ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 10, No. 1, 2025

COMPUTATIONAL EVALUATION OF LAPLACE ARTIFICIAL POTENTIAL FIELD METHODS FOR REAL-TIME OBSTACLE AVOIDANCE IN GAZEBO

Ihor Berizka, Ivan Karbovnyk

Ivan Franko Lviv National University, 107, Tarnavskoho str., Lviv, 79017, Ukraine
Authors' e-mails: *ihor.berizka@lnu.edu.ua, ivan.karbovnyk@lnu.edu.ua*

https://doi.org/10.23939/acps2025.01.001

Submitted on 11.03.2025

***Abstract*: the goal of this article is to present evaluation results for a proposed modification of the Artificial Potential Field Method (APFM). The mathematical model employs Laplace functions to compute repulsive fields to simplify calculations. Additionally, the study introduces a comprehensive evaluation framework using Gazebo and ROS2, designed to test various obstacle avoidance algorithms in simulated environments. Experiments have been conducted in a virtual room containing static obstacles of diverse shapes. The results demonstrate that the Laplace APFM effectively computes safe directional angles, enabling the robot to navigate smoothly and efficiently toward its target. The algorithm's performance have been validated through detailed analyses of LiDAR data, force calculations, and trajectory visualizations.**

***Index terms*: artificial potential field method, cyber-physical system, edge computing, information technologies, IoT concepts, obstacle avoidance, robotics, electronics.**

I. INTRODUCTION

In the domain of robotics, a critical area of focus is autonomous mobile robots. These are sophisticated machines engineered with the ability to move autonomously and are capable of execution specific decisions in a real-time context.

Instances of such autonomous mobile robots are manifold and varied. A fundamental component of the software in such machines are the algorithms for path planning and obstacle avoidance. These algorithms are pivotal as they endow the machines with capabilities such as automatic parking, circumventing emergency situations on the road, and even achieving full autonomy.

Path planning, in a broad sense, is divided into two categories: global and local. Global path planning necessitates information derived from a Geographic Information System (GIS) coupled with global localization. Local path planning requires only the relative position of the robot and the detection of obstacles in its immediate vicinity. This is more concerned with the robot's immediate surroundings and how it navigates through them.

There exists a number of algorithms designed to address both global and local planning. Each algorithm has its strengths and weaknesses, and the choice of algorithm can significantly impact the efficiency and safety of the robot's navigation. A more exhaustive review of these path planning algorithms, their methodologies, and their applications in various scenarios is provided in the works [1, 2].

Obstacle detection and avoidance is an integral part of the algorithms used in local path planning, playing a crucial role in ensuring the safety of both the robot and its operational environment. This field has been the subject of extensive research for several decades, and a wide array of approaches have been proposed. Some of these methodologies have found practical application in real-world scenarios, underscoring their effectiveness.

For a robot to avoid a collision with another object, it is imperative that it not only detects the obstacle but also recalculates its path and adjusts its current trajectory in real time. This necessitates the robot to have enough computational capabilities and the ability to make real-time decisions based on the data it collects from its sensors.

The robot must first use its sensors to detect potential obstacles in its path. Once an obstacle is detected, the robot must then calculate an alternative path or heading that avoids the obstacle. This path must be calculated quickly enough to allow the robot to adjust its trajectory in real time, thereby avoiding a collision.

Furthermore, the robot must also be able to adapt to changes in its environment. For instance, if a new obstacle suddenly appears in its path, the robot must be able to quickly detect the new obstacle, calculate a new path, and adjust its trajectory accordingly. This highlights the complexity of autonomous mobile robots and underscores the importance of ongoing research in this field.

Obstacle avoidance problem can be conceptualized as follows: a robot (denoted as "A" in our discussion) is placed in an unknown environment. The robot is given a target location or direction, and it is required to navigate towards this target while avoiding any obstacles that may be present in its path.

The environment in which the robot operates is populated with several obstacles (denoted as "O"). These obstacles could be static or dynamic and can vary in size, shape, and other properties. The robot is equipped with sensors that provide it with a certain 'sensitivity zone'

(denoted as "S"). This sensitivity zone can be thought of as the radius within which the robot can detect the presence of other objects.

The robot's current direction of movement is represented by a vector (denoted as "U"). At any given point in time, the robot's objective is to move in the direction of this vector while ensuring that it does not collide with any obstacles. Fig. 1, *a* illustrates the initial stage of the task. The robot has detected an obstacle and is moving in the given direction. If we extrapolate the robot's current direction of movement, we observe that it will lead to a collision with the obstacle.

This is where the algorithms for obstacle detection and avoidance come into play. These algorithms are designed to solve the task at hand by adjusting the robot's direction of movement, thereby ensuring safe navigation. The algorithms consider the robot's current position, the position of the target, the positions of the obstacles, and the robot's sensitivity zone to compute a safe path for the robot. The scenario depicted in Fig. 1, *b* serves as an illustrative example of an obstacle avoidance strategy. At a specific timestamp denoted as t_i, the autonomous entity, which we refer to as the robot, encounters an obstruction in its path. This triggers the robot's obstacle detection mechanism, prompting it to adjust its trajectory, represented by the variable *U*.

As the robot continues its revised path, it reaches another point in time, expressed as $t_i + T$. At this juncture, the robot identifies the presence of another obstacle. This necessitates a further adjustment to its trajectory, prompting another correction to the direction of movement *U*.

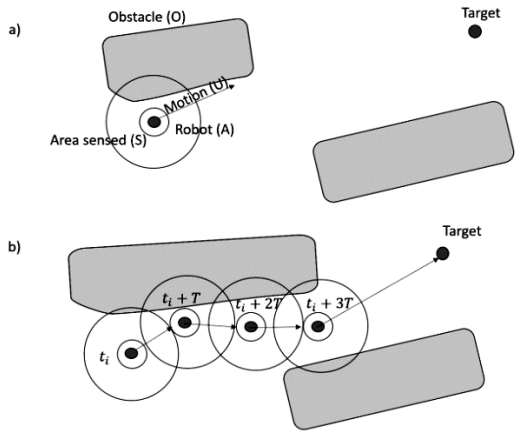


Fig. 1. Obstacle avoidance problem (a) and iterative solution of the problem (b)

This sequence of detecting obstacles and adjusting the robot's trajectory is not a one-time event but rather an iterative process. The robot continuously scans its environment for potential obstructions, making necessary corrections to its path until it successfully navigates to the pre-determined target point. This iterative process underscores the dynamic nature of autonomous navigation and the importance of effective obstacle detection and avoidance mechanisms in robotic systems.

II. LITERATURE REVIEW AND PROBLEM STATEMENT

A. MATHEMATICAL MODEL OF LAPLACE APFM

The Artificial Potential Field method (APFM) is a classic technique in the field of robotics, particularly for path planning and obstacle avoidance. This method was introduced by Khatib in 1984 [3].

In the APF method, a virtual potential field is created where the target location and any obstacles in the environment generate attractive and repulsive forces, respectively. The robot, or autonomous agent, is then guided by these forces. It experiences an attractive force towards the target and a repulsive force away from obstacles. The robot moves under the action of the resultant force [3]. Equations that represent these forces are (1)–(3).

$$f_{total} = f_{att} + f_{rep}, \tag{1}$$

$$f_{att} = k_{att} \cdot \left(\frac{r_{goal} - r}{|r_{goal} - r|} \right), \tag{2}$$

$$f_{rep} = \begin{cases} -k_{rep} \cdot \sum_{i=1}^{n} \left(\frac{1}{d_i} - \frac{1}{d_{\max}} \right) \cdot s_i, \; if \; d_i < d_{\max} \\ 0 \end{cases}. \tag{3}$$

However, the traditional APF method has some limitations. For instance, it can lead to local minima problems, where the robot gets stuck in a position that is not the target because the forces from the target and obstacles cancel each other out. It can also lead to unreachable targets and inefficient paths [4].

To overcome these issues, improved versions of the APF method have been proposed. One of such modifications of classic APFM is based on probabilities and is presented in [5]. The ODG-PF method was designed and implemented to detect obstacles and calculate the likelihood of collision with them. The paper introduces a novel attractive field and repulsive field calculation method and direction decision approach. The authors carried out simulations and experiments and compared the ODG-PF method with other potential field-based obstacle avoidance methods. The results showed that the ODG-PF method performed the best in most cases.

The algorithm under consideration involves the computation of a repulsive field. This field is a vector field that quantifies the force that repels the robot away from an obstacle.

In [6] we provided more detailed review of ODG-PF and proposed further research in this direction. We presented mathematical model of APFM which uses Laplace function to represent repulsive field.

$$f_k(\theta_i) = A_k \cdot \exp\left(-\frac{\sqrt{2}|\theta_k - \theta_i|}{\sigma} \right), \tag{4}$$

where θ_k corresponds to the central angle of the k_{th} obstacle, σ_k is half of angle occupied by the k_{th} obstacle.

The equation (4) describes the repulsive force. Analyzing the formula, this function is simpler for calculations, as it does not require squaring the value under the exponent but only taking the absolute value.

In the context of a sensor with a resolution of 1 degree, the index i can be interpreted as a discrete representation of the angular measurement in the range of 0 to 360 degrees. Each Laplace function, associated with a specific angle, contributes to the overall repulsive field, effectively creating a vector field that guides the robot away from obstacles.

The coefficient A_k is of particular interest. This coefficient is adjusted such that the Laplace function encapsulates the entirety of the obstacle. This ensures that the repulsive force is accurately represented in the vicinity of the obstacle. The derivation of this coefficient is an essential aspect of the algorithm under consideration and warrants further examination.

$$A_k = \overline{d_k} \cdot \exp(\sqrt{2}), \tag{5}$$

where $\overline{d_k} = d_{\max} - d_k$, $d_{\max}$ is sensor range distance.

Given the necessity to account for the field of each obstacle, the overall obstacle field is computed as the sum of all repulsive fields generated by the obstacles. This results in a function that is dependent on the angle θ.

$$f_{rep}(\theta_i) = \sum_{k=1}^{n} A_k \cdot \exp\left(-\frac{\sqrt{2}|\theta_k - \theta_i|}{\sigma}\right). \tag{6}$$

The subsequent stage involves the computation of an attractive field (7). This field describes the force that attracts the robot towards the target direction of movement. This attractive field, in conjunction with the repulsive field, guides the robot's trajectory in a manner that avoids obstacles while progressing towards the target.

$$f_{attr}(\theta_i) = \gamma|\theta_{goal} - \theta_i|, \tag{7}$$

$$f_{total}(\theta_i) = f_{attr}(\theta_i) + f_{rep}(\theta_i), \tag{8}$$

$$\theta_{dir} = argmin(f_{total}). \tag{9}$$

Parameter γ is selected experimentally and is set to 6.36. Equation (8) represents total field produced by the system, and the safe direction of robot movement is determined using (9).

B. ROS 2

Numerous software platforms, often referred to as middleware, have been developed to introduce modular and adaptable features that simplify the construction of robotic systems. Over time, some of these middleware platforms have evolved into comprehensive ecosystems, offering a wealth of utilities, algorithms, and sample applications. Among them, the Robot Operating System (ROS 1) stands out for its profound impact on the advancing robotics industry. Despite some limitations, ROS 1 has had a significant impact across nearly every sector involving intelligent machines.

The second generation of the Robot Operating System, ROS 2, was redesigned from the ground up to address these challenges while building on the success of its community driven capabilities [7]. ROS 2 is based on the Data Distribution Service (DDS), an open standard for communications that is used in critical infrastructure such as military, spacecraft, and financial systems [8]. It solves many of the problems in building reliable robotics systems. DDS enables ROS 2 to obtain best-in-class security, embedded and real-time support, multi-robot communication, and operations in non-ideal networking environments. DDS was selected after considering other communication technologies, e. g. ZeroMQ, RabbitMQ, due to its breadth of features including UDP transport, distributed discovery, a built-in security standard [9].

The integration of ROS 2 as a uniform middleware for deploying practical outcomes brings us one step closer to sim-to-real experiences, allowing researchers to test ideas beyond confined laboratory settings. Mathematical model of new APFM [6] was implemented as shared C++ library which simplifies its integration with other software components including integration into ROS2 ecosystem.

C. SIMULATOR AND HARDWARE PLATFORM

In [6] we conducted basic evaluation of developed mathematical model with Laplace APFM. In this paper we continue evaluating proposed model in more enhanced scenarios. Simulation is crucial for providing safely reproducible scenarios to evaluate the ever-evolving fields of computer science and robotics.

To simulate the dynamics of wheeled robots, a variety of physics engines are typically employed, including ODE, DART, MuJoCo, Bullet, SimBody, PhysX, and RaiSim. These engines are designed to solve problems related to rigid articulated bodies, collisions, and contacts. Table 1 presents a comprehensive list of platforms commonly used for robot simulation and capable of integrating with ROS.

Among the most versatile platforms are Gazebo and CoppeliaSim (formerly V-REP), both of which support multiple physics engines and programming languages like Python and C++. Gazebo is particularly noted for its realistic physics and graphics, extensibility through plugins, extensive library of models and environments, and strong community support. CoppeliaSim excels in simulating complex kinematics and dynamics and offers extensive sensor and actuator support.

According to current academic literature, Gazebo has become the primary simulator, accounting for 70.5 % of published papers in robotics simulations. This dominance can be attributed to its open-source nature, continuous community support, and modularity, as evidenced by its frequent pairing with RViz. However, Gazebo currently is preferable choice, it is worth noting a noticeable increase in usage of alternatives such as Unity and Unreal Engine.

Based on the study done in [10] and taking into account advantages of Gazebo platform it is decided to use this platform as a target simulator since its ability to easily integrate with ROS2 framework and support of C++ programming language alongside with other pros mentioned above.

The next part is selecting ROS-enabled wheeled mobile robot platform. We performed analysis of existing solutions and found out that TurtleBot series is used in most cases for research within mobile wheeled robots' domain. Also, TurtleBot series provides integration with ROS2 and Gazebo out of the box. All needed resources are available on official website and corresponding GitHub repos.

Table 1

Comparison of simulation platforms.

Name	Physics Engine	Programming Language	Key Features	Open Source
Gazebo	ODE, Bullet, Simbody, DART	• Python • C++	• Realistic physics and graphics • Extensibility with plugins • Large library of models and environments • Community support and active development	Yes
MuJoCo	MuJoCo	• Python • MATLAB • C++	• Fast and accurate physics • Seamless Reinforcement Learning (RL) Integration • Flexible API for customization • Support for soft bodies and deformable objects	Yes
Webots	ODE	• Python • MATLAB • C++ • C • Java	• Large library of robot models and environments • Offers web-based simulation • User-friendly interface • Modular and user-extensible design	Yes
CoppeliaSim (V-REP)	Bullet, ODE, MuJoCo, Vortex Studio, Newton Dynamics	• Python • MATLAB • C++ • Lua	• Simulation of complex kinematics and dynamics • Fast algorithm development • Remote API and connectivity • Wide range of sensor and actuator support	Free and paid version
ARGoS	Bullet, ODE	• Python • C++ • Lua	• Large-scale multi-robot swarm simulation • Extensibility with plugins • Customizable robot models • Easy experiment and simulation setup	Yes
PyBullet	Bullet	• Python • MATLAB • C++ • Java	• Easy integration with machine learning libraries • High performance and efficiency • Diverse robotics and soft body simulation • Real-time physics debugging	Yes
CARLA	Unreal Engine 4 (PhysX), Project Chrono	• Python • C++	• Autonomous driving scenario definition and evaluation • OpenDRIVE road network generation • Dynamic and realistic traffic simulation • Comprehensive sensor suite	Yes
RuiSim	RuiSim	• Python • C++	• Advanced sensor simulation • High-fidelity physics • Modular and lightweight architecture • Efficient GPU acceleration	No
Project Chrono	Project Chrono	• Python • C++	• Multibody dynamics and granular simulation • High-performance parallel computing • Advanced vehicle dynamics • Realistic terrain and geological simulation	Yes
USARSim	Unreal Engine 2 (PhysX)	• Python • C++	• Realistic urban environment simulation • Wireless communication simulation • Sensor and actuator fault simulation • Extensible architecture	Yes
OpenRAVE	ODE, Bullet	• Python • MATLAB • C++	• Focus on motion planning and manipulation • Extensive kinematics and dynamics solvers • Robust collision detection and grasping support • Support for multi-robot simulations	Yes
AirSim	Unreal Engine 4 (PhysX)	• Python • MATLAB • C++	• Photorealistic graphics and environments • Dynamic weather and lighting conditions • Easy integration with Machine Learning libraries • Hardware-in-the-loop (HIL) simulations	Yes
SOFA	SOFA	• Python • C++ • XML	• Advanced multi-physics simulation • Modular and extensible architecture • Real-time interaction and haptics support • Parallel computing and GPU acceleration	Yes
NVIDIA Isaac Sim	PhysX 5	• Python • C++	• Physically accurate simulations • High-Quality Visuals and Rendering with NVIDIA RTX • Deep Learning and Reinforcement Learning Support • Integration with the NVIDIA Omniverse ecosystem	Yes
Unity	PhysX	• C#	• Real-Time Physics Simulation • High-Quality Graphics and Realistic Environments • Machine Learning Integration • Vast Asset Store and Community Support	Free and paid version
Unreal Engine	Unreal Engine (PhysX)	• C++ • Blueprints visual scripting	• High-fidelity visuals and realistic environments • Robust physics simulation • Large library of high-quality assets • Physically based materials and material editor	Free and paid version

III. SCOPE OF WORK AND OBJECTIVES

The main goal of the work is to evaluate mathematical model of Laplace APFM in virtual environment with static obstacles. To perform testing and simulations we analyzed existing approaches in academia and came up with overview of software and hardware platforms that are suitable for robotics project in mobile robots' domain in the academia research. Additionally, to the main goal, this article guides on developing universal evaluation platform for autonomous mobile robots and describes pros and cons of each approach and platform. Evaluation of the mathematical model on robotics system requires well designed and layered software architecture for seamless integration with hardware platform. Within the scope of this project, three distinct applications were implemented to achieve seamless integration of software components and hardware platform within the robotic system.

IV. SYSTEM ARCHITECTURE

Interaction between software components is implemented according to ROS2 architecture. The detailed project structure is presented in Fig. 2, which follows the conventional ROS2 (Robot Operating System 2) workspace architecture. The directory named OA_Simulations contains all the critical code and configuration files that are related to the mathematical modeling and control algorithms for the robot. Furthermore, the TurtleBot3 directory is dedicated exclusively to the TurtleBot3 specific components. This directory includes the software development kits (SDKs), detailed robot models, and other specific resources necessary for the operation and customization of TurtleBot3. The segregation of these components ensures a clear separation between the generic simulation environment, mathematical model implementation and the TurtleBot3-specific components.

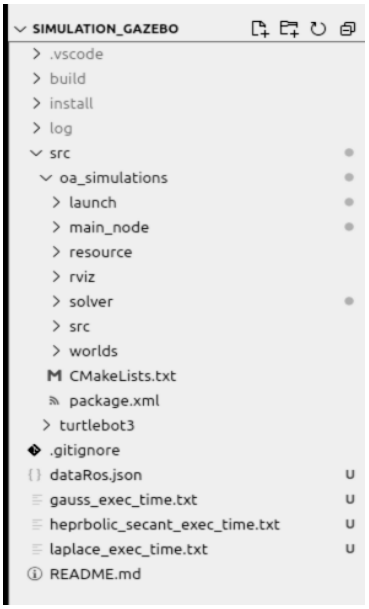


Fig. 2. Detailed project structure

Main_node: application subscribes to the LiDAR sensor data, performing real-time processing of the incoming data. It analyzes the data to calculate a safe angle direction for the robot's movement, ensuring obstacle avoidance and safe navigation. Subsequently, the Main_Node dispatches command signals to the robot's actuators, facilitating movement of the robotic platform. Also, it is designed to export intermediate forces data into separate JSON files, following the schema illustrated in Fig. 3. The primary object encapsulates an array of forces objects. Each force object stores three distinct sub-objects, representing the repulsive, attractive, and total potential field forces, respectively. Each of these sub-objects (repulsive, attractive, and total forces) contains data in the form of two number pairs, where each first value corresponds to a measurement angle (in degrees) and second one denotes the force magnitude at that particular

angle. Given that the LiDAR resolution is set to 1°, there are 360 data points (readings) for each type of force, with each data point representing the force magnitude at a specific angle, ranging from 0 to 359 degrees. Gazebo simulator: it is selected as simulation environment. It offers real-time visualization of the robot's movements and the configuration of its workspace. This application simulates sensor readings, including those from LiDAR, allowing for accurate replication of real-world scenarios.

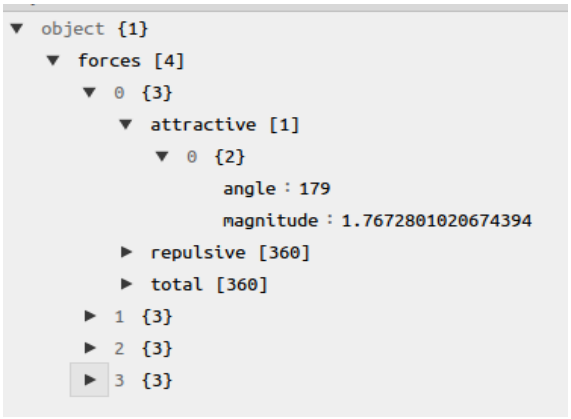


Fig. 3. JSON file schema with forces data

Rviz: application serves as a powerful tool for the visualization of the robot's movements and sensor readings. It provides an intuitive and interactive interface that enables developers to monitor the robot's performance in real-time. The application displays various sensor data, including LiDAR readings, in a visually coherent manner, path traversed by the robot, thereby facilitating the analysis and debugging of the robotic system's behavior.

Fig. 4 illustrates the detailed workspace configuration in the Gazebo simulator, which is used for testing the proposed mathematical model of the obstacle avoidance method. In this setup, the TurtleBot3 Burger is positioned at the center of the room (small cylinder with white dot in the center). The physical properties of TurtleBot3 are obtained from its official documentation and set to corresponding parameter values in the algorithm. The surrounding environment is a square room with walls and contains a set of obstacles with varying geometrical shapes, specifically cylindrical and square.

Table 2

Laplace APFM parameters and corresponding values

Parameter	Value
Threshold distance	1 m
Robot diameter	0.2 m
LiDAR max range	6 m
LiDAR resolution	–179–180, step 1°
Gamma	6.36
Goal direction	0°
Robot linear speed	0.1 m/s

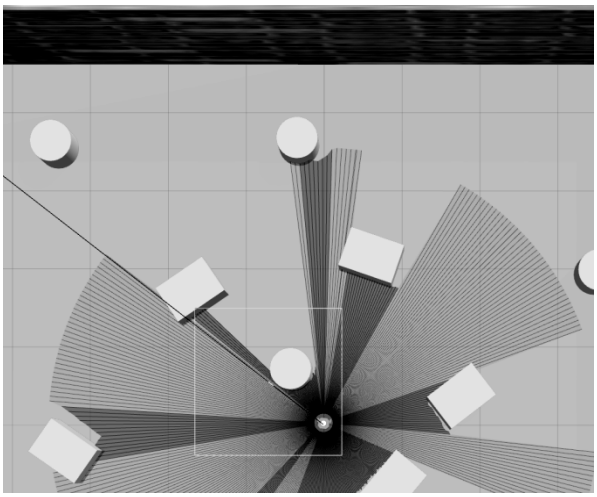


Fig. 4. Workplace configuration in Gazebo simulator

The physical properties of these obstacles are configured such that, upon collision, the robot is unable to move them. This ensures a realistic and rigorous evaluation of the robot's obstacle avoidance capabilities. Additionally, lines which are radiated from the robot center, represent the one-dimensional (1D) LiDAR path rays. LiDAR sensor resolution is set to 1°. In the current study, it is established that a single LiDAR scan corresponds to one complete rotation of the LiDAR sensor. This implies that the data collected during each scan encompasses a full 360° sweep of the surrounding environment, providing comprehensive spatial information for analysis.

Given that the simulation environment requires the startup of three distinct applications, it is considered a best practice to develop a launch file for system startup. This file is a Python script responsible for configuring and initializing all software components: the main_node, Gazebo, and RViz and is located in the launch directory within the ROS2 workspace.

Additionally, a fourth application, named Plotter, has been developed for the visualization and analysis of artificial potential fields data. It is developed using C++ and Qt framework. This application operates independently from the ROS2 project components and is not launched by Python script. Plotter is capable of reading JSON files that adhere to a schema presented on Fig. 3. Furthermore, the app supports a playback mode, which allows for the display of sensor readings and potential forces at each timestamp, providing a dynamic and comprehensive analysis of the data over time.

Proposed project architecture facilitates advanced testing methodologies and seamless integration with the actual TurtleBot robot, without modifications to the existing codebase.

V. SIMULATION IN GAZEBO

To evaluate proposed mathematical model of Laplace APFM [6] we used virtual room with obstacles which is presented in Fig. 4. Parameters and its values needed for Laplace APFM are presented in Table 2.

According to specified configuration and parameters values robot is expected to move forward until it reaches the wall and should avoid collision with obstacles on its path. The one iteration of the robot navigation algorithm is given in the pseudocode in Fig. 5.

Algorithm 1 General Robot Navigation Algorithm based on APFM

while target location is not reached **do**
  Scan Environment:
    - Acquire sample using a 1D LiDAR sensor
    - Detect obstacles
    - Enlarge obstacles
  Calculate Forces:
    - Calculate repulsive force using Eq. 6
    - Calculate attractive force using Eq. 8
    - Calculate total force using Eq. 9
  Determine Safe Direction: Calculate the safe directional angle using Eq. 10.
  if safe direction $\neq$ target direction **then**
    Rotate robot to align with the safe directional angle.
    Move forward for 1 second.
    Rotate back to the original target direction.
  else
    Move forward for 1 second.
  end if
end while

Fig. 5. Pseudocode of general robot navigation algorithm based APFM family algorithms

Next, we propose to discuss in detail the first 3 steps of an algorithm. At the starting point (the first iteration at the timestamp t_1) robot is located in the middle of the room has been shown in Fig. 4. Data from 1D LiDAR scan is shown in Fig. 6. The solid curve represents LiDAR scan. The dashed line represents threshold distance for an algorithm and is set according to the value in Table 2. All objects, which are closer to the robot that given threshold distance is considered as obstacles. Analyzing this scan, we observe that there are two continuous regions under the threshold line approximately at angles [–170°, –100°] and [+10°, +45°] and APFM is expected to detect two obstacles. The actual position of the robot within the simulation environment is depicted in the top-right corner of Fig. 6. Notably, the detected obstacles, represented by red figures, are observed to align precisely with the angular positions indicated in the LiDAR scan.

Fig. 7 illustrates the calculated repulsive, attractive, and total forces for a given timestamp t_1. The dashed line represents the attractive force, while scatter circles depict the repulsive force and scatter triangles indicate the total force. An analysis of the repulsive force curve reveals the presence of two Laplace peaks, approximately at angles [–170°, –100°] and [+10°, +45°]. These peaks correspond to the LiDAR sample shown in Fig. 6 where obstacles are detected. Notably, the Laplace peaks exhibit a slightly broader distribution along the angular axis, which enhances safety and reduces the likelihood of collisions. Furthermore, it is observed that the magnitude of the repulsive force is directly proportional to its effect: the greater the magnitude of this force, the stronger the repelling action exerted on the robot, effectively increasing the distance from the obstacle. On the other hand, the attractive force reaches its minimum at the target angle direction (which is set to 0° according to the Table 2). This indicates that the robot is naturally drawn to move along this direction, as it offers reduced resistance, guiding the system toward its intended path. The safe direction angle is determined by the minimum value of the total force. A detailed analysis of Fig. 7 reveals that the minimum total force is slightly shifted to the left along the angular axis compared to the attractive force minimum. This shift signifies that the robot must adjust its heading to avoid collisions with detected obstacles. Based on the Laplace APFM, the safe direction of movement at the given timestamp t_1 is calculated using Eq. (10) and is equal to –14°.

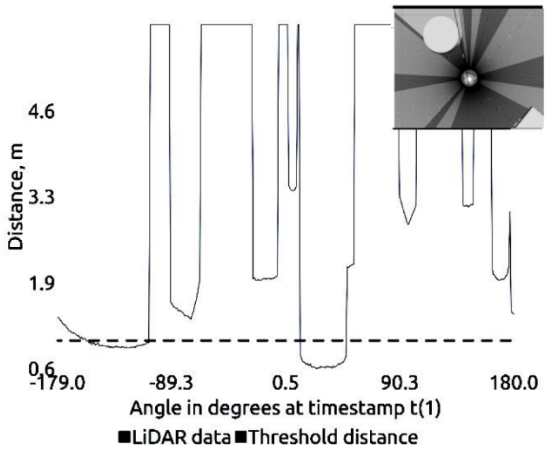


Fig. 6. 1D LiDAR scan at the timestamp t_1. Solid curve shows data from one complete LiDAR scan. Dashed line indicates threshold distance. Objects closer to the robot than threshold distance are marked as obstacles. Top-right corner picture depicts the actual position of the robot in simulator environment

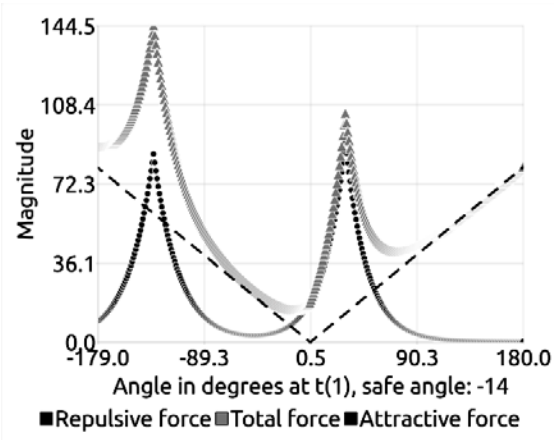


Fig. 7. Calculated repulsive, attractive and total forces at the timestamp t_1

As it has been outlined in the algorithm depicted in Fig. 5, the robot begins by rotating to the calculated safe angle, ensuring its heading is adjusted to avoid detected

obstacles. It then proceeds to move forward for a duration of one second, advancing incrementally toward its target. Following this movement, the robot reorients itself to align with the original target direction, resuming its trajectory.

This sequence of operations is iterative, continuing systematically until the robot either successfully reaches the target location or encounters a local minimum, where further progress becomes unattainable due to environmental constraints.

Based on the current experimental setup, the robot successfully navigated the environment in 28 discrete steps to reach the target location. To further validate the proposed method's reliability and performance, we propose to conduct a detailed analysis of two specific iterations. The first iteration will be examined at the intermediate location between the starting point and the target destination, specifically at timestamp t_{16}. The second iteration will focus on the target location itself, recorded at timestamp t_{28}, situated near a wall.

In both cases, a comprehensive evaluation of the forces acting on the robot during navigation will be conducted. Finally, the overall trajectory of the robot, as visualized using the RViz tool, will be reviewed thoroughly to assess the accuracy and efficacy of the proposed approach.

Analyzing data in Fig. 8, we observe one continuous region under the threshold line approximately at angles [–110°, –45°] and APFM is expected to detect single obstacle. Furthermore, it is evident that the position of an obstacle in the simulation environment corresponds precisely to those identified in the LiDAR scan.

Moreover, it is important to emphasize that the single detected obstacle is situated very close to the robot, approximately 0.3 meters from the side of the TurtleBot. This proximity suggests that the obstacle enlargement step will result in the formation of a wider Laplace peak during force calculations step.

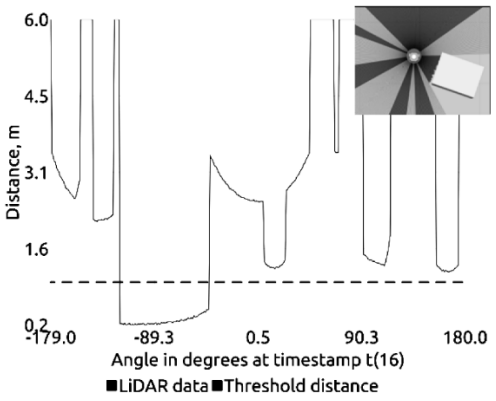


Fig. 8. 1D LiDAR scan at the timestamp t_{16}. Solid curve shows data from one complete LiDAR scan. Dashed line indicates threshold distance. Objects closer to the robot than threshold distance are marked as obstacles. Top-right corner picture depicts the actual position of the robot in simulator environment

Fig. 9 illustrates the calculated repulsive, attractive, and total forces for a given timestamp t_{16}.

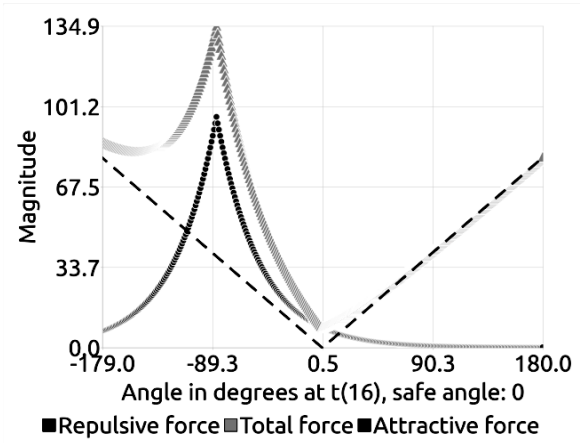


Fig. 9. Calculated repulsive, attractive and total forces at the timestamp t_{16}

An analysis of the repulsive force curve reveals the presence of single Laplace peak, approximately at angles [–180°, 0°]. This peak corresponds to the LiDAR sample shown in Fig. 8 where obstacles are detected. In this observation, it is evident that the Laplace peak spans a significantly greater range of angles compared to the actual area occupied by the obstacle, as indicated by the LiDAR sample presented in Fig. 8. This discrepancy can be attributed to the proximity of the obstacle to the robot, as it has been previously mentioned. The obstacle enlargement process amplifies this effect, leading to a broader Laplace coverage. This wider peak plays a critical role in ensuring that the robot perceives the obstacle as more prominent, effectively enhancing the repulsive force's influence for safer navigation around the obstacle.

However, since the obstacle is positioned behind the robot, the calculated safe angle remains unchanged and aligns precisely with the original target direction angle. This outcome highlights that the obstacle's influence on the robot's navigation is minimal in this scenario, as its placement does not interfere with the robot's forward trajectory.

Finally, Fig. 10 presents the LiDAR data sample captured during the robot's final step, at the point where it reached the wall.

Analyzing data in Fig. 10, we observe two continuous regions under the threshold line approximately at angles [–10°, 60°] and at angles [110°, 170°] and APFM is expected to detect both obstacles. Upon analyzing the plot, it is evident that the first detected obstacle corresponds to a wall. This conclusion is drawn from the geometric characteristics of the obstacle, which has a circular-like shape in the LiDAR data sample. Such a configuration is indicative of a wall, as it aligns with the expected uniform and smooth surface typical of this type of object. The second continuous region observed in the LiDAR data corresponds to the second obstacle. Its position can be inferred from the angular range in which it appears,

indicating that this obstacle is located slightly behind the robot. This observation is substantiated by the image depicting the robot's actual position within the simulation, as it has been shown in the Fig. 10 in top-right corner. In the simulation, a wall is clearly present in front of the robot, while an additional obstacle is located slightly behind it.

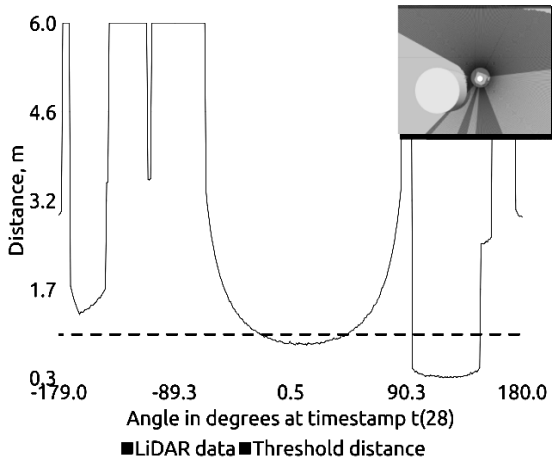


Fig. 10. 1D LiDAR scan at the timestamp t_{28}. Solid curve shows data from one complete LiDAR scan. Dashed line indicates threshold distance. Objects closer to the robot than threshold distance are marked as obstacles. Top-right corner picture depicts the actual position of the robot in simulator environment

Fig. 11 illustrates the calculated repulsive, attractive, and total forces for a given timestamp t_{28}.

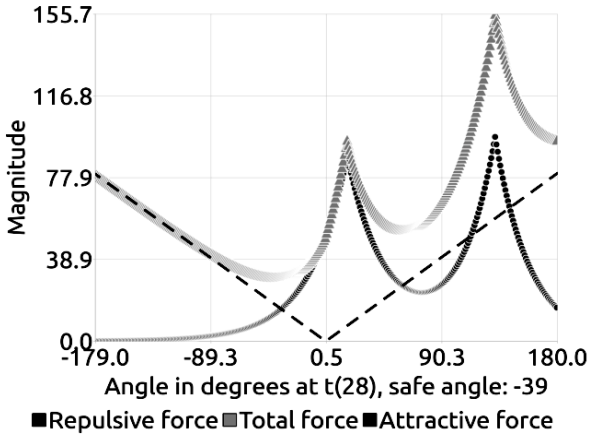


Fig. 11. Calculated repulsive, attractive and total forces at the timestamp t_{28}

An analysis of the repulsive force curve reveals the presence of two Laplace peaks, approximately at angles [–80°, +90°] and [+90°, +180°]. These peaks correspond to the LiDAR sample shown in Fig. 10. The first Laplace peak corresponds to the wall, and due to the wall's close proximity to the robot, this peak is significantly wide along the angular axis. Its substantial width results in an overlap with the Laplace peak associated with the second obstacle. The minimum of the total force aligns with an angle of –39°, which is identified as the safe direction by the algorithm. However, since the robot has already reached the target location, this calculated safe direction is disregarded. This decision reflects the algorithm's design to prioritize the final destination once it is achieved, rather than continuing to adjust navigation based on force computations.

Fig. 12 presents a trajectory visualization generated in RViz, illustrating the robot's navigation path.

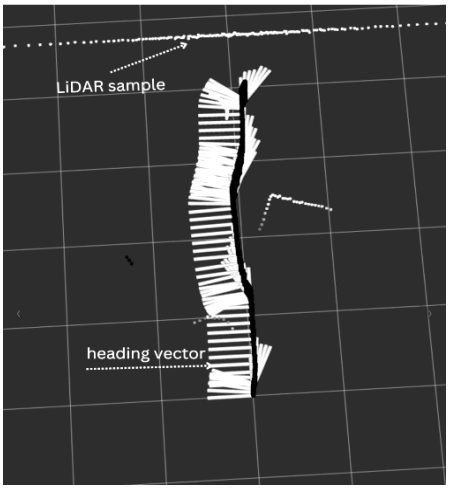


Fig. 12. Traversed path visualization in RViz

The visualization consists of several key elements. Heading vectors represent local heading adjustments made by the robot when some obstacles are detected and calculated safe direction angle is not equal to the target direction angle. LiDAR sample arrow points to markers that indicate the presence of obstacles identified by the onboard LiDAR sensor. The visualization confirms the effectiveness of our obstacle avoidance algorithm. As the robot encounters obstacles, the APF method generates repulsive forces that modify its trajectory, steering it away from potential collisions while ensuring forward progress. The smooth corrections in the path suggest stable and reactive control adjustments. Minor trajectory oscillations, observable in the path, could be attributed to sensitivity in repulsion force calculations, which may be further optimized for smoother navigation.

VI. CONCLUSION

The results of this study demonstrated the efficacy and accuracy of the proposed Laplace Artificial Potential Field Method (APFM) for robot navigation, validated through comprehensive analyses of LiDAR data and force calculations. By examining multiple iterations of the robot's trajectory, we confirmed the reliability of the approach across selected environmental conditions. The alignment of detected obstacles in LiDAR scans with their simulated positions further substantiated the precision of the method in mapping and obstacle detection.

Laplace APFM algorithm successfully computed safe directional angles, allowing the robot to maintain a smooth and efficient path toward its goal. The trajectory visualized

in RViz confirms the robustness of the algorithm, illustrating effective path corrections and collision avoidance.

In conclusion, this study established a solid framework for evaluation in simulated environments of autonomous robot navigation using different modifications of APFM, offering valuable insights into the interplay between potential fields and real-world sensor data. In future research we are going to explore the application of this method in dynamic environments and evaluate other functions to describe artificial potentials.

References

[1] Katona, K., Neamah, H. A., & Korondi, P. (2024). Obstacle avoidance and path planning methods for autonomous navigation of mobile robot. *Sensors*, 24(11), 3573. DOI: https://doi.org/10.3390/s24113573.

[2] Berizka, I. (2024). Path planning and obstacle avoidance methods for autonomous mobile robots. *Electronics and information technologies*, (28), 123–142. DOI: https://doi.org/10.30970/eli.28.11.

[3] Siciliano, B. Khatib, O. "Springer Handbook of Robotics", 2nd Edition, 2016, p. 1189.

[4] Fan, X., Guo, Y., Liu, H., Wei, B., & Lyu, W. (2020). Improved artificial potential field method applied for AUV path planning. *Mathematical Problems in Engineering*, *2020*(1), 6523158. DOI: https://doi.org/10.1155/2020/6523158.

[5] Cho, J. H., Pae, D. S., Lim, M. T., & Kang, T. K. (2018). A Real-Time Obstacle Avoidance Method for Autonomous Vehicles Using an Obstacle-Dependent Gaussian Potential Field. *Journal of Advanced Transportation*, *2018*(1), 5041401. DOI: https://doi.org/10.1155/2018/5041401.

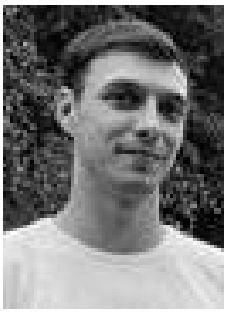
[6] Berizka, I. and Karbovnyk, I. (2024). Mathematical Model Of Modified Real-Time Obstacle Avoidance Method Based On Laplace Artificial Potential Field. *Applied Problems of Computer Science, Security and Mathematics. 3 (Sep. 2024)*, 12–22. Available: https://apcssm.vnu.edu.ua/index.php/Journalone/article/view/123.

[7] Macenski, S., Martín, F., White, R., & Clavero, J. G. (2020, October). The marathon 2: A navigation system. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 2718–2725). IEEE. DOI: https://doi.org/10.1109/IROS45743.2020.9341207.

[8] Pardo-Castellote, G. (2003, May). Omg data-distribution service: Architectural overview. In *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.* (pp. 200–206). IEEE. DOI: https://doi.org/ 10.1109/MILCOM.2003.1290110.

[9] Kim, J., Smereka, J. M., Cheung, C., Nepal, S., & Grobler, M. (2018). Security and performance considerations in ROS 2: A balancing act. *arXiv preprint arXiv:1809.09566*. DOI: https://doi.org/10.48550/arXiv.1809.09566.

[10] Kargar, S. M., Yordanov, B., Harvey, C., & Asadipour, A. (2024). Emerging trends in realistic robotic simulations: A comprehensive systematic literature review. *IEEE Access*. DOI: https://doi.org/10.1109/ACCESS.2024.3404881.



Ihor Berizka was born in 1996 in Lviv region, Ukraine. Starting from 2020, he has been studying for a PhD in Computer Science at the Faculty of Electronics and Computer Technologies of Ivan Franko National University of Lviv. With over 7 years combined of academic research and industry experience, he specializes in automotive domain, Internet of Things and robotics.

Ivan Karbovnyk, PhD, Dr. Sci., was born in Lviv, Ukraine, in 1978, is the Chair of Radiophysics and Computer Technologies Department at Ivan Franko National University of Lviv. With over 20 years of research experience, he specializes in automation, embedded systems, Internet of Things, computer modeling and electronics