IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 15 November 2023, accepted 28 November 2023, date of publication 1 December 2023,
date of current version 13 December 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3338566

RESEARCH ARTICLE

A Hybrid AI-Based Adaptive Path Planning for Intelligent Robot Arms

ALI ABDI AND JU HONG PARK, (Member, IEEE)
Department of Convergence IT Engineering, Pohang University of Science and Technology (POSTECH), Pohang 37673, South Korea

Corresponding author: Ju Hong Park (juhpark@postech.ac.kr)

This work was supported in part by the FoodTech RnD Center Development and Support Program through the Gyeongbuk Technopark (GBTP) funded by Gyeonsangbuk and Pohang under Grant GBTP2023129001; and in part by the National Research Foundation of Korea (NRF) Grant funded by the Korea Government (MSIP–Ministry of Science, ICT and Future Planning) under Grant 2019R1G1A1010859.

ABSTRACT Intelligent robot arms are advanced robotic systems used in Industry 4.0 to perform complex tasks. Unlike conventional robot arms, which perform predefined tasks, intelligent robot arms have autonomy and can operate in changing environments, interact with other machines, and collaborate with humans. In this regard, adaptive path planning is crucial for intelligent robot arms, involving real-time environment monitoring and path generation to continuously update the robot's trajectory based on changes in the surroundings. This paper presents an adaptive path planning method for intelligent robot arms to be used in dynamic environments. The proposed method is based on a hybrid active-passive approach and has been tested in a dynamic workspace simulation environment. The results indicate the ability of the proposed method to respond dynamically in a complex scenario where the target is fluctuating, and an obstacle is intentionally placed in the robot's path. Additionally, real-time analysis results show that the method can be categorized as real-time path planning with less than 100 ms reaction time for grid sizes with less than $96 \times 96 \times 96$ cells. This insight presents opportunities for the establishment of smart factories, smart homes, and smart cities, where the presence of intelligent robot arms in dynamic environments becomes essential.

INDEX TERMS Adaptive path planning, target reaching, obstacle avoidance, dynamic environment, intelligent robot arm.

I. INTRODUCTION

Intelligent robot arms are advanced robotic systems designed to perform complex tasks in various industrial settings. These robot arms are typically equipped with sensors, cameras, and other technologies that enable them to ''see'' and ''feel'' their surroundings, as well as to communicate with other machines and systems. In the context of Industry 4.0, intelligent robot arms play a critical role in the ''smart factory'' concept, which involves the use of advanced technologies such as the Internet of Things (IoT), Artificial Intelligence (AI), and big data analytics to create more efficient and connected industrial systems. By integrating intelligent robot arms into these systems, businesses can achieve greater flexibility, adaptability, and efficiency, enabling them to better meet customers' needs and respond to marketplace changes. Intelligent robot arms are also a key component of automation, which involves using technology to streamline and optimize industrial processes. By automating certain tasks, businesses can increase their productivity, reduce costs, and improve the quality and consistency of their products.

Conventional robot arms are typically pre-programmed to perform specific tasks in a controlled environment with fixed locations of objects. These robots have a limited ability to adapt to changes in their environment and require human intervention to reprogram them if the task or environment changes. In contrast, intelligent robot arms have a higher level of autonomy and are designed to operate in dynamic and unstructured environments with changing locations of objects. They are equipped with advanced sensors, cameras, and machine learning algorithms that enable them to perceive their environment, make decisions based on that perception, and adjust their movements accordingly. Intelligent robot arms can also interact with other machines, systems,

The associate editor coordinating the review of this manuscript and approving it for publication was Thomas Canhao Xu.

and humans in a collaborative manner. For example, they can work with humans to perform tasks that require both manual dexterity and cognitive skills, such as assembling complex products. They can also communicate with other machines and systems in the factory to optimize production processes and reduce downtime. In this regard, path planning for intelligent robot arms is entirely different from that for conventional robot arms. In the conventional robot arm workspace, if a collision-free path is generated, the robot can track it without any concern for collision with obstacles or loss of targets because their locations are fixed during the time. Conversely, in an intelligent robot arm workspace, even though a collision-free path is generated at the initial time, the safety of the path for the next moment cannot be guaranteed because the location of the objects may change in a fraction of a second. Therefore, for intelligent robot arms, the generated path must be updated continuously based on the environment changes, which is called adaptive path planning.

Adaptive path planning typically involves two main components: real-time environment monitoring and real-time path generation. Real-time environment monitoring involves continuously sensing and monitoring the environment around the robot arm, using sensors such as cameras, lidar sensors, or other types of sensors. This allows the robot arm to detect changes in the environment, such as the presence of new obstacles, and to adjust its trajectory accordingly. Real-time path generation involves dynamically generating a new path for the robot arm based on the current state of the environment and the robot arm's feedback. This can involve using algorithms such as motion planning, trajectory optimization, or Reinforcement Learning (RL) to generate a new path that optimizes the robot arm's performance, minimizes energy consumption, and avoids collisions with obstacles. By combining real-time environment monitoring with real-time path generation, adaptive path planning enables the robot arm to navigate through complex, dynamic environments while adapting to changes in the environment in real time.

Considering the preceding discussion, the primary purpose of this paper is to develop an adaptive path planning method for intelligent robot arms which typically work in dynamic environments. To this end, we first examine related works toward path planning in the next section.

II. RELATED WORKS

In this section, a review of path planning methods is presented. The review is split into two subsections, including conventional path planning, which is mainly in static environments where robot arms perform predefined tasks, and adaptive path planning, which is mainly in dynamic environments where robot arms must be able to update the path in real time. Finally, the contribution of our work to existing papers is explained in detail.

A. CONVENTIONAL PATH PLANNING

In recent decades, researchers have introduced numerous techniques to enhance the path planning of robot arms. These techniques encompass various algorithms such as Artificial Potential Field (APF), Probabilistic Road Maps (PRM), Rapidly Exploring Random Trees (RRT), polynomial-based approaches, Machine Learning-based methods, and Hybrid methods.

Several scholars have employed the Artificial Potential Field method for robot arm path planning. Xinkai Xia et al. have devised a proficient and secure algorithm for the path planning of medical robot arms. They utilized the velocity potential field (VPF) algorithm, a modified version of the APF method. This innovation ensures both safety and efficiency in the functioning of the medical robot arms [1]. Sun-Oh Park et al. introduced an advanced numerical technique based on Jacobian calculations. Their method seamlessly integrated the modified APF approach, enabling efficient real-time computations of inverse kinematics and collision-free path planning. This approach catered to both redundant and non-redundant scenarios, further enhancing its versatility and applicability [2]. Gai et al. developed a comprehensive path planning algorithm for a 6-degree-of-freedom (DOF) industrial robot. Their approach relied on the APF technique, enabling effective navigation and trajectory planning for the entire robotic arm. This algorithm offers a practical solution for optimizing the movement of the industrial robot and enhancing its overall performance [3]. Lin et al. put forth a novel approach for the path planning of robot arms by introducing a three-dimensional APF method. Their research offered valuable insights and strategies for effectively navigating and avoiding collisions in complex environments [4].

Other scholars utilized the Probabilistic Road Maps and Grid Roadmap (GRM) methods for robot arm path planning. Iqbal and his team presented a path planning algorithm that utilizes the PRM technique [5]. Bahar et al. introduced a novel approach centered around the GRM technique, which stands out for its edgeless roadmap design [6]. In a separate publication, Bahar et al. also suggested a technique for reducing the desired time required for path planning. Their method relied on the utilization of a GRM [7].

Some scholars focused on the Rapidly Exploring Random Tree algorithm for robot arm path planning. Wei et al. introduced an autonomous obstacle avoidance method for robotic manipulators, which involved dynamic path planning. The proposed approach incorporated an enhanced RRT algorithm [8]. Liu et al. presented an enhanced version of the RRT algorithm by introducing a random point generation mechanism based on probability. However, this method faces several challenges, such as generating cubic graphs, producing irregular paths, and resulting in non-optimal paths [9].

Polynomial-based methods were employed by some scholars for robot arm path planning. Lai et al. introduced a collision avoidance path planning method that utilizes non-uniform rational B-splines (NURBS) and a continuous polynomial function [10].

Machine Learning-based methods were also utilized for robot arm path planning. Ionescu attempted to tackle the challenge of employing dynamic path planning with obstacle avoidance through the utilization of machine learning techniques [11]. Khan et al. devised a technique enabling a robot with redundant manipulation capabilities to accurately track its real-time movement without relying on a model. This method employs a controller that utilizes the Zeroing Neural Network with Beetle Antennae Search (ZNNBAS) algorithm to guide the robot's motion [12]. Wen et al. introduced a novel obstacle avoidance algorithm that builds upon the deep deterministic policy gradient (DDPG) framework, an established deep reinforcement learning approach [13]. Prianto et al. team put forward a path planning algorithm based on the Soft Actor-Critic (SAC) approach. This algorithm was specifically designed to address the challenges associated with reinforcement learning in path planning [14]. Zhang et al. introduced a modular approach aimed at effectively acquiring and transferring visuomotor policies from simulation to real-world scenarios. This method was developed to enhance the efficiency of learning and deployment processes [15]. Ji et al. employed approximate regions rather than precise measurements to define a new state space and joint actions within the Q-learning method [16].

Hybrid methods combining different techniques were employed by some scholars for robot arm path planning. Qingni Yuan et al. presented a hybrid algorithm, APF-RRT (Artificial Potential Field - Rapid Expansion Random Tree), as an improved robot arm path planning method. This algorithm combines the strengths of an enhanced artificial potential field and the rapid expansion random tree technique [17]. Abdi et al. employed a hybrid path planning approach for a 2D workspace, integrating both Q-learning and neural networks [18]. Furthermore, they expanded their research to encompass more intricate scenarios, such as a 3D workspace [19]. This approach encompasses a two-phase path planning process: active and passive. During the active phase, the Q-learning algorithm is utilized to determine a series of basic actions (such as up, down, left, and right) required to navigate through a gridded workspace and reach the target cell. In the passive phase, the trained neural network is employed to determine the joint angles of the robot arm based on the identified actions.

B. ADAPTIVE PATH PLANNING

In contrast to conventional path planning methods, adaptive path planning requires real-time capabilities due to the dynamic nature of the robot's environment, where object positions may change. Researchers have made efforts to develop real-time path planning by enhancing existing methods and introducing new algorithms. Various real-time algorithms based on Artificial Potential Field, Probabilistic Road Maps, polynomial functions, and Machine Learning have been explored.

Some scholars have utilized improved APF methods for adaptive path planning of robot arms. Lufeng Luo et al. introduced a path planning method for six-degree-of-freedom harvesting robot arms operating in dynamic, uncertain environments. They utilized energy optimization techniques and APF to devise an efficient approach [20]. Chen et al. employed the APF approach to enable real-time path planning for manoeuvring manipulators while avoiding obstacles [21].

Improved PRM methods have also been utilized for adaptive path planning. Kunz et al. proposed a method for real-time movement planning of robot arms in dynamic environments. They utilized data from 3D sensors and a technique known as a Dynamic Roadmaps (DRM) to enable the arms to navigate through changing surroundings [22].

Additionally, researchers have investigated enhanced approaches that utilize polynomial-based for adaptive path planning. Lim et al. presented a novel polynomial-time approximation algorithm called Recursive Adaptive Identification (RAId) for adaptive informative path planning. This algorithm enables efficient and effective determination of paths in real-time [23]. Guanglei et al. devised a real-time movement planning approach for robots that involves the utilization of polynomial curves and non-uniform rational B-spline (NURBS) curves. This method enables the design of precise trajectories for the robot's movement [24]. Hameed et al. introduced an innovative real-time path planning technique for a stationary robot arm aimed at collision avoidance with obstacles. This method effectively combines the advantages of both global and local techniques. It revolves around the concept of configuring the robot arm to manipulate its joints based on a specific polynomial selection, thereby addressing the inverse kinematics problem [25].

Furthermore, scholars have explored improved Machine Learning-based methods for adaptive path planning. Li et al. put forth an enhanced autonomous learning algorithm based on Q-Learning. The algorithm was specifically developed to tackle the challenge of adaptive path planning for space manipulators operating in unknown environments [26]. Wang et al. devised an adaptive path planning approach for a welding robot system. This method utilizes a discrete genetic algorithm that operates on an inverted pyramid structure (GA-IPS). The algorithm enables efficient and effective path planning tailored to the specific requirements of the welding tasks [27]. Aoughellanet et al. introduced a trajectory planning method with real-time capabilities that incorporates obstacle avoidance. This method utilizes a recurrent neural network with carefully constrained interconnections to ensure efficiency and effectiveness in avoiding obstacles during trajectory planning [28]. Kamali et al. presented an approach known as DGDRL (Dynamic-goal Deep Reinforcement Learning) for real-time movement planning of robots engaged in telemanipulation tasks. This method utilizes reinforcement learning techniques to adapt the robot's movements according to its defined goals, ensuring effective and goal-oriented control [29]. Wang et al. introduced a cutting-edge method for real-time mapping and dynamic shortest path planning of robot arms. This method harnesses the power of the D* algorithm and is specifically designed to be highly

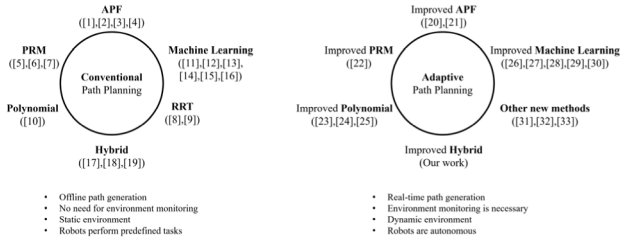

FIGURE 1. Conventional and adaptive path planning algorithms. The contribution of our work (improved hybrid method) to existing papers.

efficient and responsive. It is suitable for deployment with low-cost depth cameras and robot arms, enabling accurate and timely path planning in dynamic environments [30].

Other approaches for adaptive path planning have been explored as well. Hanna Berggren developed an advanced planning system comprising three key components: human tracking, robot path planning, and a decision algorithm. This high-level system integrates these elements to facilitate effective coordination between humans and robots, enabling smooth and efficient execution of tasks [31]. Ding et al. introduced a novel strategy for adaptive path planning in the wire-feed additive manufacturing process. This innovative approach utilizes medial axis transformation (MAT) to enable effective and efficient path planning, ensuring optimal trajectory generation for the additive manufacturing process [32]. Li et al. focused on Ping-Pong robotics, a well-known case study that demands the high-speed vision to precisely control dynamic movements. Their research specifically tackled the challenges associated with achieving accurate motion control in fast-paced Ping-Pong-playing scenarios [33].

Based on the analysis in section II-A, although the hybrid method discussed addressed complexity and slowness, it cannot be categorized as adaptive path planning since it lacks real-time capabilities. However, with improvements, this hybrid method can potentially become a real-time path planning approach suitable for dynamic environments. Thus, this study aims to enhance the hybrid method [18], [19] to increase path-finding speed, transforming it into a real-time method applicable to adaptive path planning.

Similar to the presented hybrid methods [18], [19], our approach involves active and passive phases in the path generation process. However, to enhance it into a real-time method, we will utilize a faster algorithm instead of Q-learning during the active phase.

In this study, we will demonstrate our method's ''real-time'' performance, which is crucial for adaptive path planning, by analyzing the overall execution time of the path planning process, including environment monitoring and path generation. We will validate our approach through simulations conducted in dynamic environments. The contribution of our method to existing works is depicted in Figure 1.

The structure of the paper is as follows. Section III provides a comprehensive explanation of the theoretical foundation of the methodology. The preparation and implementation of the method are detailed in Section IV. Section V presents the real-time analysis, simulation results, and a discussion. Finally, the paper is concluded in Section VI.

III. METHOD DESCRIPTION

In order to improve the hybrid method presented in [18], [19] to become a real-time path planning, in the first phase, the D* Lite algorithm is used to find a sequence of nodes to navigate from the start cell to the target cell while avoiding obstacles in a gridded workspace. The output of this phase is a set of cell indexes. In the second phase, a pre-trained neural network is used to convert these cell indexes into joint angles. This is beneficial because, in the node-finding process, there is no need to deal with the kinematics of the robot arm to find joint angles since we find the angles through the neural network in the passive section. Also, there is no need to run a time-consuming neural network process since the neural network's weights have already been trained, which saves computational time.

The first phase of hybrid path planning, which involves finding nodes, is referred to as the active phase. During this phase, a heuristic search is conducted to identify the best path, which requires the robot to utilize an active section of memory and a computer brain. The second phase, which involves finding angles, is referred to as the passive phase. In this phase, the neural network does not require training each time it is used, as it has already been trained once and can be used indefinitely. This means that the robot uses a passive memory section during this phase without needing learning or training processes.

Combining these two active and passive phases improves the speed of path planning, enabling real-time path planning. The proposed approach to path planning optimizes each phase separately, with the active phase utilizing a heuristic search and the passive phase relying on a pre-trained neural network. The overall approach is more efficient compared to previous methods that involved finding joint angles during the search process, which was time-consuming.

In this section, first, we will explain our environment monitoring algorithm. Then, we will describe our real-time hybrid path planning method thoroughly.

A. ENVIRONMENT MONITORING

As mentioned in the last section, real-time environment monitoring is essential in adaptive path planning. In this regard, in this subsection, we are going to explain our combined computer vision algorithm for real-time monitoring. In real-time environment monitoring, the locations of the obstacles and the target in the 3D workspace should be continuously measured. Consequently, the first step is determining the 3D coordinates of the locations of the objects in the robot arm's workspace. To do this, we use a combination of the You Only Look Once (YOLO) object detection algorithm, specifically the variant called tiny-YOLOv4, and a neural network. The YOLO algorithm [34] is used to detect the bounding boxes and classes of objects in different views of 2D images. Then,

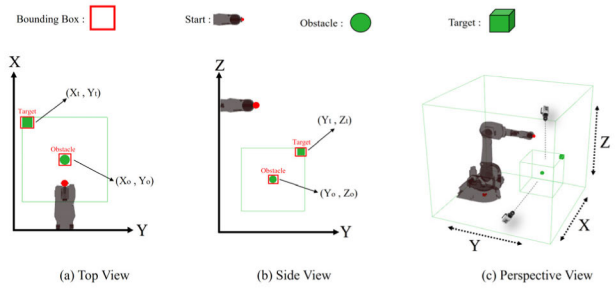

FIGURE 2. Three-dimensional workspace generated by a combination of top and side views: (a) Top view; (b) Side view; (c) Perspective view.

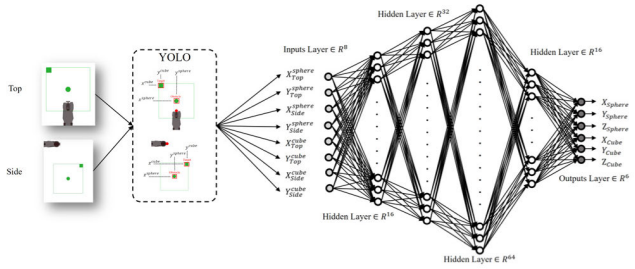

FIGURE 3. Combined computer vision algorithm comprising the YOLO object detection algorithm and coordinate-finding neural network.

a neural network is used to convert the 2D bounding boxes to 3D coordinates.

Accordingly, two cameras capture 2D images from the top and side views. These pairs of images produce a unique configuration of the obstacle and target in the 3D workspace. Notably, we use two perpendicular cameras for the following two primary reasons.

First, the precise depth of objects cannot be determined using only one simple RGB camera as this type provides information about planner images from which only 2D coordinates can be extracted. Instead, we may need to use a depth or stereo camera, which can also provide information about the depth of objects, to determine the accurate depth in future work.

Second, when objects are overlapped, and one is hidden behind the other, detecting the hidden object would be impossible with a single camera. In this case, neither a simple RGB nor a depth camera can see the hidden object. Therefore, we use two RGB cameras with two different views to determine the depths of the objects as well as ensure that no objects are not missed. However, when an object is hidden behind the other, determining the depth is still a problem; this case will be considered in our next study. For this case, the solution is to use two perpendicular depth cameras, each of which can determine the depth independently. This idea will be discussed in the future work section.

To better understand the environment monitoring method, we explain the process of our combined computer vision-based environment monitoring process using simple objects in a simple workspace. To this end, similar to our previous study [19], we use a 5 cm $\times$ 5 cm $\times$ 5 cm cube as a target object for the robot arm. Moreover, we use a 5 cm diameter sphere as an obstacle object with which the robot arm should avoid a collision. Unlike the paper mentioned above, we no longer use an object as the starting object in this study. Instead, we use the current location of the end-effector as the initial point since we want to update the current path instead of planning a path. The current location of the end-effector can be either determined using a localization method or measured via forward kinematics using current joint angles. Based on the above explanations, a simple schematic of the 3D workspace and objects is shown in Figure 2.

In the localization method, initially, two pairs of images are input to the YOLO algorithm to obtain the size and location of bounding boxes of the cube and sphere in both the top- and side-view images. We use this information to compute the center coordinates of objects in the images. As there are two objects, and each object has two X and Y central coordinates for each bounding box, each view has two central points with four parameters (two sets of X and Y coordinates). Consequently, a total of eight parameters can be extracted from the top and side images. However, we need to determine the coordinates of objects in a 3D workspace. These eight parameters are input into the neural network in the next step to determine the 3D coordinates of the sphere and cube. Hence, the neural network computes six output parameters: the X, Y, and Z coordinates of the sphere and cube (stereo depth approach). This network contains four hidden layers with 16, 32, 64, and 16 neurons, respectively.

It is worth mentioning that the discussion of this so-called stereo depth algorithm in our simulation is a key part of our plan to make sure our method works well in real-life situations with actual robots. We have intentionally included this algorithm in our simulation to deal with the challenges that come up when robots operate in the real world.

This is important because it is like a link between our theoretical ideas and making them work in real situations. We know that our method needs to handle the difficulties of real-world robot tasks, especially when it comes to figuring out how far things are. So, the stereo depth algorithm is a crucial tool. It helps us fine-tune and check our method in a simulation that closely mimics real-world conditions.

Basically, our simulation is a place where we can test how well the stereo depth algorithm deals with the same problems robots will face in the real world. This careful preparation ensures that our method can handle the ups and downs of real-world robot situations. It is not just about testing ideas on a computer; it is about getting ready for the challenges that our method will face in the unpredictable environments where real robots operate.

The architecture of the combined computer vision algorithm is depicted in Figure 3.

Once the positions of the obstacle and target in the spatial workspace are determined, we can use them to update the collision-free path from the current location of the

end-effector to the detected target. This process is explained in the next section.

B. PATH GENERATION

Each path generation algorithm must cover both target reaching and obstacle avoidance [35]. That is, the primary objective is to determine a sequence of joint angles that provide a path from the current location of the end-effector to the target point while avoiding collisions with obstacles. As already mentioned, in this paper, we use a hybrid path generation method to find a safe sequence of joint angles, which active and passive phases are explained in the following subsections.

1) DETERMINING NODES (ACTIVE PHASE)

In order to find a path from the end-effector's current location to the target's location while avoiding obstacles, we split the workspace into identical grids called cells. The main aim is to determine a sequence of collision-free cell indexes from the current cell to the target cell, where the center of the target object is located. In our last two papers [18], [19], we used the Q-learning algorithm, which is a kind of reinforcement learning algorithm, for this purpose. It determined a sequence of actions, including up, down, left, right, backward, and forward from the initial cell to the target cell. An end-effector could reach the target following this sequence of actions.

Although this method has several advantages, it still suffers from non-real-time path generation because reinforcement learning algorithms are uninformed algorithms that work with no information about the search space. In contrast, informed algorithms aid in path generation with certain available information about the target and obstacle. In this study, to take advantage of informed algorithms for real-time path generation, we substitute an informed algorithm with Q-learning to increase the speed of path generation, which leads to real-time path planning.

To achieve the desired results, we use D* Lite, one of the fast algorithms among informed-type algorithms [36]. It is a sophisticated algorithm for calculating the optimal path while considering a variety of situations, particularly in the field of robotics. The algorithm is able to find the shortest route and can also adjust the path if needed. One of the key features of D* Lite is its ability to handle dynamic environments, which is particularly useful in situations where objects or obstacles may be constantly moving. This can be especially significant when dealing with fast-moving objects that require a quick response.

D* Lite tends to be more efficient in known environments as it utilizes existing information about the map or terrain, reducing redundant exploration and focusing on the most promising paths. Additionally, as D* Lite utilizes a heuristic based on prior information, it often converges to optimal paths swiftly in environments where the map is known. Q-learning, on the other hand, may take longer to converge and may not guarantee optimality. Furthermore, D* Lite's informed approach allows it to use resources more effectively by focusing on areas of interest or changes in the environment, optimizing path updates and recalculations. Moreover, Q-learning requires exploration to learn optimal paths, which can be time-consuming and resource-intensive, especially in complex or large environments. D* Lite, by using known information, reduces the need for extensive exploration.

Unlike Q-learning, which is used to determine a sequence of actions, D* Lite is used to determine a sequence of nodes (cell center indexes) from the current cell to the target cell. First, the cost of each edge connecting the current node to its neighboring nodes is calculated to determine the shortest path from the starting node to the target node. If an obstacle is encountered, the most cost-effective route is recalculated, and a revised list of nodes is created [37]. At the end of the node-finding process, we have a sequence of cell indexes that begin from the current cell and terminate at the target cell. The algorithm finds cells that are not in the obstacles category, implying that the path is collision-free.

To understand this method, as an example, imagine that the current positions of the end-effector, desired target, and obstacle are in cells with indexes (i_C, j_C, k_C), (i_T, j_T, k_T), and (i_O, j_O, k_O), respectively. A sequence of nodes is determined to connect the current cell to the target cell, while the obstacle cell is not among the path set elements. The path set has "n + 2" elements, including the current cell at the beginning, the target cell at the end, and "n" other elements between them, where "n" is a natural number. This is expressed as follows:

$$(i_O, j_O, k_O) \notin \{(i_C, j_C, k_C), (i_1, j_1, k_1), (i_2, j_2, k_2), \ldots (i_n, j_n, k_n), (i_T, j_T, k_T)\} \quad (1)$$

Notably, the target index at the end of the path sequence illustrates the target-reaching concept, and the absence of the obstacle index in the path sequence illustrates the obstacle-avoidance concept, which are the two essential requirements in robot arm path generation in dynamic industrial environments where intelligent robot arms work.

Apart from the difference in the type of elements in the path sequences, there is another difference between path-finding using the Q-learning and D* Lite algorithms: their speed. D* Lite, as an informed algorithm, is significantly faster than Q-learning. As Q-learning is uninformed, the grid size significantly influences the running time. Conversely, in D* Lite, the grid size has fewer effects on the running time. That is because, in an uninformed algorithm, all cells should be considered during the path search. In contrast, in an informed algorithm, only the cells related to the potential path, that is, the cells between the initial and target cells, are considered.

2) FINDING ANGLES (PASSIVE PHASE)

After determining the sequence of nodes for the new path, these must be converted into joint angles to be implemented in the robot arm. Similar to our previous studies, we use a trained neural network as a passive phase. Indeed, utilizing the neural network depicted in Figure 4, we map each point

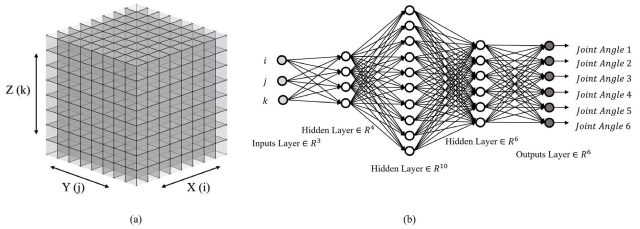

FIGURE 4. (a) Three-dimensional gridded workspace; (b) Structure of the angle-finding neural network.

TABLE 1. Overview of algorithms.

Stage	Algorithm	Input	Output
Environment monitoring	YOLO	Two images	Four 2D points
	Neural Networks 1	Four 2D points	Two 3D points
Path generation	D* Lite	Two 3D points	Sequence of path indexes
	Neural Networks 2	Path indexes	Joint angles

of the 3D workspace to the corresponding joint angles in a particular orientation. This method splits the path generation task into two different functions: node-finding (active) and angle-finding (passive) processes. As explained in the last section and our previous papers, this approach significantly increases the speed of path planning.

The structure of the angle-finding neural network used in this study is marginally different than that in the previous paper, as shown in Figure 4. This is because, unlike the last article in which we used action and cell indexes as inputs, here, we use only cell indexes because D* Lite, unlike Q-learning, does not have action indexes. Therefore, three inputs are used, cell indexes i, j, and k, representing the X, Y, and Z axes, respectively, and six outputs are obtained (joint angles one to six) as the robot arm has six degrees of freedom. The neural network also includes three hidden layers with four, ten, and six neurons, respectively. A rectified linear unit (ReLU) is the activation function for the input and output neurons.

At the end of the angle-finding process, we have a sequence of joint angles (a path) that guides the end-effector from the current to the target position. The robot follows a path that is continuously updated. This path does not change so long as the locations of the objects are the same concerning the last moment. Once the locations change, the path is updated, and the robot follows the new path. Therefore, we can implement the determined joint angles to the robot arm and test it in the simulation environment.

In Table 1 we summarized each algorithm used in both environment monitoring and path generation stages, including YOLO, first neural network, D* Lite, and second neural network.

In Figure 5, the block diagram of the proposed method is presented as an overview of the algorithm, too. In each step, the input and output of each block are shown and mentioned, trying to make the details clearer. As it shows, the algorithm

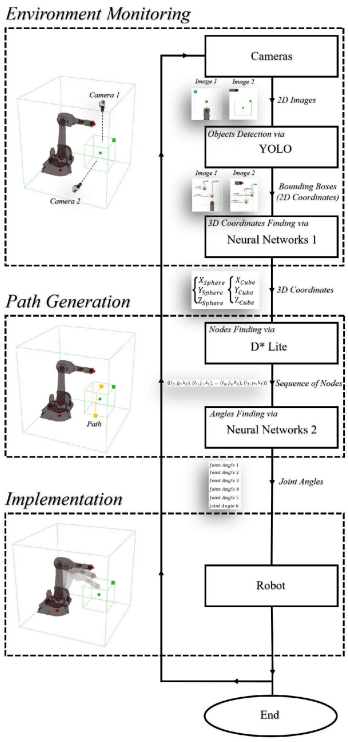

FIGURE 5. Block diagram of the proposed robot arm path planning.

monitors the workspace and finds the location of objects. Given the locations of the objects, a new path is generated and implemented in the robot. This process will continuously work until the robot reaches the target.

Implementing the method in the simulation environment is explained thoroughly in Section IV.

IV. PREPARATION AND IMPLEMENTATION OF THE METHOD

This section describes the steps to prepare the dataset, train the algorithms, and implement the method in a simulation environment. Furthermore, the simulation environment framework is explained in detail.

A. DATA GENERATION AND TRAINING

Before the implementation, we need to train the three learning parts of the algorithm: the YOLO and coordinate-finding neural network from the environment monitoring section, as well as the angle-finding neural network from the passive path generation section. Appropriate datasets are required for training these parts.

To train the YOLO, we need a dataset of images from the top and side views of the workspace. The dataset should cover the possible object arrangements to ensure that the object detection algorithm will be generalized. With respect to this, we randomly locate the sphere and box in the workspace.

For the coordinate-finding neural network, we need a dataset of image pairs in which the objects are located randomly, but their locations are known. That is because these locations are indeed the labels of the neural network, whereas the bonding boxes' information is the input. In this part,

the dataset should also cover most possibilities of object arrangements to ensure that the neural network is generalized.

We need a dataset of cell indexes and their corresponding joint angles to train the angle-finding neural network. The cell indexes are the network inputs, and the joint angles are the labels of the data points. As the number of cells increases (in fine grid sizes), the number of data points increases as well. Again, the dataset should also cover most possibilities of cell indexes and joint angles to generalize the neural network.

To collect datasets for the first two parts (YOLO and the coordinate-finding neural network), we use the Rhinoceros 3D software (Rhino). Rhino is a software program, developed by Robert McNeel and associates in 1998, used for various purposes, including computer-based designing and manufacturing products, creating prototypes, 3D printing, and analyzing existing products to understand their design and construction. To generate the dataset, we import the IRB 1600 ABB model, the model we use in our lab, into Rhino software. After that, with the help of the Python script inside the Rhino, we generate random locations of the sphere and box in the robot arm workspace and save the images of the top and side views.

For the angle-finding neural network, we use the RoboDK simulation software for robot arms that can convert the coordinates to corresponding joint angles and vice versa. In RoboDK, like Rhino, we import the IRB 1600 ABB model to collect the data points in a 3D gridded workspace. We can read the corresponding joint angles using Python script for random locations of the end-effector. By training the neural network exclusively on data from the robot's physically accessible workspace, we guarantee that any paths generated by the network will always be feasible and attainable.

Once the dataset is ready, we can train the different networks in the learning algorithm to use it in the path planning method. Well-trained YOLO and neural networks can achieve high accuracy in path planning results. In contrast, improper training can result in a collision and the inability to reach the target. For example, YOLO may not be able to detect objects well, the coordinate-finding neural network may not obtain the coordinates of objects, and the angles-finding neural network may not find appropriate joint angles.

Table 2 presents the training protocols for each training algorithm to show the data generators, algorithms' architecture, as well as other training information such as number of iterations, absolute errors, etc. Notice that the D* Lite algorithm is an incremental heuristic search algorithm, not a training algorithm, so we do not present it in this table.

After training each AI algorithm separately, they are combined with working in a synchronized manner in a simulation environment to update a collision-free path for reaching the target. The framework of this simulation environment is elaborately discussed in the next section.

B. SIMULATION ENVIRONMENT FRAMEWORK

In this section, we develop a simulated dynamic workspace environment for testing and evaluating our adaptive path planning method. We design the simulation environment framework via multi-software synchronization, which is subsequently explained in detail.

In fact, we need a framework to use trained algorithms, explained in Section IV-A, to work together as an integrated

TABLE 2. Training protocol for each algorithm.

Algorithm	Data Generator	Architecture	Training Information
YOLO	Rhino (3D model)	-Tiny_YOLOv4 structure	-Iterations: 6000 -Labeled via "labelimg" library
Neural Networks 1	Rhino	-Input layer: 8 neurons -Four hidden layers: 16, 32, 64, 16 neurons -Output layer: 6 neurons	-Iterations: 1500 -Absolute error: 0.03
Neural Networks 2	RoboDK	-Input layer: 3 neurons -Three hidden layers: 4, 10, 6 neurons -Output layer: 6 neurons	-Iterations: 150 -Absolute error: 0.01

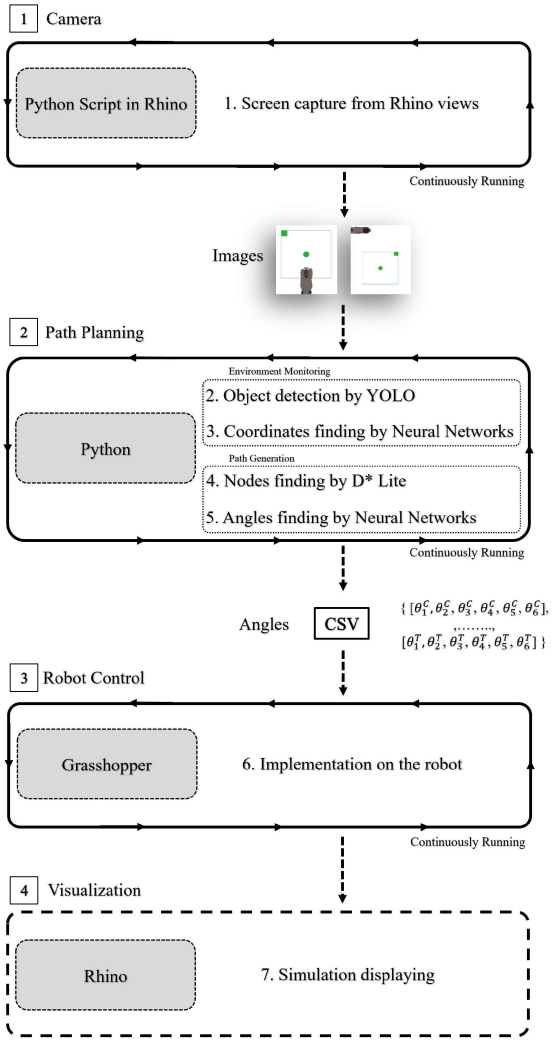

FIGURE 6. Flowchart of the simulation environment framework organized via multi-software synchronization.

system, independently and synchronised. The flowchart of this simulation environment is illustrated in Figure 6. In this framework, the results from units 1 to 3, which are continuously running, are updated and displayed in unit 4. The continuously running units are shown by the arrows in a loop in Figure 6 and are listed below:

- Python script in Rhino.
- External Python code.
- Grasshopper in Rhino.

According to the flowchart, four software programs in four different units are simultaneously running to simulate the proposed adaptive path planning method. The first unit is the camera component that simulates the functions of the top and side cameras. In this part, the Python script in Rhino is continuously run to capture images from the top and side views of the robot arm workspace. These images are updated constantly to ensure that any change in the locations of objects is identified in real time.

The second unit is the path planning component, which simulates the function of our proposed path planning method and is typically performed using a computer. This section includes the YOLO object detection, the coordinate-finding neural network for environment monitoring purposes, as well as D* Lite algorithm, and the angle-finding neural network for path generation purposes. Based on the images generated in the previous unit, Python code is used to identify the positions of the objects and generate a collision-free path for reaching the target. The updated path, which includes a sequence of nodes, is input to the angle-finding neural network to convert the nodes to the corresponding joint angles. Then, these joint angles of the updated path are saved as a comma-separated values (CSV) file. This file is continuously updated as the code constantly runs and repeats the process.

The third unit is the robot control component, which simulates the function of the robot arm control device. In this part, a Grasshopper block diagram uses the updated data from the CSV file, which contains the joint angles of the robot arm, and applies them to the robot in Rhino. Grasshopper is a visual programming language that runs within the Rhino software. Programs are created by dragging components onto a canvas. The outputs of these units are then connected to the inputs of subsequent pieces. Our Grasshopper block diagram consists of several parts containing blocks to perform specific tasks.

The fourth unit is the visualization component, where the motion of the robot arm is visualized. The Rhino software is used for displaying the simulated results of this part. Notably, each unit works independently, yet they are interlinked and use the results of other units.

V. ANALYSIS, SIMULATION, AND DISCUSSION

In this section, we present a real-time analysis of the proposed method. Thereafter, we describe its testing in the simulation environment and discuss the results. Using a simulation-based testbed helps us to perform robot arm tests in a dynamic site in a cost-effective, safe, and fast manner resulting in repeatable, controllable, measurable, and scalable effects.

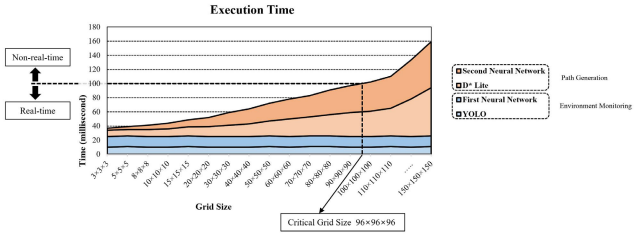

FIGURE 7. Total execution time for different grid sizes. Critical grid size for 100 ms reaction time is shown. Contribution of each section in the path planning method to the total time.

A. REAL-TIME ANALYSIS

We analyze the execution time of the flowchart's adaptive path planning unit, which is elaborately described in Section III, to determine if the method is real-time. Since the execution time is different for different grid sizes, the analysis should be conducted for different sizes. The total execution time is the summation of the execution times for the two sections, including environment monitoring and path generation. The environment monitoring section contains the YOLO object detection algorithm and the first coordinate-finding neural network. The path generation section contains the D* Lite algorithm for node finding and the second neural network for angle finding. Figure 7 shows the execution time required for the adaptive path planning by the different sections for different grid sizes.

Figure 7 also depicts the contribution of each section, including environment monitoring and path generation, to the total execution time. YOLO object detection, the first neural network for coordinate finding, D* Lite for node-finding, and the second neural network for angle-finding are represented by light blue, blue, light orange, and orange, respectively. For small grid sizes, YOLO and the first neural network have major contributions to total execution time. However, for large grid sizes, D* Lite and the second neural network majorly contribute to total execution time.

From Figure 7, we can also infer that the execution time of YOLO and the first neural network is independent of the grid size and is almost constant as the grid size increases. This is because YOLO first identifies objects from an image by dividing the image into regions. Then, it predicts bounding boxes and probabilities for each region. This implies that its operation is not based on grid cells. Moreover, the first neural network only converts the indexes of the obstacle and target cells to 3D coordinates, which is unrelated to grid cells.

In contrast, the execution time of D* Lite and the second neural network depends on the grid size. This is because the D* Lite algorithm identifies the nodes by searching for the closest collision-free cells in the path from the current cell to the target cell. Therefore, as the number of cells increases, the search process lengthens too. Furthermore, the second neural network converts the nodes to the corresponding joint angles. Thus, as the cells increase, the number of selected nodes from the starting cell to the target also increases. Consequently, the number of times the second neural network performs the

angle-finding process increases, resulting in an increase in the execution time.

It is also worth mentioning that since the structure of the first neural network is significantly more complex than that of the second neural network, the execution time of the first neural network is more than that of the second one in small grid sizes. Yet, as the number of cells increases, the execution time of the second neural network increases until it becomes more than that of the first because the first network is independent of the grid size, whereas the second network is dependent on it.

Overall, it is a general fact that as the number of cells increases, the amount of computing increases; therefore, the execution time increases too. Since some of the parts are dependent on the grid size, as the grid size increases, the total execution time increases as well. In other words, coarse motions (small grid size) need less execution time, and fine motions (large grid size) require more execution time.

Therefore, to achieve real-time path planning in our hybrid method, there is a tradeoff between the grid size and the path planning time. This implies that if the grid size increases beyond a critical size, the execution time (that is, the reaction time of the robot arm) will be too high for the process to be considered real-time path planning.

The reaction time of a robot arm varies for different applications, which implies that the definition of real-time performance is relative. In this regard, several workers may be in a dynamic industrial environment; therefore, we defined the reaction time based on human behavior. As human reaction time is approximately 180 ms [38], the robot should respond at least as fast or faster. Hence, if the execution time is less than 100 ms, we may successfully apply the algorithm for real-time response in a dynamic industrial environment where intelligent robot arms collaborate with humans. Consequently, for an execution time of 100 ms, the critical grid size is equal to $96 \times 96 \times 96$ cells. Thus, our proposed method for the industrial environment functions in real time for grid sizes up to $96 \times 96 \times 96$ cells, in which the total execution time does not exceed more than 100 ms.

B. SIMULATION RESULTS AND DISCUSSION

We design a complex scenario to evaluate the quickness of the real-time response of our adaptive path planning method. This scenario is based on changing the target and obstacle locations in the developed simulation environment to make the workspace highly dynamic. In this scenario, we are trying to test the target-reaching and obstacle-avoidance tasks in a very critical and complicated condition in terms of dynamics.

Therefore, to simulate target movements in the industrial environment, we consider that the location of the target changes randomly in a limited area. The curve created by random locations of the target is represented by green color in Figure 8. This scenario can help check the real-time target-reaching capability of the method in a dynamic industrial environment. By changing the target location, we can force the robot to update its path at each time step with respect to the target's new location.

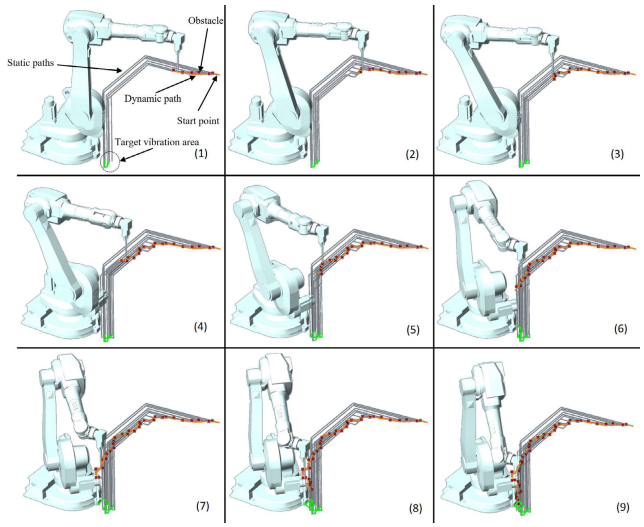

FIGURE 8. Verification of the path planning process in a complex scenario with dynamic target and obstacle locations. Orange: complete collision-free updated path (dynamic path); Green: target locations; Red dots: obstacle locations; Gray: updated path in each time step (static path).

The scenario is more complicated for the obstacles. To check the capability of real-time collision avoidance, we intentionally positioned the obstacle a few cells ahead of the current position of the end-effector in the current path. The red marks show the locations of the obstacle in each time step in Figure 8. In fact, owing to the intersection of the robot arm's current path with the obstacle, we force the robot to update its path at each time step. Otherwise, it will collide with the obstacle.

Given this scenario, the robot must update its path whenever a target or obstacle changes. The updated path is with respect to the new locations of the target and obstacle. The overall path is a combination of the paths generated in each update. This implies that there are two types of paths:

1. Static path (at each time step): The path at each time step from the end-effector to the current location of the target with respect to the current location of the obstacle. That is, if the locations of the target and obstacle were not changed, the robot would continue following this path.

2. Dynamic path (overall): A portion of each static path that the robot has passed so far. That is, a dynamic path is a collision-free path that the robot has passed at each time step to reach the target's new location at that time.

In Figure 8, the static and dynamic paths are shown in gray and orange curves, respectively. From subfigures (1) to (9), we can observe that the path is constantly updated to avoid collision with the obstacle that is intentionally placed in the way, as well as to reach the target whose location changes randomly. Thus, from the early stage of tracking (subfigure (1)) to the end of the process (subfigure (9)), the robot updates its path with respect to the new positions of the target and obstacle until it reaches the final destination.

TABLE 3. Comparative analysis of results.

Author	Path Planning Algorithm	Execution Time: t (millisecond)	Including the time of	
			Environment Monitoring	Path Generation
Our method	Hybrid	38 < t	Yes	Yes
Kunz [22]	DRM	75 < t < 100	Yes	Yes
Leven [39]	PRM	tens of ms	×	Yes
Silva [40]	A*	min < t < 40	×	Yes
Schumann-Olsen [41]	Parallel DRM	10 < t < 20	×	Yes

To make the simulation results more understandable and the paths more visible, we have removed the sphere (obstacle) and box (target) in Figure 8.

In order to evaluate our method, we compare it with other real-time path planning studies. Table 3 provides a comprehensive comparison, including algorithm types, execution times, and reported times.

The total execution time including the time of environment monitoring, as well as path generation for our method, begins from 38 milliseconds in small grid sizes and has an open-end maximum time for large grid sizes. For some of the mentioned methods, the reported execution time was only for path generation. That is, their total execution time for whole process, which includes the time for environment monitoring and path generation, is more than what is reported.

C. FUTURE WORK

Besides the advantages of our presented adaptive path planning method, this study exhibits certain limitations, which should be considered for future work. Some of the restrictions are listed below.

- As mentioned in Section II-A, obtaining depth information using simple cameras is impossible when an object is hidden behind another in the 3D workspace. The solution is to use two depth or stereo cameras for two perpendicular views to guarantee at least one image that can detect the depth. Also, in case that the objects cannot be detected by any cameras, other methods like thermal image-based method can be used [42].
- After testing in the simulation environment, the method must be tested in an experimental setup. However, to avoid a lengthy paper, our next paper will present the test results of an experimental setup.
- The algorithm is suitable for a single obstacle; however, a robot arm may simultaneously encounter multiple obstacles in real industrial environments. Thus, the next study should focus on developing a multi-obstacle path planning method.

VI. CONCLUSION

An adaptive path planning method was developed for intelligent robot arms, which typically work in dynamic environments. In the proposed method, the locations of the target and obstacle were detected through a combined computer vision approach for environment monitoring purposes. The YOLO algorithm was used to detect objects, and a neural network converted the output of YOLO to 3D coordinates. Then, the D* Lite algorithm was used to identify a sequence of collision-free nodes in a 3D gridded workspace forming a path from the current cell of the end-effector to the target cell (a so-called active process of path generation). Finally, another neural network was used to convert the path nodes to joint angles (a so-called passive process of path generation). A simulation environment framework was designed to test the method in a dynamic workspace. In this environment, an experimental test was conducted based on a complex scenario to evaluate the speed of the target-reaching and obstacle-avoidance tasks. The results demonstrated the quick reaction of the robot arm in path planning. Real-time analysis was also performed to determine the critical grid size; as far as the grid size is less than the critical grid size, the method was considered to function in real time. The analysis showed that the critical grid size was 96 × 96×96 cells for a 100 ms reaction time.

ACKNOWLEDGMENT

The authors would like to express their gratitude to their colleague, Saeji Park, for helping them to design the diagrams, which provided a better overview of their method.

REFERENCES

[1] X. Xia, T. Li, S. Sang, Y. Cheng, H. Ma, Q. Zhang, and K. Yang, "Path planning for obstacle avoidance of robot arm based on improved potential field method," *Sensors*, vol. 23, no. 7, p. 3754, Apr. 2023, doi: 10.3390/s23073754.

[2] S.-O. Park, M. C. Lee, and J. Kim, "Trajectory planning with collision avoidance for redundant robots using Jacobian and artificial potential field-based real-time inverse kinematics," *Int. J. Control, Autom. Syst.*, vol. 18, no. 8, pp. 2095–2107, Aug. 2020, doi: 10.1007/s12555-019-0076-7.

[3] S. N. Gai, R. Sun, S. J. Chen, and S. Ji, "6-DOF robotic obstacle avoidance path planning based on artificial potential field method," in *Proc. 16th Int. Conf. Ubiquitous Robots (UR)*, Jeju, South Korea, Jun. 2019, pp. 165–168, doi: 10.1109/URAI.2019.8768792.

[4] H.-I. Lin and M.-F. Hsieh, "Robotic arm path planning based on three-dimensional artificial potential field," in *Proc. 18th Int. Conf. Control, Autom. Syst. (ICCAS)*, PyeongChang, South Korea, Oct. 2018, pp. 740–745.

[5] Z. Iqbal, J. Reis, and G. Gonçalves, "Path planning for an industrial robotic arm," in *Proc. 8th Int. Conf. Intell. Syst. Appl. (INTELLI)*, Rome, Italy, Jul. 2019, p. 39.

[6] M. R. B. Bahar, H. B. Bahar, and F. Hashemzadeh, "Grid roadmap based real time path planning," in *Proc. 17th Int. Conf. Autom. Comput.*, Sep. 2011, pp. 75–79.

[7] M. R. B. Bahar, A. R. Ghiasi, and H. B. Bahar, "Grid roadmap based ANN corridor search for collision free, path planning," *Scientia Iranica*, vol. 19, no. 6, pp. 1850–1855, Dec. 2012, doi: 10.1016/j.scient.2012.02.028.

[8] K. Wei and B. Ren, "A method on dynamic path planning for robotic manipulator autonomous obstacle avoidance based on an improved RRT algorithm," *Sensors*, vol. 18, no. 2, p. 571, Feb. 2018, doi: 10.3390/s18020571.

[9] Y. Liu and G. Zuo, "Improved RRT path planning algorithm for humanoid robotic arm," in *Proc. Chin. Control Decis. Conf. (CCDC)*, Hefei, China, Aug. 2020, pp. 397–402, doi: 10.1109/CCDC49329.2020.9164659.

[10] T.-C. Lai, S.-R. Xiao, H. Aoyama, and C.-C. Wong, "Path planning and obstacle avoidance approaches for robot arm," in *Proc. 56th Annu. Conf. Soc. Instrum. Control Eng. Jpn. (SICE)*, Kanazawa, Japan, Sep. 2017, pp. 334–337, doi: 10.23919/SICE.2017.8105619.

[11] T. B. Ionescu, "Adaptive simplex architecture for safe, real-time robot path planning," *Sensors*, vol. 21, no. 8, p. 2589, Apr. 2021, doi: 10.3390/s21082589.
[12] A. T. Khan, X. Cao, Z. Li, and S. Li, "Evolutionary computation based real-time robot arm path-planning using beetle antennae search," *EAI Endorsed Trans. AI Robot.*, vol. 1, pp. 1–10, Jan. 2022, doi: 10.4108/airo.v1i.6.
[13] S. Wen, J. Chen, S. Wang, H. Zhang, and X. Hu, "Path planning of humanoid arm based on deep deterministic policy gradient," in *Proc. IEEE Int. Conf. Robot. Biomimetics (ROBIO)*, Kuala Lumpur, Malaysia, Dec. 2018, pp. 1755–1760, doi: 10.1109/ROBIO.2018.8665248.
[14] E. Prianto, M. Kim, J.-H. Park, J.-H. Bae, and J.-S. Kim, "Path planning for multi-arm manipulators using deep reinforcement learning: Soft actor–critic with hindsight experience replay," *Sensors*, vol. 20, no. 20, p. 5911, Oct. 2020, doi: 10.3390/s20205911.
[15] F. Zhang, J. Leitner, M. Milford, and P. Corke, "Sim-to-real transfer of visuo-motor policies for reaching in clutter: Domain randomization and adaptation with modular networks," *World*, vol. 7, no. 8, pp. 1–7, 2017.
[16] M. Ji, L. Zhang, and S. Wang, "A path planning approach based on Q-learning for robot arm," in *Proc. 3rd Int. Conf. Robot. Autom. Sci. (ICRAS)*, Wuhan, China, Jun. 2019, pp. 15–19, doi: 10.1109/ICRAS.2019.8809005.
[17] Q. Yuan, J. Yi, R. Sun, and H. Bai, "Path planning of a mechanical arm based on an improved artificial potential field and a rapid expansion random tree hybrid algorithm," *Algorithms*, vol. 14, no. 11, p. 321, Nov. 2021, doi: 10.3390/a14110321.
[18] A. Abdi, D. Adhikari, and J. H. Park, "A novel hybrid path planning method based on Q-learning and neural network for robot arm," *Appl. Sci.*, vol. 11, no. 15, p. 6770, Jul. 2021, doi: 10.3390/app11156770.
[19] A. Abdi, M. H. Ranjbar, and J. H. Park, "Computer vision-based path planning for robot arms in three-dimensional workspaces using Q-learning and neural networks," *Sensors*, vol. 22, no. 5, p. 1697, Feb. 2022, doi: 10.3390/s22051697.
[20] L. Luo, H. Wen, Q. Lu, H. Huang, W. Chen, X. Zou, and C. Wang, "Collision-free path-planning for six-DOF serial harvesting robot based on energy optimal and artificial potential field," *Complexity*, vol. 2018, pp. 1–12, Nov. 2018, doi: 10.1155/2018/3563846.
[21] Z. Chen, L. Ma, and Z. Shao, "Path planning for obstacle avoidance of manipulators based on improved artificial potential field," in *Proc. Chin. Autom. Congr. (CAC)*, Hangzhou, China, Nov. 2019, pp. 2991–2996, doi: 10.1109/CAC48633.2019.8996467.
[22] T. Kunz, U. Reiser, M. Stilman, and A. Verl, "Real-time path planning for a robot arm in changing environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Taipei, Taiwan, Oct. 2010, pp. 5906–5911, doi: 10.1109/IROS.2010.5653275.
[23] Z. W. Lim, D. Hsu, and W. S. Lee, "Adaptive informative path planning in metric spaces," *Int. J. Robot. Res.*, vol. 35, no. 5, pp. 585–598, Apr. 2016, doi: 10.1177/0278364915596378.
[24] G. Wu and S. Zhang, "Real-time jerk-minimization trajectory planning of robotic arm based on polynomial curve optimization," *Proc. Inst. Mech. Eng., C, J. Mech. Eng. Sci.*, vol. 236, no. 21, pp. 10852–10864, Nov. 2022.
[25] F. S. Hameed, H. M. Alwan, and Q. A. Ateia, "Obstacle avoidance method for highly redundant robotic arms," *IOP Conf. Ser., Mater. Sci. Eng.*, vol. 765, no. 1, Mar. 2020, Art. no. 012017, doi: 10.1088/1757-899x/765/1/012017.
[26] T. Li, Q. Li, W. Li, J. Xia, W. Tang, and W. Wang, "A path planning algorithm for space manipulator based on Q-learning," in *Proc. IEEE 8th Joint Int. Inf. Technol. Artif. Intell. Conf. (ITAIC)*, Chongqing, China, May 2019, pp. 1566–1571, doi: 10.1109/ITAIC.2019.8785427.
[27] X. Wang, Z. Xie, X. Zhou, J. Gao, F. Li, and X. Gu, "Adaptive path planning for the gantry welding robot system," *J. Manuf. Processes*, vol. 81, pp. 386–395, Sep. 2022, doi: 10.1016/j.jmapro.2022.07.005.
[28] S. Aoughellanet, T. Mohammedi, and Y. Bouterfa, "Neural network path planning applied to PUMA 560 robot arm," *WSEAS Trans. Syst.*, vol. 4, no. 4, pp. 446–450, 2005.
[29] K. Kamali, I. A. Bonev, and C. Desrosiers, "Real-time motion planning for robotic teleoperation using dynamic-goal deep reinforcement learning," in *Proc. 17th Conf. Comput. Robot Vis. (CRV)*, Ottawa, ON, Canada, May 2020, pp. 182–189, doi: 10.1109/CRV50864.2020.00032.
[30] L. Wang, Y. Qi, W. Li, M. Liu, and Z. Zhang, "Dynamic parallel mapping and trajectory planning of robot arm in unknown environment," *IEEE Sensors J.*, vol. 23, no. 10, pp. 10970–10982, May 2023, doi: 10.1109/JSEN.2022.3232088.
[31] H. Berggren and F. Melvås, "Real-time tracking of human motions and adaptive robot path planning for assembly cooperation," Tech. Rep., 2018.
[32] D. Ding, Z. Pan, D. Cuiuri, H. Li, and N. Larkin, "Adaptive path planning for wire-feed additive manufacturing using medial axis transformation," *J. Cleaner Prod.*, vol. 133, pp. 942–952, Oct. 2016, doi: 10.1016/j.jclepro.2016.06.036.
[33] H. Li, H. Wu, L. Lou, K. Kühnlenz, and O. Ravn, "Ping-pong robotics with high-speed vision system," in *Proc. 12th Int. Conf. Control Autom. Robot. Vis. (ICARCV)*, Dec. 2012, pp. 106–111, doi: 10.1109/ICARCV.2012.6485142.
[34] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 779–788.
[35] W. Vann, T. Zhou, Q. Zhu, and E. Du, "Enabling automated facility maintenance from articulated robot collision-free designs," *Adv. Eng. Informat.*, vol. 55, Jan. 2023, Art. no. 101820, doi: 10.1016/j.aei.2022.101820.
[36] S. Koenig and M. Likhachev, "D*lite," in *Proc. AAAI*, vol. 15, 2002, pp. 476–483.
[37] K. C. Kyu, N. H. Hung, K. D. Hwan, and H. Kyeong, "Path planning for automatic guided vehicle with multiple target points in dynamic environment," in *Proc. MATEC Web Conf.*, 2018, p. 2029, doi: 10.1051/matecconf/201815902029.
[38] J. M. Brebner, "Introduction: An historical background sketch," in *Reaction times*, A. T. Welford, Ed. London, U.K.: Academic, 1980, pp. 1–23.
[39] P. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments," *Int. J. Robot. Res.*, vol. 21, no. 12, pp. 999–1030, Dec. 2002, doi: 10.1177/0278364902021012001.
[40] J. S. Silva, P. Costa, and J. Lima, "Manipulator path planning for pick-and-place operations with obstacles avoidance: An A algorithm approach," in *Robotics in Smart Manufacturing*, 2013, pp. 223–234, doi: 10.1007/978-3-642-39223-8_20.
[41] H. Schumann-Olsen, M. Bakken, Ø. H. Holhjem, and P. Risholm, "Parallel dynamic roadmaps for real-time motion planning in complex dynamic scenes," in *Proc. 3rd Workshop Robots Clutter*, 2014, pp. 1–12.
[42] H. Wu, H. Li, H.-L. Chi, Z. Peng, S. Chang, and Y. Wu, "Thermal image-based hand gesture recognition for worker-robot collaboration in the construction industry: A feasible study," *Adv. Eng. Informat.*, vol. 56, Apr. 2023, Art. no. 101939, doi: 10.1016/j.aei.2023.101939.

ALI ABDI received the B.Sc. and first M.Sc. degrees in mechanical engineering, specializing in robotic control from the University of Tehran, Tehran, Iran, in 2016 and 2019, respectively. He is currently pursuing the second M.Sc. degree in artificial intelligence with the Pohang University of Science and Technology (POSTECH), Pohang, South Korea. His research interests include artificial intelligence-powered robots (AI), computer vision (CV), human–robot interaction (HRI), robotic control, virtual/augmented reality (VR/AR), and natural language processing (NLP).

JU HONG PARK (Member, IEEE) received the B.Eng. degree in electronic and electrical engineering and architecture from Hongik University, the M.Arch. degree from Harvard University, and the Ph.D. degree in architecture (major in design and computation and minor in education) from the Harvard Graduate School of Education, MIT. He is the Director of the Design Intelligence Laboratory, Meta Maker Space, POSTECH, and the Food Tech RnD Center. Before being appointed as an Assistant Professor with POSTECH, he was an Assistant Professor and the Coordinator of the M.S. Arch in Computation and Embedded Technology Program, School of Architecture, University of Miami. He was also a Researcher with the MIT School of Architecture and the Media Laboratory and an Architect with the Office for Metropolitan Architecture (OMA) and the Coop Himmelb(l)au. During the M.Arch. degree, he was nominated for the prestigious James Templeton Kelly Thesis Prize.