Temporary Goal Method: A Solution for the Problem of Getting Stuck in Motion Planning Algorithms

Danial Khan Mohamad Zade
Department of Electrical and Computer Engineering
Isfahan University of Technology
Isfahan 84156-83111, Iran
khanmohamadzade_danyal@iut.ac.ir

Samaneh Hosseini Semnani
Department of Electrical and Computer Engineering
Isfahan University of Technology
Isfahan 84156-83111, Iran
samaneh.hoseini@iut.ac.ir

Abstract—**Robotics is one of humans' most important achievements, which has gradually faced many challenges in various domains through its development. In order to improve the performance of robot motion planning, both classic algorithms and learning-based algorithms have been introduced. These algorithms, however, often encounter the problem of getting stuck. We introduce a method known as Temporary Goal that adds a mechanism to any motion planning algorithm in the event that the robot has been stopped and cannot move, and which is capable of overcoming this situation. This method has been implemented on the well-known RVO algorithm. Experiments conducted for the Temporary Goal method demonstrate an improvement in RVO's performance when dealing with stuck situations. According to the scenarios tested in this article, the performance of this method is on average over 4% higher than that of the base RVO algorithm, which has the problem of being stuck.**

Index Terms—**collision avoidance, motion planning, mobile robot, stuck.**

I. INTRODUCTION

Robotics is a branch of engineering with a wide range of applications. Robotics aims to improve human life by reducing time spent, lowering project costs, and so on. For example, in situations where humans cannot work in a specific environment and it may be dangerous for them, robots may be used. The field of motion planning for mobile robots is one of the most recent scientific fields active in the field of robotics. Motion planning for mobile robots in the environment presents many challenges, including avoiding collisions with static and dynamic obstacles, for which various algorithms have been proposed.

II. LITERATURE REVIEW

A mobile robot's motion planning entails finding a preferably optimal path and guiding it from its present location to its desired destination, avoiding collisions with dynamic and static obstacles, as well as adhering to any defined constraints of the problem. There are two important categories of motion planning algorithms: classic algorithms and learning algorithms. In classical methods, the representation of the problem is completely problem-dependent, i.e. the representation of the problem is customized for each problem. A classic motion planning algorithm can be divided into two categories depending on the environment in which it operates: search in discrete environments and search in continuous environments. In discrete environments, search algorithms aim to find a collision-free path from their start to their destination. There are examples of these algorithms in algorithms [1] and [2], which are based on A*, and also algorithms [3] and [4], which are based on RRT. The search algorithm for discrete environments only considers the geometric constraints of the robot in the problem environment. Moreover, search algorithms in continuous environments consider other factors including time constraints, dynamic constraints, speed constraints, and so on, as are required in real-world motion planning problems, such as reducing time to reach the goal, ensuring path traversal safety, and smoothening motion based on the dynamics of the robot. There are examples of these algorithms in [5], [6], and [7], which are velocity obstacle algorithms, as well as algorithm [8], which is inspired by algorithm [9]. Another category of methods that have been more current is learning methods, which have attracted researchers in the field of robotics. Learning methods provide more freedom in terms of robot dynamics and kinematic relationships, and their performance is relatively simpler and more powerful when dealing with complex relations and constraints. Several examples of these methods can be found in articles [10]–[17].

One of the most important problems in motion planning is the robot not reaching the destination within the specified maximum time. This problem, in which the robot does not move in a part of the path or exhibits oscillatory behavior, is called "stuck". This problem is observed in various motion planning methods, both classical and learning. In the field of motion planning, there are generally two types of getting stuck called deadlock and livelock. Deadlock occurs when one or more robots are unable to move because they are waiting for the path to clear so they can exit this state. In fact, these robots are stuck in a state where none of them can move. An example of a livelock is when robots are continuously changing their actions and states without being able to progress toward their goals. Like deadlock, livelock involves robots getting stuck, but instead of staying in a state, they continue to move. The efficiency and effectiveness of motion planning algorithms can be disrupted both by deadlock and livelock. The researchers employ various methods to overcome the problem of getting stuck in motion planning [17]–[19] such as the use of more complex motion planning algorithms, machine learning and optimization techniques, improved perception and sensing capabilities, and improved mobility and adaptability of robots.

Article [17] presents a hybrid approach that combines methods FMP [8] and GA3C-CADRL [16]. In this article, the robot uses FMP when it becomes stuck in local minima, otherwise, GA3C-CADRL is used. In Article [18], a Dynamic Replanning approach is considered to escape from being stuck in local minima. A bacterial foraging optimization (BFO) [20] algorithm is utilized in this paper, which can easily overcome circular obstacles, but gets stuck in local minima when confronted by non-circular obstacles. This article attempts to solve the problem of BFO by treating non-circular obstacles as virtual circular obstacles and passing through them in accordance with this approach. A search-based algorithm utilizing multi-resolution planning is presented in Article [19]. When the algorithm becomes stuck in continuous state space due to its complexity, this method attempts to refine the state space by converting continuous action space into discrete action space and avoiding duplicates in order to avoid being stuck in local minima.

It should be noted that these methods, like Articles [18], [19], are specifically designed to propose an algorithm to prevent getting stuck in local minima, and are built for this purpose, or, like Method [17], they embed a separate algorithm that aims to prevent getting stuck in local minima. Additionally, the proposed method is not actually a motion planning algorithm, but rather a solution for escaping stuck situations, which can be implemented regardless of the category of the base motion planning algorithm.

Since the focus of these algorithms is on preventing being stuck, each motion planning algorithm needs to specifically consider preventing being stuck, which leads to complex algorithm design. As an alternative, by using a second algorithm solely dedicated to preventing the motion from becoming stuck, they introduce a performance overhead on the motion planning algorithm and thus increase its complexity.

The innovation of this paper lies in the presentation of a solution that can be implemented on various motion planning algorithms, which may not have provisions for escaping from stuck positions. Consequently, this method can be applied to most motion planning algorithms and is not specific to any particular algorithm, while diverting the motion planning algorithm's focus from escaping from being stuck. The implementation of this method will also eliminate the need for the second algorithm that is specifically designed to prevent getting stuck. In this way, the algorithm for selecting the appropriate action during motion planning is simplified. The proposed method is very straightforward and does not introduce any disruption in the motion planning algorithm, because it activates during the execution phase of the motion planning algorithm upon detecting being stuck and forces the base algorithm to escape from being stuck by finding its own solution.

A temporary goal method is presented in this article as a solution to such a problem. This method provides a comprehensive approach to motion planning algorithms. As a result of detecting the behavior of the motion planning algorithm when it is stuck, the temporary goal enters into operation and provides the best solution for solving the stuck problem. For the purpose of testing the performance of the temporary goal method described in this article, the method has been implemented on the RVO [7] algorithm. The RVO algorithm demonstrates the behavior of livelocking in the event of a stuck condition. RVO algorithm tends to slow down the robot when it is in a stuck position. In order to recognize this behavior in the code, if a robot travels less than δ=0.11 meters in 8 consecutive time steps, each with a time step of Δ=0.2s, it is considered to have stuck behavior, and a temporary goal method is implemented for it.

III. METHODOLOGY

The robot uses the temporary goal method to determine its shortest path around an obstacle, based on the obstacle and current conditions. It temporarily sets aside its original goal when it becomes stuck and changes its goal in order to maneuver itself out of the situation. A temporary goal approach in motion planning may assist in finding a safe path around an obstacle or identifying an alternative route to reach the original destination. The temporary goal focuses on overcoming the obstacle and resuming progress toward the main goal. After overcoming the obstacle, the situation is reevaluated, and if necessary, new goals are set.

This section describes how to accomplish this temporary goal using the provided example: Suppose, in the scenario of Fig. 1, the purple robot using the RVO method as well as a temporary goal is stuck. The following scenario demonstrates the details of the temporary goal method, on which the temporary goal is executed. Fig. 1 illustrates three pedestrians in yellow, green, and red who are moving toward their goal, which is represented by the stars of the same color as the pedestrians. Other pedestrians have also reached their destination. A purple robot is trying to reach a purple star by avoiding three pedestrians who have stopped and blocked its path.

Due to the fact that the purple robot is stuck at the moment shown in Fig. 2a, the temporary goal algorithm begins to function. Initially, the algorithm only considers robots that it has caught around it. In order for the temporary goal algorithm to use the robots as obstacles, they must be close to a standstill, which means that the yellow-crossing robot is not considered an obstacle in the path of the purple robot. At this moment, as shown in Fig. 2a, the temporary goal algorithm considers the green, orange, and blue pedestrians as obstacles and tries to pass through them. Following this, the temporary goal

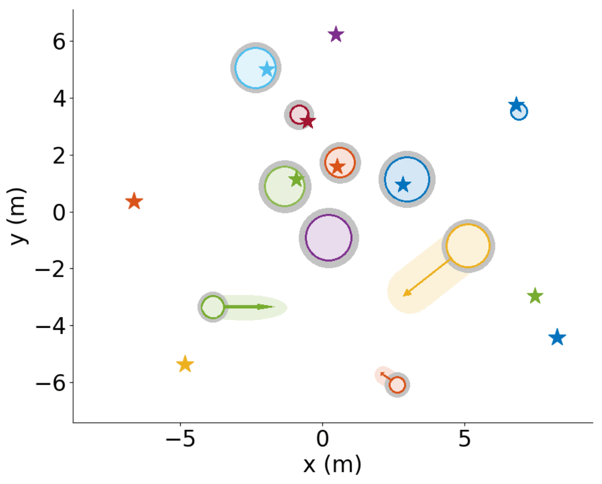


Fig. 1: A case of stuck in the combined algorithm of RVO and temporary goal is being tested.

algorithm finds the leftmost (in this case, the green pedestrian) and rightmost (in this case, the blue pedestrian) pedestrians as compared to the robot among the pedestrians it had considered obstacles.

As seen in Fig. 2b, the Euclidean distance from the starting point to the end point (the point where, if the robot is placed on the pedestrian tangent, they touch each other's surfaces) of the selected pedestrian is added to the Euclidean distance from the end point of the selected pedestrian to the main destination, and a pedestrian is chosen who has a shorter distance to the robot and the destination. We consider the pedestrian tangent instead of the pedestrian center in the calculations in order to eliminate the computational error of the distance between the robot and the pedestrians to the destination by calculating the radius of the pedestrian. As shown in Fig. 2b, using this method, the green pedestrian is selected for its shorter path.

Based on Fig. 2c, after selecting the green pedestrian at the previous stage, the temporary goal algorithm considers an imaginary robot aligned with the main robot, tangentially to the pedestrian. A safe distance of 0.2 meters is also taken into account in this tangency. By considering a temporary goal, it is necessary to adjust its angle relative to the robot and any obstructing pedestrians in such a way that the robot continues to move towards A safe distance of 0.2 meters is also taken into account in this tangency. By considering a temporary goal, it is necessary to adjust its angle relative to the robot and any obstructing pedestrians in such a way that the robot continues to move towards the temporary goal without oscillating at all and with a direct path, temporarily avoiding these pedestrians.

A temporary departure is designed to enable the temporary goal robot to return to its original goal, ensuring that it does not get stuck again in this location. The process of returning the temporary goal to the main goal is as follows: the algorithm returns to the initial stage of Fig. 2c when it detects that the distance between the robot and the temporary goal is less than the distance between the selected pedestrian(green) and the temporary goal. By tangentially creating an imaginary robot with the pedestrian (green), it reduces the angle between the destination and the robot. In the event that the angle is less than θ=30 degrees, the temporary goal is relocated to the final destination (main goal). In Fig. 2d, the robot attempts to return the temporary goal to the main goal in the second step.

It is assumed that the robot becomes stuck during the execution of the temporary goal algorithm (Fig. 2e). The algorithm in this scenario does not forget previous work to get out of being stuck and stays ready to assist the robot until it has exited the current stuck state in its entirety.

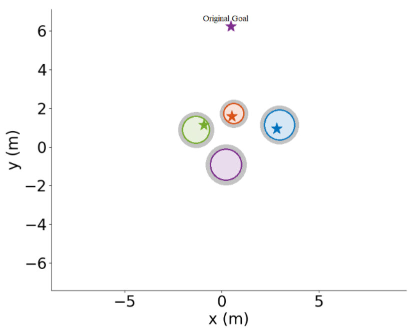
The robot, considering a wall, creates its path around pedestrians who block its path. As a result of the robot's algorithm, it is forced to stay close to the wall without colliding with it. It is actually the robot's attempt to find its way by staying close to the wall without colliding with it and continuing its journey until the end of this constraint is reached (Fig. 2f).

Based on the temporary goal method used in the RVO motion planning, Algorithm 1 shows how to select the appropriate action.

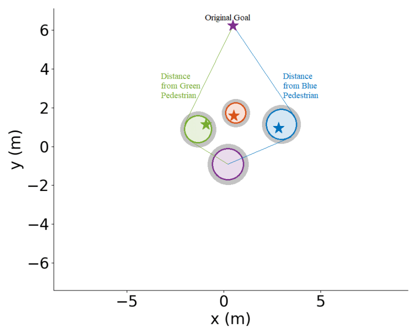
IV. RESULT

This article proposes a motion planning algorithm based on RVO with Temporary Goal called RVO+TG. The following three models of scenarios have been included in order to obtain an environment with different scenarios for comparison: circular, displacement, and completely random. A total of 100 environment models were considered for each experiment, consisting of 70% random scenarios, 15% circular scenarios, and 15% displacement scenarios. The scenarios are as follows: **Random Scenario:** In this scenario, the robots' starting and destination coordinates are placed at random within the test environment. **Circular Scenario:** In this scenario, the robots' starting and destination points are randomly positioned on a circular surface. **Displacement Scenario:** The starting and destination points of each robot are randomly selected in such a way that they end up facing each other as they attempt to reach their destination.

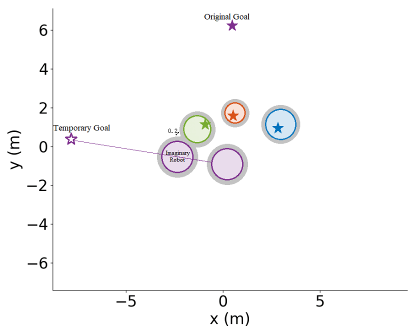
The performance of the proposed method is evaluated against other algorithms in this field in 6 evaluation criteria as follows: **Number of Stuck(Any_Stuck):** If a robot takes more than twice its average speed to reach its destination in a straight path without obstacles and fails to achieve the goal, it will be locked, and the situation will be considered stuck. **Number of Collisions:** Any scenario in which the robot under test encounters a collision is considered a collision scenario. (Note: It is possible for a scenario to simultaneously involve collisions and stucking). **Success Rate(All_Agents_at_Goal):** The test robots must reach their destination in every scenario to be considered successful. **Number of Steps:** The number of time intervals required to reach the robot's destination is counted in each scenario, and the average number of scenarios is determined. **Total Time To Goal:** For each successful scenario where all robots reach their destination, the total time



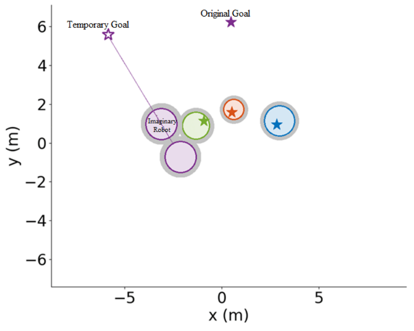
(a) The blue, orange, and green robots block the path of the purple robot to its destination.



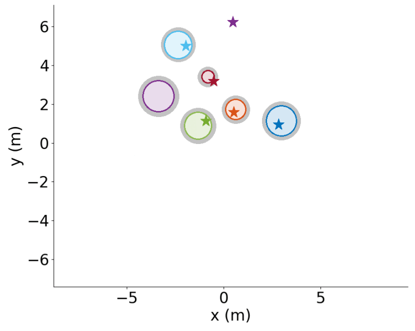
(b) Distance between the purple robot and its destination from the outermost blocking robots.



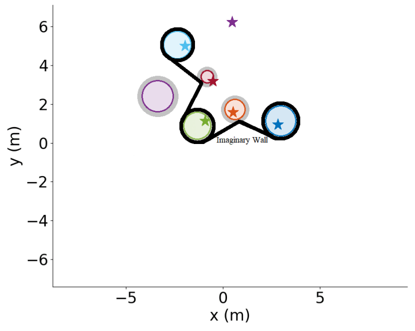
(c) Step one: Creating a imaginary robot similar to the stuck robot in contact with the last blocking robot, which is the closest path passing by.



(d) Step two: Building a imaginary robot similar to the stucked robot that is in contact with the last blocking robot.



(e) This time, three green, liver, and blue robots are blocking the path of the purple robot.



(f) A imaginary wall that the purple robot's temporary algorithm sees from its blockers.

Fig. 2: a, b, c, d, e, and f are the steps for implementing the proposed method to avoid becoming stuck in the RVO algorithm.

is calculated, and the average number of successful scenarios is calculated. **Extra Time To Goal:** In each successful scenario where all robots reach their destination, the total time is calculated, and the total time required for each robot to reach their destination directly at the desired speed is subtracted.

A test environment is defined as 4 by 4 meters when there are two to eight robots in a scenario, and 6 by 6 meters when there are 10 robots in a scenario. As shown in Table I, the temporary goal method has been able to minimize the number of collisions in the RVO algorithm. In a scenario with 10 agents in the environment, this method increased the number of agents reaching the goal by 9.2%. As a result of the use of this method in a scenario with eight agents, it has been shown that it can reduce one of the collisions in some cases, which has been deemed a significant advantage. In addition, due to the near-zero speed reduction in the RVO algorithm as a model of stuck, it has been able to find the best path to escape from these situations. This is reflected in the number of steps taken by the algorithm. As a result of reducing the number of steps and collisions, the total time to reach the agents' goal in a scenario is reduced, as evidenced by the Total_Time_to_Goal column.

V. CONCLUSION

This article presents a Temporary Goal (TG) method that reduces the stucking of the RVO algorithm significantly. In the context of stucking, by considering the deadlock or the livelock, the idea expressed in this article under the name of temporary goal can be applied to most existing algorithms in the field of motion planning. This idea is comprehensive and even learning algorithms can be adapted to this method. The concept of temporary goals can be used as a complementary method in future motion planning algorithms, where we need to eliminate stucking.

REFERENCES

[1] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke *et al.*, "Junior: The stanford entry in the urban challenge," *Journal of field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.
[2] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning a," *Artificial Intelligence*, vol. 155, no. 1-2, pp. 93–146, 2004.
[3] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects: Steven m. lavalle, iowa state university, a james j. kuffner, jr., university of tokyo, tokyo, japan," *Algorithmic and computational robotics*, pp. 303–307, 2001.
[4] J. Li, S. Liu, B. Zhang, and X. Zhao, "Rrt-a* motion planning algorithm for non-holonomic mobile robot," in *2014 proceedings of the SICE annual conference (SICE)*. IEEE, 2014, pp. 1833–1838.

TABLE I: The following table presents the results obtained by employing the temporary goal method on the RVO algorithm and its upgraded model. *3 scenarios in this section have led to the failure of the third party to reach its goal due to the risk of collision, as a result of the collision of two factors with each other and their placement on the main target.

Algorithm_Name	Num_Agents	Steps	Total_Time_to_Goal	Extra_Time_to_Goal	Collision	All_Agents_at_Goal	Any_Stuck
RVO [7]	2	51.8	2.48149	4.78399	0	100	0
RVO+TG	2	51.8	2.48149	4.78399	0	100	0
RVO [7]	4	92.86	7.04425	6.59799	2	97	1
RVO+TG	4	**89.57**	**6.78249**	**6.34299**	2	**98**	**0**
RVO [7]	6	158.23	16.82375	9.33700	**2**	95	3
RVO+TG	6	**145.86**	**15.31799**	**8.61900**	3	**97**	**0**
RVO [7]	8	214.64	23.89	**9.76099**	7	86	10*
RVO+TG	8	**187.86**	**22.80349**	9.79699	**6**	**91**	**5**
RVO [7]	10	284.42	33.38075	10.55699	4	87	10
RVO+TG	10	**209.5**	**31.01399**	**10.32599**	4	**95**	**1**

Algorithm 1

```
1: # Initial
       $define\ Global\ StuckFlag = False$
       $define\ Global\ temporary\_goal\_flag = True$
2: if $temporary\_goal\_flag == True$ then
3:     if $Agent\ Stuck\ behaviour == Stuck$ then
4:         if $StuckFlag == True$ then
5:             $change\ the\ Agent\_goal\ in\ behalf\ of\ the\ new\ pedstuck\ to\ temporary\_goal\ with\ respect\ of\ old\ side(left\ or\ right).$
6:     else if $StuckFlag == False$ then
7:         for $all\ pedestrian$ do
8:             if $closest\ less\ than\ 2 * (comfortzone)\ pedestrian\ around\ Agent$ then
9:                 if $last\ pedestrian\ on\ the\ left(right)$ then
10:                    $pedstuck = find\ pedesterian\ with\ smallest\ Euclidean\ distans\ from\ Agent\_goal\ to\ Agent\ in\ front\ of\ left(right)\ pedestrian.$
11:                $change\ the\ Agent\_goal\ in\ behalf\ of\ the\ pedstuck\ to\ temporary\_goal.$
12:                $StuckFlag = True$
13:    if $Agent\ Stuck\ behaviour \neq Stuck\ and\ StuckFlag == True$ then
14:        $\#Angle : angle\ between\ the\ three\ points\ of\ the\ Agent\_goal,\ Agent\ and\ temporary\_goal.$
15:        if $angle <= 30$ then
16:            $temporary\_goal = Agent_goal$
17:        else
18:            $angle = reduce\ angle\ to\ 30\ degrees$
19:            $StuckFlag = False$
```

[5] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Autonomous Robot Vehicles*. New York, NY: Springer New York, 1986, pp. 396–404.

[6] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The international journal of robotics research*, vol. 17, no. 7, pp. 760–772, 1998.

[7] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *2008 IEEE international conference on robotics and automation*. Ieee, 2008, pp. 1928–1935.

[8] S. H. Semnani, A. H. de Ruiter, and H. H. Liu, "Force-based algorithm for motion planning of large agent," *IEEE Transactions on Cybernetics*, vol. 52, no. 1, pp. 654–665, 2020.

[9] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Transactions on automatic control*, vol. 51, no. 3, pp. 401–420, 2006.

[10] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4298–4305, 2019.

[11] A. Francis, A. Faust, H.-T. L. Chiang, J. Hsu, J. C. Kew, M. Fiser, and T.-W. E. Lee, "Long-range indoor navigation with prm-rl," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1115–1134, 2020.

[12] H.-T. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with autorl," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2007–2014, 2019.

[13] U. Patel, N. Kumar, A. J. Sathyamoorthy, and D. Manocha, "Dynamically feasible deep reinforcement learning policy for robot navigation in dense mobile crowds," *arXiv preprint arXiv:2010.14838*, 2020.

[14] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 285–292.

[15] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1343–1350.

[16] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3052–3059.

[17] S. H. Semnani, H. Liu, M. Everett, A. De Ruiter, and J. P. How, "Multi-agent motion planning for dense and dynamic environments via deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3221–3226, 2020.

[18] M. I. I. Abdi, M. U. Khan, A. Güneş, and D. Mishra, "Escaping local minima in path planning using a robust bacterial foraging algorithm," *Applied Sciences*, vol. 10, no. 21, p. 7905, 2020.

[19] W. Du, S.-K. Kim, O. Salzman, and M. Likhachev, "Escaping local minima in search-based planning using soft duplicate detection," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 2365–2371.

[20] S. Das, A. Biswas, S. Dasgupta, and A. Abraham, "Bacterial foraging optimization algorithm: theoretical foundations, analysis, and applications," *Foundations of computational intelligence volume 3: Global optimization*, pp. 23–55, 2009.