electronics

Article

Bacterial Evolutionary Algorithm-Trained Interpolative Fuzzy System for Mobile Robot Navigation

Ferenc Ádám Szili [1], János Botzheim [2,*] and Balázs Nagy [1]

[1] Department of Mechatronics Optics and Mechanical Engineering Informatics, Faculty of Mechanical Engineering, Budapest University of Technology and Economics, 4-6 Bertalan Lajos Street, 1111 Budapest, Hungary; szilidam@gmail.com (F.Á.S.); nagybalazs@mogi.bme.hu (B.N.)
[2] Department of Artificial Intelligence, Faculty of Informatics, ELTE Eötvös Loránd University, Pázmány P. Sétány 1/A, 1117 Budapest, Hungary
* Correspondence: botzheim@inf.elte.hu

Abstract: This paper describes the process of building a transport logic that enables a mobile robot to travel fast enough to reach a desired destination in time, but safe enough to prevent damage. This transport logic is based on fuzzy logic inference using fuzzy rule interpolation, which allows for accurate inferences even when using a smaller rule base. The construction of the fuzzy rule base can be conducted experimentally, but there are also solutions for automatic construction. One of them is the bacterial evolutionary algorithm, which is used in this application. This algorithm is based on the theory of bacterial evolution and is very well-suited to solving optimization problems. Successful transport is also facilitated by proper path planning, and for this purpose, the so-called neuro-activity-based path planning has been used. This path-planning algorithm is combined with interpolative fuzzy logic-based speed control of the mobile robot. By applying the described methods, an intelligent transport logic can be constructed. These methods are tested in a simulated environment and several results are investigated.

Keywords: interpolative fuzzy system; bacterial evolutionary algorithm; neuro-activity; path planning

check for updates

Citation: Szili, F.Á.; Botzheim, J.; Nagy, B. Bacterial Evolutionary Algorithm-Trained Interpolative Fuzzy System for Mobile Robot Navigation. *Electronics* **2022**, *11*, 1734. https://doi.org/10.3390/electronics11111734

Academic Editor: Maysam Abbod

Received: 11 April 2022
Accepted: 25 May 2022
Published: 30 May 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

At present, there is a lot of research and development on robots. Depending on their function, robots can be used to perform different tasks, or sometimes, to work alongside humans. While robots are becoming increasingly common in the workplace to perform various tasks, it is expected that they will also be used in every-day life in a few decades. Moreover, robots must be prepared to live with humans. In this case, the right path planning is important for mobile robots to work among humans. This is relevant for recent mobile robots, which sometimes perform transport tasks. They need to be aware of their surroundings even if they are moving along a predefined route. The robot has to be careful to maintain the right speed, avoid obstacles, and also recognize clear terrain, so that it can move through it.

The task is to control the speed of the robot based on the safety of the environment. Such a control task can be implemented using fuzzy logic-based control. Fuzzy logic is an extremely useful tool in computer science because it provides a way to describe systems of great complexity. It also allows for the simple modeling of systems that are very difficult to be described mathematically. One of the first known applications of fuzzy logic-based control of a highly complex nonlinear system was developed by Mamdani et al. to control a complex steam engine [1].

Fuzzy logic uses rules to describe particular systems or operating mechanisms, which are unique to each system. When modeling a system, these rules must be constructed. To describe more complex systems or handle all possible inputs, in some cases, a large number of fuzzy rules are needed to describe the system. Control based on fuzzy logic

can be applied to many control tasks; it can be applied to control different electronic devices, such as battery chargers [2], or different modern energy resources [3,4]. There is also a known application concerning the IoT, which helps optimize network efficiency [5]. Currently, there are also many known applications of fuzzy control for the control of various automated vehicles. There are examples for controlling underwater and flying machines as well [6,7], and there are lots of applications for the navigation of mobile robots [8–13]. Most of these applications use a large number of fuzzy rules, and within these sets, in many cases, the sets are also Ruspini-partitioned to cover the entire domain of interpretation. However, to reduce the size of the rule base, fuzzy rule interpolation can be used for inference, which can also handle transitions between inputs defined by rules. The first fuzzy rule interpolation procedure was proposed by László T. Kóczy and Kaoru Hirota [14–16]. Fuzzy rule interpolation has been successfully applied to driverless forklift control [17]. It can also be used in a similar way to control the speed of a mobile robot.

Building a fuzzy rule base can be accomplished manually by experiments, or it can be based on expert knowledge. However, it can also be accomplished programmatically. Examples include neural networks [18], swarm intelligence algorithms [19], and the bacterial evolutionary algorithm is another possibility [20]. The advantage of the bacterial evolutionary algorithm is that it can be used for optimization tasks; in many cases, it can find the optimal or near-optimal solution. The bacterial evolutionary algorithm can also be applied to construct an interpolative fuzzy rule base, with few differences compared to a traditional rule base's construction. Such an algorithm will be exploited in this paper.

For mobile robotic applications, path planning is often a feasible task. For example, a mobile robot may need to find a path to a given target position without hitting walls or falling down stairs. There are several procedures for path planning that work [21]. Traditional path-planning algorithms are known, such as A*, D*, and Dijkstra algorithms [22–24], but some special methods can also be used, such as neural networks [25,26], ant colony algorithms [27], and even bacterial memetic algorithms [28–31], which have all been used for this purpose. However, there is a particular solution, called neuro-activity-based path planning, that works on the principle of spiking neural networks; this solution is similar to the way the human brain works [32]. It is a very complex model, with many parameters and relations, but it provides path optimization and dynamic path planning, and can be used on two and three-dimensional maps (it is developed primarily for three-dimensional maps). The presented transport logic incorporates this model and takes advantage of the possibilities offered by parameterization. This path-planning method is integrated with a speed control, based on fuzzy logic, that includes obstacle avoidance.

This paper aims to present multiple innovations. A bacterial evolutionary algorithm, fuzzy rule interpolation in vague environments, and neuro-activity-based path planning are separately known and applied. One of the new topics is speed control for mobile robots using fuzzy rule interpolation, which involves the use of fuzzy rule interpolation instead of large amounts of fuzzy rules, which is currently more common [8–13]. The other topic is the construction of an interpolative fuzzy rule base using a bacterial evolutionary algorithm. Several applications are known for building and optimizing a fuzzy rule base using a bacterial evolutionary algorithm [20,33,34], but none of these are applied to build an interpolative fuzzy rule base. The third topic is the integration of neuro-activity-based path planning with interpolative fuzzy speed control. In this context, the aforementioned known path-planning procedure [32] is applied to a two-dimensional map and supplemented with interpolative fuzzy velocity control.

The structure of the paper is as follows. In Section 2, related works to path-planning and decision-making systems are introduced briefly. In Section 3, the main challenges are highlighted. Section 4 describes the structure of the models used in this research. The experimental results for the application are presented in Section 5. Section 6 concludes the paper.

2. Related Works

Decision making is a complex task, and implementing such mechanisms into mobile robots is even more challenging. To implement decision-making algorithms, the fuzzy rule-based system was introduced [35]. Fuzzy systems can model the decision-making process of humans. According to the rules previously stored in the rule base, explainable decisions can be made. However, the robustness of the systems requires a big database with many rules, which means storing these databases is memory-consuming, and iterating through the rules in the base is time- and computation-consuming.

To address this problem, several applications have emerged. In the case of mobile devices, a widespread solution is to use cloud computing [36], or to offload the computation to an external computer [37]. The offloading strategy can even utilize deep learning-based approaches to optimize the process. However, there is a main disadvantage of this method. The process heavily relies on the usually wireless connection technology between the autonomous agent and the external computing unit. If a problem occurs regarding the connection, a loss in functionality can be present. A more robust approach is to reduce the computation cost of the decision-making algorithm so it can be implemented on the autonomous agent. In this manner, a fuzzy rule base should be operated with fewer rules. The reduced rule base can be achieved by utilizing bacterial evolutionary algorithms for optimization. The reduced rule base could lead to a decline in precision. To maintain the quality of the decision and the robustness of the application, an interpolative fuzzy decision-making algorithm is proposed.

Path planning is a challenging task which includes a lot of decision-making and optimization tasks. From this point of view, path planning is a good test-case scenario for testing and validating the method proposed in this article. A wide range of algorithms is known in connection with path planning [38]. The path-planning algorithms can be divided into several subgroups, such as global or local planning algorithms. The particle swarm optimization (PSO) algorithm [39] uses Bezier curves to optimize the path and to plan a smooth path with high-order continuity to aid in the motion control of the the mobile robots. Zhen et al. developed a bioinspired sparrow search-based algorithm [40]. The main goal of path planning is to find the quickest, shortest route, or to provide a continuous trajectory. However, in addition to the trajectory, the velocity of the moving agent could be an optimization parameter as well. In this research, the main optimization parameter is the velocity of the moving agent. Path planning with velocity optimization is the use case to test the interpolative fuzzy rule-based decision-making algorithm. Moreover, the computation efficiency of the algorithm makes it possible to use it on an autonomous agent without using cloud computing.

3. Problem Statement

The main goal is to develop an interpolative fuzzy rule-based transport logic that allows the robot to travel quickly but safe, similar to how a human would drive a car. On open roads, it can go as fast as it wants or as far as the traffic rules allow, but on more crowded or winding roads, it has to drive more carefully or it will crash into something or drift off the road. Similarly, the robot has to think in terms of priorities; that is, the more obstacles it sees in front of it, the more cautiously it has to proceed; or, in the case of a significant obstacle, such as a wall that is in the way, it must stop moving. However, as long as it sees no reason to be cautious, for example, if the terrain is clear, it can go at high speed.

All of this must be accompanied by the right trade-offs. It is safest to keep the robot moving very slowly so that it always has time to stop in the event of sudden obstacles. However, in some applications, it is advantageous if the robot can reach its destination faster, so it is necessary to find the right balance between speed and safety. It is also important to decide to what extent each obstacle poses a risk to the robot. For example, close landmarks are more of an obstacle than distant ones. Large obstacles pose a greater obstacle than small ones. How each rule will be enforced must be reflected in the control logic. The control logic is described using fuzzy rules. In order to ensure that the right

traffic is possible with as few rules as possible, fuzzy rule interpolation is used to facilitate decision making in each situation. A bacterial evolutionary algorithm is used to find the appropriate trade-offs to tune the fuzzy rules. This will ensure that transport is both fast and safe.

In addition, to ensure that the robot always travels at the optimum speed along a given path, transport also involves conscious path planning. There are several good methods for path planning, each with different advantages. Some help to plan a shorter path between point A and point B, but there is also an advantage in planning a path with less chance of colliding with objects. In other words, path planning must also find the right trade-off between speed and safety. The aim is, therefore, to develop a transport logic that allows the fastest and safest transport possible.

4. Proposed Method

The presented transport method consists of multiple components. Speed control is achieved by interpolative fuzzy control. The rule base for the control logic is constructed using a bacterial evolutionary algorithm. For path planning, the so-called neuro-activity-based path planning is used, which facilitates successful traffic management already at the planning stage.

4.1. Interpolative Fuzzy Systems

Fuzzy control is based on fuzzy set theory, in which the set of values belonging to the set is continuous instead of just 0 and 1. The control consists of rules whose input variables are the antecedents of the rule and whose output is the consequent of the rule. In practice, the inputs are the measurement results that characterize the state of the system, and the output is the intervention signal for the control.

In the case of fuzzy logic-based decision making, there may be cases where a sample does not fit into either set of antecedents but falls between them. In this case, neither of the two consequents is valid in the base case. However, the use of fuzzy rule interpolation provides the possibility of inferring an observation between the two sets of antecedents.

Fuzzy rule interpolation is illustrated by the following example. Given the following two rules:

$$R_1 : \text{If } A_1 = \text{“The tomato is red”, then } B_1 = \text{“Ripe”},$$
$$R_2 : \text{If } A_2 = \text{“The tomato is green”, then } B_2 = \text{“Unripe”}.$$

Furthermore, let the current observation be that $A^* =$ “The tomato is yellow”, whose intersection with both sets of antecedents is empty so that no inference can be made on it by most known inference methods. However, in theory, we know the notion $B^* =$ “Half-ripe”, and this inference can be given formally by fuzzy rule interpolation [16]. This mentioned example is illustrated in Figure 1 with fuzzy sets.

This inference procedure can also be used in the case of regulation, where the antecedent sets may not cover the full range of interpretation. If fuzzy rule interpolation is used for a control task, the control task can be solved by using fewer rules, simplifying the model. An important control example is driverless forklift control. In a 1999 publication [17], Szilveszter Kovács et al. showed how to control a driverless forklift using a method that applies fuzzy rule interpretation.

There are several examples of fuzzy rule interpolation strategies, one of the most basic being the so-called K–H (Kóczy–Hirota) rule interpolation procedure. The driverless forklift control model described by Szilveszter Kovács et al. uses a variant of this procedure [41], and the full name of the procedure is K–H rule interpolation in the vague environment.

In the vague environment, fuzzy models are described by scaling functions instead of membership functions. In addition, instead of membership values, the membership of an element in a given set is described by the indistinguishability of the elements. Two elements are ε indistinguishable if their distance is less than ε. The distance is the integral of the scaling functions.

$$\varepsilon > \delta_s(x_1, x_2) = \left| \int_{x_1}^{x_2} s(x)\, dx \right| \tag{1}$$

In Equation (1), δ_s is the distance between the two elements, x_1 and x_2 are the two elements, and $s(x)$ is the scaling function [41].

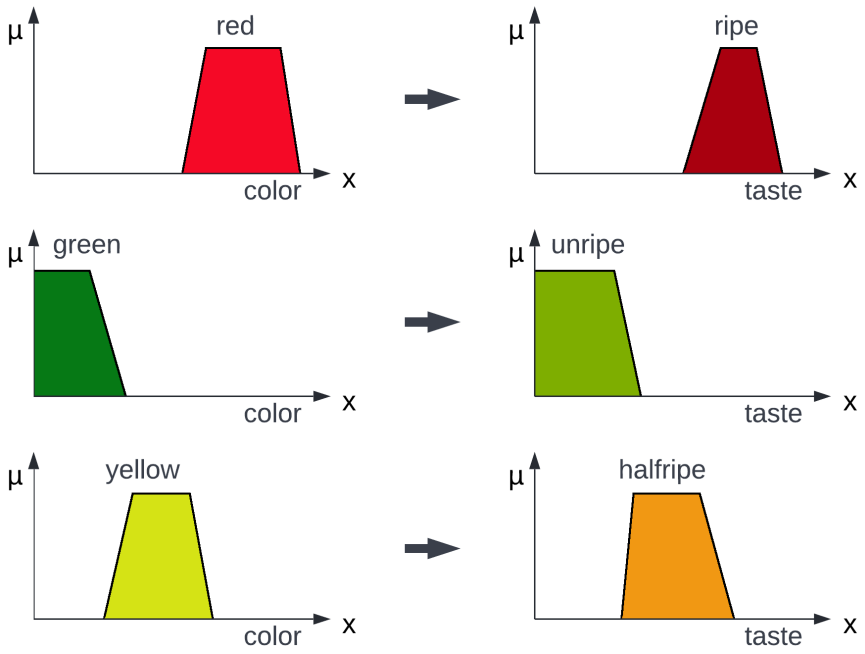

Figure 1. Example of a fuzzy rule interpolation inference.

The relationship between membership values and distance can be given by the alpha cut of a membership function, which is a set in which elements are indistinguishable from the element x_0 to the degree $(1 - \alpha)$. For any two elements of a given alpha cut, the following inequality (Equation (2)) holds [41]. The distance δ_s is illustrated in Figure 2.

$$\delta_s(x_1, x_2) \leq 1 - \alpha \tag{2}$$

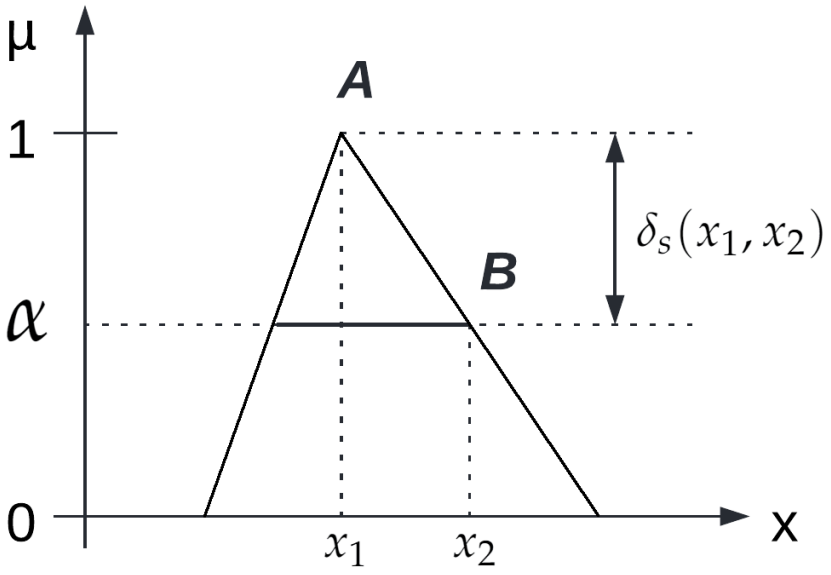

Figure 2. Vague distance.

The relationship between the membership value and the distance δ_s is described by Equation (3) [41].

$$\mu_{x0}(x) = 1 - \min\{\delta_s(a,b), 1\} = 1 - \min\left\{\left|\int_a^b s(x)\,dx\right|, 1\right\} \tag{3}$$

To specify the vague environment, it is necessary to find a suitable scaling function that gives a suitable description based on the fuzzy sets. The scaling function can be given based on the membership functions, as introduced by Frank Klawonn [42]. The relation between the membership function and the scaling function is described by Equation (4).

$$s(x) = |\mu'(x)| = \left|\frac{d\mu}{dx}\right| \tag{4}$$

The scaling function is the derivative of the membership function [41]. For this reason, membership functions with linear sections are equivalent to scaling functions with constant sections. Figure 3 illustrates an example of how a scaling function corresponding to a given membership function can be given by Equation (4).

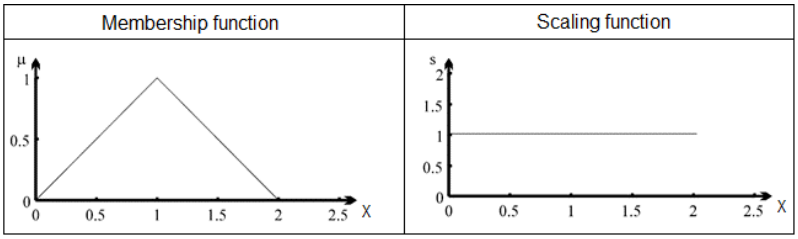

Figure 3. Fuzzy membership function (**left**) and scaling function (**right**).

When a fuzzy partition consists of more than one set, there is not always a scaling function that can describe the entire partition. Therefore, in these cases, so-called approximate scaling functions must be used to specify the fuzzy partition. The simplest way to write this is to use linear interpolation between neighboring sets. Triangular membership functions can be described by two constant sections, and if they have different slopes, the break point of the scaling function is at the core point of the membership function. Thus, such a membership function can be specified exactly by three values, namely, the x-coordinate of the core point and the slopes of the linear sections. In this case, the approximate scaling function can be given by linear interpolation of the slope of the right-hand segment of the left-hand set and of the slope of the left-hand segment of the right-hand set. This is illustrated in Figure 4, and is formally described by Equation (5).

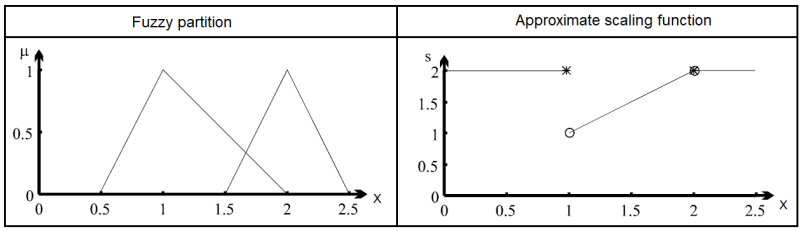

Figure 4. Fuzzy partition (**left**) and approximate scaling function (**right**).

$$s(x) = \left\{\frac{s_{i+1}^L - s_i^R}{x_{i+1} - x_i} \cdot (x - x_i) + s_i^R \mid x \in [x_i, x_{i+1}), \forall i \in [1, n-1]\right\}, \tag{5}$$

where $s(x)$ is the approximate scaling function, x_i is the location of the core point of the i-th set, s_i^R and s_i^L are the slopes of the right and left segments of the i-th set, respectively, and n is the number of sets in the fuzzy partition [41].

Fuzzy rule interpolation in vague environments is based on classical linear interpolation. In this context, if an observation infers a set of two rules, both consisting of an antecedent and a consequent set, the distances of the antecedent sets from the observation are as proportional to each other as are the distances of the consequent sets from the inference. This proportionality is expressed by Equation (6).

$$dist(A_1, x) : dist(x, A_2) = dist(B_1, x) : dist(x, B_2), \tag{6}$$

where $A_1 < x < A_2$, $B_1 < B_2$ and $R_i = A_i \rightarrow B_i$ $i \in [1,2]$. It is also assumed that the antecedent universe of fuzzy rules is multidimensional, while the consequent universe is one-dimensional, or multidimensional, but then decomposes into one-dimensional instances. Using the formula for the distance of the elements in the vague environment, i.e., Equation (1), the interpolation can be written according to Equation (7).

$$\sqrt{\sum_{i=1}^{m}\left(\int_{A_{1i}}^{x_i} s_{xi}(x_i)\,dx_i\right)^2} : \sqrt{\sum_{i=1}^{m}\left(\int_{x_i}^{A_{2i}} s_{xi}(x_i)\,dx_i\right)^2} = \left|\int_{B_1}^{y} s_Y(y)\,dy\right| : \left|\int_{y}^{B_2} s_Y(y)dy\right|, \tag{7}$$

where s_{xi} is the i-th scaling function in the m-dimensional antecedent universe and s_Y is the scaling function of the consequent [41].

The following procedure can be used for inferences based on fuzzy rule interpolation [41]. In this case, linear interpolation is extended to all rules in the rule base. The inference is given as a weighted average of the consequents, where the weights are inversely proportional to the distance between the observation and the antecedent(s) of the given sequence.

$$\delta_{sx}(x, A_i) = \sqrt{\sum_{j=1}^{m}\left(\int_{x}^{A_i^j} s_{xi}(x)dx\right)^2} \tag{8}$$

$$w_i = \frac{1}{(\delta_{sx}(x, A_i))^p} \tag{9}$$

$$\delta_{sy}(y_0, y) = \frac{\sum_{i=1}^{n} w_i \cdot \delta_{sy}(y_0, b_i)}{\sum_{i=1}^{n} w_i}, \tag{10}$$

where $\delta_{sx}(f,g)$ and $\delta_{sy}(f,g)$ are the distances of given points f and g on the antecedent and consequent sides, w is the weight factor, x is the observation, y is the inference, y_0 is the smallest element of the consequent basis set, and p is a factor that determines the sensitivity to the distant rules. Equation (10) gives the distance of the inference from the smallest element of the consequent universe, and from there we can calculate back what the inference (value of y) is. This interpolative inference is made by this procedure.

4.2. Application of Interpolative Fuzzy Systems for Controlling a Mobile Robot

Based on the requirements, the robot must control its speed based on the safety of its environment. So, the more or the larger the obstacles in the environment of the robot, the more cautiously it should move; however, if the road is unobstructed, it can move at a higher speed. To control the speed, the robot uses a fuzzy control consisting of two antecedents and a consequent. That is, it calculates the output based on two inputs. The output of the control is an intervention signal, which can be acceleration or deceleration. One input is the safety of the environment. This variable is calculated from LIDAR signals. LIDAR gives the distances of objects, but these need to be converted into safety information. The other variable is the current speed of the robot. This is needed because, for example, if the robot enters a dangerous area, the faster it moves, the more it has to brake if it needs to slow down.

To determine the amount and size of landmarks in the vicinity of the robot, the use of LIDAR is the best option. LIDAR can measure the distance of objects in 360° around

the robot by measuring their distance with a laser. In this case, the resolution of the sensor was 1°, so a total of 360 distance data points can be acquired. These 360 data points will be the measured signal, which will act as an input for speed control. The measured distance values must be converted into safety information. Figure 5 illustrates the distance safety calculation.

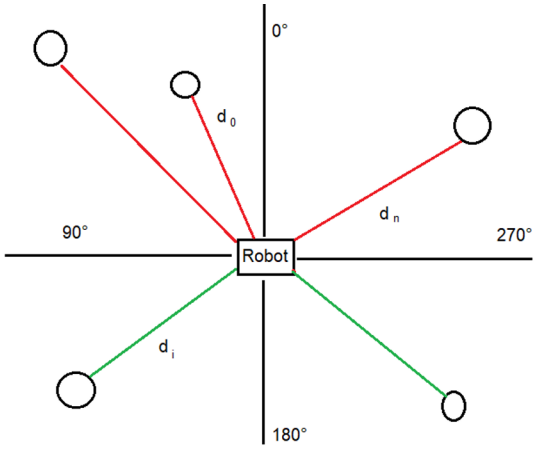

Figure 5. The calculation of safe distances.

The program evaluates the measured distance values. First of all, it is important to note that not all distance values are taken into account. Signals that indicate the distance of objects behind the robot are not taken into account. In the figure, these are the green rays; the red ones are taken into account. In addition, even those that are taken into account are not taken into account equally. The objects that are in front of the robot pose the greatest risk, and the more to the left or right of the robot they are, the less risk they pose since the robot can pass them. The distance of the objects is also taken into account because an object poses a greater risk depending on how close it is to the robot. All in all, the measured data are converted into safety information using Equation (11).

$$S = \sum_{i=0}^{90} \frac{\cos(\varphi_i)}{d_i} + \sum_{j=270}^{359} \frac{\cos(\varphi_j)}{d_j}, \tag{11}$$

where S is the degree of danger, and the higher the degree of danger, the more likely it is that deceleration is required. Here, d_i and d_j are the distance values measured at each angle, and φ_i and φ_j are the angles subtended by the 0° line in Figure 5.

The following points should be stressed in connection with this formula:

1. The robot has not yet left only those objects that are within the angular ranges 0°–90° or 270°–359°, so only within these ranges is the distance measured;
2. The dependence on the position of the objects is described by a cosine function because it takes its maximum value at 0°, and 0 at 90° and 270°, and takes non-negative values in the ranges given in the context;
3. Object distances are included in the denominator because the smaller the distances, the greater the risk.

The control uses the values of safety and speed to calculate an intervention signal using interpolative fuzzy inference, which represents some acceleration or deceleration, and this becomes the output of the control.

One of the advantages of fuzzy rule interpolation is that it guarantees that all observations include an intervention even if the sets do not form a Ruspini partition or do not cover the entire interpretation domain. So, if an observation that is received as input does not fall into one of the given sets of antecedents, but is instead in between them, then an intermediate consequent can be produced using rule interpolation. With fuzzy rule interpolation, it is possible to have fewer fuzzy sets or fewer rules to make a complete rule

base, and therefore, only three sets were used to describe safety, speed, and intervention. This model also comes with a table that shows which interventions are performed for which combinations.

The interpolative fuzzy rule base for speed control was constructed using a bacterial evolutionary algorithm. Figures 6–8 show the sets, and Table 1 shows the decision table. The identifiers of the consequent sets are also indicated here. The values in the table indicate the corresponding sets based on these identifiers. A value of −1 in this table indicates that there is no intervention for a given antecedent pair.

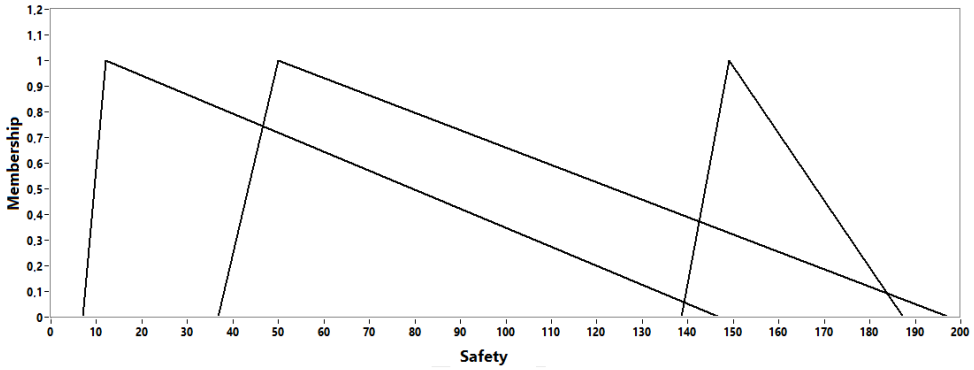

Figure 6. The sets of the safety variable; left set: safe; middle set: acceptable; right set: dangerous.

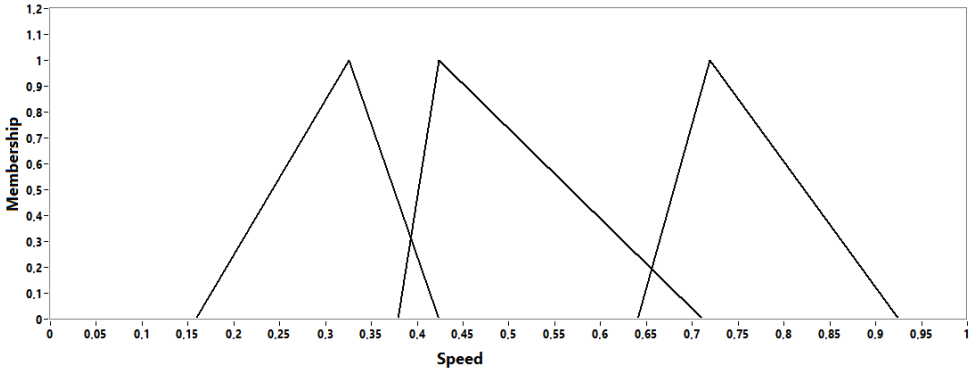

Figure 7. The sets of the speed variable; left set: slow; middle set: medium; right set: fast.

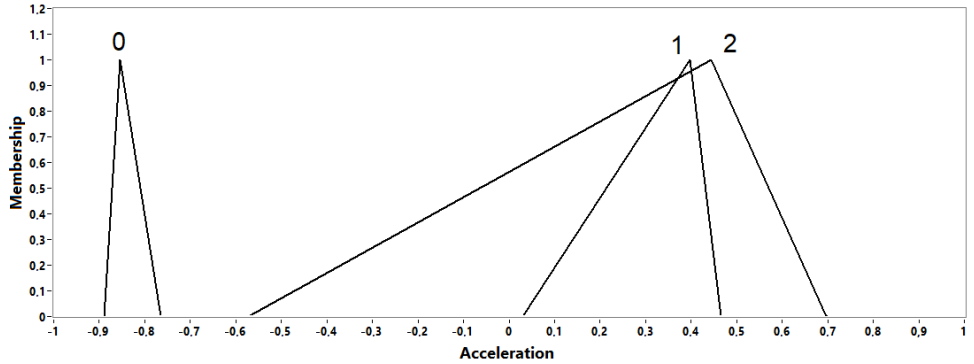

Figure 8. The sets of the consequent; 0: high deceleration; 1: low acceleration; 2: high acceleration.

Table 1. Table of control actions for each pair of antecedents.

	Safe	Acceptable	Dangerous
Slow	2	2	1
Medium	2	2	0
Fast	0	−1	0

4.3. Bacterial Evolutionary Algorithms

The bacterial evolutionary algorithm was developed in the second half of the 1990s [20], and is based on the evolution of bacteria. The idea of this method was developed based on the gene transfer ability of bacteria. Thus, one of the operators of the algorithm is the so-called gene transfer, which allows individuals in a population to pass on their traits to each other. The other is a bacterial mutation, in which the properties of each individual are randomly altered. The bacterial evolutionary algorithm is based on the process shown in Figure 9.

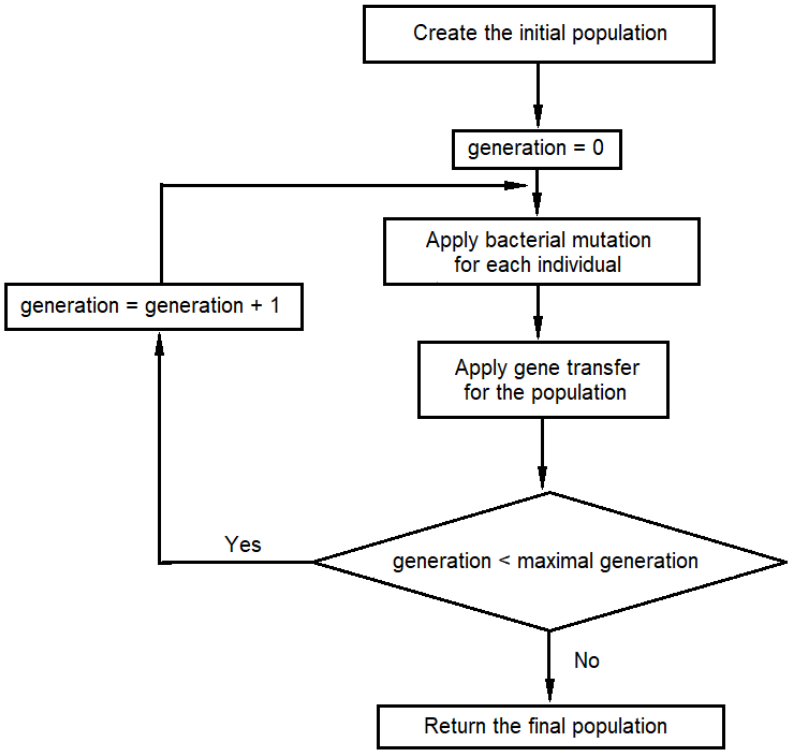

Figure 9. Bacterial evolutionary algorithm.

So, the first step is to create an initial population. This can consist of predefined or randomly generated individuals. The individuals then evolve over a given number of generations. Evolution is achieved by bacterial mutation and gene transfer. The bacterial mutation must be carried out on all individuals. In doing so, the algorithm tries to produce better individuals by randomly modifying the characteristics of the individuals. In gene transfer, the better individuals pass on some of their traits to the worse ones. The process stops when a given number of generations has been completed.

Usually, an individual is described by a sequence of numbers. Each number gives a particular property of the bacterium. The structure of a bacterium may be different for different tasks, but it should always be designed in such a way that the properties represent the solution sought for a given task. The bacterial evolutionary algorithm uses a cost function to determine how good an individual is for a given task. The lower the value of the function, the better the individual.

4.3.1. The Bacterial Mutation

The bacterial mutation operation must be performed on each element of the population individually. Figure 10 illustrates the process. During the operation, the next bacterium in line is copied each time. One of the input parameters of the algorithm is the number of copies of the individual made during a mutation. A gene (trait) is then selected and randomly altered for all but one clone, so that the original unaltered clone must remain.

After the change, the cost function must be performed for each copy. Whichever copy gives the best cost value will pass the modified gene to all other clones.

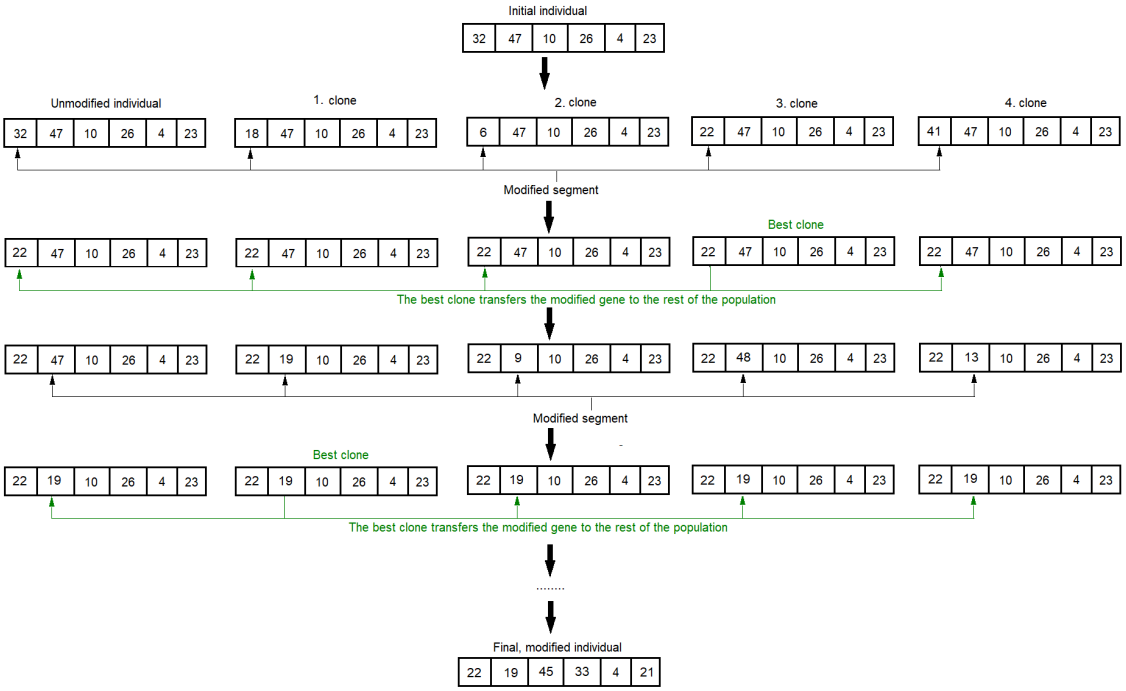

Figure 10. Illustration of a bacterial mutation.

The bacterial mutation lasts until all the genes have been modified. The original bacterium must then be replaced with the mutated bacterium, and the bacterial mutation must be carried out on the next bacterium. This must be repeated until all bacteria have been mutated. The process then continues with gene transfer.

4.3.2. Gene Transfer

Gene transfer is a population-level operation, so all individuals participate. It involves the transfer of genes (traits) from low-cost individuals, representing better solutions, to higher-cost individuals, thereby reducing potential divergence between individuals. The gene transfer process is illustrated in Figure 11.

Gene transfer starts by sorting individuals by cost. The cost function is applied to each individual, and the order is established based on the result. The population is then divided into two groups: one is the superior half, which contains the lower-cost individuals, and the other is the inferior half. The transfer of genes is accomplished by randomly selecting one individual from the superior half to be the source bacterium and one from the inferior half to be the target bacterium. A gene or a group of genes is then selected and copied from the source bacterium to the target bacterium. The gene in the target bacterium is then overwritten. It is then also necessary to check whether, after the gene transfer, the cost of the target bacterium has become low enough to be transferred to the superior half; if so, the population must be rearranged.

This operation is repeated as many times as the number of infections, which is the input parameter of the algorithm. The purpose of gene transfer is to spread the correct information between individuals within the population. After the gene transfer is complete, the process continues with the next iteration, which again starts with a bacterial mutation. The whole process is completed when the number of generations reaches a maximum, which is also an input parameter of the algorithm.

Figure 11. Illustration of a gene transfer.

4.4. Application of Bacterial Evolutionary Algorithms for Training Interpolative Fuzzy Systems

Since this is an optimization, there is no ideal set of samples to be approximated as closely as possible, but the goal is to find the best possible solution for a given problem by minimizing the cost function. Thus, the solution with the lowest cost function will be the optimal one.

4.4.1. The Structure of the Algorithm

In this optimization, the bacterial genes include the characteristic points of the sets as real numbers and the elements of the table as integers. The characteristic points of the set are the breakpoints because they specify the set. Since all sets are triangular, each one has three characteristic points. A total of nine sets are included in the model because there are three in each antecedent, as well as in the consequent universe, and the data from the cells of the nine-item table are also included in the individual, so an individual has thirty-six genes in total. The characteristics of the bacterial evolution algorithm for optimization are the following.

1. Ten individuals in the population;
2. Ten generations;
3. Five clones for each mutation. The operation is performed for each gene one by one;
4. Twenty infections in gene transfers. Each gene transfer involves the transfer of one gene;
5. Randomly generated initial population.

The algorithm has a special property that it can modify not just one gene, but a group of genes, based on a random decision with a given probability. In the bacterial mutation, there is a chance to modify not just one gene, but an entire set, as well as several random cells in the table; the same applies to gene transfer, such that sets of genes could also be transferred with some probability.

4.4.2. The Cost Function

The feature of the algorithm that differs most for different applications is the cost function. The cost function must be constructed in such a way that the lower the value of the output given as inputs for the characteristics of the fuzzy rules, the more favorable the characteristics of the fuzzy rules are for the operation of the control. Thus, the better the

features, the lower the value they should be given when they are substituted into the cost function.

In the current case, the goal is to tune the rule base of the fuzzy control. The task of the controller is to control the speed of the robot depending on the safety of the environment. The most important thing is safety, i.e., that the robot does not hit an obstacle. Secondary is speed, so if the terrain is safe, the robot should move as fast as possible. However, if it becomes dangerous, the robot should be able to slow down in time.

Taking these into account, a cost function was created that is a simulation. The simulation consists of making a robot approach an obstacle of a given width and distance. The simulation is executed for several parameter settings.

Figure 12 illustrates the initial state of the simulation. The robot is positioned at a distance of D from the wall. The width of the obstacle is w, and the initial velocity of the robot is u. At the start of the simulation, the robot starts towards the wall at speed u, and then controls its speed along the way according to fuzzy logic encoded in the current bacterium. So, from the bacterium for which the cost function is performed, information about the fuzzy set is first extracted, and the fuzzy logic is compiled and then used in the simulation. The rule base comes from the bacterium, the initial velocity is given, the subsequent velocities will be given because they are output as results at the end of each iteration, and safety can be calculated at each iteration from the laser beams using geometric operations. The simulation is run for several cases—u, w, and D. It should be added that the simulation is not a graphical simulation; the algorithm is not played out in front of the human eye, and only the calculations are performed. In each iteration step of running the cost function, the degree of safety is calculated using Equation (12).

$$x_s = \sum_{i=0}^{359} N_i \cdot \sqrt{\tan(i) \cdot d^2 + d^2} \tag{12}$$

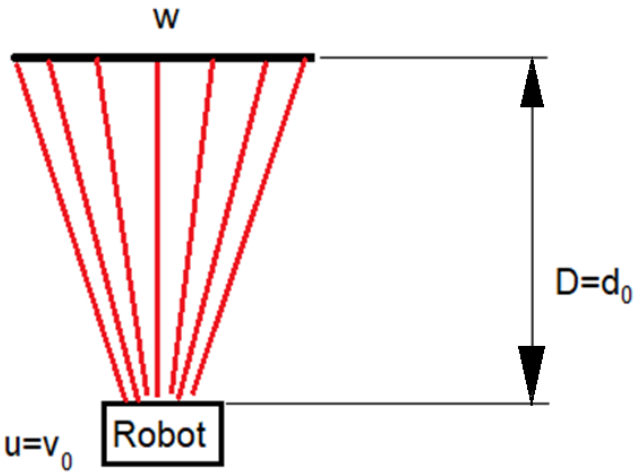

Figure 12. Representation of the cost function simulation.

Each member represents one laser beam since there are 360 laser beams in total. In the formula, N_i indicates whether a given laser beam has reached the obstacle because only the laser beam that has reached the obstacle can be used to measure distance information. d is the distance from the wall. Equation (13) shows whether a given laser beam will reach the obstacle.

$$N_i = \begin{cases} 1, & \text{if } \tan(i) \cdot d \leq \frac{w}{2} \text{ and } \cos(i) > 0. \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

The indices are the same as the indices of the quantities in Equation (12). Since N_i is 0 for rays that do not hit the barrier, these rays have no effect. The distance d is equal to the difference between the initial distance and the displacement.

$$d = d_0 - s \tag{14}$$

The speed of the robot is the displacement divided by the elapsed time.

$$v(t) = \frac{\Delta s}{\Delta t} \tag{15}$$

The speed variable associated with fuzzy control is equal to this speed.

$$x_v = v(t) \tag{16}$$

The intervention $y(x_s, x_v)$ is computed by an interpolative fuzzy inference operation. This operation is based on the fuzzy rule interpolation in the vague environment. The intervention causes a change in velocity, i.e., the velocity increases or decreases by this amount in the $t + 1$-th time step.

$$v(t+1) = v(t) + y(x_s, x_v) \tag{17}$$

The simulation ends when the robot hits the wall, or when its speed reaches zero or becomes negative. When a simulation ends, a new one starts with a different wall width, initial speed, or initial distance from the previous one. The more simulations run for different combinations, the more accurate the optimization will be, but also the longer it will take to run. At the end of each simulation, an evaluation is made. This evaluation gives the cost; the lower the cost, the better the bacterium tested, and the more likely it is to survive. The cost follows the following rules:

1. If the robot hits the obstacle, the cost will be very high. This is the worst-case scenario, because safety is the highest priority for drivers on the roads. If the robot did not hit the wall, it will stop randomly;
2. The distance from the wall after stopping is penalized. This prevents the robot from becoming stuck near an object. Keeping an appropriate distance does not require much effort to ensure the robot's safety. It is acceptable to be bold, and it is fine as long as the robot does not hit the wall;
3. The duration of the simulation is penalized in such a way that the longer the robot takes to move toward the wall, the slower it moves. Slow motion is not beneficial if the path is clear. As long as the robot does not hit the wall, or as long as there are no obstacles nearby, the faster it goes, the better.

The before-mentioned rules were used to teach the robot to drive intelligently. Equation (18) summarizes the evaluation at the end of the simulation.

$$Y = \begin{cases} 1000 + v, & \text{if } s > d_0. \\ \xi_d \cdot d + \xi_t \cdot t, & \text{otherwise.} \end{cases} \tag{18}$$

where ξ_d is the penalty term of the final distance and ξ_t is the penalty term (multiplier) of time. In the first case, the robot hits the wall. In this case, a very high cost is charged because avoiding this is most important. In the second case, if there is no collision with the wall, the value of the cost function is determined by the final distance from the wall, the travel time, and their penalty terms.

4.5. Neuro-Activity-Based Path Planning

A specific model for path planning, the neuro-activity-based dynamic path-planning model was published in 2018 [32]. The procedure is modeled on the way the human brain

works, mimicking the mechanism of the brain for planning a path at an unpredictable cost. In addition, the model uses the flow of information in the brain and the developmental process of neurons in its operation. The full structure of the procedure is shown in Figure 13.

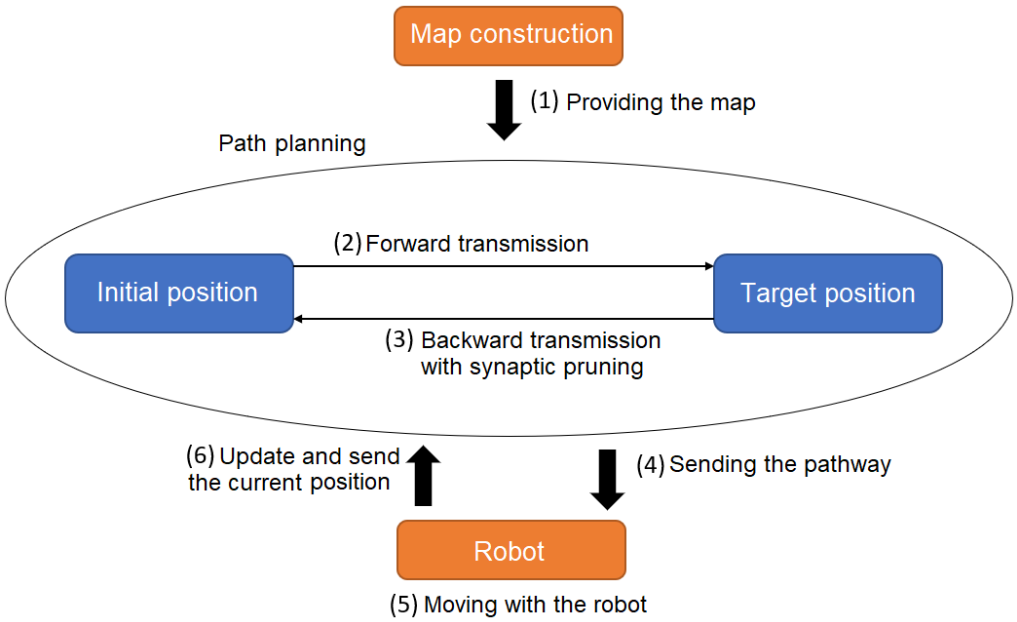

Figure 13. The path-planning process.

4.5.1. Forward Transmission

The first step in pathway planning is the forward transmission. This procedure gives detailed information about the path from the starting point to the destination. The algorithm is biologically inspired, as it performs path planning based on information transmission from the human brain. According to neurobiology, information travels from one neuron to another by an electrical or chemical process, and this "neuron transmission" is triggered by synapses. During transmission, the signal diversity between neurons increases steadily, and when it reaches a threshold, the postsynaptic neuron generates another synapse, which triggers transmission to the neurons connected to it, and information is passed through the neurons.

This biological process is mapped in the implementation by dividing the given map into small segments and modeling each segment with a single neuron. For example, a map consists of twenty times twenty segments, i.e., four hundred segments. It can be described by a square grid of size twenty times twenty; the model then contains four hundred neurons. The starting and target positions of the robot are represented by a neuron. The forward transmission simulates the information transport of the brain. The information starts from the neuron representing the starting position of the robot. The path the information takes to reach the neuron at the target position was investigated. The path the information takes from the source to the destination is the resulting path. It is also possible that the information may not reach the target, in which case the target position is considered unreachable.

Given the map and, hence, the position of the neurons, the neurons must first establish connections with each other. Information can pass through neurons that are connected. If the signal cannot pass between two neurons, the robot cannot travel along that path. According to the model, two neurons can only connect if the distance between them is less than a threshold. The threshold signal γ_{inf} is a parameter of the algorithm.

Once the connections between neurons are established, the neuron at the starting point starts the information propagation through the network, and once it reaches the destination, the algorithm stops. The propagation of information is illustrated in Figure 14.

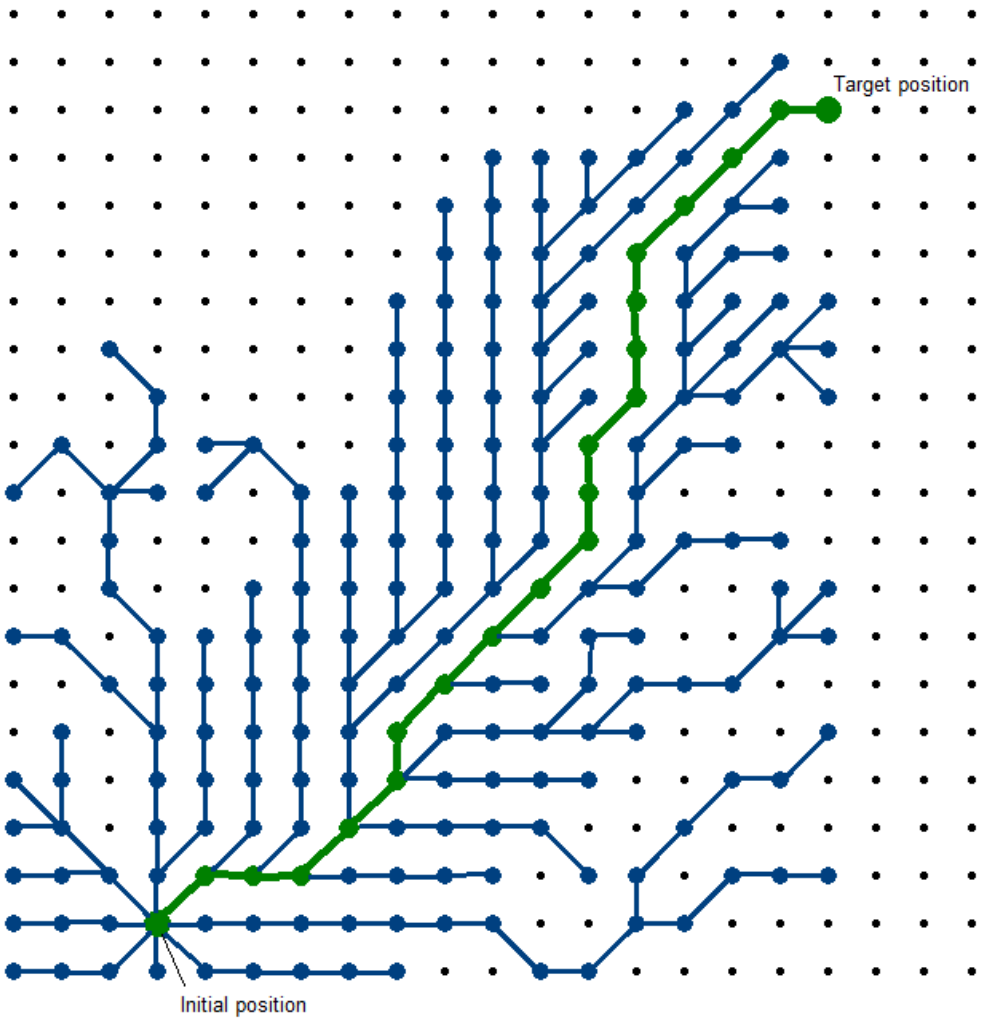

Figure 14. Illustration of the forward transmission.

An important feature of signal propagation is the signal diversity between two neurons. In the implementation, the signal diversity between neurons can be written in a matrix denoted by **X**. The other characteristic is the postsynaptic potential of the neurons ($x^{(PSP)}$). This quantity depends on the impulse time $t^{(imp)}$. For all neurons, it is true that X and $x^{(PSP)}$ have initial values of 0, and that $t^{(imp)}$ has an initial value of 10000. These initial values are given in the original description of the model [32], but the initial values of $x^{(PSP)}$ and $t^{(imp)}$ can be varied.

At the start of the forward transmission, the neuron at the starting point triggers an impulse signal to its neighboring neurons. This action is called neuron firing. Then, the signal diversity (X) between the firing neuron and its neighboring neurons starts to increase. The next neuron in line fires when X reaches a critical level, which is the distance between the two neurons. The impulse time ($t^{(imp)}$) of the firing neuron then becomes 0, and its postsynaptic potential ($x^{(PSP)}$) rises abruptly. The value of signal diversity as a function of time is given by Equation (19).

$$\dot{X}_{ij} = X_{ij}(t-1) + V_{ij}(t) - A \tag{19}$$

Signal diversity varies according to a differential equation, and time is the independent variable [32]. In Equation (19), A is the inhibitory effect that slows down the propagation of the signal. It is calculated using Equation (20).

$$A = \sum_{j=1, j\neq i}^{N} X_{ij}^{(ext)}(t-1) \cdot x_i^{(PSP)}(t-1), \tag{20}$$

where $x^{(PSP)}$ is the postsynaptic potential and $X^{(ext)}$ is the external influence between two neurons, which consists of multiple components [32]. The signal diversity is always interpreted between two connected neurons (the i-th and j-th neurons). In each case, the i-th neuron is the presynaptic neuron from which the signal originates, and the j-th neuron is the postsynaptic neuron, to which the signal arrives. The postsynaptic potential varies with time according to Equation (21) [32].

$$x_i^{(PSP)}(t) = P_i^{(PSP)} \cdot e^{\frac{-t_i^{(imp)}}{\tau_i}} \tag{21}$$

The impulse time is given by Equation (22) [32].

$$t_i^{(imp)} = \begin{cases} 0, & \text{if } X_{ij} \geq ||\mathbf{r}_i - \mathbf{r}_j||. \\ t_i^{(imp)} + 1, & \text{otherwise.} \end{cases} \tag{22}$$

That is, when fired, the impulse time is zero; otherwise, it rises. Here, $\mathbf{r}$ denotes the position vectors of each neuron, so firing occurs when the signal diversity exceeds the distance between the two neurons. As the impulse time increases steadily, the postsynaptic potential decreases exponentially, which means that the inhibitory effect also decreases steadily, as long as the external influence is constant. $P^{(PSP)}$ is the maximum potential and τ is the parameter that affects the rate of potential decay. For each transition during the propagation of the signal, the signal travel time shall be measured and stored. This quantity is the signal travel time between the i-th and j-th neuron [32].

$$T_{ij} = t_i^{(imp)}, \text{if } X_{ij} \leq ||\mathbf{r}_i - \mathbf{r}_j|| \tag{23}$$

The signal travel time will play a role in the second half of the model, the backward transmission. The signal diversity is also affected by the potential force ($V(t)$) between two neurons. When the neuron fires, energy is released, and this energy accelerates the propagation of the signal. The energy released during firing is proportional to the ratio of the changes in the postsynaptic potential of the firing neuron to the changes in its pulse width.

$$E_i = C \cdot \frac{dx_i^{(PSP)}}{dt_i^{(imp)}}, \tag{24}$$

where C is a constant and a parameter of the algorithm [32]. The potential force is increased by the energy released, and reduced by the external effect. The external effect is made up of several components. Some components are constant over time, but some components change as the synapse signal progresses.

$$X_{ij}^{(ext)} = s_{ij}^{(unp)} + s_{ij}^{(w)} + s_{ij}^{(f)} + s_{ij}^{(s)} \tag{25}$$

In Equation (25), $s^{(unp)}$ represents an unpredictable obstacle in dynamic path planning [32]. If an object has appeared, it must be given a large value to inhibit signal propagation. For each area, $s^{(f)}$ represents the friction. This value is constant, but may vary from area to area. The more the friction inhibits the movement of the robot, the higher this value can be.

The term $s^{(w)}$ represents the gravitational force, which is proportional to the mass of the robot. If there is a very large height difference between two neurons, they will not even form a connection, but if they do, the gravitational force will still make it difficult for the robot to move; thus, it must be taken into account.

$$s_{ij}^{(w)} = m \cdot g \cdot \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \cdot \frac{\mathbf{r}_i - \mathbf{r}_j}{||\mathbf{r}_i - \mathbf{r}_j||} \tag{26}$$

When planning a path based on a two-dimensional map, $s_{ij}^{(w)}$ is always zero. In the context of Equation (26), $\mathbf{r}$ is the position of each neuron in the coordinate system, m is the mass of the robot, and g is the gravity acceleration [32]. The last member, $s^{(s)}$, represents the effects of the surrounding neurons. Only nearby neurons can affect the impulse signal.

$$s_{ij}^{(s)} = \sum_{k=1}^{C_{j,0}} d_{C_{j,k}} \cdot h_{C_{j,k}} \cdot e^{-\frac{\left\| \mathbf{r}_{C_{j,k}} - \mathbf{x}_{i,j}^{(r)} \right\|}{d_{max}}} \tag{27}$$

In Equation (27), $C_{j,0}$ represents the number of neurons that are close enough to the impulse signal to be sensitive to it [32]. The index $C_{j,k}$ is the index of these neurons. In addition, d_{max} denotes the maximum range, r denotes the position vector of the $C_{j,k}$-th neuron, x is the position of the impulse signal, and d and h are additional parameters to be calculated.

$$d_{C_{j,k}} = \left(\mathbf{r}_j - \mathbf{x}_{i,j}^{(r)} \right) \cdot \left(\mathbf{r}_{C_{j,k}} - \mathbf{x}_{i,j}^{(r)} \right) \tag{28}$$

$$h_{C_{j,k}} = \left(\mathbf{r}_{C_{j,k}} - \mathbf{x}_{i,j}^{(r)} \right) \cdot \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \tag{29}$$

Equation (29) shows that $h_{C_{j,k}}$ is always zero when using a two-dimensional map, because it only takes a non-zero value in the vertical direction, and in the case of a two-dimensional map, the vertical direction is not interpreted. Since $h_{C_{j,k}}$ appears as a multiplier in Equation (27), in the case of a two-dimensional map, the value of $s^{(s)}$ is always zero as well. To calculate this quantity, the position of the impulse signal is needed, which can be given by the signal diversity X_{ij} between the two neurons. If the value of X_{ij} is 0, the position of the signal is equal to the position of the i-th neuron; if it is equal to the distance between the two neurons, the position is equal to the position of the j-th neuron. If, on the other hand, the value is between these two, the signal position is proportional to the position of the two neurons.

$$\mathbf{x}_{i,j}^{(r)} = (\mathbf{r}_i - \mathbf{r}_j) \cdot X_{ij} + \mathbf{r}_i \tag{30}$$

This allows us to calculate the value of the signal diversity (X) in each time step. If the value of the diversity between two neurons exceeds the value of their distance, the next neuron fires, sending new signals to the neighboring neurons; so, if the value of X_{ij} becomes greater than the distance between the i-th and j-th neurons, the j-th neuron fires. The process continues until an impulse signal reaches the target neuron. At that point, the pathway through which the signal has traveled from the starting point to the destination becomes the intended pathway, which is the output of the forward transmission procedure.

4.5.2. Backward Transmission with Synaptic Pruning

Once the path has been completed by the forward transmission, the robot can follow it. A path using forward transmission can be planned alone. However, it is possible that the path is unnecessarily detailed—for example, that there are too many waypoints on a completely straight stretch, or too many bends. The backward transmission model optimizes the path by reducing these, making it simpler and less time-consuming to travel.

Synaptic pruning is inspired by the biological phenomenon of the same name, which is that many synaptic connections are broken during the development of the brains of children because this optimizes the resource pool of associative memory. This phenomenon was described by Huttenlocher in 1979 [43]. The weaker the connections are, the more likely they are to disappear. The model uses neural regulation-driven synaptic pruning, the theory of which was presented in 1999 [44].

$$W'(t+1) = W(t) - (W(t))^{\alpha} \cdot \mu(t) \tag{31}$$

$$W(t+1) = W'(t+1) \cdot \frac{f_i^0}{f_i^t}, \tag{32}$$

where W represents the age of the neuron, or, in other words, the strength of the connections, because the higher the value of W, the stronger the neuron [32]. The value of W decreases steadily over time, and when it reaches 0, the neuron is removed. At this point, its connections are broken and new connections are created in their place. The degradation parameter is α, whose value is chosen in the interval $[0; 1]$. The quantity $\mu(t)$ is a limiting factor. Its value is given by Equation 33.

$$\mu(t) = \frac{1}{1 + e^{-(\eta - \varepsilon)}}, \tag{33}$$

where η is the threshold for synaptic pruning. The smaller the threshold, the slower the decrease in W [32]. The other quantity, ε, is determined by the connections between neurons. The model also includes f_i^0 / f_i^t, which represents the input field of the neuron and is biologically relevant, but in this model, it is set to 1.

During backward transmission, neurons that are less important for the pathway are removed first. Several of the many waypoints close to each other may disappear, and smaller bends may also disappear. The value of ε, which determines the extent to which a waypoint can be eliminated, can be calculated from Equation (34). The higher the value, the higher the probability that the waypoint marked by the neuron can be removed from the pathway, and the faster the aging of the neuron [32].

$$\varepsilon = h_i^{PSP}(t) \cdot \left(\alpha^{(l)} \cdot \beta_i^{(l)} + \alpha^{(d)} \cdot \beta_i^{(d)} \right) \tag{34}$$

The process of backward transmission works on the principle of a spiking neural network, where neurons are the waypoints. In this network, h_i^{PSP} represents the postsynaptic potential of the i-th neuron. Its value decreases over time. In addition, the value of ε consists of two components: $\beta_i^{(l)}$ is influenced by the distance between neurons, and $\beta_i^{(d)}$ is influenced by the smoothness of the path trajectory. Both affect how fast the neurons age, and the constant coefficients $\alpha^{(l)}$ and $\alpha^{(d)}$ affect the importance of each property during degradation. Figure 15 illustrates the synaptic pruning. It also illustrates the quantities that influence synaptic pruning.

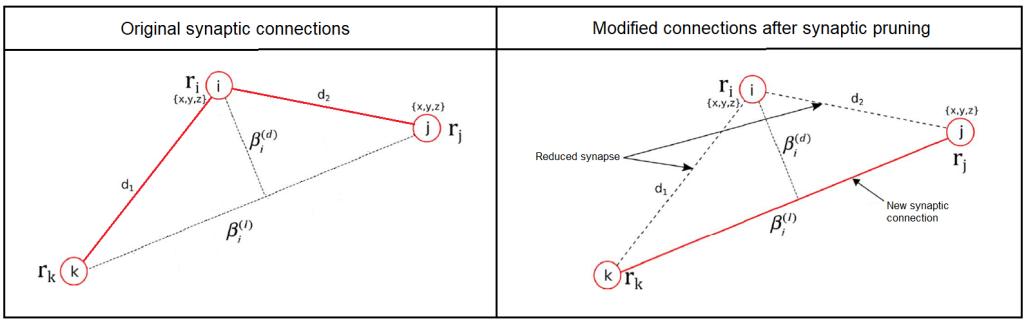

Figure 15. Reduction in the number of connections by synaptic pruning.

These quantities are calculated for the i-th neuron, but the two neurons surrounding it should be taken into account—namely, the next waypoint before and after it, and these are the j-th and k-th neurons. In the case where the age of the i-th neuron becomes 0, the i-th neuron is unbound and moves out of the pathway, and a new connection is established between the j-th and k-th neurons. These are not calculated for the origin and destination neuron, so there is no point in removing them. These important quantities are calculated from Equations (35) and (36).

$$\beta_i^{(l)} = ||\mathbf{r}_k - \mathbf{r}_j||, \tag{35}$$

$$\beta_i^{(d)} = ||(\mathbf{r}_k - \mathbf{r}_i) - ((\mathbf{r}_k - \mathbf{r}_i) \cdot \mathbf{n}) \cdot \mathbf{n}||, \tag{36}$$

where $\mathbf{n}$ is a unidirectional unit vector with the vector $(\mathbf{r}_k - \mathbf{r}_j)$ [32]. This means that the age of the neurons in each time step should be calculated during the backward transmission. Signal processing during backward transmission is performed with a simple spiking neural network model. After the neuron transmission is finished, the neuron at the target position fires. In this model, the strength of the synapse signal will be proportional to the weights calculated during the forward transmission. Each time a neuron fires, whether it can be removed must be decided based on its age. If the age (W) value reaches 0, that neuron must be removed, and a new connection be established between the neuron and the surrounding neuron. The formation of a new connection is illustrated in Figure 15. A neuron fires when its membrane potential (h) becomes higher than the threshold [32].

$$p_i = \begin{cases} 1, & \text{if } h_i \geq \Theta. \\ 0, & \text{otherwise.} \end{cases} \tag{37}$$

The membrane potential value is made up of several components [32].

$$h_i = \tanh\left(h_i^{syn}(t) + h_i^{ref}(t)\right), \tag{38}$$

where h_i^{syn} affects signal strength based on the postsynaptic potential of the neuron and the weights calculated during forward transmission [32].

$$h_i^{syn} = \gamma^{syn} \cdot h_j^{(PSP)}(t) + \sum_{j=1, j\neq i}^{N} X_{ij}^{(w)}(t+1) \cdot h_j^{(PSP)}(t) \tag{39}$$

In Equation (39), γ^{syn} is an algorithm parameter, $h^{(PSP)}$ is the postsynaptic potential of each neuron, and $X_{ij}^{(w)}$ is the weight of the connections, which can be calculated from the signal travel time (T) during the forward transmission; the weight of the connections is inversely proportional to the signal travel time because a longer time will result in a lower signal propagation speed during the backward transmission. The postsynaptic potential of neurons is similarly given as in the forward transmission. Equation (40) gives the value of the postsynaptic potential of the neurons during the backward transmission [32].

$$h_i^{(PSP)} = e^{-\frac{t-t_i^f}{\tau_i}}, \tag{40}$$

where t is the current time, t_i^f is the time of the last firing of the neuron, and τ_i is a parameter that affects how fast the postsynaptic potential should decrease, and thus, how long the increased potential at firing should affect other neurons.

The other component of the membrane potential is h_i^{ref}, and it is responsible for the refractoriness, which is often used in spiking neural networks. The value of the h_i^{ref} quantity can be determined using Equation (41). The essence of refractoriness is that, after firing a neuron, the membrane potential is significantly reduced to avoid too much firing [32].

$$h_i^{ref} = \begin{cases} \gamma^{ref} \cdot h_i^{ref}(t-1) - R, & \text{if } p_i(t-1) = 1. \\ \gamma^{ref} \cdot h_i^{ref}(t-1), & \text{otherwise.} \end{cases} \tag{41}$$

where R gives the degree of refractoriness, i.e., how much this component should decrease when fired, and the latter describes the rate of increase.

While the model is running, the points of the built path are continuously consumed at a rate and speed that depends on the parameters. The choice of parameters greatly influences the behavior of the model. If the stopping condition is satisfied, the optimized

path becomes the output, and the robot has to follow it. The stopping condition can be the number of iterations, the number of waypoints removed, the time, etc.

4.6. Integration of Neuro-Activity-Based Path Planning with Mobile Robot Control

To reach its destination, the robot must plan a path to the destination and then follow it. Safe transport requires planning a safe path and traveling to the right place. To achieve this, interpolative fuzzy speed control and neuro-activity-based path planning have been integrated. Once the robot is ordered to move to a certain location, its first task is to plan a path to said location using neuro-activity-based path planning. Once the path plan is complete, it has to follow the path while using interpolative fuzzy speed control to achieve safe travel. Thus, the robot tries to reach the specified destination as quickly and as safely as possible by controlling the speed according to the specified fuzzy rule base and control procedure. Figure 16 illustrates the process.

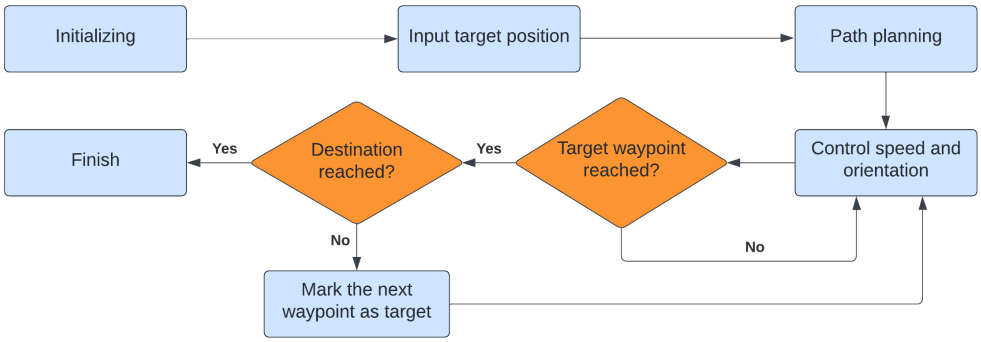

Figure 16. The control process. (Blue color represents the actions, orange color represents the decisions.)

In summary, the rule base for speed control is generated by the bacterial evolutionary algorithm, and this rule base becomes the input to the control process. At the beginning of the process, neuro-activity-based path planning generates the path from the starting position to the destination, and this path determines where the robot should go. Thus, the rule base and the route are the two inputs to the robot-guiding algorithm.

The speed control rule base is structured as shown in Figures 6–8 and in Table 1. This is how the bacterial evolutionary algorithm constructed the rule base. Neuro-activity-based path planning is a complex process with a lot of parameters. These parameters must be given in order to start the process. It is worth noting that this algorithm was developed for path planning on a three-dimensional map, but it works in two dimensions with relatively little difference. The difference between the two- and three-dimension planning is that the external effect acting on the impulse signals is different. Equation (25) describes the value of the external effect, $X^{(ext)}$. Because of the constant height, $s^{(w)}$ and $s^{(s)}$ are always zero. Moreover, since no dynamics are used in the forward transmission, $s^{(unp)}$ is always zero. The only remaining external effect is, therefore, friction. Since all fields are the same, the friction is equal in all fields. For this reason, the friction force can be considered as a parameter of the algorithm. The parameters for the forward transmission are given in Table 2.

Table 2. The parameters of the forward transmission.

d_{max}	m	g	$P_i^{(PSP)}$	γ^{inf}	τ_i	C	$s^{(f)}$
1.2	1	9.81	30	1.5	10	1	0

The quantities d_{max}, m, and g are irrelevant because the path planning is only conducted in two dimensions, and these quantities affect the external influence on the synapse signal. However, it can be said that, based on γ^{inf}, the path planning can be performed

based on the eight-neighbor principle, and the robot can traverse from one waypoint to another in horizontal, vertical, and diagonal directions. It can also be said that a neuron can form a connection with a maximum of eight other neurons, and that there is zero friction based on $s^{(f)}$; thus, nothing prevents the potential force from increasing. The backward transmission parameters are given in Table 3.

Table 3. The parameters of backward transmission.

γ^{syn}	γ^{ref}	R	Θ	$\alpha^{(l)}$	$\alpha^{(d)}$	τ_i	α	η	f_i^0/f_i^t
0.05	0.2	10	0.8	0.3	0.7	10	0.5	0.8	1

Another important point about backward transmission is that the algorithm stops running if 50% of the waypoints are lost due to synaptic prunings. This rate could be reduced, just as the parameter values are not necessarily optimal, but practical experience has shown that this setting has produced good results.

Once the path planning is complete, the robot starts moving along the planned path. On the way, each waypoint must be touched individually. If the robot draws near enough to a waypoint, that point is considered reached, and the robot can move on to the next one. While the robot is moving along the path, its speed is controlled based on the safety of the environment using the fuzzy control model. The orientation is controlled using the PD control method. The process is finished once the robot reaches its destination.

4.7. The Analysis of the Time Complexity

In Section 4.6, the elements of the control process were specified (Figure 17). The time complexity of each element was measured and evaluated. Inside the speed control of the robot, an interpolative fuzzy inference method is used. The interpolative fuzzy inference method has the advantage of allowing accurate inferences using fewer rules, but because it requires more computation, its time complexity is also higher than that of classical methods. Table 4 contains the measurement results of time complexity in the case of different fuzzy inference methods. The time requirement of one inference may depend on many different factors, but these values were measured on the same computer and with the same fuzzy rule base and inputs.

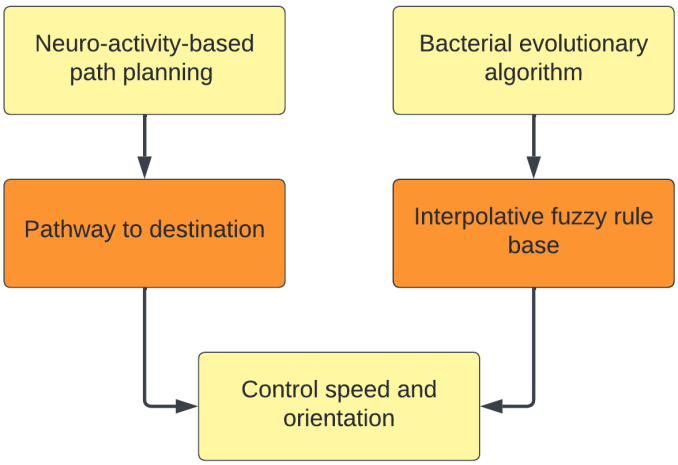

Figure 17. The interaction between the components of the control process.

It can be seen that interpolative fuzzy inference takes much longer than the others. In some applications, this difference may be important, but in the case of controlling a mobile robot, the mechanics are much slower than the computation of the inference; thus, the higher time complexity is not a disadvantage for this application.

Table 4. Comparison of the time complexities of different fuzzy control models.

	Mamdani Inference	Sugeno Inference	Interpolative Inference
Time complexity of the inference (µs)	68.964	19.996	597.452

In Section 4.4.2, that the cost function of the bacterial evolutionary algorithm actually performs a lot of calculations was discussed. For this reason, the optimization process of the fuzzy rule base requires several hours. Table 5 shows the main measurement data of the optimization process. During the optimization process, several interpolative fuzzy inferences were performed.

Table 5. Measurement data of the optimization process related to time complexity.

	Duration of One Cost Function (s)	Number of Cost Function Calls during the Optimization Process (-)	Duration of the Optimization Process (min)
Measurement result	0.9173	18,400	281.3

The time complexity of the optimization depends on several factors. These are the size of the rule base, the number of generations, the number of clones during the bacterial mutation, and the number of infections during the gene transfer. Increasing the values of these parameters increases the accuracy, but also the time complexity. The parameters of the discussed optimization task are given in Section 4.4.1.

The time complexity of the neuro-activity-based path planning depends on many different variables. The measurement and optimization of the time complexities are not within the focus of this study. Therefore, the duration of the path-planning process with the parametrization described in Section 4.6 is not the most optimal one. The time complexity of the path-planning process depends on the following parameters:

1. The size of the map;
2. The ratio of free and occupied space on the map;
3. The distance between the starting point and the destination;
4. The strength of scope, γ^{inf};
5. The constant coefficient, C;
6. The maximal potential value, $P^{(PSP)}$;
7. The initial value of $t^{(imp)}$;
8. The friction force, $s^{(f)}$;
9. τ_i, which influences the speed of the reduction in the potential energy.

These parameters affect the time complexity when using a two-dimensional map. In the case of a three-dimensional map, almost all parameters affect the time complexity, and the quality of the relief is also an important parameter in this case.

5. Experimental Results

Results on the tuning of the interpolative fuzzy rule base have been presented previously; it was accomplished by a bacterial evolutionary algorithm, and Figures 6–8 and Table 1 contain the results. This fuzzy rule base was used by a mobile robot for speed control.

Despite interpolative inference, the sets are well-distributed along with the interpretation domain. Nowhere is the full base set exploited, but this is only spectacular on the consequent side. The table also shows that acceleration occurs in the safety case, and that deceleration occurs in the dangerous case. In general, large accelerations and decelerations are characteristic.

Large speed variations were also observed in simulation testing. The robot accelerated quickly at start-up and stopped quickly in case of danger. In this test, the robot made little effort to be safe, stopping only when it was very close to the walls, but always managed to stop in time. This is the behavior that was expected.

The decline of the cost function as a function of generations is shown in Figure 18. The final value of the cost function was 15.49 units. The low cost is confirmed by the practical results.

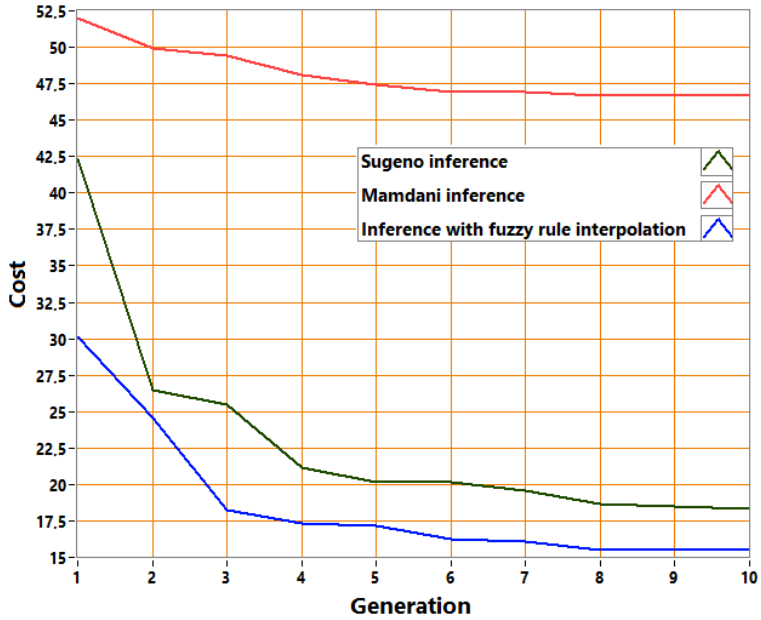

Figure 18. Cost function decrease in the case of different inference methods.

In Figure 18, the blue curve depicts the cost function reduction associated with the construction of an interpolative fuzzy rule base. The other two curves belong to classical fuzzy inference methods. In this case, the bacterial evolutionary algorithm was used to build a fuzzy rule base with the same cost function, but with a different inference procedure. In these cases, however, the sets are always Ruspini-partitioned to cover the entire interpretation domain. The red curve shows the cost function reduction associated with the construction of the rule base when using the classical Mamdani inference procedure, and the green curve shows the cost function reduction when using the Sugeno inference procedure. The cost values for each procedure are shown in Table 6 for all 10 generations.

Table 6. Comparison of fuzzy control models by cost values.

	Mamdani Inference (Unit)	Sugeno Inference (Unit)	Interpolative Inference (Unit)
1.	51.987	42.378	30.115
2.	49.912	26.433	24.540
3.	49.410	25.485	18.218
4.	48.066	21.143	17.292
5.	47.413	20.178	17.134
6.	46.940	20.174	16.285
7.	46.897	19.599	16.118
8.	46.721	18.653	15.493
9.	46.721	18.526	15.493
10.	46.664	18.343	15.493

In the case of interpolative inference, more rules do not necessarily produce better results. After trying several sets, it was found that three sets in the base set of both the

antecedents and the consequent gave the best results in terms of the cost function. When using two sets in both the antecedents and the consequent basis set, the final cost is 17.587 units, whereas if four sets are present in the safety basis set—three in the speed basis set and four in the consequent basis set—the cost is 17.72 units. Both are very good results, but not as good as the original.

The advantage of the interpolative fuzzy model over the others is that it has a lower cost and can therefore be applied more accurately to the control task at hand. A second advantage is that it can work with fewer rules, thereby simplifying the rule base, and a third is that the sets do not need to form a Ruspini partition; thus, the rule base is less bounded. The disadvantage is the higher time complexity of the inference, which does not allow for particularly low cycle times.

Fuzzy speed control and path planning were tested separately and integrated with each other. The tests were performed in a ROS environment using the Gazebo simulator. The robot model that was used for the simulation tests is a specific mobile robot called Turtlebot3 Burger. The model of the Burger is shown in Figure 19, and the simulated world used for testing is shown in Figure 20.

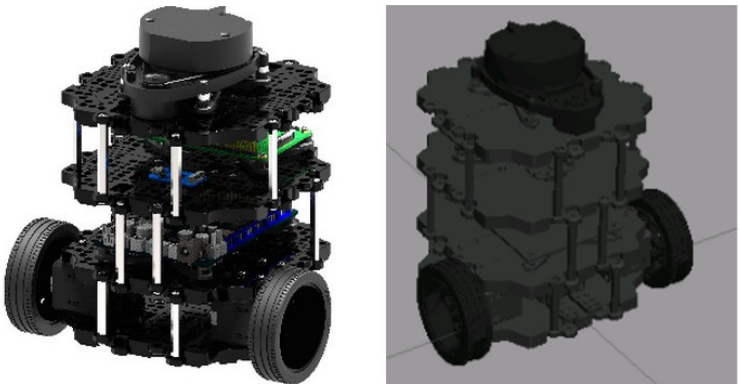

Figure 19. The real and simulated model of Turtlebot3 Burger.

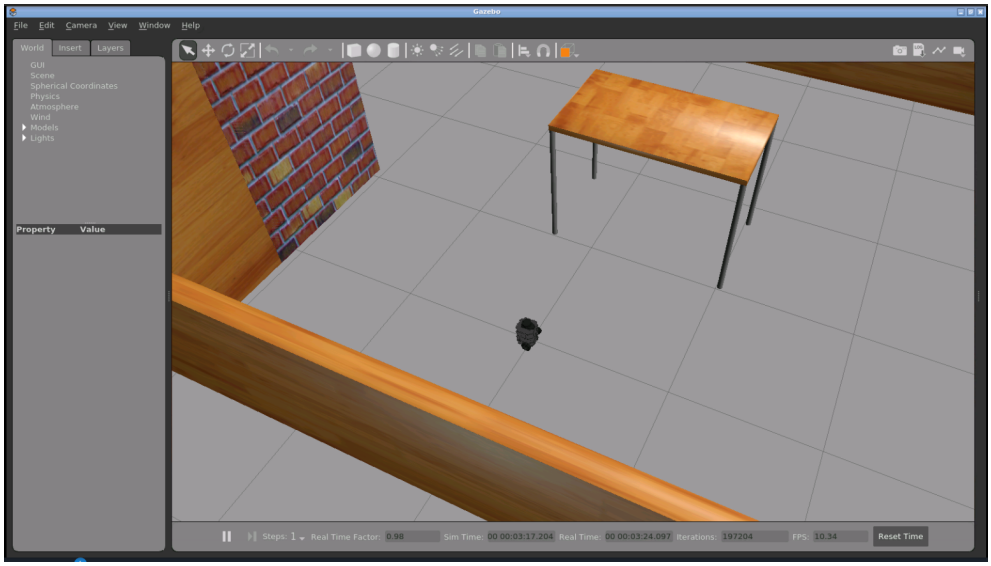

Figure 20. Simulated environment.

In the ROS environment, when using a robot equipped with LIDAR, it is possible to create a map of the environment of the robot. With LIDAR, the robot can detect landmarks around itself in 360° using laser range finding. Based on the distance information of the landmarks, a map of the environment can be generated using the appropriate algorithm. The map can then be used for path planning. To create a map, the ROS provides ready-to-

use applications that can be launched by the robot to start building the map and to then build it continuously. For example, the Burger robot is provided with the Turtlebot3 SLAM (simultaneous localization and mapping) package. Figure 21 indicates the experimental results provided by different path-planning algorithms. Each experiment was performed on the same map, and this map was generated with the SLAM algorithm.

In neuro-activity-based path planning, the forward transmission alone can generate a path, but if the backward transmission is used, the planned path is different. It is worth observing the difference between pathways designed with and without backward transmission. A pathway designed by running neuronal transmission alone consists of more waypoints, and therefore, has a different structure than one optimized by synaptic pruning.

Figure 21a,b illustrates the designed path using each algorithm. In each case, the task is to move from the starting position to the target position by avoiding the legs of the table that are shown in Figure 20. When only neuron transmission was run, and no backward transmission was used for optimization, the resulting path became longer and more tortuous.

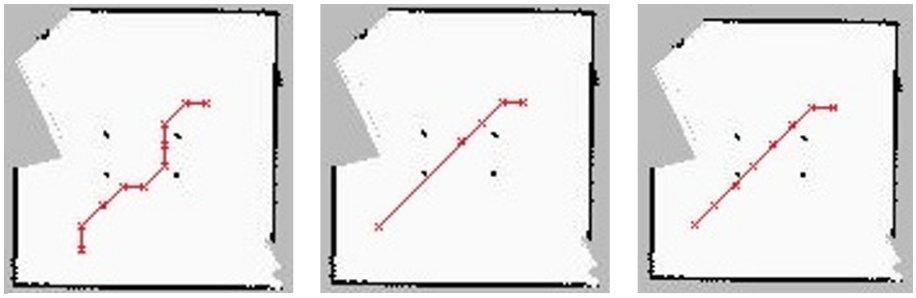

Figure 21. Planned pathways with forward transmission (**a**), with forward and backward transmission (**b**), and with the A* algorithm (**c**).

Figure 21c illustrates the designed path using the classic A* algorithm. This algorithm is not parameterizable to the same extent as the neuro-activity-based one, so the goal is always to find the shortest pathway. It can be observed that the path designed in this way, although consisting of more points, has the same trajectory as the one in Figure 21b.

In neuron transmission, the propagation of the synaptic signal usually reaches the target position by a path that extends a relatively long distance from the walls. When the synaptic signal reaches a neuron adjacent to a wall, it typically cannot travel any further from there, so the pathway typically does not run adjacent to the wall. This may be the reason why pathways designed in this way avoid obstacles by a larger arc than in other cases. In synaptic prunings, the path can be optimized and shortened, but then the path is also closer to the obstacles. The backward transmission was run until half of the points designed during the neuron transmission were removed. If fewer points were removed, the path would probably change, be longer, and avoid obstacles in a larger arc, and thus be closer in characteristics to that created by the forward transmission alone. This leads to the conclusion that in neuro-activity-based path planning, it is the effect of backward transmission that determines how short or safe the path will be. The safer the planned path to the goal is, the less synaptic pruning is needed, and the faster the goal is reached, the more synaptic prunings are needed. As a result, the advantages of neuro-activity-based path planning over classical algorithms are greater detail and more flexibility.

6. Conclusions

This paper presents a method that provides an intelligent transport solution for mobile robots. This method includes pathway generation with neuro-activity-based path planning, and speed control for the safety of the environment, using an interpolative fuzzy speed control method. This method was developed and tested using a mobile robot equipped with LIDAR, which enables it to observe its environment and evaluate its safety. Then,

the robot can adjust its speed according to the safety of the environment using fuzzy rule interpolation in the vague environment. The advantage of this method over other fuzzy models is that it can provide correct inferences with a simpler structure and a fuzzy rule base with few rules. The fuzzy rule base was built from scratch using a bacterial evolutionary algorithm. The path-planning process was the neuro-activity-based path planning, which uses a natural mechanism of the human brain.

The experimental results show that the bacterial evolutionary algorithm can construct the whole interpolative fuzzy rule base from scratch when the correct cost function is used. The more cases the cost function examines, the more accurate the result, but the more computationally intensive and time-consuming the process is. In any case, the optimization resulted in a good solution with a low cost value. It was also shown that neuro-activity-based path planning can construct different pathways according to its parametrization. When only the forward transmission is applied, the longest, but safest, pathway is generated. The backward transmission removes waypoints from the path, making it shorter, but the pathway becomes less safe with each waypoint removed. This provides another choice between faster and safer transport.

It is worth mentioning that some limitations of the presented methods have to be taken into account. One such limitation is that safety cannot always be accurately measured because the robot cannot detect objects lower than the robot using LIDAR. For this reason, the robot can hit objects of small size. In addition, the possible movement of obstacles is not taken into account when determining safety. Other than that, it should be added that the time complexity of neuro-activity-based path planning is higher than that of the A* algorithm. The runtime can be reduced by appropriate parameterization, but more computation also results in higher runtimes. For this reason, if the goal is to find the shortest path in all cases, this method may not be the most suitable. In addition, currently, when running the bacterial evolutionary algorithm, the number of fuzzy rules is predetermined and the algorithm is not trained to optimize the number of rules. For this reason, it is also an opportunity for improvement.

As future work, the model can be further improved in a lot of ways. Some of the options for future developments are as follows:

1. Optimizing the time complexity of neuro-activity-based path-planning process;
2. Adding more variables to the speed control, which can be extended to any number of variables. In this case, a rework of the optimization algorithm is needed;
3. Optimizing the number of fuzzy rules using the bacterial evolutionary algorithm;
4. Making path planning dynamic, so the robot can react to sudden obstacles;
5. Enabling path planning on a topographic map, which requires a different mapping method because Turtlebot3 can only produce a two-dimensional map;
6. Taking into account the movement of obstacles for the safety of the environment;
7. Incorporating a reconnaissance function, i.e., planning a route through an unknown area;
8. Designing an implementation for a robot that can transport objects; such a robot should strive for safety rather than speed if the object is heavy or fragile;
9. Detecting obstacles with image processing. Security information should be based on the image. This would also avoid collisions with low obstacles;
10. Upgrading optimization to a bacterial memetic algorithm;
11. Integrating neutrosophic statistics into the control process. Neutrosophic statistics is an extension of classical statistics, and it is applied when the data is coming from an uncertain environment. This type of statistics uses sets instead of numbers to represent uncertain data. The sets contain numerical values that the data can be equal with. Neurosophic statistics was first proposed at the end of the 20th century, but it was only developed recently [45,46]. One of the potential applications is the generalization of fuzzy sets [47], which can also be applied in the current control process. Furthermore, it is worth mentioning that several parameters can have uncertain values. For example, the degree of safety can be uncertain because of measurement uncertainty, and there

are different measurements that indicate the same level of safety. The other example is friction force in the neuro-activity-based path planning, which indicates how hard it is for the robot to move on the ground. Friction, in a physical sense, can have uncertain values, but this quantity in the neuro-activity-based path planning has a role that is similar to the cost values in case of cost function minimization-based algorithms, and these cost values can also be considered uncertain [48,49];

12. Performing sensitivity analysis and investigating the stochastic nature of the applied evolutionary algorithm.

Fast and safe transport is important for many mobile robots. Future applications of the presented process can take advantage of this. By investing the right amount of time and energy, with sufficient optimization and program extension, a waiter or assistant robot can be created using the presented method and the right traffic rules.

The code and measurement data used in this paper are available online at [50].

Author Contributions: Conceptualization, F.Á.S. and J.B.; methodology, F.Á.S. and J.B.; software, F.Á.S.; validation, F.Á.S.; formal analysis, F.Á.S.; investigation, F.Á.S.; resources, F.Á.S.; data curation, F.Á.S.; writing—original draft preparation, F.Á.S., J.B. and B.N.; writing—review and editing, F.Á.S., J.B. and B.N.; visualization, F.Á.S.; supervision, J.B. and B.N.; project administration, B.N.; funding acquisition, J.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mamdani, E.; Assilian, S. An experiment in linguistic synthesis with a fuzzy logic controller. *Int. J. Man-Mach. Stud.* **1975**, *7*, 1–13. [CrossRef]
2. De Luca, F.; Calderaro, V.; Galdi, V. A Fuzzy Logic-Based Control Algorithm for the Recharge/V2G of a Nine-Phase Integrated On-Board Battery Charger. *Electronics* **2020**, *9*, 946. [CrossRef]
3. Mosavi, A.; Qasem, S.N.; Shokri, M.; Band, S.S.; Mohammadzadeh, A. Fractional-Order Fuzzy Control Approach for Photovoltaic/Battery Systems under Unknown Dynamics, Variable Irradiation and Temperature. *Electronics* **2020**, *9*, 1455. [CrossRef]
4. Anthony, M.; Prasad, V.; Kannadasan, R.; Mekhilef, S.; Alsharif, M.H.; Kim, M.K.; Jahid, A.; Aly, A.A. Autonomous Fuzzy Controller Design for the Utilization of Hybrid PV-Wind Energy Resources in Demand Side Management Environment. *Electronics* **2021**, *10*, 1618. [CrossRef]
5. Memon, I.; Hasan, M.K.; Shaikh, R.A.; Nebhen, J.; Bakar, K.A.A.; Hossain, E.; Tunio, M.H. Energy-Efficient Fuzzy Management System for Internet of Things Connected Vehicular Ad Hoc Networks. *Electronics* **2021**, *10*, 1068. [CrossRef]
6. Domingos, D.; Camargo, G.; Gomide, F. Autonomous Fuzzy Control and Navigation of Quadcopters. *IFAC-PapersOnLine* **2016**, *49*, 73–78. [CrossRef]
7. Smith, S.; Rae, G.; Anderson, D.; Shein, A. Fuzzy logic control of an autonomous underwater vehicle. *Control. Eng. Pract.* **1994**, *2*, 321–331. [CrossRef]
8. Cupertino, F.; Giordano, V.; Naso, D.; Delfine, L. Fuzzy control of a mobile robot. *IEEE Robot. Autom. Mag.* **2006**, *13*, 74–81. [CrossRef]
9. Faisal, M.; Hedjar, R.; Sulaiman, M.A.; Al-Mutib, K. Fuzzy Logic Navigation and Obstacle Avoidance by a Mobile Robot in an Unknown Dynamic Environment. *Int. J. Adv. Robot. Syst.* **2013**, *10*, 37. [CrossRef]
10. Singh, N.H.; Thongam, K. Mobile Robot Navigation Using Fuzzy Logic in Static Environments. *Procedia Computer Science* **2018**, *125*, 11–17. [CrossRef]
11. Hartono, R.; Nizar, T. Speed Control of a Mobile Robot Using Fuzzy Logic Controller. *IOP Conf. Ser. Mater. Sci. Eng.* **2019**, *662*, 022063. [CrossRef]
12. Mac, T.; Copot, C.; Keyser, R.; Tran, T.; Vu, T. MIMO Fuzzy Control for Autonomous Mobile Robot. *J. Autom. Control. Eng.* **2015**, *3*, 65–70. [CrossRef]
13. Omrane, H.; Masmoudi, M.S.; Masmoudi, M. Fuzzy logic based control for autonomous mobile robot navigation. *Comput. Intell. Neurosci.* **2016**, *2016*, 9548482. [CrossRef]
14. Kóczy, L.T.; Hirota, K. Rule interpolation by α -level sets in fuzzy approximate reasoning. *BUSEFAL* **1991**, *46*, 115–123.

15. Kóczy, L.T.; Hirota, K. Rule interpolation in approximate reasoning based fuzzy control. In Proceedings of the 4th IFSA World Congress, Brussels, Belgium, 7–12 July 1991; pp. 89–92.
16. Kóczy, L.T.; Hirota, K. Approximate reasoning by linear rule interpolation and general approximation. *Int. J. Approx. Reason.* **1993**, *9*, 197–225. [CrossRef]
17. Kovács, Sz.; Kóczy, T.L. Application of an approximate fuzzy logic controller in an AGV steering system, path tracking and collision avoidance strategy. *Fuzzy Set Theory Appl.* **1999**, *16*, 456–467.
18. Rubaai, A.; Young, P. Hardware/Software Implementation of Fuzzy-Neural-Network Self-Learning Control Methods for Brushless DC Motor Drives. *IEEE Trans. Ind. Appl.* **2016**, *52*, 414–424.
19. Castillo, O.; Lizárraga, E.; Soria, J.; Melin, P.; Valdez, F. New approach using ant colony optimization with ant set partition for fuzzy control design applied to the ball and beam system. *Inf. Sci.* **2015**, *294*, 203–215.
20. Nawa, N.; Furuhashi, T. Fuzzy system parameters discovery by bacterial evolutionary algorithm. *IEEE Trans. Fuzzy Syst.* **1999**, *7*, 608–616. [CrossRef]
21. Shin, H.; Chae, J. A Performance Review of Collision-Free Path Planning Algorithms. *Electronics* **2020**, *9*, 316.
22. Dijkstra, E. A Note on Two Problems in Connexion with Graphs. *Numer. Math.* **1959**, *1*, 269–271. [CrossRef]
23. Likhachev, M.; Gordon, G.J.; Thrun, S. ARA* : Anytime A* with Provable Bounds on Sub-Optimality. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver and Whistler, British Columbia, Canada, 8–13 December 2003; Thrun, S., Saul, L., Schölkopf, B., Eds.; pp 767 – 774.
24. Ferguson, D.; Stentz, A. Field D*: An Interpolation-Based Path Planner and Replanner. In Proceedings of the 12th International Symposium on Robotics Research, San Francisco, CA, USA, 12–15 October 2005; pp. 239–253.
25. Quoy, M.; Moga, S.; Gaussier, P. Dynamical neural networks for planning and low-level robot control. *IEEE Trans. Syst. Man Cybern. Part A Syst. Hum.* **2003**, *33*, 523–532. [CrossRef]
26. Yang, S.; Luo, C. A neural network approach to complete coverage path planning. *IEEE Trans. Syst. Man Cybern. Part B Cybern.* **2004**, *34*, 718–724. [CrossRef] [PubMed]
27. Chen, Y.; Shen, T.; Yang, S.; Liu, X.; Yang, R.; Cheng, L. A Path Planning Strategy with Ant Colony Algorithm for Series Connected Batteries. *Electronics* **2020**, *9*, 1816. [CrossRef]
28. Botzheim, J.; Toda, Y.; Kubota, N. Bacterial memetic algorithm for offline path planning of mobile robots. *Memetic Comput.* **2012**, *4*, 73–86. [CrossRef]
29. Botzheim, J.; Toda, Y.; Kubota, N. Path Planning in Probabilistic Environment by Bacterial Memetic Algorithm. In *Intelligent Interactive Multimedia: Systems & Services*; Watanabe, T., Watada, J., Takahashi, N., Howlett, R.J., Jain, L.C., Eds.; Smart Innovation, Systems and Technologies; Springer: Berlin/Heidelberg, Germany, 2012; Volume 14, pp. 439–448. [CrossRef]
30. Botzheim, J.; Toda, Y.; Kubota, N. Bacterial memetic algorithm for simultaneous optimization of path planning and flow shop scheduling problems. *Artif. Life Robot.* **2012**, *17*, 107–112. [CrossRef]
31. Lonklang, A.; Botzheim, J. Improved Rapidly Exploring Random Tree with Bacterial Mutation and Node Deletion for Offline Path Planning of Mobile Robot. *Electronics* **2022**, *11*, 1459. [CrossRef]
32. Saputra, A.A.; Toda, Y.; Botzheim, J.; Kubota, N. Neuro-Activity-Based Dynamic Path Planner for 3-D Rough Terrain. *IEEE Trans. Cogn. Dev. Syst.* **2018**, *10*, 138–150. [CrossRef]
33. Drobics, M.; Botzheim, J. Optimization of fuzzy rule sets using a bacterial evolutionary algorithm. *Mathw. Soft Comput.* **2008**, *15*, 21–40.
34. Horvath, C.M.; Botzheim, J.; Thomessen, T.; Korondi, P. Bacterial memetic algorithm trained fuzzy system-based model of single weld bead geometry. *IEEE Access* **2020**, *8*, 164864–164881. [CrossRef]
35. Kant, S.; Agarwal, D.; Shukla, P.K. A Survey on Fuzzy Systems Optimization Using Evolutionary Algorithms and Swarm Intelligence. In *Computer Vision and Robotics*; Springer: Singapore, 2022; pp. 421–444.
36. Shahidinejad, A.; Ghobaei-Arani, M.; Masdari, M. Resource provisioning using workload clustering in cloud computing environment: a hybrid approach. *Clust. Comput.* **2021**, *24*, 319–342. [CrossRef]
37. Shakarami, A.; Shahidinejad, A.; Ghobaei-Arani, M. An autonomous computation offloading strategy in Mobile Edge Computing: A deep learning-based hybrid approach. *J. Netw. Comput. Appl.* **2021**, *178*, 102974. [CrossRef]
38. Sánchez-Ibáñez, J.R.; Pérez-del Pulgar, C.J.; García-Cerezo, A. Path Planning for Autonomous Mobile Robots: A Review. *Sensors* **2021**, *21*, 7898. [CrossRef] [PubMed]
39. Song, B.; Wang, Z.; Zou, L. An improved PSO algorithm for smooth path planning of mobile robots using continuous high-degree Bezier curve. *Appl. Soft Comput.* **2021**, *100*, 106960. [CrossRef]
40. Zhang, Z.; He, R.; Yang, K. A bioinspired path planning approach for mobile robots based on improved sparrow search algorithm. *Adv. Manuf.* **2022**, *10*, 114–130. [CrossRef]
41. Kovács, S.; Kóczy, T.L. The use of the concept of vague environment in approximate fuzzy reasoning. *Fuzzy Set Theory Appl.* **1997**, *12*, 169–181.
42. Klawonn, F. Fuzzy Sets and Vague Environments. *Fuzzy Sets Syst.* **1994**, *66*, 207–221. [CrossRef]
43. Huttenlocher, P.R. Synaptic density in human frontal cortex—Developmental changes and effects of aging. *Brain Res.* **1979**, *163 2*, 195–205.
44. Chechik, G.; Meilijson, I.; Ruppin, E. Neuronal Regulation: A Mechanism for Synaptic Pruning During Brain Maturation. *Neural Comput.* **1999**, *11*, 2061–2080. [CrossRef]

45. Smarandache, F. *Introduction to Neutrosophic Statistics*; Sitech & Education Publishing: Craiova, Romania, 2014. [CrossRef]
46. Patro, S.; Smarandache, F. The Neutrosophic Statistical Distribution, More Problems, More Solutions. *Neutrosophic Sets Syst.* **2016**, *12*, 73–79.
47. Smarandache, F. Neutrosophic set—A generalization of the intuitionistic fuzzy set. *J. Def. Resour. Manag.* **2010**, *1*, 107–116.
48. Smarandache, F.; Vladareanu, L.; Broumi, S.; Bakali, A.; Akram, M. Applying Dijkstra Algorithm for Solving Neutrosophic Shortest Path Problem. In Proceedings of the 2016 International Conference on Advanced Mechatronic Systems, Melbourne, Australia, 30 November–3 December 2016; p. 1.
49. Jan, N.; Aslam, M.; Ullah, K.; Mahmood, T.; Wang, J. An approach towards decision making and shortest path problems using the concepts of interval-valued Pythagorean fuzzy information. *Int. J. Intell. Syst.* **2019**, *34*, 2403–2428. [CrossRef]
50. Szili, F.A. Intelligent Travel with Fuzzy Control for Mobile Robot. Available online: https://gitlab.com/Szilidam/intelligent-travel-with-fuzzy-control-for-mobile-robot (accessed on 10 April 2022).