Robot Obstacle Avoidance Using Bumper Event

Neerendra Kumar*, Zoltán Vámossy **, Zsolt Miklós Szabó-Resch**
* John von Neumann Faculty of Informatics, Obuda University, Budapest, Hungary
* Department of Computer Science & IT, Central University of Jammu, India
** John von Neumann Faculty of Informatics, Obuda University, Budapest, Hungary
e-mail: * neerendraiimt@gmail.com, ** vamossy.zoltan@nik.uni-obuda.hu, szabo.zsolt@nik.uni-obuda.hu

Abstract—**In this paper we have developed a simple algorithm for robot obstacle avoidance using bumper event. This algorithm is implemented on Turtlebot (a robot model) in the Gazebo simulator and on a real Turtlebot robot. The bumper and state fields of robot's bumper event are studied for the different command velocities. The experimental results are shown by Mat plots using ROS (robot operating system) tool 'rqt'.**

I. INTRODUCTION

In robot navigation, obstacle avoidance is a key task. Obstacle avoidance in unknown environment is much more complicated than in known environment. Since the map containing the information about the obstacles and their respective positions cannot be provided to robot for the navigation in the former case. Therefore, in the case of unknown environment, the robot needs to avoid the obstacles by using its inbuilt sensors. Recently, many researchers considered the robot obstacle avoidance in their study including [1–4]. The SLAM (simultaneous localization and mapping) and path planning are necessary for the autonomous navigation process [5]. The obstacles can be treated as integrated part of the environment. The velocities of moving obstacles can be ignored for collision-free navigation of the robot [6]. Human-like approaches to avoid the obstacles can be used in robot navigation [7]. An optimal and collision-free path can be achieved by Bacterial Potential Field method as given in reference [8]. In the environments with moving and deforming obstacles, a purely reactive algorithm to navigate a mobile robot mathematically guarantees collisions avoidance [9]. Neural networks have the ability to work with imprecise information and are excellent tools applicable in obstacles avoidance by mobile robots. Reliable and fault-tolerant control can be obtained with neural control systems [10]. Efficient artificial bee colony (EABC) algorithm is effective for obstacle avoidance in multiple mobile robots system [11].

ROS (Robot Operating System) is an efficient software library which provides many useful tools and packages for the robotic programming. ROS is an open-source and available for free. As a result, ROS is widely used in robotic research and development thought the globe. Starting from oldest to newest, nine versions of ROS, released so far, are 'ROS Box Turtle' (2010), 'ROS C Turtle' (2010), 'ROS Diamondback' (2011), 'ROS Electric Emys' (2011), 'ROS Fuerte Turtle' (2012), 'ROS Groovy Galapagos' (2012), 'ROS Hydro Medusa' (2013), 'ROS Indigo Igloo' (2014), 'ROS Jade Turtle' (2015). The latest version of ROS, 'ROS Jade Turtle', is not recommended for the stability or long-term service. Therefore, for long-term service, 'ROS Indigo Igloo' is a good option. For the present study, we are using 'ROS Indigo Igloo' on Ubuntu operating system. Modern control theory and robotics are advancing greatly due the development of new technologies [12]. The combination of several sensor systems is used in mobile robots. For making navigation decisions, Sensor fusion is the task of combining the information into a usable form. Hybrid navigation system, combining the perception and dead reckoning, provides satisfactory results [13]. The complex task of programming is preventing the use of industrial robot at large extent [14]. In addition, there may be situations in the autonomous navigation in an unknown environment [15, 16] that some of the sensors of the robot are not working properly or removed to minimize the programming and system complexity. In this paper, we are considering that the bump sensor is the only sensor to handle the obstacles.

The rest of the paper is organized as follows: Section II is about the related basic concepts. In Section III, the proposed algorithm is given. Experimental verification and results are presented in Section IV. Section V concludes the paper. Section V is followed by acknowledgment and references.

II. RELATED KNOWLEDGE

A. Turtlebot and Gazebo Simulator

Turtlebot is a wheeled mobile robot. This robot kit is useful for the researcher due to its low cost and open source. The main hardware components of Turtlebot are 'Turtlebot Structure', 'Turtlebot Module Plate', 'Kinect', 'ROS compatible Netbook', 'Asus Xion Pro Live' and 'Kobuki Base'. In addition to the software development kit (SDK) the robotic software development environment of the Turtlebot provides libraries for the several useful tools like visualization, control and error handling. To bring up the Turtlebot in the real system, the ROS command '$ roslaunch turtlebot_bringup minimal.launch' can be used in the Ubuntu (Operating System) terminal.

Gazebo is a simulator which can be used for robot simulation. It can efficiently simulate and visualize the robotic acts in a three-dimensional environment. Robot simulation using different types of robot models in indoor and outdoor environment is possible in Gazebo. In this paper, Turtlebot as a robot model in Gazebo simulator is used. The ROS command '$ roslaunch turtlebot_gazebo turtlebot_world.launch' brings up the Turtlebot in Gazebo simulator. This simulated world can be edited and saved according to the requirements using the various available tools.

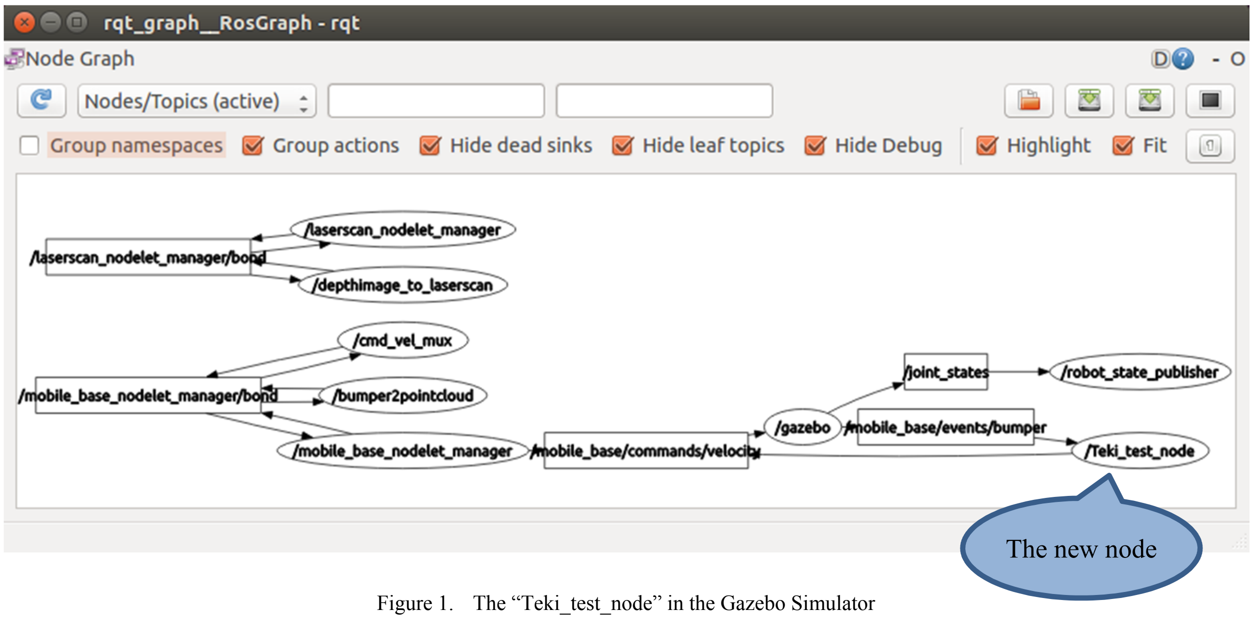

Figure 1. The "Teki_test_node" in the Gazebo Simulator

B. *Communication in ROS and RQT Tool*

The user defined nodes and packages can be created in the ROS. These nodes can communicate with each other by mean of message passing on the concerned topic. The user defined nodes can also be linked with the ROS to perform the desired operations on or by the robot by publishing or subscribing the available topics. Publisher-subscriber mechanism is used in the communication of any message on a topic in the ROS.

'rqt' is a cross-platform application framework of ROS. It implements the graphical user interface (GUI) tools. Some basic plugins of rqt package are rqt_bag, rqt_console, rqt_graph, rqt_logger_level, rqt_plot. 'rqt_bag' provides a GUI plugin for recording and managing bag files. 'rqt_console' provides a GUI plugin to display the messages being published to 'rosout' topic. The logger level of ROS nodes can be configured using 'rqt_logger_level'. 'rqt_graph' is used to find the ROS graph of the system. The 'rqt_plot' plugin construct the two-dimensional plots for the given ROS topics.

We have developed a node named as 'Teki_test_node' to implement the algorithm proposed in Section III of this paper. Using the rqt_graph, the node graph of the system with Gazebo simulator is presented in Fig. 1 and Fig. 2 shows the node graph of the system with real Turtlebot. In these node graphs our node, 'Teki_test_node', is pointed out by round call-outs separately. In the node graph, the nodes are represented by ellipses and the topics enclosed with rectangles. The arrow directed from topic to the node represent that the topic is subscribed by the node. On the other hand, the arrow pointing towards the topic from the node shows that the topic is published by the node.

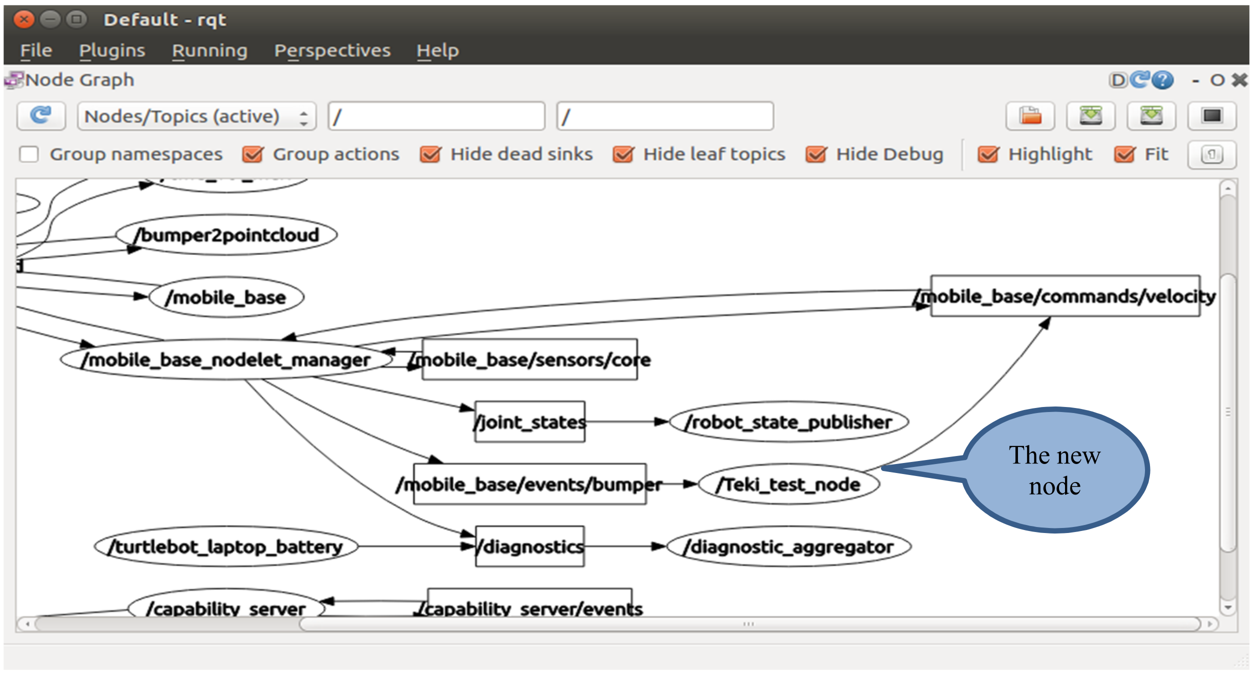

Figure 2. The "Teki_test_node" in the real Turtlebot system

III. ALGORITHM

The Turtlebot mobile base (Kobuki base) is equipped with a bump sensor. When the bump sensor is hit or pressed by some object, the bumper event is generated. There are two field in bumper event: bumper and state. Bumper field can have any value from LEFT(0), CENTER(1) or RIGHT(2) depending upon the corresponding bump sensor has been pressed. The state field can take the values RELEASED(0) or PRESSED(1). In ROS, there are two types of velocity commands: linear ans angular. The linear velocity moves the robot in straight line and the angular velocity is used to turn the robot. To minimize the complexity, if the robot does not have any sensor except the bump sensor to avoid the obstacles than the Turtlebot will collide the obstacles in its way. In this situation, the robot must recover from the collision and move to the other possible way. For this purpose, Fig. 3 presents the algorithm by a flowchart.

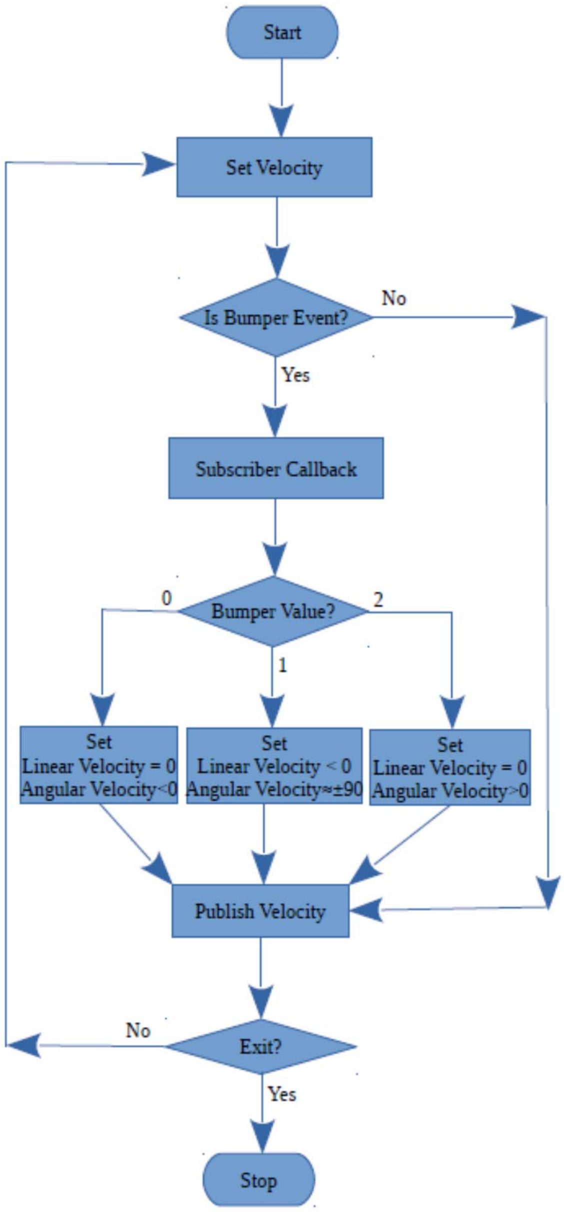
Start
Set Velocity
Is Bumper Event?
No
Yes
Subscriber Callback
Bumper Value?
0
1
2
Set Linear Velocity = 0 Angular Velocity<0
Set Linear Velocity < 0 Angular Velocity≈±90
Set Linear Velocity = 0 Angular Velocity>0
Publish Velocity
Exit?
No
Yes
Stop

Figure 2. Flowchart of the proposed algorithm

IV. IMPLEMENTATION AND EXPERIMENTAL RESULTS

The proposed algorithm is implemented in C++ code. The velocity commands to move the Turtlebot can be published in the form of geometry message: 'Twist' on the ROS topic '/mobile_base/commands/velocity'. Our node, Teki_test_node, is publisher on the topic '/mobile_base/commands/velocity'. The ROS command for the publisher is 'ros::Publisher pub=nh.advertise<geometry_msgs::Twist>("/mobile_base /commands/velocity",1000);' where 'pub' is user defined name of Publisher object, 'nh' is user defined node handle and 1000 is the user defined buffer size. The bumper event can be subscribed in the form of Kobuki message 'BumperEvent' using the ROS topic '/mobile_base/events/bumper'. The topic '/mobile_base/events/bumper' can be subscribed using the ROS command 'ros::Subscriber bumper = nh. Subscribe("/mobile_base/events/bumper", 1, callback);' where 'callback' is the user defined name of the callback function, bumper is user defined name of the object, 1 is user defined buffer size and 'nh' is the user given name of the node handle. The callback function of the subscriber can be called by using the ROS function 'ros::spinOnce()' or 'ros::spin();'. The 'ros::spin()' function blocks the execution of the program only for callback function call and in this situation the publisher related task cannot be executed outside the definition of the callback function. To publish the messages outside the callback function definition we can use 'ros::spinOnce()' for callback function call. In this study, we are using 'ros::spinOnce()' function for the callback function call so that we can publish the twist messages outside the callback function definition as well. Therefore, whenever the bumper event occurs the callback function is called once and then after the remaining code of the program continues to be executed. The twist messages for velocity commands are published at the rate of 10 messages per seconds. The *x*, *y* and *z* components of linear and angular velocities are taken as can be seen in Fig 4.

```
The default velocity is taken as:
  {
  linear.x=0.1, linear.y=0, linear.z=0
  angular.z=0, angular.x=0, angular.y=0
  }
If bumper 0 hit then:
  {
  linear.x=0, linear.y=0, linear.z=0
  angular.z=−0.35, angular.x=0, angular.y=0
  }
If bumper 1 hit then:
  {
  linear.x=−0.25, linear.y=0, linear.z=0
  angular.z=−1.25, angular.x=0, angular.y=0
  }
If bumper 2 hit then:
  {
  linear.x=0, linear.y=0, linear.z=0
  angular.z=0.35, angular.x=0, angular.y=0
  }
In the case of bumper hit the ROS sleep function for is used so
that the appropriate velocity command come into effect.
```

Figure 3. The x, y and z components of linear and angular velocities

A. With Gazebo Simulator

For Simulation in Gazebo, we have constructed a simple maze (only boundaries) in Gazebo environment by inserting four jersey barrier and a Turtlebot as shown in Fig. 5. The proposed algorithm is tested in this maze. For the future use, the maze can be saved in the computer using the '.world' extension. There the following two main steps to execute our node in the Gazebo:

Step 1: Bring up Turtlebot in our own Gazebo world using the ROS command '$ roslaunch turtlebot_gazebo turtlebot_world.launch world_file:=path' in a terminal. Here 'path' represents the path of our prior saved maze.

Step 2: In a new terminal execute the following series of commands:

~$ cd catkin_ws (to enter into the 'catkin_ws' directory)

~$ catkin_make (to compile our code)

~$ source devel/setup.bash (to set environment variables)

~$ rosrun nkt Teki_test_node (to execute our node, here 'nkt' is name of user created package)

The bumper hit events for right and center bumper are shown in Fig. 6. Further, Fig. 7 shows the left and center bumper hit events.

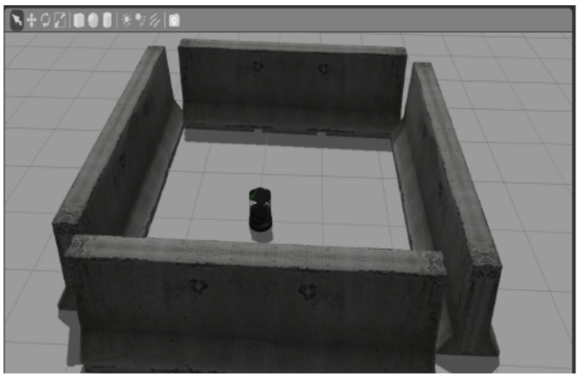
Figure 5. Introductory, simple maze developed in Gazebo simulator

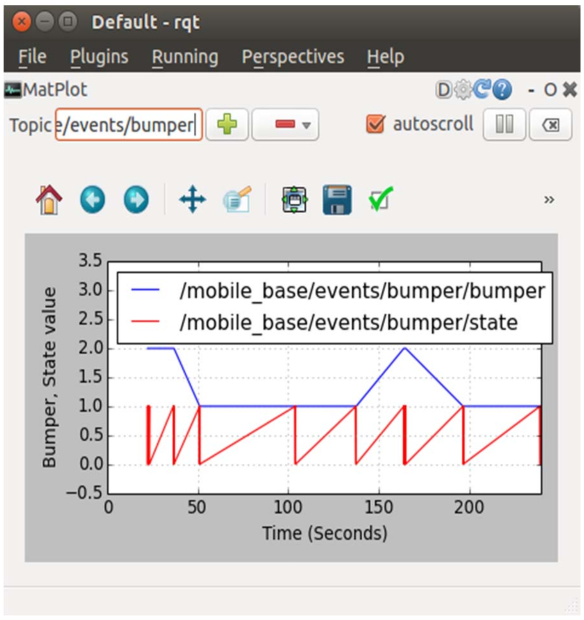

Figure 6. Right and center bumper hit events

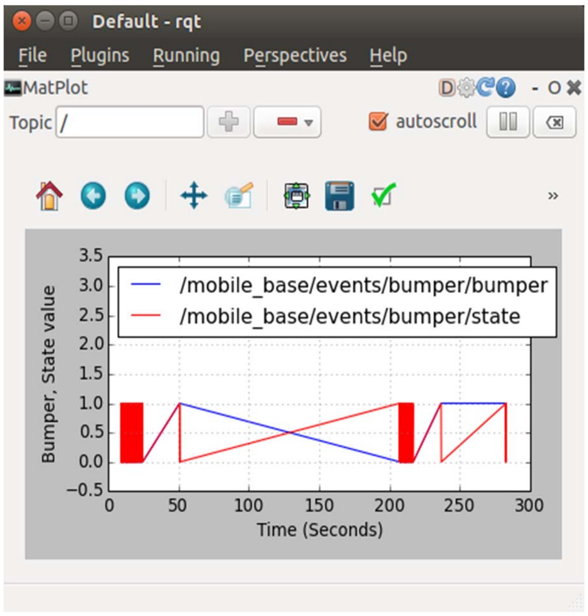

Figure 4. Left and center bumper hit events

B. With Real Turtlebot

The implementation of the algorithm with real Turtlebot is exactly same as in the case of Gazebo. The same C++ code is applicable to the real Turtlebot without any change. There the following two main steps to execute our node in the system with real robot:

Step 1: Bring up the real Turtlebot in the system using the ROS command '$ roslaunch turtlebot_bringup minimal.launch' in a terminal.

Step 2: In a new terminal execute the same series of commands as given above in step 2 of Gazebo simulator. However, if we are using the same terminal which we have used in step 2 with Gazebo then we will already be in the catkin working directory. Further, there is no change in the C++ code so we do not need to compile the code again. Furthermore, the environment variables are

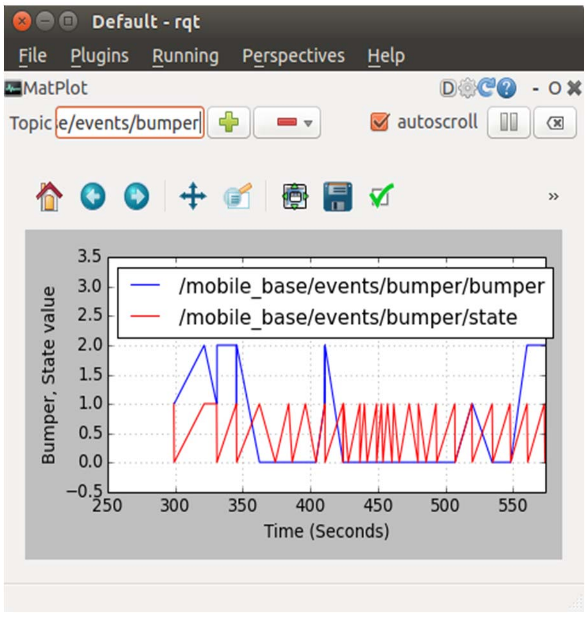

Figure 7. Bumper hit events by real Turtlebot

already set. Therefore, if we are using the same terminal in which we have executed our node previously then the only one command: '~$ rosrun nkt Teki_test_node' is sufficient where 'nkt' is the name of our package and 'Teki_test_node' is the name of our node.

Fig. 8 presents the bumper events generated with left, right and center bumper values.

V. Conclusion

The implementation of the new algorithm shows that the obstacle avoidance task can be handled by using the bumper events of the Turtlebot. The Turtlebot successfully recovers from the collisions and follows the new velocity commands. In case autonomous navigation in unknown environment, this algorithm is very useful when other sensors for obstacle avoidance are got damaged or removed to minimize the complexity.

The main disadvantage of the proposed algorithm is that this algorithm does not lead to a collision-free navigation. Therefore, the other sensors of robot like laser and camera may get priority over bump sensor for collision-free navigation. Consequently, the methods based on the sensors which lead to obstacle avoidance without collisions during navigation can be taken into consideration for the future work.

Acknowledgment

This work is a part of the TekiControl project which is an undergoing research project at John von Neumann Faculty of Informatics, Obuda University, Budapest, Hungary.

The authors would like to say thanks for the help of the Hungarian National Talent Program (NTP-HHTDK-0015-0036).

References

[1] C. Giannelli, D. Mugnaini and A. Sestini, “Path planning with obstacle avoidance by G1 PH quintic splines”, Computer-Aided Design, vol. 75–76, pp. 47-60, June 2016.

[2] A.S. Matveev, A.V. Savkin, M. Hoy and C. Wang, “Biologically-inspired algorithm for safe navigation of a wheeled robot among moving obstacles”, Safe Robot Navigation Among Moving and Steady Obstacles, pp. 161-184, 2016.

[3] M. Wang, J. Luo and U. Walter, “A non-linear model predictive controller with obstacle avoidance for a space robot”, Advances in Space Research, vol. 57, Issue 8, pp. 1737-1746, 15 April 2016.

[4] Y. Chen and J. Sun, “Distributed optimal control for multi-agent systems with obstacle avoidance”, Neurocomputing, vol. 173, Part 3, pp. 2014-2021, 15 January 2016.

[5] S . Wen, X. Chen, C. Ma, H.K. Lam, and S. Hua, “The Q -learning obstacle avoidance algorithm based on EKF-SLAM for NAO autonomous walking under unknown environments”, Robotics and Autonomous Systems, vol. 72, pp. 29–36, 2015.

[6] A.V. Savkin and C. Wang, “Seeking a path through the crowd: Robot navigation in unknown dynamic environments with moving obstacles based on an integrated environment representation”, Robotics and Autonomous Systems, vol. 62, pp. 1568–1580, 2014.

[7] T.D. Frank, T.D. Gifford and S. Chiangga, “Minimalistic model for navigation of mobile robots around obstacles based on complex-number calculus and inspired by human navigation behavior ” Mathematics and Computers in Simulation, vol. 97, pp. 108–122, 2014.

[8] O. Montiel, U. Orozco-Rosas and R. Sepúlveda, “Path planning for mobile robots using Bacterial Potential Field for avoiding static and dynamic obstacles”, Expert Systems with Applications, vol. 42, pp. 5177–5191, 2015.

[9] A.S. Matveev, M.C. Hoy and A.V. Savkin, “A globally converging algorithm for reactive robot navigation among moving and deforming obstacles”, Automatica, vol. 54, pp. 292–304, 2015.

[10] A. Medina-Santiago, J.L. Camas-Anzueto, J.A. Vazquez-Feijoo, H.R.H.-de León and R. Mota-Grajales, “Neural Control System in Obstacle Avoidance in Mobile Robots Using Ultrasonic Sensors ”, Applied Research and Technology, vol. 12, pp. 104-110, 2014.

[11] J.H., Liang and C.H. Lee, “Efficient collision-free path-planning of multiple mobile robots system using efficient artificial bee colony algorithm”, Advances in Engineering Software, vol. 79, pp. 47–56, 2015.

[12] K. Alisher, K. Alexander and B. Alexandr, “Control of the mobile robots with ROS in robotics courses”, Procedia Engineering, vol. 100, pp. 1475 – 1484, 2015.

[13] A.M. Zaki, O. Arafa and S.I. Amer, “Microcontroller-based mobile robot positioning and obstacle avoidance”, Electrical Systems and Information Technology, vol. 1, pp. 58–71, 2014.

[14] M. Ragaglia, A. M. Zanchettin, L. Bascetta and P. Rocco, “Accurate sensorless lead-through programming for lightweight robots in structured environments”, Robotics and Computer-Integrated Manufacturing, vol. 39, pp. 9-21, June 2016.

[15] D. Stojcsics, “Autonomous waypoint-based guidance methods for small size unmanned aerial vehicles”, Acta Polytechnica Hungarica, Vol. 11, No. 10. pp. 215–233, 2014

[16] Árpád Takács, Levente Kovács, Imre J Rudas, Radu-Emil Precup, Tamás Haidegger, Models for Force Control in Telesurgical Robot Systems, Acta Polytechnica hungarica, Vol 12:(8) pp. 95-114. 2015