adjust the direction of vehicle's movement so the robot remains on a safe, non-colliding path. Changing the vehicle's movement is achieved by changing speeds of both motors.

Each motor is controlled by a PWM signal. Changing its speed is obtained by changing the duty cycle (the higher the duty cycle, the higher the speed). Although motors that were used are exactly the same model, produced by the same company, they reach different angular speeds with the same control signal applied. This issue causes the vehicle to move on a curve instead of a straight line. Hence, creating a speed controller that would help reach and maintain a desired speed becomes essential. A classical PI controller has been arbitrarily chosen for this purpose. It allows for a short settling time and eliminates the steady state error [6]. The derivative term has not been used in order to avoid disturbance amplification which would have a significant effect on control quality, especially when considering the low resolution of encoders. Coefficients of the controller have been chosen experimentally.

The structure of controlling vehicles has been split into two modules (see Figure 13). A high-level module is a currently active obstacle avoidance algorithm which is, among others, setting speeds for both motors. A low-level module is a PI speed controller, which is responsible for reaching and maintaining a certain speed, set by the other module.

where Vset refers to velocity set point, V is the current vehicle's velocity and V* is velocity measured by equipment. Signals e, u and ω refer to error, control signal and motor's angular speed respectively. Z is the disturbance signal.

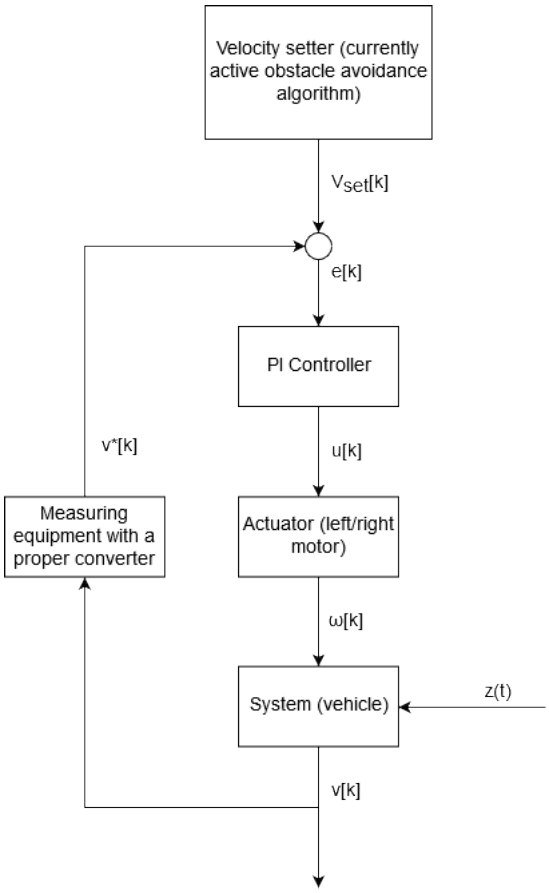

Fig. 13. Structure of vehicles' control

Two obstacle avoidance algorithms were implemented and tested. The first one is based on a simple set of rules. A vehicle can be in one of three states available – "Move forward", "Stop" or "Turn right". Transitions between these states are determined by current sensor readings, whether or not any obstacle has been detected within the sensors' range. This algorithm has been called the Rule Set [7]. An illustration depicting its functioning is presented in Figure 14.

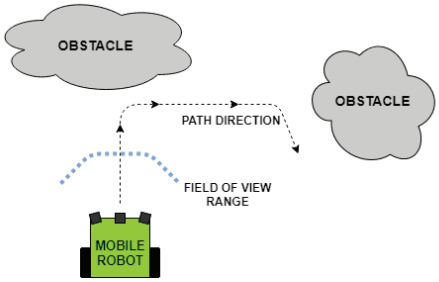

Fig. 14. Rule Set principle

The second algorithm is based on searching new path by calculating the derivative of arithmetic mean of measured distances. When an obstacle is detected within the specified range, the vehicle starts turning around, scanning the environment and calculating the difference between current and previous readings' mean. The scanning is finished when the derivative is small enough which means that the longest path without any obstacles has been found.

In an ideal situation the longest path would be found if the derivative was equal to 0. However, in practical terms it is impossible due to the noise coming from sensors and floating point precision.

It is also worth mentioning that, from the mathematical perspective, the path that has been found is a local maximum but not necessarily a global one. This approach may not find the longest possible path, but works significantly faster and adds smoothness to the vehicle's movement. The discussed algorithm has been called the Derivative Search. An illustration depicting its functioning is presented in Figure 15.

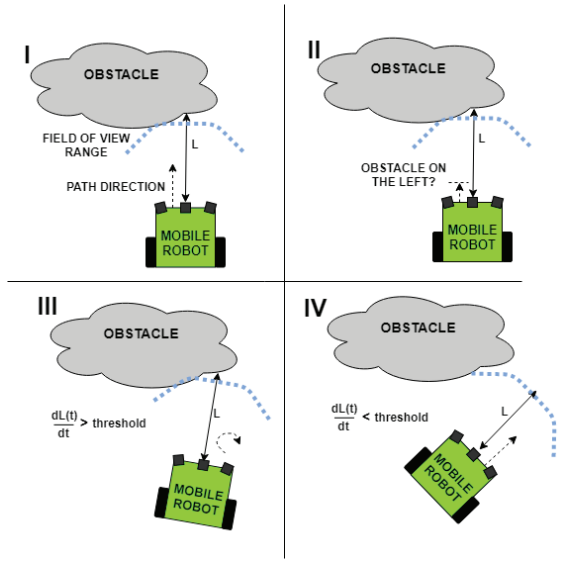

Fig. 15. Derivative Search principle

4. Software Development

Software itself is a key part of the presented work. It includes many aspects such as choosing the right platform and Software Development Kit (SDK) related to it, developing a unified Application Programming Interface (API) which makes teamwork much more comfortable, implementing obstacle avoidance algorithms and implementing communication via Bluetooth.

Arduino has been chosen as the platform [8]. It's a widely used, well documented platform, supporting many different microcontrollers and has a large base of libraries, simplifying usage of various peripherals. Arduino platform allows to write code in both C and C++ languages, which means it supports usage of Object Oriented Programming [9].

When it comes to IDE, one could potentially use Arduino IDE, but despite its advantages and simplicity of use it also has its flaws, making code management harder, especially in more advanced projects. That's why it was decided to use Atom with PlatformIO extension [10]. It is an open-source project, developed by the community which supports microcontrollers based on the Arduino platform. Contrary to Arduino IDE, it allows to create a complex project tree, provides code-hints and autocompletion and supports Git source control [11].

Concerning implementation of obstacle avoidance algorithms, two most important peripherals are motors and sensors. An API has been created for both of these. Each peripheral has its own, dedicated class, located in a separate header file. A class exposes a public interface containing only a few methods with self-explanatory names, therefore, introducing a new level of abstraction. All low-level logic is encapsulated and contains actions such as configuration of proper physical ports, receiving or transmitting electrical signals on corresponding ports. Public interfaces of these classes have been presented on Listings 1a and 1b.

Listing 1. Public interfaces responsible for:
sensors

```
void Begin(byte pin);
int GetVoltage();
float GetDistance();
```

motors

```
void Begin(byte colour, byte side);
void SetPWM(int duty);
long GetEncoderTicks();
```

Used sensors have nonlinear voltage-distance characteristics. In order to achieve the distance measured given in the SI units, the approximation of characteristics has been made, with the following model [12]:

$$L = a * \exp(b * u[k]) + c * \exp(d * u[k]) \qquad (1)$$

where L is the distance given in centimeters, u[k] is an output signal from A/D converter and a, b, c, d are coefficients.

Coefficients of this model were calculated using MATLAB's Optimization Toolbox and are equal to: a = 71.83; b = -0.016; c = 16.99; d = -0.00283.

In order to compare averaged data collected with the model, a standard deviation of error has been calculated. Its value was equal to 0.0136 which has been considered satisfying. A visual comparison between the mathematical model and averaged data collected from sensors is presented in the Figure 16.

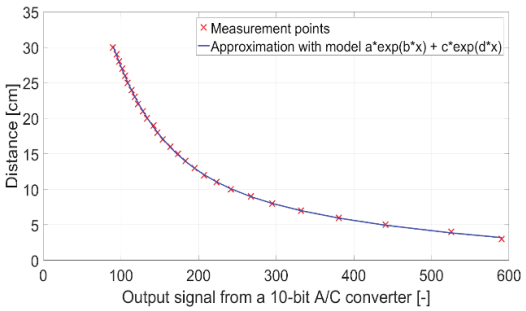

Fig. 16. The curve fitted to measurement points

After finishing the fundamental elements of a software, the next step was to define the program's core structure. The main goal was to create software architecture that would allow to introduce separation

of concerns as well as to help write clean, readable and maintainable code and make it more extensible. In particular, implementing a new obstacle avoidance algorithm should have no impact on existing and already tested code.

In order to separate concerns and meet the Single Responsibility Principle, a structure called the Result has been created [13]. This structure is created and returned by current algorithm in its every iteration and contains information about desired motor speeds and Light Emitting Diodes (LED) states.

Furthermore, to meet the extensibility requirement, an abstract class, called ObstacleAvoidanceAlgorithm has been created. This is a base class for all other classes that implement the algorithms. It contains two abstract methods (Run and SetLEDs) that define the functionality of a specific algorithm as well as implementation of all common features for algorithms. In addition, each algorithm has its field of view range and hysteresis defined in order to reduce the impact of sensors' noise. Their values can be changed via methods that are meant to be used by supervisory control. The ObstacleAvoidanceAlgorithm's code can be seen in Listing 2.

Listing 2. A header file containing definition of the ObstacleAvoidanceAlgorithm class

```
protected:
 bool obstacleDetected;
 float hysteresis;
 float boundary;
 float sensorReadings[3];
 bool checkForObstacles();
 virtual void setLEDs(Result &r) = 0;
public:
 ObstacleAvoidanceAlgorithm();
 void GetDistances(float left,
   float middle, float right);
 void SetBoundary(float newBoundary);
 void SetHysteresis(floatnewHysteresis);
 float GetBoundary();
 float GetHysteresis();
 virtual Result Run() = 0;
```

With that being done, a clean architecture is established. Implementing a new algorithm has absolutely no impact on existing code. Moreover, it opens up an opportunity to switch between various algorithms without shutting down or even stopping the vehicle – a sort of "hot swapping".

The procedure of adding a completely new obstacle avoidance algorithm looks as follows:
Create a new class and put it in a separate file,
Make this class inherit from ObstacleAvoidanceAlgorithm,
Override two abstract methods from the base class,
Create any number of fields and helper methods needed.

Points 3 and 4 can be reordered or performed in parallel, depending on the programmer's approach.

Two algorithms described in the previous section, that is Rule Set and Derivative Search have been implemented in respect to that manner. Their flowcharts are presented in Figures 17-18.

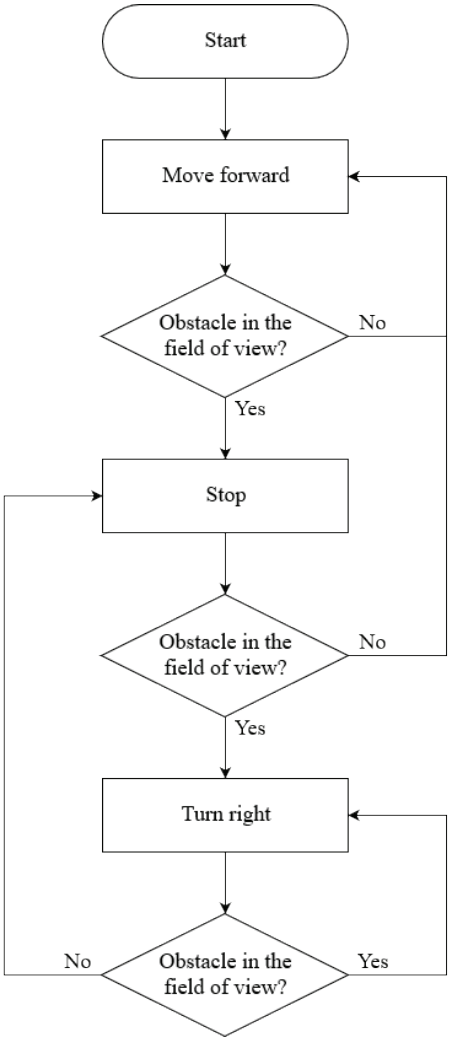

Fig. 17. Flowchart of the Rule Set algorithm

Apart from that, the structure of the main program's loop has been established. It contains two parts – the first one is responsible for executing the algorithm, setting speeds of motors using PI controller and flashing diodes. The second one takes care of communication via Bluetooth, which in turn is discussed further in this section.

First of all, to make communication via Bluetooth possible, an application that would allow user to interact with robots and implemented program was needed. The decision was made to make an application for mobile devices with Android as their operating system.

The MIT App Inventor has been chosen as the environment because of its accessibility and simplicity [14]. It was created by employees and students of MIT in order to make it possible for everybody to easily create applications for Android devices. It allows for programming an application and designing its graphical interface by simply pulling over certain parts of the interface or programming functions from a spe-

cially provided toolbox. The final look of the application is presented in the Figure 19.

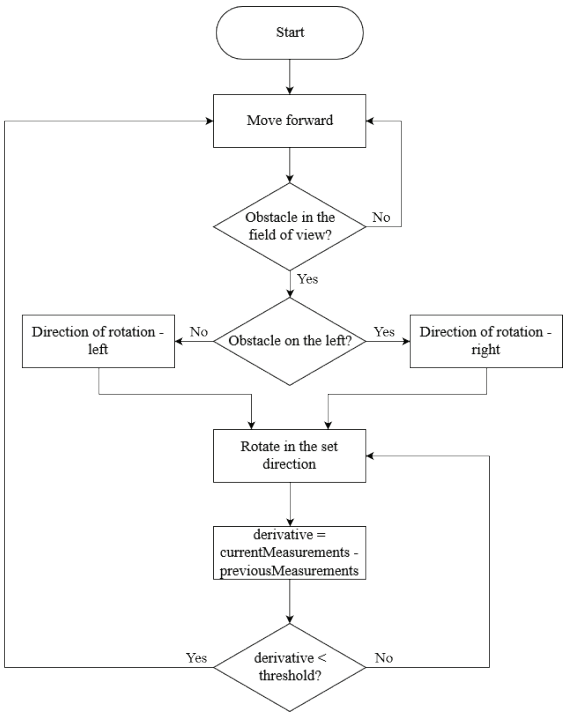

Fig. 18. Flowchart of the Derivative Search algorithm

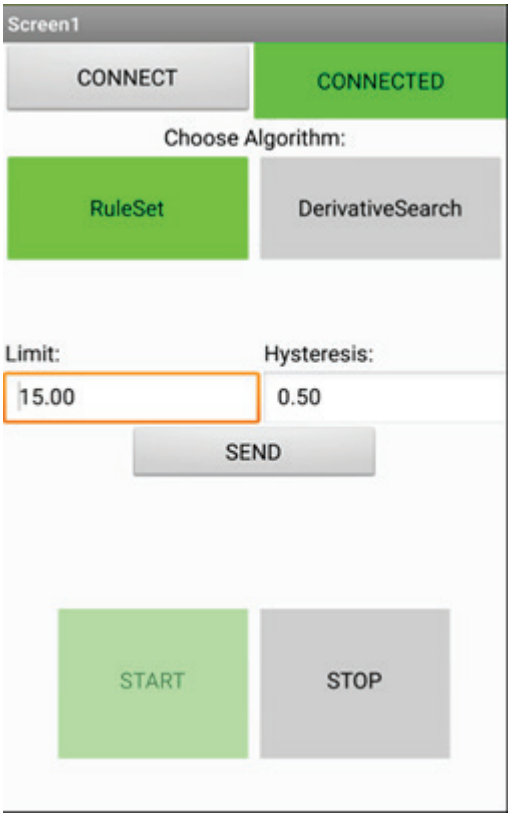

Fig. 19. Graphical interface of the mobile application

The application allows for connecting to one robot at a time by tapping the CONNECT button. It also shows information about connection status in the top right corner. After establishing a connection the application sends a certain set of messages that allows it to download data from the connected robot about its currently running algorithm and its parameters, as well as the state in which robot is. Two buttons are located below which allow to change running algorithm and inform the user about the one being currently used by highlighting its background green. Beneath, two textboxes are located that allow user to input the field of view range and hysteresis values. The input is being taken from a numerical keyboard to avoid encountering an error with the data type provided. Furthermore, in case of providing field of view range larger than 30 cm or below 4 cm, which are limits dictated by sensors used, the application sets this value automatically to maximum or minimum values respectively. Moreover, it is worth mentioning that sending values of parameters to the robot is only possible when any of the provided values have changed from the ones already in the robot's program. On the bottom of the screen buttons responsible for turning the robot on and off are placed.

Moreover, further enhancements to process messages coming via Bluetooth were added to the created application. However, in consideration of eventual future development of the project it was made in a manner that other ways of communication would work, without any changes in the software, despite Bluetooth being the only one possible for the time being.

In order to achieve this, a new class named Communication was created. It consists of a set of generic methods which allow for two-way communication patterns, essential for supervisory control. This class internally uses Stream pointer which supports any stream-based media e.g.: Wi-Fi, Ethernet, USB, as well as many others. This issue is resolved by Dependency Injection. Due to the specific structure of programs based on Arduino platforms a setter injection type has been used. The class Communication provides Begin method which injects the dependency that is a concrete implementation of a certain communication object. The class' header file is presented on a Listing 3.

Listing 3. Header file containing definition of the Communication class

```
private:
 Stream *stream;
public:
 String ReadMessage();
 bool Available();
 void SendMessage(String msg);
 void Begin(Stream *str);
```

5. Verification Tests

A series of verification tests was performed for each of the algorithms. Tests were carried out in an environment with static, dynamic or both types of obstacles in order to measure the quality of implemented algorithms.

In an environment with only static obstacles three boxes were used, two cuboid ones as well as one in

the shape of a cylinder. For both algorithms the results of this test were positive. In case of RuleSet algorithm collisions with obstacles were sporadic, occurring when turning or were caused by wheels that are protruding out of general shape of a robot. Changing values of the range of field of view and its hysteresis for this algorithm barely affected the performance of the mobile vehicles. Lowering the values resulted in slightly fewer collisions. On the other hand, for DerivativeSearch algorithm, changing values of parameters caused a significant difference. In narrow spaces robots used to zigzag a lot while lowering the field of view range resulted in moving more smoothly. In addition, with high values set, normal collisions happened to occur more often than with lower values.

As of environment with dynamic obstacles the mobile robots themselves were obstacles for each other. In this case, results of conducted tests were also positive. The functioning of both algorithms was satisfying. Changing values of parameters impacted only the DerivativeSearch algorithm, where again, lowering the range of field of view influenced the smoothness of movement of vehicles.

The last type of environment was nearly identical to the static one with the only difference being two robots running at the same time instead of one. Results of tests for both algorithms were positive. In case of RuleSet algorithm collisions only occurred with static obstacles for the very same reasons as described in previous paragraphs, no collision occurred between the two robots. However, in case of DerivativeSearch algorithm, collisions happened with static as well as dynamic obstacles. Same as before, lowering the range of field of view resulted in better performance.

It is worth to mention that each environment was limited by board's walls to ensure testing in the same conditions. The arrangement of obstacles was identical throughout all series of tests that were performed. The tests were 10 minutes long each and consisted of gathering data about performance of robots after changing currently running algorithm as well as parameters which are field of view range and hysteresis.

Tab. 1. A comparison between two algorithms

Algorithm	Type of obstacles	Parameter	
		Voltage drop on a battery over 10 minutes of work [V]	Number of collisions occurred during tests [-]
Rule Set	Static	0.11	5
	Dynamic	0.11	0
	Mixed	0.10	11
Derivative Search	Static	0.10	6
	Dynamic	0.11	5
	Mixed	0.11	11

To compare both algorithms two conditions were taken into consideration. First of them being number of collisions that occurred during tests and the second one being the energy consumption over time of the test. All of these results are presented in Table 1.

6. Conclusions

The implemented obstacle avoidance algorithms perform satisfyingly. However, limitations resulting from the used hardware and precision of sensors do not allow for flawless results.

In case of RuleSet algorithm it performed well in overall, despite its simplicity. Continuity of movement is not preserved due to the way of handling the avoidance of obstacles by the algorithm. Furthermore, changing values of field of view range and its hysteresis affected the performance of the robot. Moreover, it is worth to mention that the vehicle with RuleSet algorithm implemented did not use a lot of energy. The best environment for this algorithm was the one with dynamic obstacles in consideration of number of collision occurring during tests.

The second algorithm, DerivativeSearch, is more complex in the way it works. Due to this fact, changing the range of field of view and its hysteresis affected its performance. The only problems for robots running this algorithm occurred in narrow spaces when obstacles were closer than the provided value of field of view range and its hysteresis, which resulted in zigzagging and stopping. However, changing values of these parameters affected the performance of this algorithm resulting in a smoother movement under these circumstances. The energy drain of this algorithm was the same as when running the RuleSet algorithm. Furthermore, once again the environment where this algorithm performed best was the environment with dynamic obstacles.

A lot of different factors affect the performance of obstacle avoidance algorithms, one of them being precision of sensors. Both algorithms performed worse in an environment with static and dynamic obstacles at the same time, mostly due to high density of obstacles on the board. The main reasons of collisions occurring during tests where wheels protruding out of the general shape of the robot as well as a narrow field of view. Moreover, when the vehicle was turning and an obstacle appeared in its field of view closer than 4 cm it was not detected due to the precision of sensors.

The loosely-coupled, clean and maintainable code architecture makes further extensions fairly easy and does not impact the existing code. Thanks to that, future works may include implementing additional obstacle avoidance algorithms that are more sophisticated and require more computational power such as the Curvature-Velocity Method (CVM) and the Vector Field Histogram (VFH) described in [15] and [16]. Further enhancements may also include hardware changes, mainly adding more sensors in order to increase the view angle, which will improve control

quality. They may also include a whole redesign of the board in order to shift the center of mass closer to the motors' axis. This would result in better stability of the vehicle, allowing it to take sharper turns at higher speeds.

Hardware improvements may also include adding peripherals to support various communication media e.g. USB or Wi-Fi. Thanks to the unified software interface that has been developed, any of these peripherals can be used and swapped at any time with all software remaining intact.

AUTHORS

Robert Piotrowski – Gdańsk University of Technology, Faculty of Electrical and Control Engineering, email: robert.piotrowski@pg.edu.pl.

Bartosz Maciąg – Gdańsk University of Technology, Faculty of Electrical and Control Engineering, email: maciagbartosz96@gmail.com.

Wojciech Makohoń – Gdańsk University of Technology, Faculty of Electrical and Control Engineering, email: wojtek.makohon@gmail.com.

Krzysztof Milewski* – Gdańsk University of Technology, Faculty of Electrical and Control Engineering, email: milewskik51@gmail.com.

*Corresponding author

REFERENCES

[1] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots". In: *1985 IEEE International Conference on Robotics and Automation Proceedings*, vol. 2, 1985, 500–505
DOI: 10.1109/ROBOT.1985.1087247.

[2] P. Szulczyński, D. Pazderski, K. Kozłowski, "Real-time obstacle avoidance using harmonic potential functions", *Journal of Automation, Mobile Robotics and Intelligent Systems*, vol. 5, no. 3, 2011, 59–66.

[3] I. Ulrich, F. Mondada, J. Nicoud, "Autonomous vacuum cleaner", *Robotics and Autonomous Systems*, vol. 19, no. 3, 1997, 233–245
DOI: 10.1016/S0921-8890(96)00053-X.

[4] J. P. Grotzinger, "Analysis of Surface Materials by the Curiosity Mars Rover", *Science*, vol. 341, no. 6153, 2013, 1475–1475
DOI: 10.1126/science.1244258.

[5] J. Li, H. Liu, "Design Optimization of Amazon Robotics", *Automation, Control and Intelligent Systems*, vol. 4, no. 2, 2016
DOI: 10.11648/j.acis.20160402.17.

[6] U. Tochukwu, "Effects of PID Controller on a Closed Loop Feedback System". In: *Proceedings of Conference: Ternopil National Technical University. Ukraine*, 2013
DOI: 10.13140/2.1.2650.0167.

[7] G. Narvydas, R. Simutis, V. Raudonis, "Autonomous Mobile Robot Control Using IF-THEN Rules and Genetic Algorithm", *Information Technology And Control*, vol. 37, no. 3, 2008.

[8] L. Louis, "Working Principle of Arduino and Using It as a Tool for Study and Research", *International Journal of Control, Automation and Systems*, vol. 1, no. 2, 2016, 21–29
DOI: 10.5121/ijcacs.2016.1203.

[9] A. Urdhwareshe, "Object-Oriented Programming and its Concepts", *International Journal of Innovation and Scientific Research*, vol. 26, no. 1, 2016, 1–6.

[10] "PlatformIO Documentation, Release 4.2.1". PlatformIO, https://buildmedia.readthedocs.org/media/pdf/platformio/stable/platformio.pdf. Accessed on: 2020-01-20.

[11] N. N. Zolkifli, A. Ngah, A. Deraman, "Version Control System: A Review", *Procedia Computer Science*, vol. 135, 2018, 408–415
DOI: 10.1016/j.procs.2018.08.191.

[12] H. J. Motulsky, L. A. Ransnas, "Fitting curves to data using nonlinear regression: a practical and nonmathematical review", *The FASEB Journal*, vol. 1, no. 5, 1987, 365–374
DOI: 10.1096/fasebj.1.5.3315805.

[13] D. Riehle, H. Züllighoven, "Understanding and using patterns in software development", *Theory and Practice of Object Systems*, vol. 2, no. 1, 1996, 3–13
DOI: 10.1002/(SICI)1096-9942(1996)2:1<3::AID-TAPO1>3.0.CO;2-#.

[14] K. Prutz, H. Abelson, "Expanding Device Functionality for the MIT App Inventor IoT Embedded Companion", *Term paper,* Massachusetts Institute of Technology, May 17, 2018.

[15] R. Simmons, "The Curvature-Velocity Method for Local Obstacle Avoidance". In: *Proc. of the IEEE International Conference on Robotics and Automation*, 1996, 3375–3382.

[16] J. Borenstein, Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots", *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, 1991, 278–288
DOI: 10.1109/70.88137.