sciendo

Int. J. Appl. Math. Comput. Sci., 2025, Vol. 35, No. 1, 45–57
DOI: 10.61822/amcs-2025-0004

amcs

FROM ROUGH MEREOLOGY TO ROBUST NAVIGATION: EXPANDING MOBILE ROBOT STRATEGIES WITH DIVERSE MAPS AND ENHANCED GOAL–TARGETING MECHANISMS

ALEKSANDRA SZPAKOWSKA [a], PIOTR ARTIEMJEW [a,*], WOJCIECH CYBOWSKI [b]

[a] Faculty of Mathematics and Computer Science
University of Warmia and Mazury in Olsztyn
ul. Słoneczna 54, 10-710 Olsztyn, Poland
e-mail: {ola.szpakowska,artem}@matman.uwm.edu.pl

[b]Security Team
Elympics
Plac Europejski 1, 00-844 Warsaw, Poland
e-mail: wojciech.cybowski@elympics.ai

Path planning is one of the most important problems in mobile robotics. Particularly challenging in selecting an appropriate path planning method is the choice of a method for complex obstacle configurations. The problems of path planning, among many other methods, come into play with approaches based on the application of potential fields methodology based on physical anomalies with gravitational or electromagnetic fields. This idea makes it possible to navigate in complex maps. The idea of applying these fields in terms of rough mereology was developed by Polkowski and Ośmialowski (2008), who introduced the method of the mereological potential field in the framework of mereological spatial reasoning. This particular work is one of a series of extensions of this method where our final goal is to apply the idea of path planning in a 3D environment. To this end, we are preparing and testing our own library for controlling mobile robots, improving the real-time path planning capability and implementing a set of algorithms for practical testing.

Keywords: rough mereology, mobile robotics, path planning.

1. Introduction

The history of path planning techniques for mobile robots goes back deep into the past, but has developed rapidly with advances in technology (Brooks, 1986; Latombe, 1991; Hwang and Ahuja, 1992; Kavraki *et al*., 1996; Arkin, 1998; Choset *et al*., 2005; LaValle, 2006; Sun and Liu, 2021; Raj and Kos, 2022). Initially, the simplest path planning methods were based on simple heuristic algorithms that selected the shortest path from point A to point B, avoiding obstacles. Over time, with the development of computer science and artificial intelligence, more advanced techniques have emerged. One of the most popular is the A* algorithm (and its dynamic versions), which allows finding an optimal path in complex environments full of obstacles. This algorithm uses a combination of heuristics and a search algorithm to find the best path. The next step in the development of path planning techniques was the use of SLAM (simultaneous localization and mapping) and machine learning techniques. With these, mobile robots can effectively plan paths in real time, taking into account changing environmental conditions. Today, the latest path planning methods for mobile robots use advanced technologies such as artificial intelligence, deep learning or evolutionary algorithms. Thanks to these, robots can move efficiently even in highly complex and dynamic environments.

In our work, we develop the idea of path planning for mobile robots using a mereological potential field (Ośmiałowski, 2011). It is a technique used in robotics and automation to precisely guide robots or mobile devices through complex environments. It is based on mereology, or the theory of parts and wholes and

*Corresponding author

© 2025 A. Szpakowska et al.
This is an open access article distributed under
the Creative Commons Attribution-NonCommercial-NoDerivs license
(http://creativecommons.org/licenses/by-nc-nd/3.0/).

their relationship to each other, and the concept of a potential field, which models the space around the robot as a continuum of different potential values that guide the device from a starting point to a destination (Polkowski and Ośmiałowski, 2008; Ośmiałowski, 2011). The foundational principles of rough mereology, which form the basis for the development of this family of techniques, were introduced by Polkowski and Skowron (1996). In this work, we extend the conference publication (Szpakowska *et al*., 2023) with new maps and map filtering improvements and discuss the difficulties encountered during our work. The basic concepts that allow us to define the environment for designing our method are as follows:

1.1. Integrating rough mereology in the control environment of intelligent agents. This section explores the application of rough mereology for generating potential fields (Polkowski and Ośmiałowski, 2008; Ośmiałowski, 2011). The introduction of rough inclusion, denoted by $\mu(x, y, r)$, asserts that x is part of y to a degree of at least $r \in [0, 1]$ (Polkowski and Ośmiałowski, 2008; Ośmiałowski, 2011). Specifically focusing on spatial objects, rough inclusion is expressed as $\mu(X, Y, r)$ if and only if

$$\frac{|X \cap Y|}{|X|} \geq r,$$

where X and Y represent n-dimensional solids, and $|X|$ signifies the n-volume of X.

The predicate μ captures basic intuitions about nature contained to a significant degree (Polkowski and Ośmiałowski, 2008; Ośmiałowski, 2011). In the specific context of this research, we consider a planar scenario involving an autonomous mobile robot navigating within a 2-dimensional space. Consequently, our spatial objects X and Y are conceptualized as regions, with $|X|$ representing the area of X. The role of rough inclusion $\mu(X, Y, r)$ is crucial in shaping the rough mereological potential field (Polkowski and Ośmiałowski, 2008; Ośmiałowski, 2011). The components of this field take on a square shape, and their relative distance is defined as:

$$K(X, Y) = \min\{\max(r|\mu(X, Y, r), \max(s|\mu(Y, X, s)\}.$$

Let us explain the idea of what the transition is from the defined distance $K(X, Y)$ to its application in the square fill algorithm.

In the context of quadratic fields of different sizes in the plane, $K(X, Y)$, can be interpreted as a measure that determines the minimum degree of inclusion of two fields relative to each other in the space defined by the tolerance radii r and s. (X, Y, r) determines the degree of inclusion of a set X into a set Y, given a neighborhood of radius r. In the context of quadratic fields, r can be interpreted as a margin or tolerance in space that allows the boundaries of the square Y to be extended to be more inclusive of X. The $\max_r(X, Y, r)$ is the search for the maximum degree of inclusion of X in Y, given an optimal radius r.

Intuitively, this means how much we can extend the boundaries of Y to maximally include X. The measure $K(X, Y)$ considers both the inclusion of X in Y and the inclusion of Y in X, choosing the smallest of the maxima. This is a conservative similarity assessment and it considers two fields to be close if, even under the most favourable conditions (with tolerance r, s), their mutual inclusion does not fall below a certain level. The distance $K(X, Y)$ for Y representing an obstacle can be used to define a repulsive or attractive force.

If we assume that the fields X and Y are identical squares or circles (as in our implementations), the distance $K(X, Y)$ can be reduced to using the distance between the centres of these squares (the usual Euclidean distance). When the squares have different sizes, shapes and orientations, the rule that reduces the distance $K(X, Y)$ to the distance between the centers of these squares no longer works, because $K(X, Y)$ varies between both ways. The defined distance is universal; we use a simplified version of it in the current calculations.

A comprehensive explanation of the field's construction is detailed in Section 2. The robot's trajectory within the field towards its destination is determined by waypoints. These waypoints are identified inductively, with the subsequent waypoint recognized as the centroid of the combination of field squares close to the square encompassing the current waypoint, taking into account the distance $K(X, Y)$.

The upcoming sections of the paper cover the following content. Section 2 introduces the methodology employed for path planning, clarifying the use of the rough mereological potential field. Section 4 provides a detailed description of the experimental setup. Finally, Section 5 presents a concise summary of our publication.

2. Methodology

This section will thoroughly examine various methods employed in developing a robot navigation system aimed at navigating through rough mereological potential fields.

2.1. Square fill algorithm. In this chapter, we will present our interpretation of the square fill algorithm introduced by Ośmiałowski (2011). This algorithmic method has undergone modifications, as discussed by Polkowski *et al*. (2018), Zmudzinski and Artiemjew (2017) or Gnyś (2017). First of all, we assume that the shape of the field is a square whose size depends on the size of the robot, and we have a set of obstacles O where

each obstacle is a square of a given size. The fundamental steps to initialize the algorithm and the obtained results are outlined below:

1. Initiate the values:
   - Set the current distance to the goal, $d = 0$.
     The distance value initialized as 0 defines the distance from which we start generating potential fields. The initial value allows us to show that the potential fields start forming exactly from the point where the target is. With each subsequent iteration of the loop, the distance value will increase accordingly, allowing the fields to be dispersed across the map, moving proportionally away from the target.
   - Set the algorithm direction to *clockwise*.
     The definition of the direction in which we create potential fields is crucial. By declaring logical variables clockwise/anticlokwise it is possible to avoid the problem of potential fields dispersing only in a specific configuration.
2. Define an empty list F that will contain a set of potential fields meeting the conditions described in Step 4 of the algorithm.
3. Define an empty queue $Q = \emptyset$. The list Q was created to store information about potential fields.
4. Insert the first potential field, expressed as $p(x, y, d)$, into the generated queue Q. In the first iteration, the goal coordinates are assumed as the initialization point of the algorithm. In subsequent loop iterations, the point that next appears on the Q list will be taken into account. Here, x and y represent the location coordinates of the already created field, and d signifies the current distance to the goal,
$$Q = Q \cup \{p(x, y, d)\}.$$
5. Iterate over the elements in Q:
   (a) In the second iteration change the distance to $d = 5$.
   (b) Identify neighbors influenced by the current direction:
   If *clockwise* is true then
$$N = \begin{cases} p_0 = p(x-d, y, d), \\ p_1 = p(x-d, y+d, d), \\ p_2 = p(x, y+d, d), \\ p_3 = p(x+d, y+d, d), \\ p_4 = p(x+d, y, d), \\ p_5 = p(x+d, y-d, d), \\ p_6 = p(x, y-d, d), \\ p_8 = p(x-d, y-d, d). \end{cases}$$
   If *anticlockwise* is true then
$$N' = \begin{cases} p_0 = p(x-d, y-d, d), \\ p_1 = p(x, y-d, d), \\ p_2 = p(x+d, y-d, d), \\ p_3 = p(x+d, y, d), \\ p_4 = p(x+d, y+d, d), \\ p_5 = p(x, y+d, d), \\ p_6 = p(x-d, y+d, d), \\ p_8 = p(x-d, y, d), \end{cases}$$
   where x and y represent the current first element of Q.
   (c) During the formation of potential fields, it is possible to create several identically located potential fields. Compute the Euclidean distance from the previous and existing potential field neighbors in the Q list to eliminate redundancy.
   - If the Euclidean distance of the current potential field $p_k(x, y, d)$ is less than 15 and $p_k(x, y, d) \in F$, or $p_k(x, y, d) \in O$, where O is the set of obstacle coordinates, or $p_k(x, y, d) \in Q$, then reject the current field and return to Step 5(b).

   (d) Insert the potential field neighbours of $p_k(x, y, d)$ at the end of list Q,
   (e) Increase the distance to the goal,
$$d = d(p_k) + 0.1.$$
   (f) Invert the direction (*clockwise* to *anticlockwise, anticlockwise* to *clockwise*)
   (g) Remove the current element $p_k(x, y, d)$ from the queue Q and append it to the list of potential fields, F.
   * After each iteration go back to Step 5(b). The algorithm terminates when the queue Q is empty.

According to the outlined approach, the distance is initialized as 0. This value will increment in subsequent iterations, as our algorithm generates potential fields from the goal towards the starting point. The only exception is the first iteration of the algorithm, where the distance changes from 0 to 5. Applying this operation will force the algorithm to create all possible neighbors that are closest to the target point and are necessary to start exploring the map (distance = 0). By significantly changing the distance in the second iteration of the algorithm, we will avoid excessive crowding on the map around the target, without disturbing the algorithm (distance = 5). In the next iterations, the distance value starts

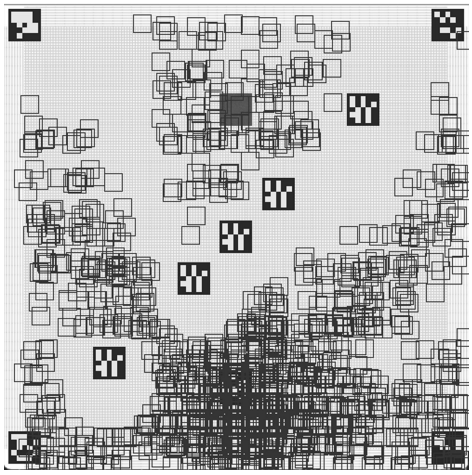


Fig. 1. Rough mereological potential field is generated using the square fill algorithm. The edge AR markers outline the map borders, while the central markers signify obstacles. The marker beneath the drawn squares designates the goal. The one-color painted square indicates the robot's initial position.

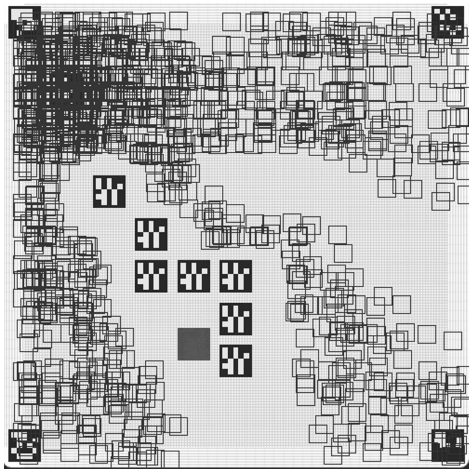


Fig. 2. Square fill algorithm used to generate a rough mereological potential field. The AR markers along the edges define the map boundaries, while the central markers represent obstacles. The marker beneath the drawn squares in the top-left corner indicates the goal. The solid-colored square marks the robot's starting position.

to increase proportionally in Step 5(e). The generated potential fields are denoted as $p(x, y, d)$, where x and y represent the coordinates, and d signifies the distance value. This distance value plays a crucial role in creating new neighbors during the operation.

When considering the Euclidean distance conditions (Step 5(c) of the algorithm), the number 15 is used, which avoids excessive accumulation of potential fields around the target. The value 15 is the minimum value that does not break the continuity of the algorithm. A larger distance indicates that the potential field is generated farther from the goal. The values 15 and 0.1 used to increase the distance correspond to maps obtained from a camera with a frame rate of 30 fps, with a resolution of 640×480. The size of the map we work on is not constant. It all depends on the position of the markers defining the map boundaries. By placing AR markers on our workspace in the lab, the size of the map for experiments was in the range of 450×500. Furthermore, in each iteration, the direction of generating neighbors must be altered to avoid stagnation and ensure the exploration of the entire map. Figures 1–3 show how a mereological potential field is generated using the square fill algorithm for different obstacle and target alignments.

2.2. Circle fill algorithm. Instead of using squares in our algorithm, we investigate the option to generate a force field using circles (Ośmiałowski, 2011). Following the generation and testing of several map combinations, clear differences were not evident. In the upcoming section concerning path generation, we will investigate whether there are any distinctions between the resulting tracks. Figures 3–8 show how a mereological potential field is generated using the circle fill algorithm for different obstacle and target alignments. After analyzing the performance of the circle fill algorithm, no significant differences were found when comparing it with the method using square field shapes.

3. Obtaining a path using potential fields

The potential fields obtained during the compilation of the square circle fill algorithm are used to find a path for the mobile robot. The generated mereological potential fields explored the map, avoiding defined obstacles and respecting the board boundary. In this section, we will discuss the individual steps of obtaining the path. First, we will apply filters to select potential fields, then we will smooth the generated path.

3.1. Path finding. To determine our path, we initially use a variation of the algorithm proposed by Ośmiałowski (2011), known as the path search algorithm. The algorithm operates within the provided potential field and is outlined as follows:

```
IF
robot location is equal or very
close to goal position:
END
APPEND the current field to the closest
points robot list
WHILE target is not found:
IF
distance between the current point
```

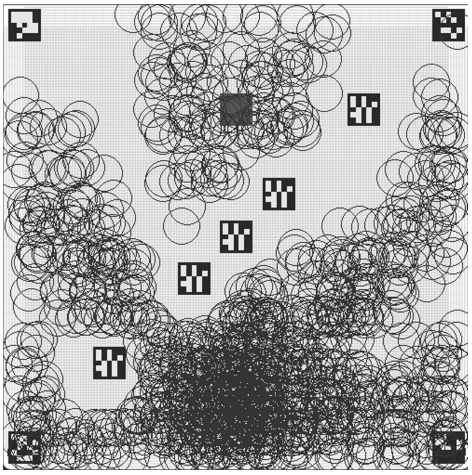


Fig. 3. Rough mereological potential field is generated by the square fill algorithm but with circles instead of squares. As in Section 2.1, the edge AR markers represent the borders of the map, and the central markers represent obstacles. The marker situated in the most occupied area on the map designates the goal. If attention is directed to a color-painted square, it indicates the robot's initial position.

```
and the next potential field is smaller
than the currently smallest value
THEN
APPEND the current field into
closest points robot list
ELSE
DROP this field and go to the next
potential field
END
```

The path finding algorithm uses a list of potential fields and the calculated Euclidean distance as a selection criterion. The function starts by checking if the currently considered point is close enough to the target – if so, it terminates. Otherwise, in each iteration it calculates the distances between the current point and all available fields from the considered list, moreover taking into account the goal point. The field with the minimum distance had been chosen as the next point in the path. The selected field is added to the path and then removed from the list of potential fields so that the algorithm can continue the process in subsequent iterations until the target is reached or the number of iterations is exhausted. In practice, we proceed from the starting point one by one, always choosing the closest potential field, approaching the target in small steps by using the weighted Euclidean distance.

The generated path for both map filling algorithms (square and circle fill algorithms) are in Figs. 5–8.

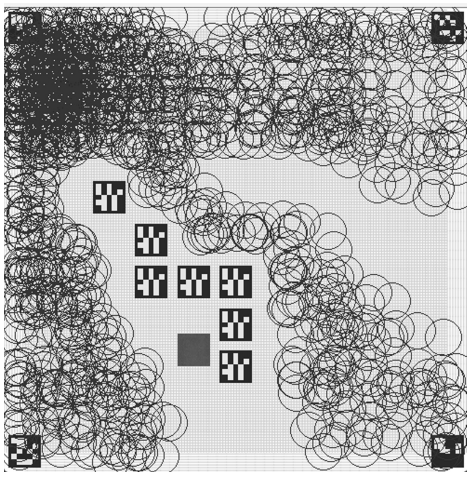


Fig. 4. Different map with generated potential, circle shape fields.

3.1.1. Variation in the Euclidean distance: The weighted Euclidean distance. To measure the distance between the current point and potential fields, we employed the Euclidean distance

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2},$$

in the two-dimensional plane where p represents the point with potential field coordinates and q describes the point with goal coordinates. To enhance the accuracy of pathfinding, we employ two Euclidean distances. One is focused on the distance between the current potential fields, referred to as the classical Euclidean distance. The second, used for generating the first path, is named the weighted Euclidean distance. The weighted distance takes into account the distance between the goal and the current point. Based on this result, we apply a weight (by multiplying it by the appropriate floating-point number, which is defined as 0.4) between two potential fields. The mentioned weight is a numerical factor that is attached to all observations appearing in a function describing a specific potential field to indicate varying degrees of importance. We use the weighted distance to avoid selecting a suboptimal path, preventing a direct jump to the target without avoiding obstacles.

3.2. Operation of field filtering: Optimization. The path obtained from the path search algorithm is not as optimal and clear as expected. To reduce noise in the path, we applied a filter for path optimization. This filter focuses on the Euclidean distances between points on the generated path and the goal. The main condition is as follows: Starting from the first element of the path, we calculate the Euclidean distance between the currently

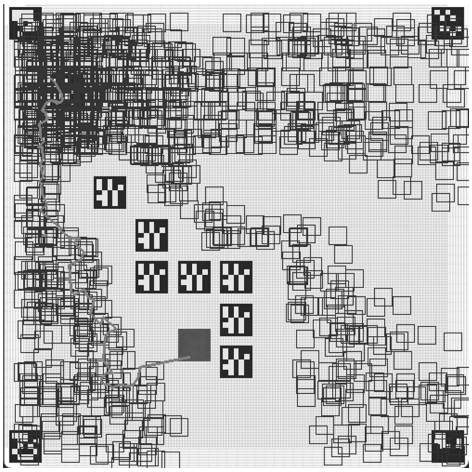


Fig. 5. Generated path using the path search algorithm from start point (one-color painted square) to the goal (marker located under the biggest amount of drawn squares), concerning the fields in a square shape.

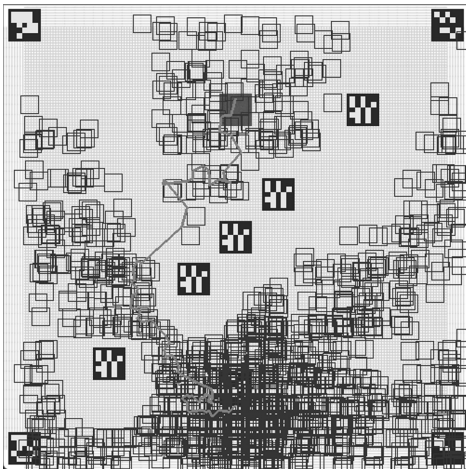


Fig. 6. Path, generated using a path search algorithm, extends from the starting point (solid-colored square) to the goal (marker positioned beneath the largest cluster of drawn squares), while considering square-shaped fields.

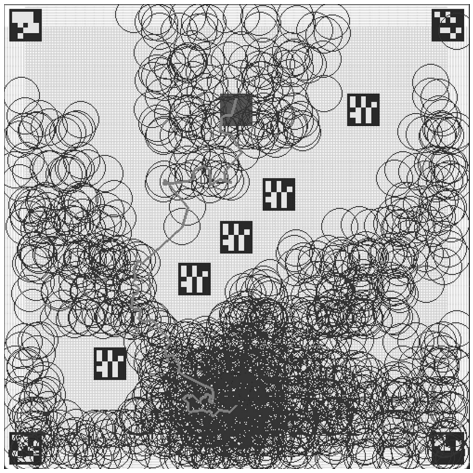


Fig. 7. Path, computed using a path search algorithm, starting from the initial position (solid-colored square) to the goal (marker located beneath the highest-density cluster of drawn circles), considering circular-shaped fields.

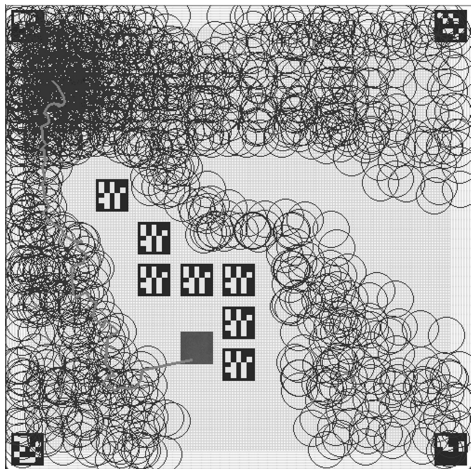


Fig. 8. Path, generated by a path search algorithm, stretches from the initial position (solid-colored square) to the goal (marker situated beneath the densest grouping of drawn circles), taking circular-shaped fields into account.

considered point of the path and the target. If we have the same or greater distances from the current points to the target, we skip those points. If we have multiple points with the same distance, we need to calculate the distances from the points of the next neighbors to the target and compare them. The points with the smallest distance values will be kept. Finally, we get a sorted and reduced list of points, with the help of which we will get a clear path from the starting point to the target.

That optimization process significantly refines the path, ensuring a clearer and more efficient trajectory for the robot. By filtering out points that do not contribute to a more direct path to the goal, we enhance the overall path quality. This step is crucial for improving the robot's navigation and ensuring it follows an optimal trajectory through the environment. Below we show the most important conditions in the function responsible for path filtering.

```
for i in range(len(path)):
  euclid_dist = d_eucl
  (path[i][0], path[i][1],
  goal_coord[0][0], goal_coord[0][1])
        .
        .
    if euclid_dist < path_distances[-1]:
      optimal_path.append(path[i])
      path_distances.append((euclid_dist))
```

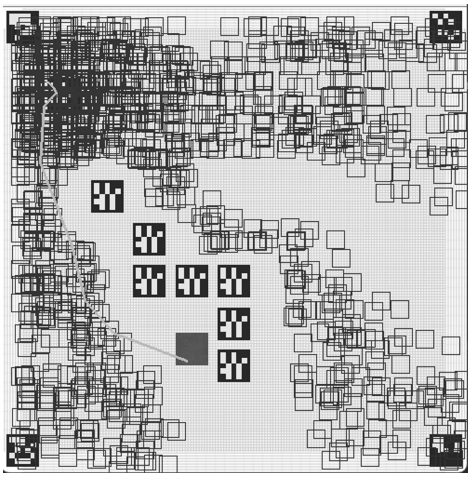


Fig. 9. Paths improved through optimization filtering with respect to square-shaped fields.

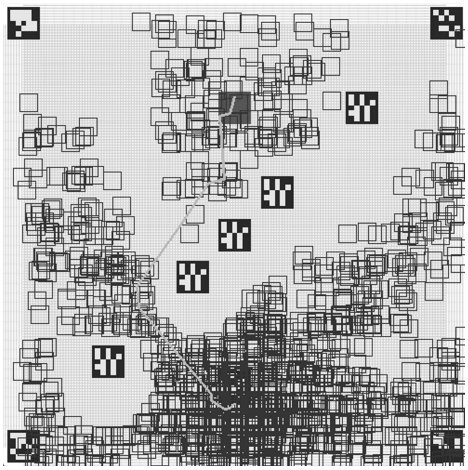


Fig. 10. Path after applying optimizing filtering based on square shape fields.

```
if euclid_dist == path_distances[-1] and
i+ 1 < len(path)-2:
  euclid_superset = d_eucl(path[i+1][0],
  path[i+1][1], goal_coord[0][0],
  goal_coord[0][1])
  if euclid_superset < euclid_dist:
    optimal_path.pop(-1)
    optimal_path.append(path[i])
    path_distances.append
    (euclid_superset)
  if euclid_superset == euclid_dist and
    i + 2 < len(path)-2:
              .
              .
return(optimal_path)
```

We can see the effect of the filtering in images in Figs. 9–12.

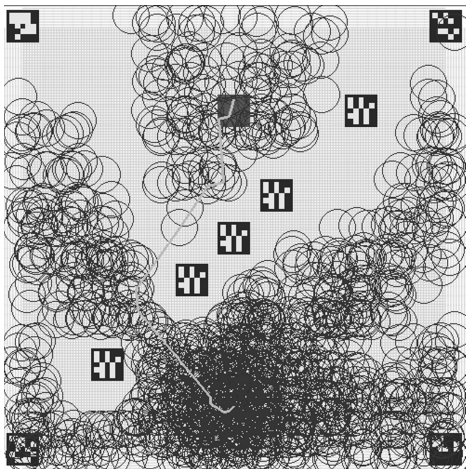


Fig. 11. Paths improved through optimization filtering using circle-shaped areas.

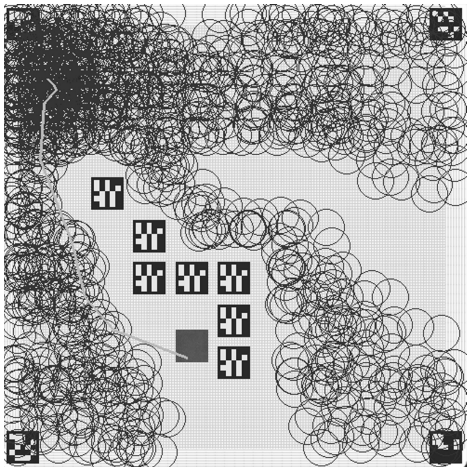


Fig. 12. Paths enhanced by applying optimization filtering tailored to circular-shape fields

3.3. Path smoothing. After visualizing the optimal path from the robot's starting location to the target, we initiated the path smoothing algorithm (Zmudzinski and Artiemjew, 2017) to further optimize our path. We iteratively apply the algorithm n times until the result and path shape are satisfactory:

1. We minimize the distance between points by employing the variable α, which determines how quickly we move away from the original position x_k, considering the preceding point x_{k-1} and the subsequent point x_{k+1},

$$x_k = x_k + \alpha(x_k - 1 + x_k + 1 - 2x_k).$$

2. Next, we balance the point x_k by applying the variable β and calculating y_k, representing the new position of the point. This operation helps us avoid straight lines in the path,

$$y_k = y_k + \beta(x_k - y_k).$$

The discussed method of path modification leads to the creation of a piecewise linear function, the construction of which consists in iteratively transforming the points x_k by taking into account the neighboring points $x_k - 1$ and $x_k + 1$. Each point on the path is adjusted to minimize the distance between successive points, which creates a series of rectilinear segments between points. As a result, each segment connecting successive points can be described as a straight segment, which results in a path structure composed of linear segments. Using an additional smoothing operation using the parameter β introduces point balancing, minimizing the sharpness of the angles between neighboring segments and giving the path a smoother character. Reducing sharp breaks makes the path resemble a continuous line, eliminating sudden changes in direction.

The effect of smoothing the path can be seen in Figs. 13–16.

4. Experimental results

In this section, we will discuss the technical side of our experiments finalizing it by sharing a demonstration of our project in a real environment. Also, we will show difficult cases, according to the results given by the path search algorithm.

4.1. Difficulties in the path search algorithm. While testing different types and setups of the map, we encountered a critical case, cf. Figs. 17 and 18. It turns out that there is a combination of map borders and obstacle locations when our algorithm could not find the path. We are speculating that the phenomenon appears when there are no more attractive potential fields taking into account the weighted and normal Euclidean distances. This means the rest of the possible potential fields have a bigger distance to the goal than the current, cf. Section 3.1.1. Another disadvantage is that the algorithm works on the elimination principle. This means that once a potential field is selected for generating a path, it is removed from the list of possible potential fields. It has been observed that the algorithm does not cope when the map resembles a maze. Despite the correct generation of potential fields, the selection of fields used in the path finding algorithm does not take into account the backward action.

4.2. Technical aspects of the experiments. We conducted real-world testing in an intelligent robotics laboratory using key devices, including a top camera for capturing point coordinates and a Smart Element Hub cube from the Lego Robot Inventor kit. For semi-autonomous control, where the computer serves as the computing unit, we used our custom Python library,

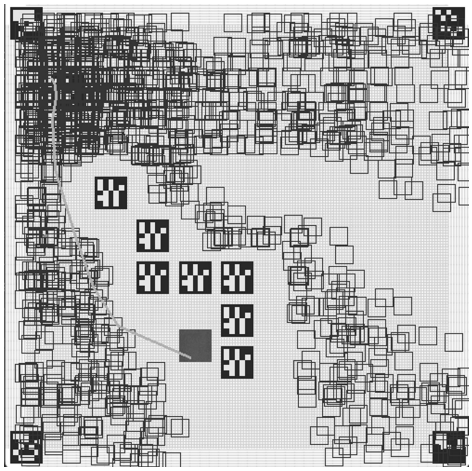


Fig. 13. Our previous filtered path based on square shape potential fields after using path smoothing algorithm .

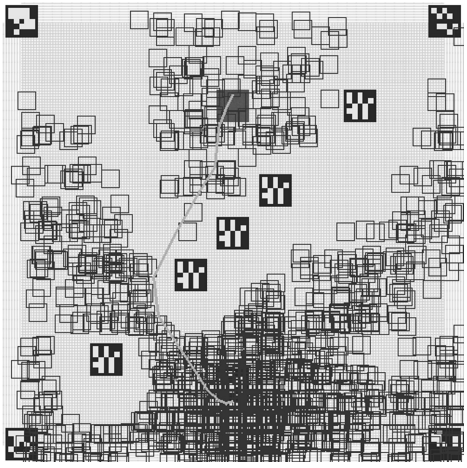


Fig. 14. Path, previously filtered using squared potential fields, improved through the path smoothing algorithm.

accessible through the work of Cybowski (2025). The validation code is available on GitHub (Szpakowska, 2025a).

4.3. Map generation using augmented reality markers.. To assess the described algorithms, we needed to create a customized environment for our mobile robot. To implement our concept of the robot's world map, we employed AR markers to establish map boundaries, define obstacles, and designate the goal point. Additionally, the robot's location was represented by a one-color square applied on top of the machine. Specific elements on the map were configured to facilitate the testing process:

- Boundaries (4 markers),
- Obstacles (2 markers),
- Designed goal (1 marker),

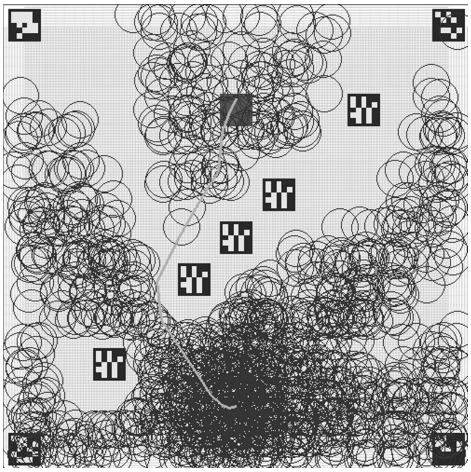


Fig. 15. Previously filtered path, derived from circular potential fields, after applying the path smoothing algorithm.

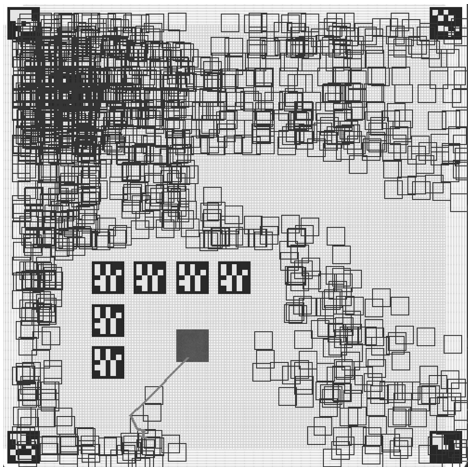


Fig. 17. Critical case visualize using square shape potential fields.

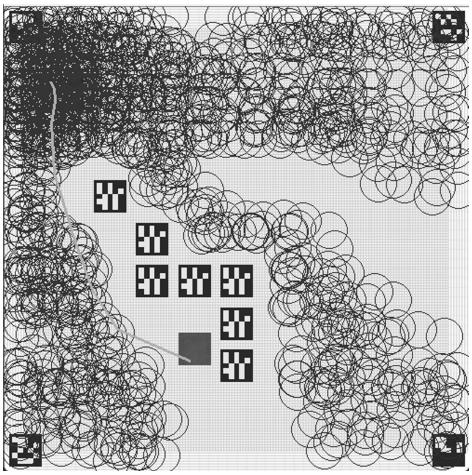


Fig. 16. Filtering path has undergone smoothing with the path smoothing algorithm.

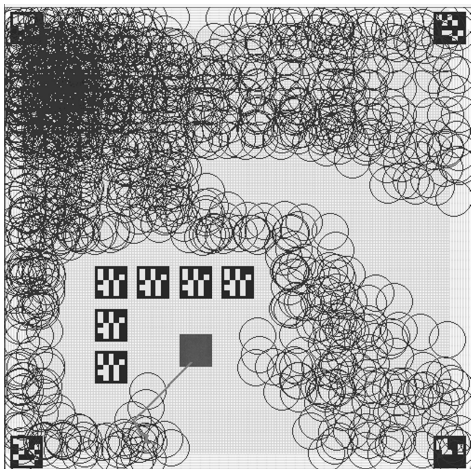


Fig. 18. Critical case representation utilizing a circular potential field.

- Robot position (one color square).

To identify the listed elements and enable the definition of each position on the virtual map, we used OpenCV (2025) and Python AR markers (Brauer and Rouneau, 2025).

After obtaining the camera view, we had to customize our window. The coordinates of all points were now determined by the pixels captured by the camera, shifting the map for the robot agent to start from $(40, 46)$ instead of the origin $(0, 0)$. To adjust the top-left border point, we implemented the following mapping:

```
for i in range(len(x1)):
    a = x1[i] - x_min
    x_borders.append(a)

for i in range(len(y1)):
    a = y1[i] - y_min
    y_borders.append(a)
```

where $x_{\min}$ determines the minimum of X values, $y_{\min}$ is the minimum y value in border points (x, y).

4.4. Connection to the robot and basic features. For our test, we assembled a robot using a Smart Element Hub cube with an LED screen from the Lego Robot Inventor kit and servomotors, using the Python language. The crucial step to commence working with this robot was the implementation of a library containing the necessary functions (Cybowski, 2025).

The main features of the robot are the following:

- the current position (obtained from the camera),
- the design goal (a sequence of points on the path to the target),

- compass sensor access (built in the hub),
- current direction access (read from the hub),
- motor access (Ports A and B).

The initial step involves establishing a Bluetooth connection between the robot and our device. To achieve this, we needed to select the appropriate ports, either COM3 or COM4, depending on the port assigned to our robot. Then, we had to map the robot's position point from the camera to the defined map, normalizing it as the center of the one-color painted square located on top of the machine, as well as all the boundaries and obstacles outlined in the upper part. We achieved this by establishing a new point $(0, 0)$ defined by reading from the camera the coordinate of the map boundary having the smallest values of the x and y coordinates. This action was necessary for the correct functioning of the P-controller.

4.5. Lego library. In this section, we will focus on the invented library, responsible for communication with the Lego robot, which was used in experiments. The library, named $le_mind_controller$ and available at the following address as open source software (Cybowski, 2025) enables communication and control of the hub, a key component of the Lego Mindstorms set number 51515.

In addition, it should also work with the hub from the Lego Spike Prime set numbered 45678, because the hubs in the two sets differ only in external appearance, but tests have not been conducted on the hub from the 45678 set. Connection to the hub can be made via a USB cable as well as via Bluetooth. The type of connection does not affect the operation and use of the library.

The library is divided into four modules:

1. `Helpers.py`, which contains helper functions, responsible for listing the serial ports available on the system and for establishing a connection through the selected port. The open-source pySerial library, available at the following address: `github.com/pyserial/pyserial`, is responsible for the technical, operating system-dependent aspects of handling serial ports.

2. `MindComm.py`, which is responsible for formatting and sending control commands to the hub. It also receives responses and data sent by the hub and then directs them to a parsing function in another module. When sending commands, it is important to remember that each must contain an individual identifier. It is randomly generated, has a length of four characters, and consists of uppercase and lowercase letters, numbers, dashes or underscores. When the hub executes a command, it sends back

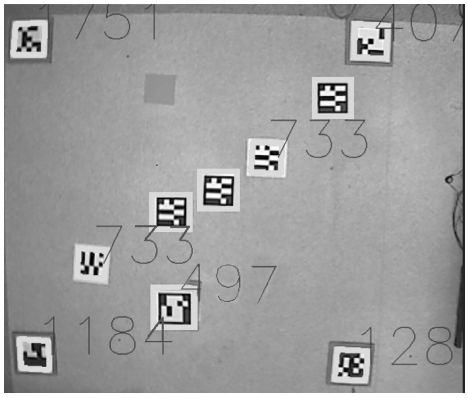


Fig. 19. AR recognition. The one color-painted point on a capture describes the start point—the robot's actual position.

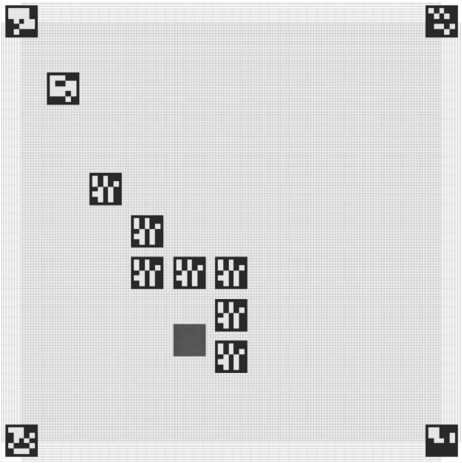


Fig. 20. Results of map generating including the start point as a one-color square.

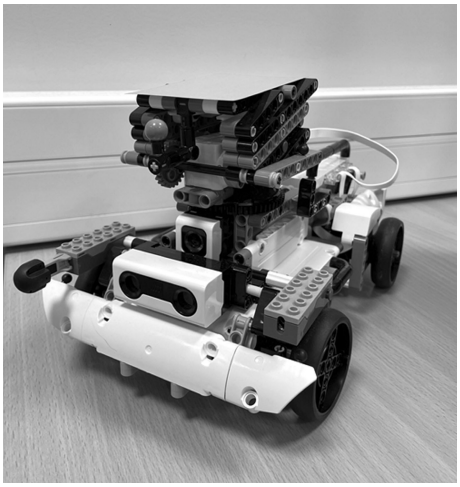


Fig. 21. Robot used in the experimental part, based on a Smart Element Hub cube with an LED screen of the Lego Robot Inventor kit.

a message with the same identifier as the command sent. This makes it easy to control the status of command execution.

3. `MindData.py`, which processes the data received from the hub, as well as contains the definition of constant values used by the modules connected to the hub and the hub itself, such as the color seen by the sensor or the type of module connected. The functions contained in this module allow us to easily obtain the information of interest from the hub itself as well as the modules connected to it.

4. `SerComm.py`, which is used to handle events related to the connection with the hub. Here one can find, among others, functions called in the case of connection loss or receiving a new line of data from the hub.

4.6. Controlling the driving part of the robot. The robot that we used to conduct the experiments is equipped with a compass sensor, which allows us to capture and fix the current robot's direction. Taking into account that our path is created and smoothed including obstacle avoidance, we get the set of coordinates that single variables represent the individual points of the optimized path. Using this set we can reach a goal. This experiment was working in real-time and our machine was localized by the top camera using a one-color recognition.

The main purpose of designing a P-controller for our robot is to make it move independently, taking into account the generated path. Below we focused on the main steps needed for our steering.

4.6.1. Conversion of the compass reading values. The proposed conversion is based on the current rotational direction of the robot concerning the built-in compass sensor value. The calculation spectrum covered angles from 0 to 359, degrees. This methodology segregated a given set, of angles into two subsets. Following the conversion, the range of values was adjusted to a spectrum from -180 to 180. The split was implemented as follows:

```
def convert(k,x):
    x = x - k
    if (x > 180):
        x = x - 360
    if ( x < -180):
        x = 360 + x
```

Suppose k represents the actual direction that needs adjustment relative to the robot's current position obtained from the camera, and x denotes the robot's current direction.

Following the conversion, angles ranging from 0 to 180 degrees exhibit positive values consistently, whereas values from 181 to 359 span from -179 to -1, respectively. These numerical representations correspond to the global directions:

- 0° North,
- 90° East,
- 180° South,
- -90° West.

The function *convert* returns the converted angle, taking into account both angles: the actual robot's direction (read from the hub) and the goal direction (elements in the path). Obtaining directions is described below.

4.6.2. Calculation of the actual rotation. In the conversion, we use the actual direction concerning the robot's current position. To compute this value, we employed the following property:

$$direction = \mathrm{Atan2}(x - y', y - x') \times \frac{180}{\pi}$$

Here, (x, y) is the current path element and the (x', y') is the actual robot position on map then.

```
act_dir = math.atan2(path_points_x[i] -
        - actual_y, path_points_y[i] -
        - actual_x) * 180 / 3.14
```

4.6.3. Control system. The control function employs a conversion function for angles and wheel speeds, treating them separately. The algorithm modifies the speed initially assigned to one of the wheels to initiate the rotation of the robot. The fundamental concept behind the applied algorithm is to minimize the speed of the selected wheel, determined by calculating the robot's new direction (Åström and Hägglund, 1995). When the velocity value on the right wheel is lower than the velocity value on the left wheel, the device turns to the right side. The same procedure is followed when turning to the left side.

```
cte = convert(actual_direction,
current_direction)
if cte <= 0:
    if abs(cte) > precision:
        wheel2 = (wheel1 * (-1))
    else:
        wheel2 = (wheel1 * (-1)) -
        (ksi * cte)
if cte > 0:
    if cte > precision:
        wheel1 = (wheel2 * (-1))
else:
    wheel1 = (Wheel2 * (-1)) -
```

```
        (ksi * cte)

    mc.motor_double_turn_on_deg
    (HubPortName.A, HubPortName.B,
    wheel1, wheel2,
    degrees=actual_direction)
```

Here, `wheel1` and `wheel2` indicate the speed of each robot motor. The parameter `ksi` specifies the corresponding reduction in wheel speed, taking a value within the range of 0.1 to 0.5.

4.7. Project access to codes and demonstration video. The action of our project in practice can be seen in the work of Szpakowska (2025b), in which we show the steps of our research and visualization of the results. The code of our library responsible for steering the robot is available in the work of Cybowski (2025). The rest of our codes and additional map configurations along with a demonstration of the algorithm operation are available in the work of Szpakowska (2025a). The AR tags we employed are available at through Brauer and Rouneau (2025).

5. Conclusions

In this research paper, we have successfully implemented path planning using the rough mereological potential field, coupled with a weighted Euclidean distance to the target. We developed and implemented a dedicated library designed for a specific mobile robot. Virtual reality markers were instrumental components used for mapping purposes. The control mechanism employed a P-controller. Through experimentation, we identified a critical case for our algorithm, highlighting areas for improvement in the current version of the path search algorithm. Our future plans involve addressing and resolving this critical case. Nevertheless, the primary objective of the work has been accomplished, as we have adapted the new library and hardware to enable optimal real-time navigation of the mobile robot across a map with obstacles. This study marks an initial step towards applying the rough mereological potential field for three-dimensional path planning.

References

Arkin, R.C. (1998). *Behavior-Based Robotics*, MIT Press, Cambridge.

Åström, K.J. and Hägglund, T. (1995). *PID Controllers: Theory, Design, and Tuning*, International Society of Measurement and Control, Research Triangle Park.

Brooks, R. (1986). A robust layered control system for a mobile robot, *IEEE Journal on Robotics and Automation* **2**(1): 14–23.

Brauer, M. and Rouneau, A. (2025). Detection of Hamming markers for OpenCV in Phyton, `https://github.com/DebVortex/python-ar-markers`.

Cybowski, W. (2025). Library for smart element hub cube lego robot inventor kit, `https://github.com/wcyb/le_mind_controller`.

Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G.A. and Burgard, W. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations*, MIT Press, Cambridge.

Gnyś, P. (2017). Mereogeometry based approach for behavioral robotics, *in* L. Polkowski *et al.* (Eds), *Rough Sets: International Joint Conference*, Lecture Notes in Computer Science, Vol. 10314, Springer, Berlin/Heidelberg, pp. 70–80.

Hwang, Y.K. and Ahuja, N. (1992). Gross motion planning—A survey, *ACM Computing Surveys* **24**(3): 219–291.

Kavraki, L.E., Svestka, P., Latombe, J.-C. and Overmars, M.H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces, *IEEE Transactions on Robotics and Automation* **12**(4): 566–580.

Latombe, J.-C. (1991). *Robot Motion Planning*, Kluwer Academic Publishers, Dordrecht.

LaValle, S.M. (2006). *Planning Algorithms*, Cambridge University Press, Cambridge.

OpenCV (2025). OpenCV computer vision library, `https://opencv.org/`.

Ośmiałowski, P. (2011). *Planning and Navigation for Mobile Autonomous Robots: Spatial Reasoning in Player/Stage System*, Polish-Japanase Academy of Information Technology, Warsaw.

Polkowski, L. and Ośmiałowski, P. (2008). A framework for multiagent mobile robotics: Spatial reasoning based on rough mereology in player/stage system, *in* C.-C. Chan *et al.* (Eds), *Rough Sets and Current Trends in Computing: 6th International Conference*, Springer, Berlin/Heidelberg, pp. 142–149.

Polkowski, L. and Skowron, A. (1996). Rough mereology: A new paradigm for approximate reasoning, *International Journal of Approximate Reasoning* **15**(4): 333–365.

Polkowski, L., Zmudzinski, L. and Artiemjew, P. (2018). Robot navigation and path planning by means of rough mereology, *2nd IEEE International Conference on Robotic Computing, Laguna Hills, USA*, pp. 363–368.

Raj, R. and Kos, A. (2022). A comprehensive study of mobile robot: History, developments, applications, and future research perspectives, *Applied Sciences* **12**(14): 6951, DOI: 10.3390/app12146951.

Sun, K. and Liu, X. (2021). Path planning for an autonomous underwater vehicle in a cluttered underwater environment based on the heat method, *International Journal of Applied Mathematics and Computer Science* **31**(2): 289–301, DOI: 10.34768/amcs-2021-0020.

Szpakowska, A. (2025a). Rough mereology potential field 2D algorithm, `https://github.com/aleksandraszpakowska/Recognition_opencv`.

Szpakowska, A. (2025b). Path planning using rough mereological potential field: Project demonstration, https://www.youtube.com/watch?v=hUHCbkKCDpY.

Szpakowska, A., Artiemjew, P. and Cybowski, W. (2023). Navigational strategies for mobile robots using rough mereological potential fields and weighted distance to goal, *in* A. Campagner *et al.* (Eds), *International Joint Conference on Rough Sets*, Springer, Berlin/Heidelberg, pp. 549–564.

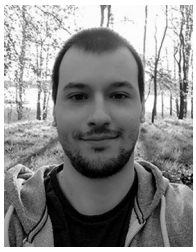
Zmudzinski, L. and Artiemjew, P. (2017). Path planning based on potential fields from rough mereology, *Rough Sets: International Joint Conference, IJCRS 2017, Olsztyn, Poland*, pp. 158–168.



Aleksandra Szpakowska was born in 1998 in Olsztyn, Poland. She earned her MSc degree in 2022 from the University of Warmia and Mazury, where she currently holds the position of an assistant professor. Her research interests revolve around autonomous aerial and mobile robotics, with the focus on path planning and decision making in three-dimensional environments. Her expertise includes rough sets, optimization algorithms, and artificial intelligence applications in robotic systems. She is particularly interested in the development of intelligent navigation strategies for drones, enhancing their autonomy in dynamic and uncertain conditions.



Piotr Artiemjew received his PhD and DSc degrees in computer science from the Polish Japanese Academy of Information Technology in Warsaw in 2009 and 2016, respectively. He is a professor at the Faculty of Mathematics and Computer Science, University of Warmia and Mazury in Olsztyn. Focusing on applied machine learning and decision support systems, he collaborates actively with the Polish AI Society, EurAI, the International Rough Set Society, and other AI-focused groups.



Wojciech Cybowski earned his MSc degree in 2020 from the University of Warmia and Mazury and currently works as an independent consultant in the field of cybersecurity. His expertise spans reverse engineering of hardware and software, electronic device design, and software development. He is also a frequent speaker at Poland's largest cybersecurity conferences, sharing insights on emerging threats, secure system design, and advanced security research. In addition to his consulting work, he explores various cybersecurity-related topics, including low-level programming, embedded systems security, and innovative approaches to cyber defense.

Received: 1 April 2024
Revised: 23 November 2024
Re-revised: 12 January 025
Accepted: 21 January 2025