TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

ACTA TECHNICA NAPOCENSIS

Series: Applied Mathematics, Mechanics, and Engineering
Vol. 64, Issue Special IV, December, 2021

DESIGN-CENTRIC OBSTACLE AVOIDANCE ALGORITHM FOR AN AUTONOMOUS MOBILE ROBOT AND ITS TESTING USING VIRTUAL PROTOTYPING TECHNOLOGIES

Stelian BRAD, Damaris Naomi DOLHA

Abstract: *Autonomous mobile robots are required in many applications, from industry to services, security, and defense. The capacity to navigate autonomously in a smooth way in environments with various obstacles is essential for adoption in the practice of these types of robotic systems. Smooth navigation and obstacle avoidance significantly depend on the right correlation between the obstacle avoidance algorithm and the robotic system design (type of sensors, sensor location, type of control, mechanical construction of the mobile platform). This paper investigates the possibility to design the obstacle avoidance algorithm from the perspective of the particular design of the robot. Early validation of the algorithm is a cost-effective approach. In this respect, this paper also introduces an architectural construct of various open-source technologies to test and validate the algorithm via a digital prototype (or digital twin) that embeds the physical properties of the real robot and of the obstacles within the simulation environment. Tests run in the virtual environment show that the proposed algorithm, embedded in a wider Simultaneous Localization and Mapping (SLAM) algorithm, is capable to ensure a smooth avoidance of obstacles. Results can be easily implemented in a physical mobile robot for intelligent autonomous navigation.*
Key words: *autonomous navigation, virtual prototyping, mobile robots, SLAM, obstacle avoidance, robot operating system, digital twin, robot simulation.*

1. INTRODUCTION

The global market of autonomous mobile robots (AMRs) is constantly growing due to their capacity to navigate in highly dynamic environments without human interventions [1]. There are many applications where these types of robots are implemented, from parts transfer in manufacturing plants, cleaning and logistics in hospitals, logistics in warehouses, to patrol and surveillance in ports, material carrying in defense, inspection and operations in agriculture, transportation in malls, logistics in airports, cleaning in public spaces and houses, search, and rescue [2]. In the last decade, many autonomous navigation and mapping algorithms have been proposed, but very few are related to the configuration of the robotic system.

One recent research, published in [3], indicates the role of LiDAR (light detection and ranging) configuration on motion accuracy. Using a nodding LiDAR configuration, combined with GPS waypoint, the robot navigates with a root mean squared error of 0.0778 m, which is 0.2% of the total travelled distance. This research does not consider other aspects of configuration, even if it highlights the relevance of configuration for the navigation system concerning to a given application (i.e., in this case, the data collection in the field on phenotyping of crops).

Paper [4] proposes a new positioning method based on multiple ultrasonic sensors for AMRs that can realize higher positioning accuracy. A measurement model is established for sensor configuration. Three time-of-flight signals are obtained from three ultrasonic sensors. Despite its merits to improve navigation accuracy, this paper does not consider the dynamics of the mechanical system and its influence on navigation accuracy.

A review paper that describes various navigation techniques for AMRs is referenced at [5]. After surveying about 1000 papers, it was found that the heuristic approaches (e.g., fuzzy logic, neural network, neuro-fuzzy, genetic algorithm, particle swarm optimization, ant colony optimization, and artificial immune system) provide the most suitable and effective results for AMR navigation (both target-reaching and obstacle-avoidance) in an unknown and dynamic environment. Using a heuristic approach, an AMR can navigate among obstacles without hitting them.

Regardless of the developments in the navigation and path planning algorithms, the accuracy of navigation strongly depends on the symbiosis between the mechanical design of the robot, types of sensors used for navigation, location of these sensors, and the control algorithms.

Our additional search in the Web of Science database for the combination of keywords "AMR" AND "mechanical design" AND "navigation accuracy" indicates two relevant titles [6], [7]. A closer look at the content of these papers reveals that they do not treat navigation accuracy in conjunction with the architectural design of the robotic system. Thus, we see here a window of opportunity for further investigations. In this line, we introduce in this paper a heuristic algorithm for obstacle avoidance that is aligned to a particular design of the mechanical structure of the robot.

2. METHODOLOGY

The key research question we investigate in this paper is "What algorithm to select or design for obstacle avoidance by an AMR that navigates in a dynamic environment, considering a given type of sensors and a given mechanical structure (both being imposed in the design specifications and considered as design constraints)?

A cost-effective approach, already practiced by researchers and engineers to investigate designs, is to model and simulate the system in a virtual environment where properties of the physical system are incorporated. The traditional path is to select and/or design more solutions and to test them on a virtual prototype that operates in a virtual environment. This is an empirical approach, which does not necessarily indicate the best solution, but rather the best one from the set of tested solutions.

In this paper, we consider a systematic design algorithm, called complex system design technique (CSDT) [8], which depicts an abstract problem into small steps of ideation, and uses systematic creativity techniques to tackle each step. In this paper, we consider TRIZ [9] and SAVE [10] methods for guiding the ideation process at the lower layers of the design's granularity.

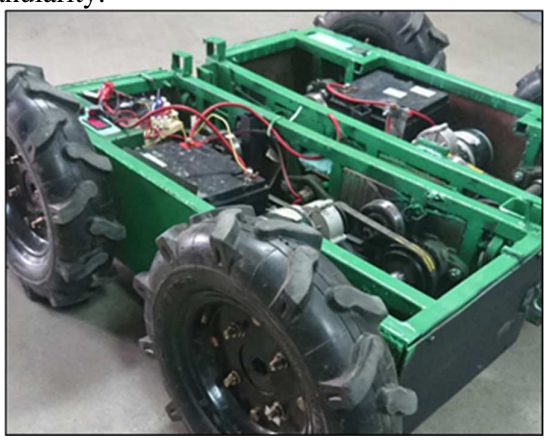

Fig. 1. The mobile robot(With courtesy: Dan Hăşmăşan)

3. THE MOBILE ROBOT PLATFORM

The mobile robot under investigation is illustrated in Figure 1. The CAD model of the robot, done in SolidWorks, is shown in Figure 2. It consists of two modules that are connected by a rotary axle. On each module, there are two wheels. Two motors are used to drive the robot, each motor being linked to one of the rear wheels. By combining the direction and rotational speed on each driving shaft, the mobile unit is driven forward, backward, or angular.

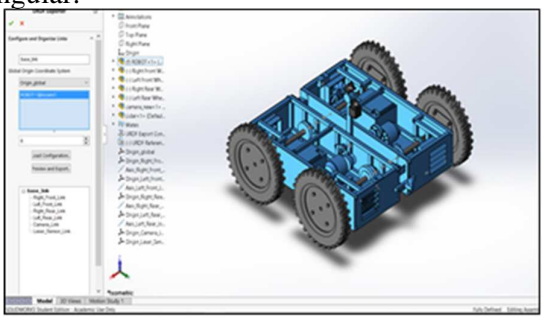

Fig. 2. The CAD model of the analyzed mobile robot

4. VIRTUAL PROTOTYPE SETTINGS

4.1. From the CAD model to the virtual prototype

The CAD model of the robot accurately reproduces the geometrical parameters of the physical robot. To create the virtual prototype of the robot, we used ROS (robot operating system) and the Gazebo platform. To define the kinematic and dynamic models of the robot in the virtual prototype, the CAD model from SolidWorks was converted into a URDF format (Unified Robotic Description Format) – a special XML file that is recognized by ROS.

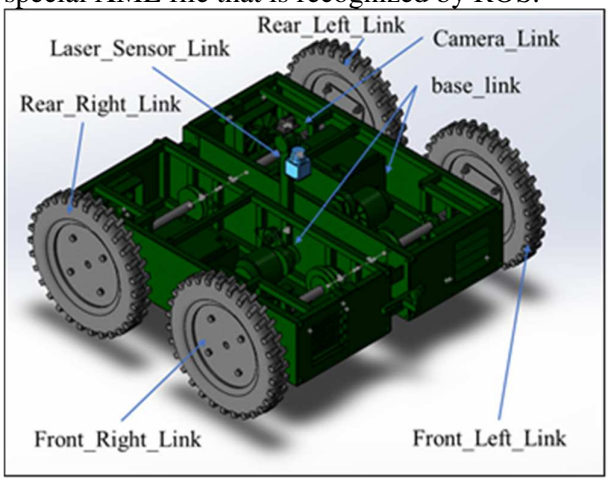

Fig. 3. Main modules for CAD model conversion into URDF to be recognized by ROS and Gazebo

To export the CAD model of the mobile robot as URDF and to spew it correctly inside the Gazebo simulator, it is mandatory to set it up properly by configuring the parent link of the assembly (the chassis and the components inside it, labelled as "base_link"), as well as its child links (the four wheels called "Front_Right_Link", "Front_Left_ Link", "Rear_Right_Link", "Rear_Left_Link", the camera ("Camera_Link") and the laser scanner ("Laser_Sensor_Link") (see Figure 3).

Each link, except for the "base_link", is defined by a joint name (that is being defined by the user), a reference coordinate system, a reference axis, and a joint type (continuous joints for wheels and fixed joints for camera and LiDAR sensor). The parent link is attached to the other links through a joint and the movements of the parent influence the motion of the connected links, which represent, in fact, the child links.

The robot is described through a collection of links that are connected by a set of joints. A rigid body with inertia, visual characteristics, and collision qualities is described by the link element, while its kinematics and dynamics are described by the joint element.

4.2. Consideration of the physical properties

For each element of the structure, physical properties are stored in a special XML file. The exemplification for the case of "base_link" element is shown in Figure 4.

```
<!--- Base link element -->
<!--- Attribute representing the name of the link -->
<link
  name="base_link">
  <!--- Inertial characteristics of the link -->
  <inertial>
    <!-- Mass value -->
    <mass
      value="64" />
    <!-- The 6 above-diagonal elements of the 3x3 symmetric rota
    <inertia
      ixx="5.0618"
      ixy="0.044213"
      ixz="-0.0034029"
      iyy="6.2021"
      iyz="-0.064867"
      izz="10.33" />
  </inertial>
  <!-- Visual properties of the link -->
  <visual>
    <geometry>
      <!-- Trimesh element -->
      <mesh
        filename="package://mobile_platform/meshes/base_link.STL
    </geometry>
    <material
      name="">
      <color
        rgba="0.79608 0.82353 0.93725 1" />
    </material>
  </visual>
  <!--- Collision properties of the link -->
  <collision>
    <geometry>
      <mesh
        filename="package://mobile_platform/meshes/base_link.STL
    </geometry>
  </collision>
</link>
```

Fig. 4. The XML code to describe properties for "base_link"

The <inertial> component (Figure 4) greatly influences the physics engine in the simulation platform (i.e., Gazebo in this case), where the mass of a link, defined in kilograms, must be bigger than zero, for Gazebo to not disregard it. Furthermore, in any finite torque application, links with the value of the principal moment of inertia that is equal to zero, might lead to an infinite acceleration.

Thus, we must consider these dynamic parameters. They are represented by the <inertia> property in Figure 4, which represents a 3x3 rotational matrix, defined by only six parameters, since the matrix is symmetrical: ixx, ixy, ixz, iyy, iyz, izz.

The <collision> property enhances the feature that allows the collision detection engine from Gazebo to better determine the borders of the object, along with the <visual> attributes.

Moreover, in the case of a robot with visual characteristics that are more complicated, the collision property is simplified by specifying a mesh file of the respective part of the assembly, inside the <mesh> element, which significantly improves the collision detection accuracy.

Gazebo plugins extend the capability of the URDF models by connecting ROS messages and service calls for sensors and motors. Different types of plugins are supported by Gazebo and all of them may be linked to ROS, but only a handful of them can be referenced via a URDF file, such as “ModelPlugins”, “SensorPlugins” and “Visual Plugins”.

Cameras and sensors in Gazebo are designed to be associated to links, therefore the <gazebo> element that describes them must include a link reference, to allow permission to use their APIs. In this case study we consider the Hokuyo LiDAR. For this LiDAR model, the <visualize> element emits a semi-translucent laser ray that is visible in the Gazebo simulator when it is true, within the scanning area of the gpu laser.

Other elements worth mentioning are <min_angle> and <max_angle>, that represent the angular scan range of the sensor, and the lowest and maximum distances at which the laser may operate (described by the <min> and <max> ranges). Increasing the value of the <noise> tag, enhances the volume of the feedback unit, depending on how realistic the simulation is required to look [11].

The noise is distributed in a Gaussian-style, with a standard deviation of 0.01 m. As a result, 99.7% of samples are within 0.03 m of the real reading, resulting in a +/- 30 mm accuracy for distances less than 10 m [3]. All this information, whatsoever, is specified either by the manufacturer or resulted from calibration tests.

4.3. 3D visualisation of robot simulation

Rviz or "ROS visualisation" is the 3D visualization graphical tool that incorporates plugins to visualize information provided by sensors, cameras, and other 3D gadgets, to generate an accurate representation of the surroundings of the robot. Rviz shows what the robot thinks is happening, while Gazebo shows what is really happening. Additionally, Rviz includes a Graphical User Interface (GUI) that offers the user the possibility to display data relevant to their project.

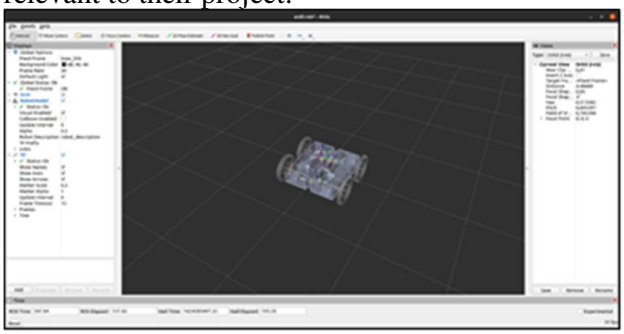

Fig. 5. Virtual robot visualization in Rviz

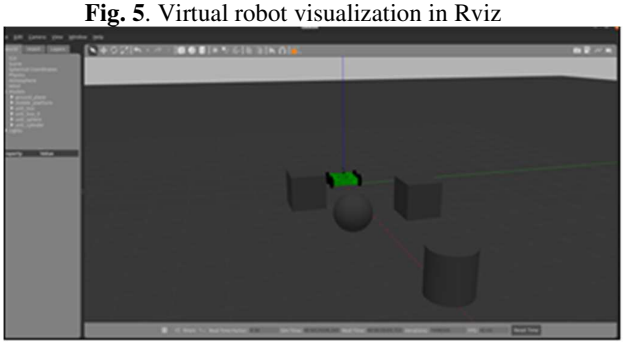

Fig. 6. Random obstacles in Rviz

¶ The virtual prototype for simulation of the mobile robot from this paper is shown in Figure 5. The random generation of obstacles in the working space of the robot is shown on Figure 6. The simulation environment includes specialized plugins for visualising the behaviour of cameras and LiDARs. For our case, this is shown in Figure 7.

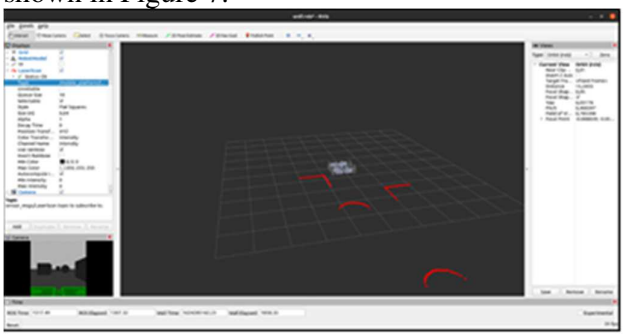

Fig. 7. Camera and LiDAR plugins in Rviz

To provide the robot with the ability to interact in the Gazebo simulator, as in to actuate the joints of the robot, simulated controllers must be included. These simulated controllers rely mostly on ROS, within the “ros_control” packages. “ros_control” receives joint status data and an input set point (target) and delivers the necessary commands to the actuators as an output. Controller manager, controller, transmission, and hardware interfaces, as well as

control toolbox, are all part of the “ros_control” package set.

4.4. Control in the virtual environment

To achieve control of the robot with the “ros_control” package, some additions have been integrated into the URDF, starting with the <transmission> element. This element represents a component of the URDF robot description model, that enhances the interaction between an actuator and a joint. Figure 8 depicts the “transmission” element between the joint of the right wheel and one of the two motors of the mobile platform.

```
<!-- Gazebo transmissions -->
<transmission name="Right_Front_Wheel_Transmission" type="SimpleTransmission">
  <!-- Specifies the transmission type -->
  <type>transmission_interface/SimpleTransmission</type>
  <!-- A joint the transmission is connected to -->
  <joint name="Right_Front_Joint">
    <!-- Specifies a supported joint-space hardware interface -->
    <hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
  </joint>
  <!-- An actuator the transmission is connected to -->
  <actuator name="Right_Front_Wheel_Motor">
    <!-- Specifies a supported joint-space hardware interface -->
    <hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
    <!-- Specifies a mechanical reduction at the joint/actuator transmission -->
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

Fig. 8. URDF transmission element of the right front wheel

In addition to the comments of this snippet of code, it is essential to discuss a few other aspects, including the “name” parameter inside the <joint> element, that must be related to one of the joints previously described inside the URDF.

Besides that, the “gazebo_ros_control” plugin receives information from the <hardwareInterface> parameter, within the <joint> and <actuator> tags, about which hardware interface to fetch (in this case, velocity interface).

The control of the robot wheels is distinct from the control techniques of sensors. For this job, an open-source ROS package is implemented. It is named “diff_drive_controller”. Velocity command is divided and transmitted to the wheels via a differential drive wheelbase. The differential drive controller package is set up using the joints and transmissions specified in a so-called Xacro file, as well as robot dimensions. In this respect, “geometry/Twist message” type is accepted by the “diff_drive_controller”, which describes velocities (linear and angular) along the x, y and z axes. The controller extracts both, the x component of the linear velocity and the z component of the angular velocity. Additionally, a velocity interface connects the controller to the wheel joints. The corresponding .yaml file containing the differential drive controller is detailed in Figure 9.

```
# Differential drive controller
mobile_base_diff_controller:
  type      : "diff_drive_controller/DiffDriveController"
  left_wheel  : ['Left_Front_Joint', 'Left_Rear_Joint']
  right_wheel : ['Right_Front_Joint', 'Right_Rear_Joint']
  publish_rate: 50.0  # default: 50

  # Odometry covariances for the encoder output of the robot
  pose_covariance_diagonal: [0.001, 0.001, 0.001, 0.001, 0.001, 0.03]
  twist_covariance_diagonal: [0.001, 0.001, 0.001, 0.001, 0.001, 0.03]

  # Wheel separation and diameter
  wheel_separation : 0.965
  wheel_radius : 0.206

  # Velocity commands timeout [s], default 0.5
  cmd_vel_timeout: 0.25

  # Top level frame (link) of the robot description
  base_frame_id: base_link

  # Odom frame_id
  odom_frame_id: odom

  # Publish the velocity command to be executed. It is to monitor the effect of
controller input
  publish_cmd: true

  # Setting this to true will allow more than one publisher on the input topic,
  allow_multiple_cmd_vel_publishers : true

  # The number of velocity samples to average together to compute the odometry
twist.angular.z velocities
  velocity_rolling_window_size : 6

  linear:
    x:
      # Whether the controller should limit linear speed or not
      has_velocity_limits    : true
      # Maximum linear velocity (in m/s)
      max_velocity           : 5.5  # m/s
      # Whether the controller should limit linear acceleration or not
      has_acceleration_limits: true
      # Maximum linear acceleration (in m/s^2)
      max_acceleration       : 1.2  # m/s^2
  angular:
    z:
      has_velocity_limits    : true
      max_velocity           : 5.5  # rad/s
      has_acceleration_limits: true
      max_acceleration       : 0.9  # rad/s^2
```

Fig. 9. Differential drive controller of the robot wheels

The term "differential" refers to the ability of the robot to shift direction by altering the relative rate of rotation of its wheels, without requiring extra steering motion.

4.5. Driving system modelling

The robot comprises two driving wheels, positioned on a shared axis, each of them being able to independently rotate clockwise or anticlockwise. To examine the point around which the robot can revolve, the ICC (Instantaneous Centre of Curvature) parameter is introduced (see Figure 10).

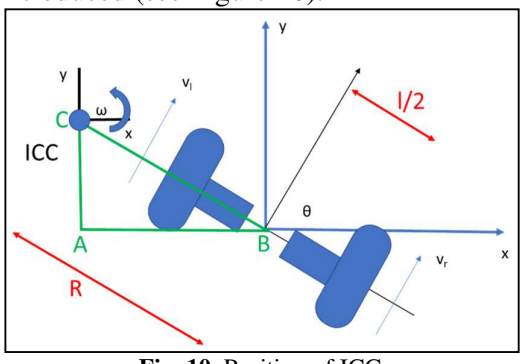

Fig. 10. Position of ICC

In Figure 10, V represents the velocity of the robot, R is the signed distance between the ICC

and the midway of the wheels, ω represents the angular velocity, V_r and V_l are the velocities of the right and left wheels, respectively, and l is the distance between the centres of the wheels.
Changing the velocities of the wheels imply different trajectories of the robot and because both wheels must rotate at the same rate of rotation (ω) around the ICC, we get the following relationships:

$$\omega = \frac{V_r - V_l}{l} \quad (1)$$

$$R = \frac{1}{2} \cdot \frac{V_r + V_l}{V_r - V_l} \quad (2)$$

Relationships (1) and (2) are essential for navigation and obstacle avoidance algorithms. When $V_r = V_l$, there is a forward linear motion in a straight line. R approaches infinite and the rotation is zero ($\omega = 0$).When $V_l = -V_r$, R is zero. It implies a rotation of the robot about the centre of the wheel axis, known as in-place rotation. When either $V_r = 0$ or $V_l = 0$, there is a rotation about the right wheel and left wheel, respectively.
The mobile robot platform meets certain restrictions, meaning that it can only move in the xy plane, having three degrees of freedom (DOF): orientation and position in two axes. Nevertheless, the robot only comes with two degrees of freedom that are controllable: speed and steering angle; the term for such a driving system is non-holonomic.
Using the rotation data of the wheels, forward kinematics are applied to compute the orientation and position of the robot. For this study, it was of interest to characterize the robot as being situated at a point in space defined by x and y coordinates, which moves towards a direction, creating an angle θ with the x axis. Assuming that the robot is in the middle of the wheel axle, it can move to different positions and orientations, by adjusting the control parameters V_r and V_l.
At the time ($t + \delta t$), the new position of the robot can be defined as a simple 2D rotation model, represented by a rotation matrix around ICC frame by $\omega\delta t$, multiplied by the pose of the robot in the ICC frame, to which the pose of ICC in the world coordinate system is added (see (3) and Figure 10).

(3)

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \cos(\omega\delta t) & -\sin(\omega\delta t) & 0 \\ \sin(\omega\delta t) & \cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{bmatrix}$$

In the equation (3), parameters determine the new positions of the robot in the world coordinate system, with the heading. Therefore, equation (3) depicts the motion of a robot rotating around its ICC, with an angular velocity of.
In ROS, the "geometry_msgs/Twist" message type is commonly used to transmit movement orders to the robot. The data saved in the motor driver node should then be used to control the motor.

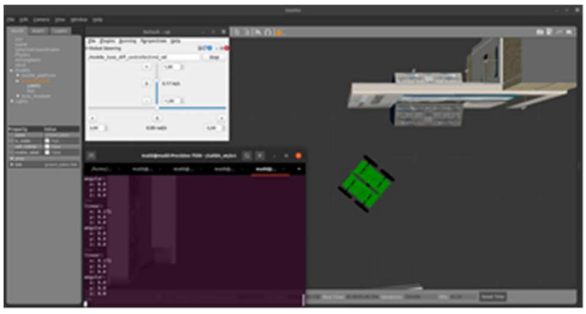

Fig. 11. Current linear and angular velocities displayed inside the terminal

(Figure 11 shows the x component of the linear speed vector and the z component of the angular speed vector used to operate the mobile_platform in the xy plane.

5. AUTONOMOUS NAVIGATION AND MAPPING

Autonomous mobile robot platforms can navigate in their surrounding environment and accomplish cognitive tasks on their own. Simultaneous localization and mapping is the key technology that allows the robot to estimate its own position using sensors, and create a map of the surrounding environment. These two tasks are interdependent.
The ROS Navigation package includes several navigation-related algorithms that may be used to quickly create autonomous navigation in mobile robots. The user must provide the Navigation package with the input consisting of the target location of the robot and the odometry data from sensors, while the output is represented by the velocity commands that will move the robot to the target position.

In ROS, Hector SLAM is a LiDAR-based, open-source package that can create 2D and 3D maps of an unknown environment, and simultaneously identifying the position of the robot. Its main modules are the following: “hector_slam”, “hector_ mapping”, “hector_map_server”, “hector_geotiff” and “hector_trajectory_server”. The module “hector_mapping” estimates the current 2D posture information of the mobile robot platform using the ultra-high scanning frequency of LiDAR. The path planning algorithm is shown in Figure 12.

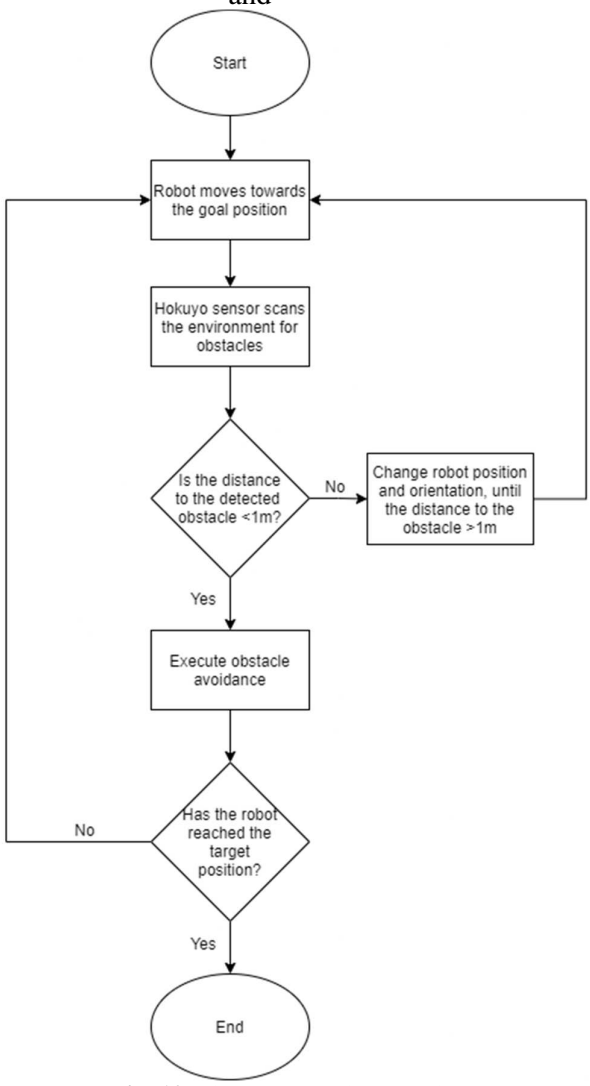

Fig. 12. The path planning algorithm

“RobotModel” and “LaserScan” plugins in Rviz, are used to generate the map, as it is seen in Figure 13.

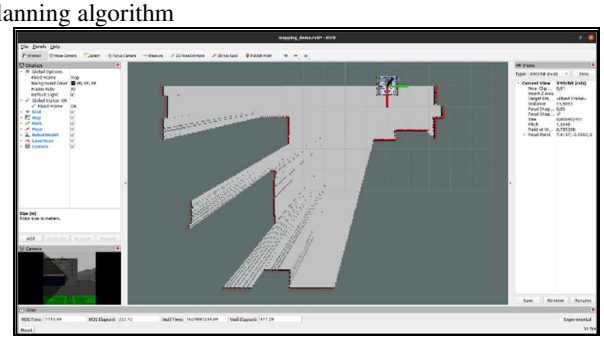

Fig. 13. Rviz display with the “Map” plugin while running the “hector_slam” package

The map will be updated anytime the robot travels a distance given by the "map_update_distance_thresh" parameter or rotates by an angle defined by the "map_update_angle_thresh" parameter, mentioned in “mapping_default” file.

In a dynamic environment, local route planning is required to address the deviation produced by the dynamic, complex environment. Local path planning prioritises obstacle avoidance, whereas global path planning prioritises the shortest route. Using a mix of global and local planning algorithms the robot can accomplish precise navigation.

The results of the path planning in the simulated environment for the robot platform analysed in this paper is shown in Figure 14. On the left side it is seen the environment mapped by the robot’s sensor. On the right side it is shown the environment (designed in Gazebo). On the top side it is shown the starting phase. On the bottom side it is shown the result of the path execution.

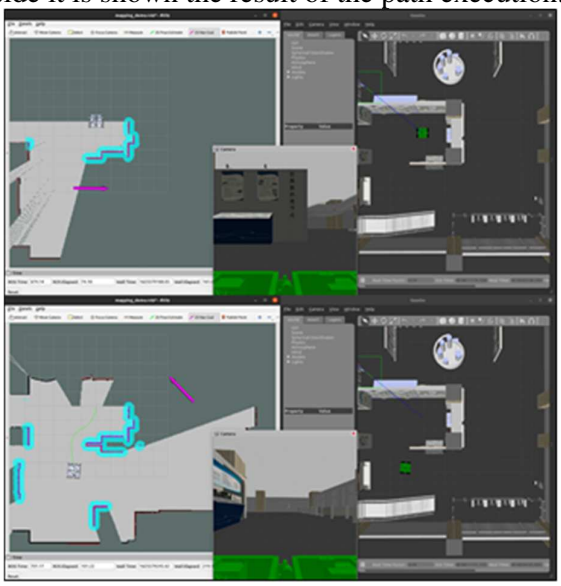

Fig. 14. Path execution with the path planning algorithm

The pink arrow points from Figure 14 indicate the goal position and orientation of the robot. The cyan-coloured space is the security space for the robot.

6. DESIGN-CENTRIC OBSTACLE AVOIDANCE ALGORITHM

For designing the algorithm dedicated to obstacle avoidance, we consider that mechanical construction influences, through its dynamics, the performances for obstacle avoidance. This means, for a given longitudinal speed of the mobile robot, for a given geometrical shape and volume of the mobile robot, and for a given aggregated mass and aggregated inertial moment around the z axis of the mobile robot, as well as for a given distance between the robot and the obstacle, the algorithm for obstacle avoidance must consider not only the kinematics of the robot, but also its dynamics. This is illustrated in Figure 15, which shows the model for the dynamic analysis of the robot. For the four wheels, masses are concentrated in one point, whereas for each of the two bodies of the robot, masses are dynamically concentrated in three points.

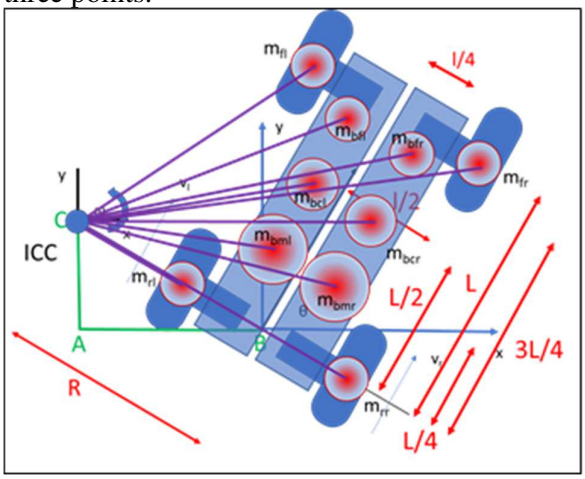

Fig. 15. Dynamic model of the robotic platform

It is not the purpose of this paper to provide more information about the dynamic modelling of the robotic platform. Data about masses and inertial moments around the z axis, reduced to the pivotal centre (see the Huygens–Steiner theorem of the parallel axes), have been introduced in the simulation software to adjust the level of torques on each motor.

To identify or design the obstacle avoidance algorithm (in the case none of the existent ones are appropriate), we consider the CSDT method [8]. It creates in the first stage relationships between the performance metrics of the algorithm and the constitutive generic components of the algorithm, as well as correlations between metrics. Based on this result, a roadmap for ideation is generated by the CSDT algorithm. Following the elementary steps of the roadmap, with assistance provided by systematic inventive problem-solving tools

(e.g., in this case TRIZ and SAVE, as it was mentioned in section 2 of this paper), a suitable solution must be identified to achieve the established performance indicators.

The following metrics (performance indicators) have been considered: (A) capacity to avoid obstacles that occur in less than 1 m; (B) capacity to realize a smooth path for collision avoidance such as to reduce shocks on the motors; (C) capacity to ensure obstacle avoidance with a single LiDAR; (D) capacity to move at a speed equivalent of half than the speed of human when it walks, while respecting metrics (A), (B), and (C).

The generic components of the algorithm are: (K1) division of the space; (K2) decision according to information about the scene; (K3) angle of rotation; (K4) direction of rotation.

The CSDT planning framework is shown in Figure 16. Further, the investigation and ideation roadmap is illustrated.

	D	-1954	-792	-1954	<-K				
	C	-10952	-4440		-3				
	B	-2220		-2	-3				
	A	C->	-1	-2	-3				
		A	B	C	D				
	r/d	0.1	0.3	0.1	0.5	d	H	J	O
K1	0.2	1	9	9	3	5.2	26	1.04	4.4
K2	0.3	9	9	9	3	6	20	1.8	9
K3	0.3	9	9	9	9	9	30	2.7	10.8
K4	0.2	9	9	1	1	4.2	21	0.84	4
	W	7.4	9	7.4	4.4				
	I	74	30	74	8.8				
	Z	0.74	2.7	0.74	2.2				
	Q	2.8	10.8	2.8	8				

Fig. 16. The CSDT planning framework

The following CSDT coefficients are introduced (see Figure 16): value weight (W), technical index of priority (I), relative technical effort (Z), impact depreciation (Q), technical depreciation (O), input risk (J), difficulty to satisfy inputs (d), correlation index of priority (K), and input index of priority (H). These coefficients are introduced in a logic algorithm [8] to generate the ideation roadmap. These are technicalities that can be consulted in [8] and we consider that there is no value-added by displaying the whole demarche here. To visualize the design flow (the ideation roadmap), some conventions are used. They are further introduced: (1) symbol "<>" indicates a link between two subsequent steps from the flow; (2) the symbol "&" describes the request to analyse the correlation between two outputs; (3) the symbol "|" asks to apply a given method (here TRIZ or SAVE) to solve a negative correlation between two outputs; (4) the symbol "—" represents the process of conceptualizing, finding a partial or complete solution for a given input with respect to a given output or a pair of outputs. The ideation roadmap is further introduced.

Flow 1: K3—(A&C) | TRIZ <> K2—(A&C) | TRIZ <> K1—(A&C) | SAVE <> K4—(A&C) | TRIZ

Flow 2: K3—(B&C) | TRIZ <> K2—(B&C) | TRIZ <> K1—(B&C) | SAVE <> K4—(B&C) | TRIZ

Flow 3: K3—(A&B) | TRIZ <> K2—(A&B) | TRIZ <> K1—(A&B) | SAVE <> K4—(A&B) | TRIZ

Flow 4: K3—(A&D) | TRIZ <> K2—(A&D) | TRIZ <> K1—(A&D) | SAVE <> K4—(A&D) | TRIZ

Flow 5: K3—(C&D) | TRIZ <> K2—(C&D) | TRIZ <> K1—(C&D) | SAVE <> K4—(C&D) | TRIZ

Flow 6: K3—(B&D) | TRIZ <> K2—(B&D) | TRIZ <> K1—(B&D) | SAVE <> K4—(B&D) | TRIZ

We exemplify only one elementary step of the ideation process: K3—(A&C) | TRIZ. This means "find a solution for deciding the elementary angle of rotation such that to ensure the capacity to avoid obstacles that occur in less than 1 m using a single LiDAR". In TRIZ language this means "reduce criticality without increasing complexity of the tool". This conflict leads to a set of TRIZ inventive vectors [9]: periodic action (increase frequency of impulses); increase the degree of segmentation (cover 180 degrees with one LiDAR and divide it into several segments); change the conditions (e.g., location of LiDAR).

Following the steps indicated by the roadmap, we finally identified an algorithm that adequately satisfies our requirements. It was developed in 2019 and described in [12].

The "obstacle avoidance" algorithm analyses a large quantity of data from the LiDAR and conducts sophisticated robot actions, combining linear and angular velocities to allow the robot to follow a curved path, while avoiding collisions with obstacles. The sensor scans a space of 180°, which is divided into 5 distinct

sets, as highlighted in Figure 17, where each set is the shortest distance recorded on a 36-degree sector, hence the total of 5 sectors ($5 \cdot 36° = 180°$).

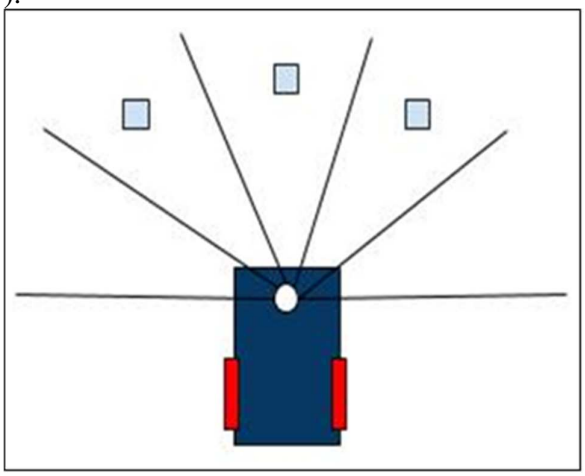

Fig. 17. Five divisions of the scanned space [12]

According to Figure 18, front left, front and front right regions are of major interest for the avoidance of obstacles. For each of these regions, eight possibilities can occur, depending on the position of the detected obstacle.

	Front Left	Front	Front Right
Case 1			
Case 2		x	
Case 3			x
Case 4	x		
Case 5		x	x
Case 6	x	x	
Case 7	x	x	x
Case 8	x		x

Fig. 18. Three obstacle detection areas studied through eight possible cases [12]

In this regard, the Python script "obstacle_ avoidance" considers each case and publishes linear and angular velocity commands to the "/mobile_ base_diff_controller/cmd_vel" topic, in conformity with each situation, as it is reflected in Figure 19.

```python
# Implement the obstacle avoidance logic
def take_action(regions):
    msg = Twist()
    # When linear_x > 0 = go straight
    linear_x = 0
    # When angular_z > 0 = turn to the left/right; rotate along z axis
    angular_z = 0

    state_description = ''

    d = 1

    # 8 cases disussed for 3 regions: Front Left, Front & Front Right
    # Object is considered an obstacle if it is situated at 1 m from the robot (at least)
    if regions['front'] > d and regions['front_left'] > d and regions['front_right'] > d:
        state_description = 'case 1 - nothing'
        linear_x = 0.9
        angular_z = 0
    elif regions['front'] < d and regions['front_left'] > d and regions['front_right'] > d:
        state_description = 'case 2 - front'
        linear_x = 0
        angular_z = 0.5
    elif regions['front'] > d and regions['front_left'] > d and regions['front_right'] < d:
        state_description = 'case 3 - front_right'
        linear_x = 0
        angular_z = 0.5
    elif regions['front'] > d and regions['front_left'] < d and regions['front_right'] > d:
        state_description = 'case 4 - front_left'
        linear_x = 0
        angular_z = -0.5
    elif regions['front'] < d and regions['front_left'] > d and regions['front_right'] < d:
        state_description = 'case 5 - front and front_right'
        linear_x = 0
        angular_z = 0.5
    elif regions['front'] < d and regions['front_left'] < d and regions['front_right'] > d:
        state_description = 'case 6 - front and front_left'
        linear_x = 0
        angular_z = -0.5
    elif regions['front'] < d and regions['front_left'] < d and regions['front_right'] < d:
        state_description = 'case 7 - front and front_left and front_right'
        linear_x = 0
        angular_z = 0.5
    elif regions['front'] > d and regions['front_left'] < d and regions['front_right'] < d:
        state_description = 'case 8 - front_left and front_right'
        linear_x = 0.9
        angular_z = 0
    else:
        # Reach an impossible point
        state_description = 'unknown case'
        rospy.loginfo(regions)

    # Print the case number
    rospy.loginfo(state_description)

    msg.linear.x = linear_x
    msg.angular.z = angular_z
    # Publish message to the topic
    pub.publish(msg)
```

Fig. 19. Obstacle avoidance logic in Python

The "take_action" function (Figure 19) implements the logic for avoiding obstacles – depending on the distances in each of the five regions – analysing various obstacle combinations and steering the robot in a different direction, avoiding collision. The logic behind the algorithm is quite straightforward: for example, "case 1" depicts the circumstance where the robot moves forward, since there is no obstacle situated at 1 meter from the robot. Taking "case 3" and "case 5" as another example, the obstacle detected on the right side determines the robot to turn left. The function concludes with specifying and publishing the "twist" message (the values for the robot velocities according to the selected case).

7. RESULTS

Because it provides a safe trajectory and guarantees convergence, a collision-free algorithm is a vital requirement for autonomous mobile robots. Figure 20 and Figure 21 show tests of the robot on obstacle avoidance during autonomous navigation, using two simulation platforms.

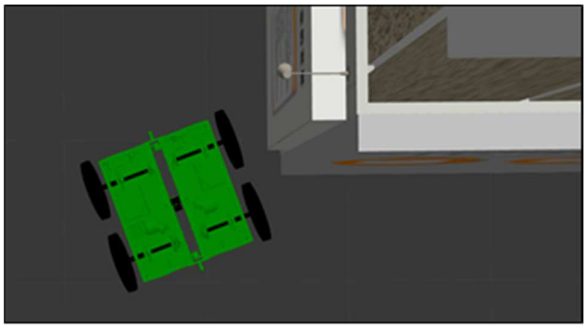

Fig. 20. Mobile robot avoiding obstacle (Gazebo)

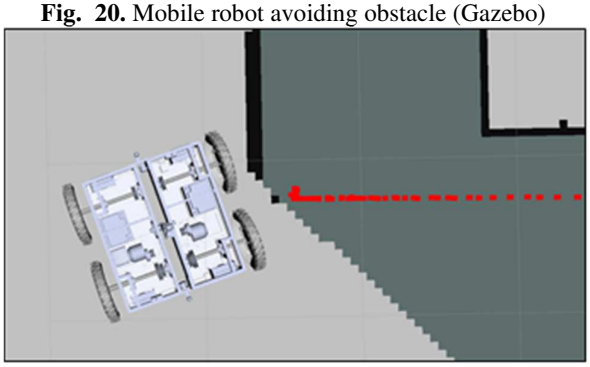

Fig. 21. Obstacle avoidance visualization (Rviz)

In this manner, the implemented obstacle avoidance algorithm allows the robot to travel the complete perimeter of the obstacle and choose the appropriate position from which to depart towards the goal, as demonstrated in the figures above. Although the advantage of this approach is that it facilitates the detection of every obstacle inside the environment, the mobile robot requires a longer time to complete its task, because of the impact of linear and angular components of velocity.

Regarding the performances of Hector SLAM algorithm with the proposed obstacle avoidance algorithm integrated in its logic, we noticed that it generates a superior quality mapping, due to its dependence on accurate scan matching. This is achieved when the navigation of the mobile robot is done with low linear and angular velocities (approximately 0.3 m/s). Quality refers to a successful overlap of the laser scans in real-time. Opposite results have been obtained when the latter velocity parameters were increased to 1 m/s. This situation is explained by the fact that this algorithm demands a laser range finder with high update rates (i.e., the update rate is 20 Hz for the Hokuyo LiDAR sensor); thus, restrictions concerning the values of linear and angular components of velocity are necessary for a high-quality mapping.

8. CONCLUSIONS

Autonomous navigation of mobile robots requires the integration of many technologies. Sensors that capture data from the external dynamic environment must be filtered and analysed in due-time and converted into adequate formats to be processed by the control unit and transfer rapid commands to the motors. The capacity to embed both kinematic and dynamic models of the robot within the decision algorithm is very important to ensure accurate navigation and obstacle avoidance, with smooth paths and low shocks to the motors. Thus, the obstacle avoidance algorithm necessitates customization to the specificities of the mechanical construction of the robotic platform, but also to the construction of the sensing system (sensor type, sensor location, number of sensors). We have found that the more accurate the dynamic model is, the better the obstacle avoidance algorithm behaves. In this paper we have tested an existing algorithm for obstacle avoidance, because the systematic investigation roadmap indicated it to be valuable. However, there is still potential for improvement this algorithm with the consideration of more divisions of the scanned space and interlinked with the geometrical dimensions of the platform.

9. REFERENCES

[1]. Akiyoshi, K., Chugo, D., Muramatsu, S., Yokota, S., Hashimoto, H., *Autonomous mobile robot navigation considering the pedestrian flow intersections*, 2020 IEEE/SICE International Symposium on System Integration (SII), pp. 428-433, doi: 10.1109/SII46433.2020.9026277, Honolulu, USA, (2020).

[2]. Alatise, M.B, Hancke, G.P., *A review on challenges of autonomous mobile robot and sensor fusion methods*, IEEE Access, Vol. 8, pp. 39830-39846, (2020).

[3]. Iqbal, J., Xu, R., Sun, S., Li, C., *Simulation of an autonomous mobile robot for LiDAR-based in-field phenotyping and navigation*, Robotics, Vol. 9, No. 46, pp. 1-19, (2020).

[4]. Shen, M., Wang, Y., Jiang, Y., Ji, H., Wang, B., Huang, Z., *A new positioning method based on multiple ultrasonic sensors for autonomous mobile robot*, Sensors, Vol. 20, No. 17, pp. 1-15, (2020).
[5]. Mohanty, P.K., Parhi, D.R., *Controlling the motion of an autonomous mobile robot using various techniques: a review*, Journal of Advanced Mechanical Engineering, Vol. 1, pp. 24-39, (2013).
[6]. Du, Y.C., Ai, C.S., Feng, Z.Q., *Research on navigation system of AMR based on ROS, 2020 IEEE International Conference on Real-Time Computing and Robotics*, pp. 117-121, Shandong, China, (2020).
[7]. Elkilany, B.G., Abouelsoud, A.A., Fathelbab, A.M.R., Ishii, H., *A proposed decentralized formation control algorithm for robot swarm based on an optimized potential field method*, Neural Computing & Applications, Vol. 33, No. 1, pp. 487-499, (2020).
[8]. Brad, S., *Complex system design technique*, International Journal of Production Research, Vol. 46, No. 21, pp. 5979-6008, (2008).
[9]. Altshuller, G., *The Innovation Algorithm. TRIZ. Technical Innovation Center*, Worcester, USA, (2000).
[10]. Brad, S., *Structured activation of vertex entropy (SAVE): another way around creative problem solving for non-technical applications*, Innovator Journal of the European TRIZ Association Vol. 1, pp. 76-81, (2017).
[11]. Bailey, M., Gebis, K., Žefran, M., *Simulation of Closed Kinematic Chains in Realistic Environments Using Gazebo. Springer*, Chicago, USA, (2016).
[12]. Arruda, M., *Exploring ROS with a 2 wheeled robot #5: obstacle avoidance. Retrieved from The Construct:* https://www.theconstructsim.com /exploring-ros-2-wheeled-robot-part-5/, (2019).

ALGORITM DE EVITARE A OBSTACOLELOR CENTRAT PE DESIGN PENTRU UN ROBOT MOBIL AUTONOM ȘI TESTAREA ACESTUIA FOLOSIND TEHNOLOGII DE PROTOTIPARE VIRTUALĂ

Rezumat: Roboții mobili autonomi sunt necesari în multe aplicații, de la industrie la servicii, securitate și apărare. Capacitatea de a naviga autonom într-un mod lin în medii cu diverse obstacole este esențială pentru adoptarea în practică a acestor tipuri de sisteme robotizate. Navigarea lină și evitarea obstacolelor depind în mod semnificativ de corelația corectă între algoritmul de evitare a obstacolelor și proiectarea sistemului robotizat (tipul de senzori, locația senzorului, tipul de control, construcția mecanică a platformei mobile). Această lucrare investighează posibilitatea de a proiecta algoritmul de evitare a obstacolelor din perspectiva designului particular al robotului. Validarea timpurie a algoritmului este o abordare rentabilă. În acest sens, această lucrare introduce, de asemenea, o construcție arhitecturală folosind diferite tehnologii open-source pentru a testa și valida algoritmul prin intermediul unui prototip digital (sau digital twin) care încorporează proprietățile fizice ale robotului real și ale obstacolelor din mediul de simulare. Testele efectuate în mediul virtual arată că algoritmul propus, încorporat într-un algoritm mai larg de Localizare și Cartografiere Simultană (SLAM), este capabil să asigure o evitare fără probleme a obstacolelor. Rezultatele pot fi implementate cu ușurință într-un robot fizic mobil pentru navigare autonomă inteligentă.

Stelian BRAD, Professor dr. Eng. Technical University of Cluj-Napoca, Department of Engineering Design and Robotics, E-mail: stelian.brad@staff.utcluj.ro.
Naomi Damaris DOLHA, Eng., Technical University of Cluj-Napoca, Department of Engineering Design and Robotics, E-mail: damarisdolha@gmail.com.