technologies

MDPI

Article

Intelligent IoT-Based Network Clustering and Camera Distribution Algorithm Using Reinforcement Learning

Islam T. Almalkawi [1,*], Rami Halloush [2], Mohammad F. Al-Hammouri [1], Alaa Alghazo [3], Loiy Al-Abed [1], Mohammad Amra [1], Ayooub Alsarhan [4] and Sami Aziz Alshammari [5,*]

1 Department of Computer Engineering, Faculty of Engineering, The Hashemite University, Zarqa 13133, Jordan; alhammouri@hu.edu.jo (M.F.A.-H.); luaiehsan@outlook.com (L.A.-A.); mohammadamra00@gmail.com (M.A.)
2 College of Engineering and Technology, American University of the Middle East, Egaila 54200, Kuwait; rami.halloush@aum.edu.kw
3 Department of Mechatronics Engineering, Faculty of Engineering, The Hashemite University, Zarqa 13133, Jordan; alghazo@hu.edu.jo
4 Department of Information Technology, Faculty of Prince Al-Hussein Bin Abdallah II for Information Technology, The Hashemite University, Zarqa 13133, Jordan; ayoubm@hu.edu.jo
5 Department of Information Technology, Faculty of Computing and Information Technology, Northern Border University, Rafha 91431, Saudi Arabia
* Correspondence: eslam.malkawi@hu.edu.jo (I.T.A.); sami.alshammari@nbu.edu.sa (S.A.A.)

Abstract: The advent of a wide variety of affordable communication devices and cameras has enabled IoT systems to provide effective solutions for a wide range of civil and military applications. One of the potential applications is a surveillance system in which several cameras collaborate to monitor a specific area. However, existing surveillance systems are often based on traditional camera distribution and come with additional communication costs and redundancy in the detection range. Thus, we propose a smart and efficient camera distribution system based on machine learning using two Reinforcement Learning (RL) methods: Q-Learning and neural networks. Our proposed approach initially uses a geometric distributed network clustering algorithm that optimizes camera placement based on the camera Field of View (FoV). Then, to improve the camera distribution system, we integrate it with an RL technique, the role of which is to dynamically adjust the previous/existing setup to maximize target coverage while minimizing the number of cameras. The reinforcement agent modifies system parameters—such as the overlap distance between adjacent cameras, the camera FoV, and the number of deployed cameras—based on changing traffic distribution and conditions in the surveilled area. Simulation results confirm that the proposed camera distribution algorithm outperforms the existing methods when comparing the required number of cameras, network coverage percentage, and traffic coverage.

Keywords: Internet of things (IoT); surveillance systems; camera field of view (FoV); machine learning; reinforcement learning; Q-learning; deep neural Q-network (DQN)

Received: 18 November 2024
Revised: 10 December 2024
Accepted: 12 December 2024
Published: 24 December 2024

Citation: Almalkawi, I.T.; Halloush, R.; Al-Hammouri, M.F.; Alghazo, A.; Al-Abed, L.; Amra, M.; Alsarhan, A.; Alshammari, S.A. Intelligent IoT-Based Network Clustering and Camera Distribution Algorithm Using Reinforcement Learning. *Technologies* **2024**, *13*, 4. https://doi.org/10.3390/technologies13010004

1. Introduction

Advances in camera sensing technology have facilitated seamless integration with a variety of wireless devices. This integration enables the development of smart camera surveillance systems, a prominent application of the Internet of Things (IoT) paradigm [1]. The increasing demand for intelligent surveillance solutions in both civilian and military contexts has driven the deployment of camera sensors capable of real-time detection, recognition, and tracking of target objects within their Field of View (FoV) [2]. To optimize

energy consumption, cameras are strategically deployed with overlapping FoVs. Abnormal objects are selectively identified and tracked, while normal objects are disregarded. Communication between overlapping cameras or within the same location is established to ensure continuous tracking of objects as they transition between FoVs [3,4].

For large-scale surveillance networks, the effective integration of multiple camera Field of Views (FoVs) and precise sensor calibration are crucial for optimal system performance. In such systems, cameras must collaborate and communicate with neighboring cameras to leverage overlapping coverage for target recognition and tracking. However, excessive camera overlap in dense networks can increase costs, energy consumption, and redundant sensing, processing, and communication overhead. On the other hand, sparse camera deployment reduces network coverage percentage and may miss target detection and tracking. Therefore, maximizing target coverage with minimal deployed cameras is essential for efficient and cost-effective surveillance systems.

AI and machine learning techniques offer significant potential to revolutionize smart surveillance systems. By leveraging advanced algorithms, these technologies can optimize camera placement, dynamically adjust camera parameters, and intelligently analyze video streams to detect anomalies, track objects of interest, and improve overall system efficiency.

Many existing solutions for camera distribution rely on traditional approaches that suffer from limitations, such as the deployment of a sparse number of high-resolution, high-power cameras, the use of complex and computationally intensive algorithms for dynamic camera movement and orientation, or the installation of redundant simple cameras to achieve adequate coverage. To overcome these limitations, we propose an intelligent reinforcement learning-based approach for camera distribution. Our approach optimizes camera placement to minimize redundancy while ensuring maximum coverage, leading to significant cost savings and energy efficiency. Additionally, our RL-based solution addresses the computational challenges associated with traditional optimization techniques, enabling efficient real-time operations once the optimal policy is learned. Furthermore, we will provide specific examples and quantitative analysis to illustrate the advantages of our proposed method over existing techniques. For instance, we will compare the performance of our RL-based approach with traditional grid-based and/or random placement strategies in terms of the required number of cameras, network coverage percentage, and traffic coverage.

Building upon our previous work [3], which introduced an efficient camera distribution strategy based on triangle geometry, this paper further enhances the system performance through the integration of machine learning techniques to refine the initial camera placement. We apply reinforcement learning algorithms, specifically Q-Learning [5] and Deep Neural Q-Network (DQN) [6], to optimize camera distribution and target coverage in smart surveillance systems. Our new algorithm aims to smartly distribute the cameras in the monitored area where the network and target coverage are maximized with the least number of nodes. We deploy reinforcement learning to dynamically adjust the overlapping distance, the FoV of the cameras, and camera positions based on the target traffic in the surveillance area. As a result, our proposed smart camera distribution system can be universally applicable to various surveillance applications, including industrial operations, traffic monitoring, and public safety, among others.

In summary, this paper makes the following major contributions:

- A deployment strategy involving overlapping camera views on a generated map of the target area facilitates object detection and tracking.
- An adaptive cooperation among the adjacent cameras exploiting their overlapping FoVs to achieve maximum coverage.
- A smart camera distribution approach based on Q-Learning RL traffic conditions.

- A further improvement on intelligent camera distribution approach using DQN RL technique.
- A new reward calculation formula that helps the reinforcement agent reach an optimal policy.

This paper is structured as follows: The required background information is demonstrated in Section 2. Section 3 provides an overview of previous studies on distributing surveillance cameras. In Section 4, we introduce our smart camera distribution algorithm. Section 5 presents our findings and analysis and compares our algorithm to previous approaches. Finally, Section 6 summarizes the research work and the paper's conclusions.

2. Background

This section aims to familiarize the reader with basic concepts before delving into our research specifics. We begin by defining the Field of View (FoV) of a camera, and we focus on defining important terminologies associated with reinforcement learning and the epsilon-greedy strategy [7].

Camera FoV stands for "Field of View" and measures the extent of the observable scene that a camera can capture or see. FoV is typically expressed in degrees and represents the angle between the two outermost rays of the camera's field of vision. An illustration of FoV with different degrees can be observed in Figure 1. The FoV of cameras directly impacts the overall coverage and effectiveness of a surveillance system. A well-designed camera placement strategy, informed by an intelligent technique, can optimize the FoV to maximize coverage while minimizing overlaps and redundancy.

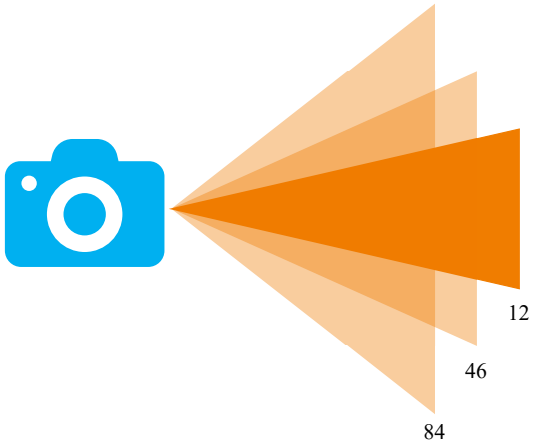

Figure 1. Illustration of camera FoV.

Reinforcement learning (RL) is a machine learning technique where the RL agent learns how to make decisions in an environment by interacting with it and receiving feedback in the form of rewards or penalties. RL aims to learn a policy that maximizes the cumulative reward over time. The agent uses trial and error to learn the optimal actions in different situations and gradually improves its performance through experience as illustrated in Figure 2. RL has been applied to a wide range of applications, including robotics, game-playing, and autonomous driving. RL-based techniques utilize the dynamic nature of surveillance environments and the need for adaptive camera placement strategies. RL's ability to learn optimal policies through interaction with the environment makes it a suitable approach for this task. In this paper, we show how RL can be used to dynamically adjust camera parameters such as positions and FoV in response to changing traffic conditions.

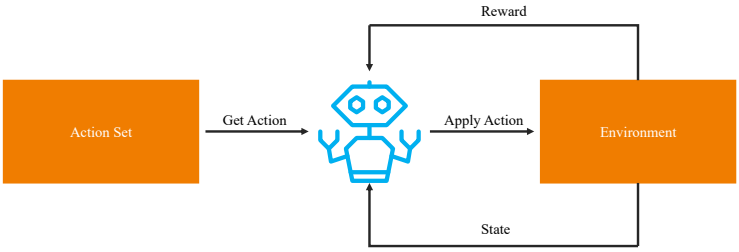

Figure 2. Workflow of reinforcement learning.

The epsilon-greedy strategy is a simple exploration-exploitation strategy commonly used in RL. In this strategy, the agent chooses a random action with a probability of ϵ (a small value), otherwise, it chooses the action with the highest estimated reward. The purpose of this strategy is to balance the exploration of new actions (exploration) with exploiting the knowledge the agent has gained so far (exploitation) to maximize the cumulative reward. The epsilon-greedy strategy is widely used in many RL applications.

Both Q-Learning (Q-table) and Deep Neural Q-Network (DQN) are popular algorithms for solving reinforcement learning problems, but they employ distinct approaches to representing and learning policies. Q-Learning utilizes a tabular method, where state–action pairs are explicitly stored and updated. In contrast, DQN leverages neural networks to approximate the Q-value function, enabling it to handle larger and continuous state–action spaces more effectively.

Optimal camera placement can be formulated as a reinforcement learning problem. The agent, or system, is tasked with determining the most strategic camera positions within a specified environment to achieve maximum coverage. The environment is divided into sections that require monitoring, and the reward function is designed to evaluate the effectiveness of camera coverage in these sections.

3. Related Work

This section presents a comprehensive review of the state-of-the-art research on applying various machine learning techniques to optimize camera distribution and related applications. A summary and comparison of these related works are provided in Table 1.

Table 1. Comparison of Related Works.

Reference	Used Method	Advantages	Limitations
[8]	QMIX, MADDPG, IPPO, MAPPO, TarMAC, and I2C.	Modularity, flexibility, and scalability.	The performance of the algorithms and models developed in this environment may not necessarily translate to real-world applications.
[9]	Reinforcement learning.	Prioritizes maximizing longer-term visibility while simultaneously navigating towards the goal camera poses.	The surgical environment presents unique challenges, such as deforming, dynamic, and highly reflective surfaces, and highly constrained operating physical space, which may limit the effectiveness of the proposed method.
[10]	Deep reinforcement learning.	Handles visibility constraints by keeping the feature away from the dangerous area, which is not guaranteed by other methods such as the Kalman filter-based method and the potential field-based method.	Considering a method to handle the feature loss when switching to the obstacles avoidance controller suggests that there may be limitations in the current approach when dealing with obstacles and feature loss.
[11]	Reinforcement learning	The total error in all test scenes is also less than 90% of the baseline ones.	Only guarantee the completeness and accuracy of the results within a few meters.
[12]	Deployment of acoustic and visual sensor nodes.	Energy efficient surveillance.	Does not consider the effect of real-world factors.
[13]	Occupancy distribution estimation.	Improved energy efficiency and evacuation.	Requires processing of camera videos.
[14]	Reinforcement learning.	Maximizes area coverage while considering UAV Limitations.	The score used to evaluate patrolling performance is limited by the environment's size.
[15]	Distributed initial orientation algorithm.	Addresses directional sensor network coverage.	May not be computationally efficient.
[16]	Greedy approach and reinforcement learning.	Overcomes limitations of fixed networks in public surveillance.	The experiments are limited to a fixed number of pedestrians and movement patterns.
[17]	Greedy heuristic approach.	Achieves near-optimal but not efficient solutions.	Similar overhead and reaction time to distributed solutions.
[18]	Network Decomposition.	Results in coverage quality close to the centralized optimal solution.	Overhead and reaction time similar to distributed solution.
[19]	Overlapped region detection clustering.	Reduces network communication load and energy consumption.	Relatively low reduction rates.
[20]	Long-range image sensor.	Cost-effective and can be deployed in real-world scenarios.	Requires Semtech LoRa technology.

Table 1. *Cont.*

Reference	Used Method	Advantages	Limitations
[21]	Modified greedy algorithm.	Reduces redundancy and computational complexity.	Requires the deployment of a significant number of sensors.
[22]	Dynamic programming algorithm.	Models camera placement problem.	Assumes a fixed FoV and does not consider dynamic changes in the monitored area.
[23]	Particle Swarm Optimiser.	Presents multiple contributions and analyses.	Self-organized approach may not be suitable for immediate action scenarios.
[24]	(MCLP-CC) model.	Generates optimal camera configurations with lower occlusion and overlapping.	Line-of-sight algorithm may produce different optimization results.
[25]	Shortest-path algorithm.	Accurately estimates 3D observations from multiple camera views.	Tracking occluded objects may be challenging.
[26]	Combinatorial optimization.	Addresses the lack of research on OCP.	Low density of problem instances used may have influenced the results.
[27]	Reinforcement learning.	Optimizes energy efficiency and coverage for multi-UAV navigation.	Limited focus on a particular situation of multi-UAV navigation.
[28]	Centralized heuristics.	Surpasses pre-existing sensor-oriented heuristics.	Assumes all active sensors will monitor targets until their batteries are entirely depleted.

Guzman proposed an RL-based distributed control scheme for cooperative intersection traffic control. The proposed approach combines real-time data from traffic detectors at the intersection with past and future traffic information from the ego and neighboring intersections, encoded by a GNN traffic prediction model [29].

Awaisi discussed the use of deep learning and image processing techniques for efficient car parking management. The paper proposes a fog-enabled architecture for car parking that uses deep neural networks (DNNs) and convolutional neural networks (CNNs) for vacant parking slot detection [30].

Yuanbin Wang addressed the problem of surface defect detection on complex products using machine vision, specifically deep learning methods. The authors highlight that capturing high-quality images is crucial for accurate defect detection, but it becomes challenging for complex products due to issues such as occlusion, illumination, and other factors [31].

Xuehai Pan [8] presented an overview of baseline algorithms, including QMIX, MADDPG, IPPO, MAPPO, TarMAC, and I2C, and discussed the heterogeneity, asymmetry, variety, flexibility, and scalability of the MATE environment.

Yun-Hsuan Su [9] proposed a method that involves using a combination of a context-aware method and deep reinforcement learning approach for automatic multi-camera positioning and camera motion and exploration in between viewpoints. The deep reinforcement learning approach governs camera motion and exploration in between viewpoints, intending to maximize longer-term visibility and simultaneously navigate towards the goal camera poses.

Zhehao Jin [10], proposed an approach using policy-based deep reinforcement learning (DRL) for image-based visual servoing (IBVS) control of mobile robots with visibility constraints. It uses a DRL algorithm to design an adaptive law for tuning the controller gain in the continuous space, which can maintain the feature in the Field of View of the camera, as well as improve the servo efficiency.

Yichuan Chen [11] proposed an online solution to the optimal camera placement (OCP) problem for depth observation of indoor scenes based on reinforcement learning. The proposed system comprises a simulation environment that implements scene observation and reward estimation using shadow maps and an agent network containing a soft actor–critic (SAC)-based reinforcement learning backbone and a feature extractor to extract features from the observed point cloud layer-by-layer.

Yanjie Ze [32] proposed a method for visual RL using self-supervised 3D representations. The authors propose a joint training approach that combines reinforcement learning with a 3D reconstruction task to learn a robust visual representation for robotic manipulation tasks.

Qiaofeng Ou [33] proposed a camera calibration method based on RL for large FoV cameras. The proposed method starts by establishing a Markov decision process (MDP) model of the calibration procedure. A reward function is then designed, considering both calibration accuracy requirements and state-space constraints. The method involves continuous interaction with the calibration environment using Q-Learning, which guides the camera calibration process.

The work in [12] proposed an energy-efficient mechanism for environment surveillance in Wireless Multimedia Sensor Networks (WMSNs) [34] using a mixed random deployment of acoustic and visual sensors. However, their simulation has limitations, including the absence of real-world factors like environmental interference, hardware malfunctions, and unexpected events.

Zhang proposed using surveillance cameras to estimate the occupancy distribution in buildings [13], which can improve energy efficiency and evacuation planning. The method

records the number of passing occupants at a region's entrance by processing camera videos, eliminating the need for additional sensors.

An optimal patrolling strategy for UAVs with a camera sensor was proposed in [14], using a relevance map to represent coverage requirements. The system optimizes important areas while considering UAV limitations, using a deep learning network trained with RL to choose actions that maximize a reward function expressed in terms of area coverage.

Esmaeilzadeh and Abbaspour, in [15], addressed directional sensor network coverage, proposing two solutions for the temporal coverage problem: an ILP optimization approach and a local-information-based heuristic.

Bisagno proposed smart camera networks to overcome the limitations of fixed networks in public surveillance by using a decentralized approach for network reconfiguration using PTZ and UAV-based cameras [16]. Two policies for camera reconfiguration are presented and evaluated in a simulated environment.

Bilal Gonen [17] proposed an optimization problem to maximize camera coverage in smart camera networks, using a computationally efficient heuristic and a spatially decomposed algorithm. The approach achieves near-optimal but not efficient solutions with similar overhead and reaction time to distributed solutions.

Vikram P. Munishwar and Nael B. Abu-Ghazaleh [18] proposed an algorithm that spatially decomposes the network and computes optimal solutions for individual partitions. The approach results in coverage quality close to the centralized optimal solution, with an overhead and reaction time similar to those of distributed solutions.

Kheirkhah and Khansari proposed a clustering method and camera activation algorithm to conserve energy and maximize system lifetime for WMSNs [19]. Their methods reduce network communication load and energy consumption up to 75% and 66.6%, respectively, which are relatively low.

Pham discussed [20] a low-cost, long-range image sensor that enables remote visual surveillance using off-the-shelf components and packet loss-tolerant image compression. The sensor uses Semtech LoRa technology and can be deployed in real-world scenarios using battery-operated image nodes.

Raqeebir Rab proposed a modified greedy algorithm to solve the "Maximum Coverage with Minimum Sensors" problem in visual sensor networks [21]. The algorithm could reduce redundancy and computational complexity.

A. Altahir [22] covered two approaches to model the camera placement problem and proposed a dynamic programming algorithm to solve the visual sensor coverage problem. However, it has limitations, such as assuming a fixed FoV and not considering dynamic changes in the monitored area.

Esterle [23] explored maximizing coverage in PTZ camera networks with varying levels of information and cooperation. It presents three contributions, including a comparison of ARES and PSO, an analysis of distributed versions of PSO and ARES, and a self-organized approach. The study discusses the results and limitations of the self-organized approach, which may not be suitable for immediate action scenarios.

Zhigang Han [24] discussed optimizing surveillance cameras in public spaces using location-based data. The proposed method uses building footprints, POI, and social network data to infer camera coverage and optimize placement. The experiment shows the method generates optimal camera configurations with lower occlusion and overlapping. However, the use of the line-of-sight algorithm to find the view shed may produce different optimization results due to the size of the grid cells.

Xiaoayan Jiang [25] solved multi-object tracking in visual sensor networks for IoT by proposing a graph-based data fusion algorithm that uses 2D and 3D features to estimate accurate 3D observations from multiple camera views. Particle filters are used for tracklet

extraction and the approach is evaluated thoroughly. However, tracking objects that remain occluded or out of view for prolonged periods may be challenging due to the reliance on temporally connected 3D observations from the greedy data association algorithm for particle filter initialization and updates.

Julien Kritter [26] proposed a framework for generating real-world instances of the OCP problem and evaluated several algorithms using numerical benchmarks. The authors aim to address the lack of research on OCP by using findings from the set cover problem literature. They use OpenStreetMap and NASA's SRTM data to sample the surveillance area and simplify computations in a two-step conversion process. However, the study notes that the low density of the problem instances used may have influenced the results, and the performance of the algorithms may differ significantly for instances with different densities.

Chi Harold Liu [27] proposed a distributed DRL algorithm for multi-UAV navigation to optimize energy efficiency and coverage. However, the experiment only focused on a particular situation of multi-UAV navigation for extending communication coverage over an extended period. As a result, the outcomes may not be transferable to other circumstances, necessitating additional trials to evaluate the model in varying environments.

Hafsa Zannat [28] presented a target-focused method and put forward three heuristics for achieving target coverage in smart camera networks that surpass pre-existing sensor-oriented heuristics. The paper lays out the problem and includes a discussion of the incentive, system model, and a formal definition. It is worth noting, however, that the study assumes that all active sensors will monitor targets until their batteries are entirely depleted, which may not be feasible in real-world circumstances.

Table 1 presents a comprehensive comparison of related works in terms of their proposed solutions, advantages, and main limitations.

4. The Proposed Camera Distribution Approach

This section is structured into six parts, each providing a comprehensive insight into the proposed solution. The first part presents robot mapping using Turtlebot2 [35], explaining how it is employed to perform initial area mapping before camera distribution. Subsequently, the second part summarizes our proposed initial camera distribution based on the triangle geometry approach [3]. In the third part, we examine the use of motion sensors and radiation patterns, explaining how they complement the proposed solution and contribute to its overall functionality. In the fourth part, we take a closer look at how our solution enhances detection and tracking, and the potential impact this could have on a wide range of applications and industries. Finally, in the last two parts, we discuss the application of reinforcement learning techniques, specifically Q-Learning and DQN, to optimize camera placement and coordination for effective area monitoring and dynamic adaptation to changing traffic conditions.

4.1. Robot Mapping

For the process of identifying ideal locations for placing cameras in a specific area, we chose the Turtlebot2 robot platform [35], as shown in Figure 3, which is a robot that hosts the Robot Operating System (ROS), an open-source framework for robot development [36]. Developers can use ROS to build complex robotic systems in research, academia, and industry.

Turtlebot2 uses ROS to communicate with sensors, motors, and other components, and to process data collected by sensors. ROS provides a modular architecture that allows developers to create software components called nodes that can communicate with each other using a publish/subscribe messaging system. Turtlebot2 is equipped with various sensors such as a 2D laser scanner, a 3D depth camera, and an Inertial Measurement Unit

(IMU). These sensors provide the robot with information about its surroundings, such as the position and orientation of obstacles and other objects.

The robot's sensors are connected to the Turtlebot2 computer that runs ROS to process sensor data. ROS provides various libraries and tools for sensor data processing, including point cloud processing and Simultaneous Localization and Mapping (SLAM) algorithms.

SLAM is a key feature of Turtlebot2, allowing the robot to create a real-time map of its surroundings. The robot uses sensors to collect data about its surroundings and uses that data to estimate its position and orientation within a map. As the robot moves through the environment, the map is updated with new information to provide a more accurate representation of space.

Figure 3. The Turtlebot2 robot platform.

ROS also provides tools for controlling Turtlebot2 movements, such as speed control and path planning. These tools enable the robot to navigate its environment and perform complex tasks.

To take this further, Turtlebot2 collects data from sensors and uses advanced algorithms to process and analyze this information. A map is then generated to provide a clear visualization of the surrounding area, including any obstacles, structures, or other objects that may be present. We can use this map to plan and perform various tasks such as navigating the robot to a specific location or identifying a good location for a camera. By mapping the region of interest using Turtlebot2, we obtained a detailed and comprehensive spatial representation of our research lab. The resulting map produced by the robot can be seen in Figure 4. This figure shows the level of detail and accuracy achieved by this approach. The blue arrow in the figure indicates the last location of Turtlebot2 after scanning the entire map. The blue dots and the black lines indicate obstacles such as walls in the map. Later on, after the scanning of the area is complete, the map is fed into our algorithm, through which the initial camera distribution is achieved.

4.2. The Triangle Geometry Approach

The proposed approach utilizes the principles of triangle theory and geometry to initially distribute the surveillance cameras in the intended area. This method involves first

calculating the maximum Cluster Covering Domain (CCD) [3] for each node to determine the monitoring domain of each cluster $C(j)$, where $j = 1, 2, \ldots, N$, and N represents the total number of clusters present in the area. The size of each cluster is denoted by C_{Size}, while the area covered by a single node is represented by A_{FoV} (triangle area). Given that each cluster is established based on a clustering threshold of μ, the CCD can be estimated using Equation (1). Please note that all parameters related to this section are listed in Table 2.

$$CCD \approx \mu A_{\text{FoV}} + (1 - \mu) = A_{\text{FoV}}(C_{\text{Size}} - \mu(C_{\text{Size}} - 1)) \quad (1)$$

This section presents two methods for determining optimal camera placement within a defined area (e.g., rectangular or circular). The first method focuses on minimizing camera overlap. In this approach, if an object moves between adjacent camera fields of view (FoVs), the corresponding cameras are activated to ensure continuous tracking. However, this method does not explicitly optimize for object tracking efficiency. Equation (2) is used to calculate the minimum number of non-overlapping cameras required to cover the specified area. To ensure full coverage of the designated area, cameras should be positioned at the intersections of the overlapped areas, with their orientation being opposite to that of the previously mentioned cameras. Equation (3) calculates the number of cameras in the opposite direction, where $Cameras_{No-overlap}$ is the number of cameras in the front direction, and $Cameras_{Opposite}$ is the number of cameras in the opposite direction.

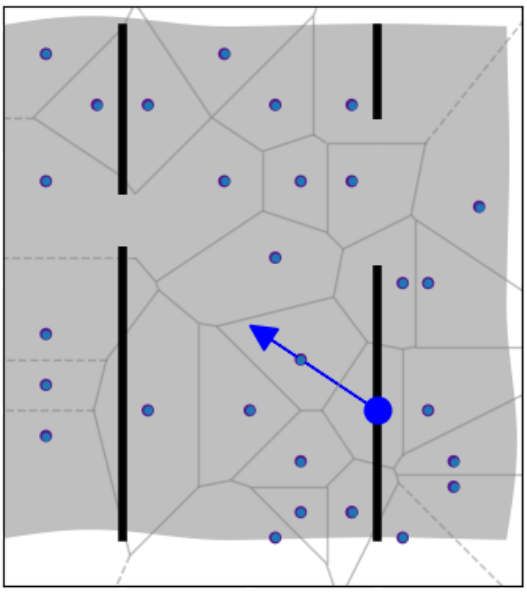

Figure 4. Map of our lab area generated by Turtlebot2.

Table 2. Triangle Geometry Parameter Definitions.

Symbol	Definition
μ	Minimum similarity threshold between points to be grouped into the same cluster (a value between 0 and 1)
A_{FoV}	Area of FoV being analyzed
C_{Size}	Number of points in each cluster
A	Area to be monitored
Overlap	Overlap distance between the two adjacent cameras
H_{FoV}	Height of the FoV.
ω	Overlapping factor (Overlapping percentage)
θ	Angle of FoV
L_{Area}	Length of the original area.
$C_{Per\ Section}$	Number of cameras per section

$$\text{Cameras}_{No-overlap} = \lceil Area/CCD \rceil \tag{2}$$

$$\text{Cameras}_{Opposite} = \text{Cameras}_{No-overlap} - 1 \tag{3}$$

The second approach for determining camera positions considers overlapping FoVs to enable continuous object tracking without the need for individual camera activation and deactivation. Unlike Equation (2), which requires individual cameras to activate and deactivate based on object movement, the second approach (deployment of overlapping cameras) incorporates a specific overlap ω value (e.g., if we want 30% overlap, then ω = 0.3) to facilitate object tracking between cameras [3]. Equation (4) calculates the additional number of cameras that covers the redundant area that resulted from the overlapping. M is the number of cameras found by Equation (2).

$$\text{Cameras}_{overlap} = \frac{Area - Area^*}{CCD} \tag{4}$$

where $Area^* = \sum_{i=1}^{M} A_{FoV_i} - \sum_{i=1}^{M} Overlap_i$.

Once the total number of cameras has been determined for both non-overlapping and overlapping scenarios, we proceed to divide the area into sections to distribute the cameras, as shown in Figure 5a. Equation (5) calculates the number of sections required to partition the total monitored area, where H_{FoV} is the height of camera FoV and L_{Area} is the length of the area.

$$\text{Number of Sections} = \frac{Area}{H_{FoV} \times L_{Area}} \tag{5}$$

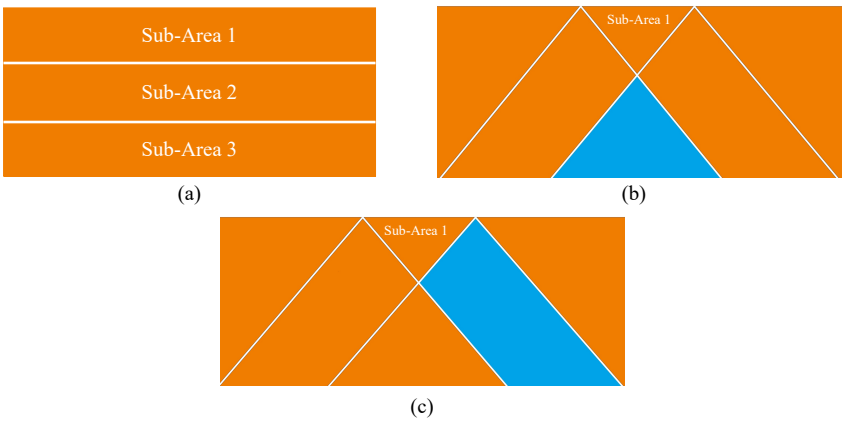

Figure 5. Detection and tracking.

Algorithms 1 and 2 detail the implementation of the non-overlapping and overlapping camera distribution strategies, respectively.

Algorithm 1 Non-Overlapped Camera Distribution Pseudocode

Input: Area Size, Cluster Size, Clustering Threshold
Output: No. Cameras, Coverage Area, Dark (not covered) Area

```
Calculate the maximum Cluster Covering Domain (CCD) //Equation (1)
Calculate the number of non-overlapping cameras per section //Equations (2) and (3)
Calculate number of area sections //Equation (5)
Calculate total number of cameras // (Equation (2) + Equation (3)) * Equation (5)
Calculate coverage area
Drawing area: steps 7 to 19
cameras_drawn ← 0
for i ← 1 to no_sections do
    draw(section)
    no_cameras_in_section ← section_width / FoV_base
    for j ← 1 to no_cameras_in_section do
        if cameras_drawn ≥ no_cameras then
            break
        else
            draw(camera)
            cameras_drawn ← cameras_drawn + 1
            if j < no_cameras_in_section − 1 and cameras_drawn < no_cameras then
                draw(camera)
                cameras_drawn ← cameras_drawn + 1
```

Algorithm 2 Overlapped Camera Distribution Pseudocode

Input: Area Size, Cluster Size, Clustering Threshold, Ω
Output: No. Cameras, Coverage Area, Dark (not covered) Area

```
Calculate the maximum Cluster Covering Domain (CCD) //Equation (1)
Calculate the number of additional overlapping cameras per section //Equation (4)
Calculate number of area sections //Equation (5)
Calculate total number of cameras // (Equation (2) + Equation (4)) * Equation (5)
Calculate coverage area
Drawing area: steps 7 to 19
cameras_drawn ← 0
for i ← 1 to no_sections do
    draw(section)
    no_cameras_in_section ← section_width / FoV_base
    for j ← 1 to no_cameras_in_section do
        if cameras_drawn ≥ no_cameras then
            break
        else
            draw(camera)
            cameras_drawn ← cameras_drawn + 1
            if j < no_cameras_in_section − 1 and cameras_drawn < no_cameras then
                draw(camera)
                cameras_drawn ← cameras_drawn + 1
```

4.3. Motion Sensors and Radiation Pattern

To enhance object detection within the area, we propose a distributed network of motion sensors. These sensors are randomly deployed throughout the region and wirelessly connected to cameras equipped with directional antennas. The antenna's radiation pattern is designed to be wider than the camera's FoV, enabling connectivity with neighboring cameras in overlapping areas. Additionally, the pattern's peak intensity is directed toward

the next camera in the sequence to facilitate communication. Only motion sensors within the antenna's radiation range will respond to the camera's queries.

Following the deployment of cameras and motion sensors, an initial connectivity phase is initiated. Cameras broadcast signals that are received and analyzed by neighboring cameras and sensors. If the signal strength exceeds a predefined threshold, the receiving device responds with its location and type information. Each camera maintains a local database to store the details of connected devices, including their ID addresses, locations, and types.

To ensure comprehensive coverage, neighboring cameras positioned at overlapping area intersections, along with the central camera, must form a kind of cluster. This cluster facilitates coordinated operation, where activating one camera triggers the awakening of the others. At any given time, a minimum of three cameras within the cluster must be active.

This interconnected network enables seamless communication between cameras and motion sensors. Upon object detection, the system strategically activates the necessary cameras to track the object's trajectory. A sensor-triggered signal prompts a camera to wake up and initiate object detection. If no such signal is received, the camera continues tracking the existing object.

4.4. Detection and Tracking

Object detection is a fundamental task in numerous applications, often hindered by the presence of distortions in video sequences. While visual tracking techniques can mitigate these distortions, they can also introduce noise and disturbances. To address these challenges, various object tracking algorithms have been proposed [37]. In our system, the region of interest is partitioned into subareas, each corresponding to a camera's FoV, as illustrated in Figure 5a. Cameras are strategically deployed based on one of our proposed mechanisms, while sensors are randomly distributed across the entire area. Initially, all cameras remain in a power-saving state.

Object detection is initiated when an object traverses subarea 1 (section). Upon detecting motion, the corresponding sensor activates the camera. If the object remains within the camera's FoV, tracking continues. However, if the object exits the FoV, another sensor detects the motion and triggers the nearest camera to resume tracking. When an object enters an overlapping area, as depicted in Figure 5b, the sensor activates three cameras: the two cameras within the overlapping area and the camera positioned at their intersection.

Motion sensors continuously monitor the environment. When an object enters a camera's FoV, the sensor detects the motion and transmits a signal. If the signal strength surpasses a predetermined threshold, the sensor triggers the nearest camera, which activates and commences object tracking. As the object moves between subareas, the corresponding sensors detect the motion and activate the relevant cameras.

When an object traverses an overlapping area, the sensor transmits a signal to the nearest camera. If the sensor receives signals from two cameras within a specific proximity, it identifies the object's location as being within an overlapping region and sends signals to both cameras in that area.

4.5. Q-Learning RL-Based Camera Adjustment

Given that overlapping accounts provide redundant coverage essential for effective tracking detection, intelligent mechanisms that adapt to changing traffic conditions are needed. The proposed solution uses RL with the states represented as (Area, FoV, Traffic Location, Overlap Distance). FoV is used to allow the cameras to extend their views. The actions the RL agent can take are represented as a pair of overlapping distance and FoV

values. In addition to these two actions, the RL agent can add a camera in a traffic location if one of the cameras fails. For the RL agent to learn how to cover an object, we simulated an environment in which we generate random traffic in each episode of the agent training phase, within each episode the RL agent tries to increase the overlapping distance between the cameras by changing the FoV to cover the object. If the taken action can cover the object, it is awarded a positive reward. Meanwhile, if it cannot, it is awarded a negative reward. Updating the Q-table by the RL agent is given by Equation (6):

$$Q_{New} = Q_{Old} + \alpha \cdot [R + \gamma \cdot \max Q - Q_{Old}] \quad (6)$$

Overall, the 3D function of the RL agent can be illustrated in Figure 6.

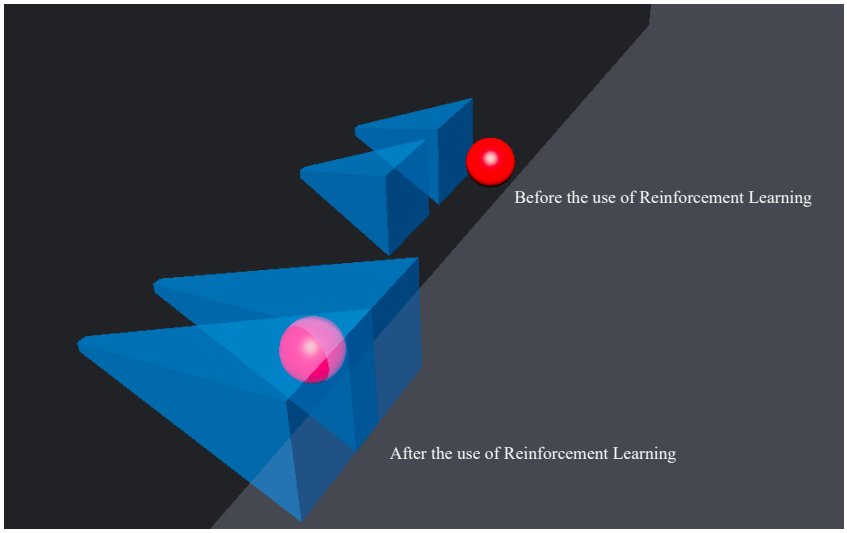

Figure 6. Reinforcement agent function.

In our Q-Learning RL technique, we specifically utilize the epsilon-greedy strategy. The epsilon-greedy strategy is a trade-off between exploration and exploitation, where the agent chooses a random action with a probability of ϵ and the best-known action with a probability of $1 - \epsilon$. This approach enables the agent to explore different actions while still focusing on actions that have previously yielded favorable outcomes. The definitions of all parameters of the equations used in this section are in Table 3.

Table 3. RL Parameter Definitions.

Symbol	Definition
Q	Current estimate of the Q-value for the current
α	Learning rate, which determines the weight given to the new information compared with the old estimate of the Q-value.
R	Reward received for taking the current action in the current state.
γ	Discount factor, which determines the importance given to future rewards compared with immediate rewards
Max_Q	Maximum Q-value over all possible actions in the next state. This represents the estimated maximum long-term reward achievable from the next state

In order to calculate the reward for the Q-Learning technique, we measure the object coverage by the distributed overlapped cameras. A certain number of traffic objects are randomly located within the monitored area, and then the traffic coverage percentage is calculated (assuming that the object is covered when it is located within the FoV of a camera, as explained in Figure 6).

Then, the Q-Learning RL agent uses an epsilon-greedy strategy to balance exploration and exploitation. Initially, the agent explores various camera setups (such as camera FoV, overlapping distance, and camera placement), gradually shifting towards exploiting learned better configurations. The state space represents the environment's conditions (camera positions, FoV, ω), and actions involve moving or adjusting camera parameters. Rewards are computed based on improvements in object coverage and used number of cameras.

Algorithm 3 summarizes how the Q-Learning RL agent works in our proposed camera distribution approach. As an example of how our RL agent works, we distributed 10 random traffic points in a 20 $\times$ 20 area and deployed the RL agent after it had been trained. The RL agent's decision to cover the traffic points can be observed in Figure 7.

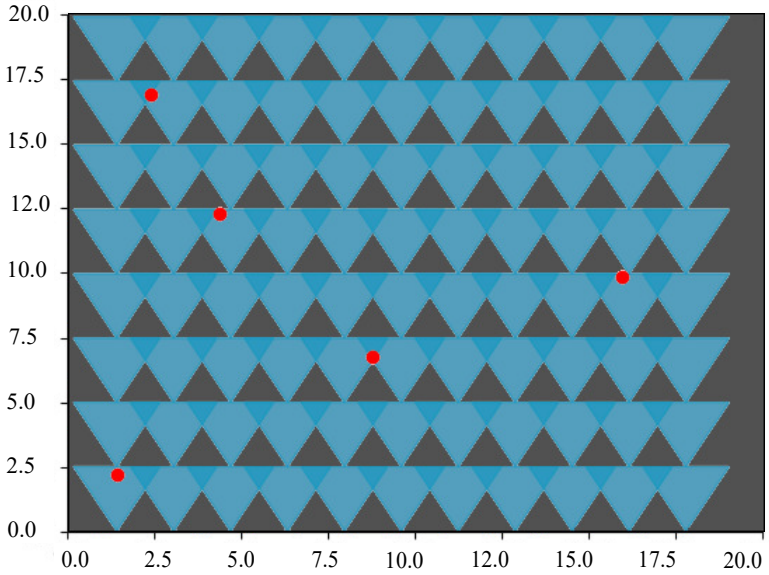

Figure 7. Reinforcement agent's decision.

Algorithm 3 Q-Learning RL agent

Initialize Q-table with all zeros
Set learning rate α, discount factor γ
Set exploration rate ϵ
Set ϵ decay
Randomly distributed objects and calculate traffic coverage
for each episode **do**
    Initialize environment, obtain initial state s
    while episode not terminated **do**
        Choose action a using ϵ-greedy strategy:
        With probability ϵ, choose a random action
        With probability $1 - \epsilon$, choose action with highest Q-value for state s
        Perform action a, observe new state s' and reward
        Update Q-value using Q-Learning update rule:
        $Q[s, a] \leftarrow Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$ ▷ Equation (6)
        Update state: $s \leftarrow s'$
    Reduce exploration rate ϵ if necessary (e.g., using decay function)
Evaluate learned Q-values and/or policy

A final visualization of how the RL agent works is shown in Figure 8. The RL agent interacts with the environment by choosing actions based on their policies, receiving

rewards, and updating their value functions. This cycle continues until the RL agent's policy converges to better performance in maximizing the coverage percentage.

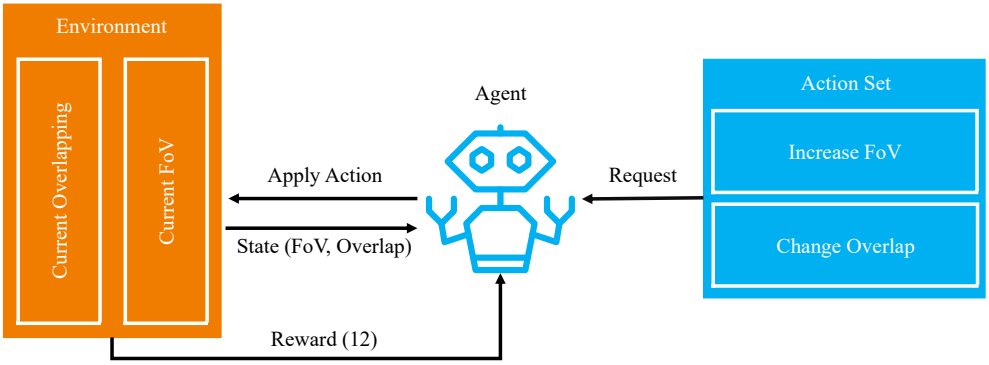

Figure 8. Workflow of our reinforcement agent.

4.6. DQN RL-Based Camera Adjustment

A Deep Q-Network (DQN) is a type of RL that uses neural networks rather than Q-tables to evaluate the Q-values and obtain learning-optimal actions in a given environment. This approach leverages deep learning to approximate the Q-value function, which is used to predict the expected future rewards for taking specific actions in given states, enabling the RL agent to learn optimal policies over time. The integration of Deep Q-Networks (DQNs) into IoT-based smart surveillance systems provides a sophisticated approach to optimizing camera placement and coordination for effective area monitoring, as shown in Figure 9. This strategy seeks to maximize surveillance coverage while minimizing camera redundancy. By dynamically adjusting the camera parameters such as FoV and camera placement in response to real-time traffic conditions, DQN ensures optimal performance.

The proposed DQN algorithm begins by initializing a replay buffer with capacity N to store experiences and a policy network (Q-network) with random weights, alongside a target network initialized with identical weights to stabilize training. The learning rate α, discount factor γ, exploration rate ϵ, and exploration decay ϵ_{decay} are set accordingly. During each episode, the environment is initialized, and the RL agent, following the ϵ-greedy strategy, either selects a random action with probability ϵ or exploits the policy network by choosing the action with the highest Q-value.

The possible actions are defined as either Move Left to shift the camera to the left, reducing the x-coordinate, or Move Right to shift the camera to the right, increasing the x-coordinate. These actions allow the system to dynamically adjust the camera positions and their overlapping distance to optimize target coverage. The state represents the current configuration of the surveillance system. It includes the position of a camera and which area section it is in. For example, a state could be (x, y), where x denotes the horizontal position, and y represents the section being monitored.

The RL agent then observes the new state s' and reward r, and stores the experience tuple $(s, a, r, s', done)$ in the replay buffer. A mini-batch of experiences is sampled for training, and for each experience, if the episode has ended, the target Q-value is set to r; otherwise, it is calculated using Bellman's equation, as follows:

$$r + \gamma \cdot \max(Q(s', a'; target\ network)) \tag{7}$$

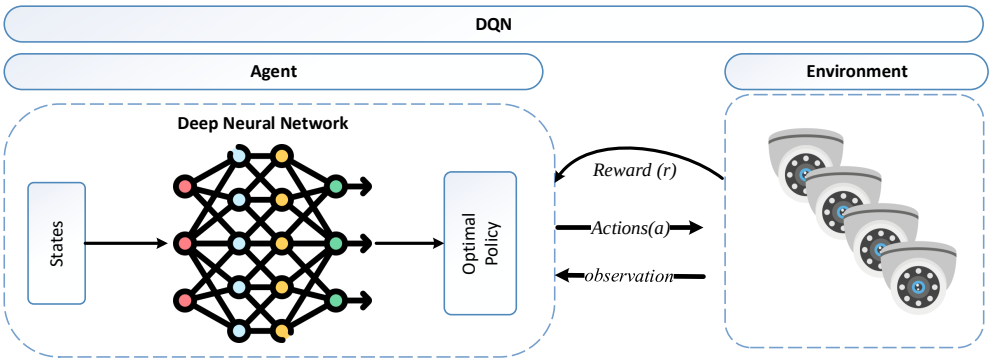

Figure 9. DQN approach.

The policy network is updated by minimizing the Huber Loss or Mean Squared Error (MSE) between the predicted Q-value and the target. The state is updated to s', and the target network is updated every C step to match the policy network. The exploration rate ϵ is decayed progressively, and the Q-values and policy are evaluated to assess the RL agent's performance. The pseudo code of our DQN algorithm is shown in Algorithm 4.

$$a = \begin{cases} \text{random action} & \text{with probability } \varepsilon \\ \arg\max_a Q(s, a; \text{policy\_net}) & \text{with probability } 1 - \varepsilon \end{cases} \tag{8}$$

$$s' = (x + (1 - 2a), y) \tag{9}$$

$$\text{Loss}(q, \text{target\_Q}) = \begin{cases} 0.5(q - \text{target\_Q})^2, & \text{if } |q - \text{target\_Q}| < 1, \\ |q - \text{target\_Q}| - 0.5, & \text{otherwise.} \end{cases} \tag{10}$$

$$\varepsilon = \max(\varepsilon_{\min}, \varepsilon \times \varepsilon_{\text{decay}}) \tag{11}$$

The proposed solution's workflow is visualized in Figure 10, where the step-by-step process of the algorithm can be observed and analyzed, providing a clear understanding of how the solution functions and enabling potential improvements to be identified and implemented.

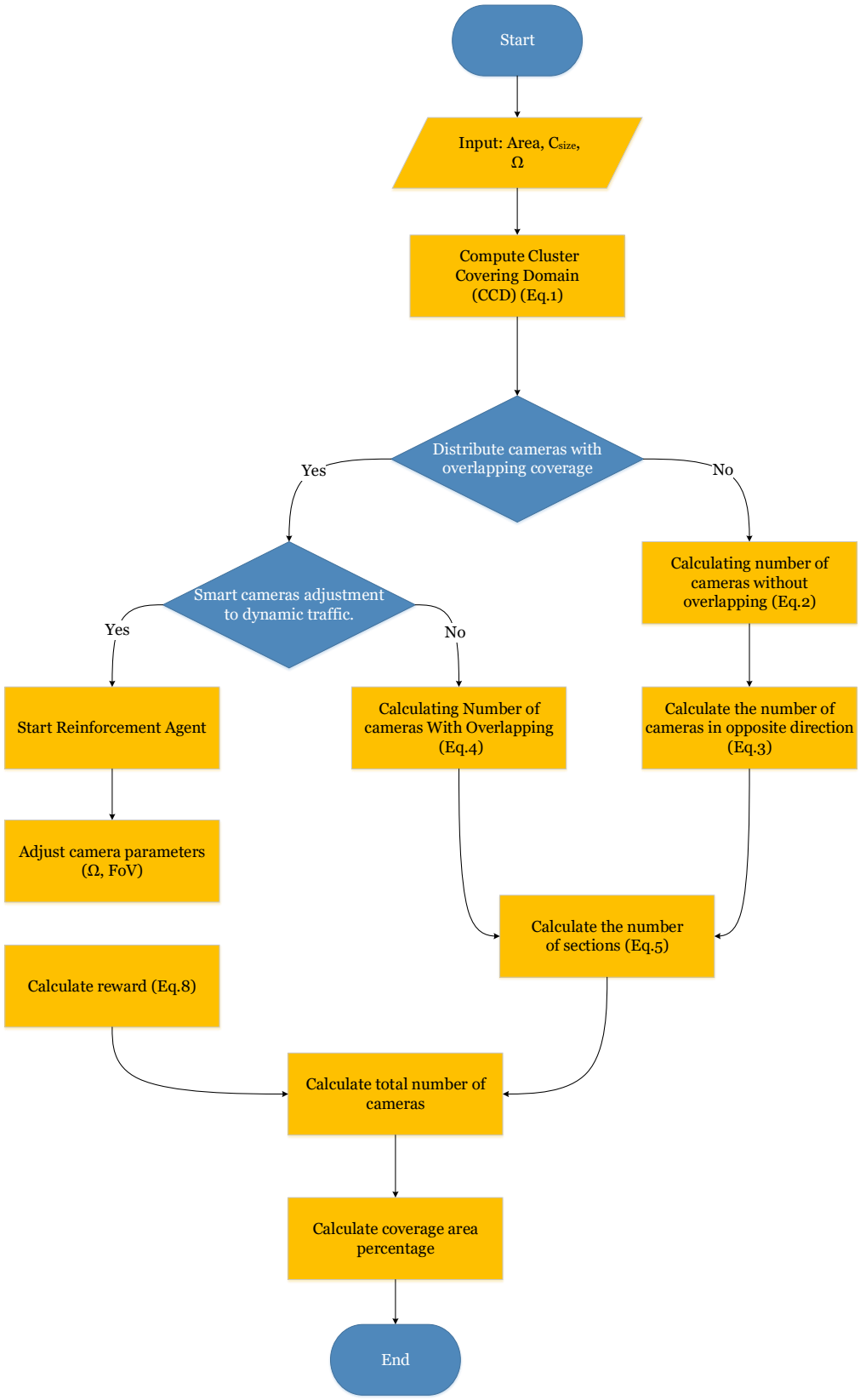

Figure 10. Workflow of our proposed solution.

Algorithm 4 DQN RL agent

```
Initialize ReplayBuffer with capacity N
Initialize policy network (Q-network) with random weights
Initialize target network with the same weights as the policy network
Set learning rate α, discount factor γ
Set exploration rate ϵ and exploration decay ϵ_decay
traffic_coverage ← is_traffic_covered()
for each episode do
    Initialize environment, obtain initial state s
    while episode not terminated do
        Choose action a using ϵ-greedy strategy:
          With probability ϵ, choose a random action.
          With probability 1 − ϵ, choose action with
          highest Q-value from policy network.
        Perform action a, observe new state s′ (see Equation (9))
        and reward r.
        Store experience (s, a, r, s′, done) in ReplayBuffer.
        Sample a mini-batch from ReplayBuffer.
        for each experience (s, a, r, s′, done) in the
            mini-batch do

            if done then
                target ← r
            else
                target ← r + γ ∗
                max_a′ Q(s′, a′, target_network)
            Update Q-value (policy network) for (s, a)
            using the loss function (Huber Loss or MSE)
        Update state: s ← s′
        Update target network weights every C steps
    Reduce exploration rate ϵ if necessary
    (using decay function ϵ_decay)
Evaluate learned Q-values and/or policy
```

5. Simulation Results and Performance Analysis

In this section, we present the performance outcomes of our proposed approach to triangle geometry. We examine two scenarios for the triangle geometry algorithm camera coverage: non-overlapping and overlapping scenarios. This analysis includes discussions on the optimal camera distribution and the maximum number of cameras needed for comprehensive coverage. Following that, we evaluate the results of the reinforcement agent and provide a critique of its optimal hyperparameters. Subsequently, we delve into the intricacies of the triangle geometry and reinforcement learning algorithms, breaking down their complexities. Lastly, we conduct a comparative analysis of our proposed solution against recent algorithms that address similar problems.

5.1. Non-Overlapping Camera Distribution Results

This section of this study focuses on the scenario of non-overlapping cameras, where each camera covers a distinct section of the area being monitored. To investigate this scenario, we conducted a series of trials, in which we maintained a constant area of 20×20 and varied the camera parameters in each trial. Specifically, we analyzed the coverage and dark areas of each trial, the number of cameras required to cover the entire area, and the number of distinct sections created by the cameras. The camera distribution plots for each trial are illustrated in Figure 11 and the results can be observed in Table 4. In this table,

maximum area coverage can be achieved by altering several factors: camera parameters (Fov and CCD) and the number of deployed non-overlapping cameras. By carefully analyzing these results, we can gain a better understanding of the trade-offs between camera parameters and the number of cameras required for monitoring a given area.

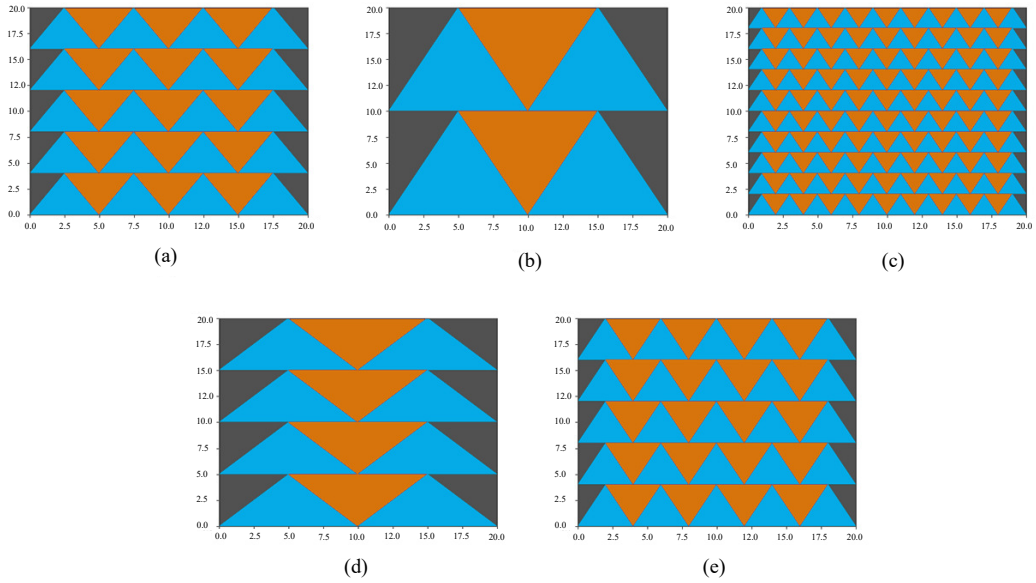

Figure 11. Non-overlapping camera distribution scenario.

Table 4. Non-overlapping camera distribution scenario results.

Trial	Cameras	FoV Area	CCD	Coverage Area
a	35	10.0	11.599	87.50%
b	6	50.0	58.0.0	75.00%
c	190	2.0	2.0	95.50%
d	12	25.0	25.0	75.00%
e	45	8.0	8.0	90.00%

5.2. Overlapping Camera Distribution Results

This series of trials aimed to investigate the impact of the parameter ω on both the FoV and the required number of cameras. We set the parameters for the cameras to be of 2 units for the area of FoV and 2 units for CCD. ω is a critical parameter that indicates the degree of overlap between adjacent cameras, which ultimately determines the level of redundancy in the monitored area. Our goal was to determine how adjusting the value of ω can affect the FoV and the number of cameras needed to achieve complete coverage of the area. We concluded that a value of ω below 0.5 would require additional cameras to be deployed to achieve greater coverage and reduce the dark areas. This is because a lower value of ω means that there is less overlap between adjacent cameras, which results in more gaps in the monitored area. By deploying additional cameras, we can achieve greater coverage and reduce the number of dark areas, but this also comes at the cost of increased complexity and higher expenses.

Overall, our findings highlight the importance of carefully adjusting the parameter ω to achieve optimal coverage with the minimum number of cameras possible. By finding the right balance between the degree of overlap and the number of cameras deployed, we can effectively monitor a given area and detect any potential threats or incidents in a timely and efficient manner. The camera distribution plots and the results of each trial can be observed in Figure 12 and Table 5. The distance factor is used to represent the distance between the angle vertices of FoVs between two adjacent cameras. In this table, maximum area coverage can be

achieved by increasing the overlapping ratio between the adjacent cameras but at the expense of increasing the number of deployed cameras. Therefore, there is a trade-off between maximizing the coverage and increasing the cost, assuming fixed camera parameters.

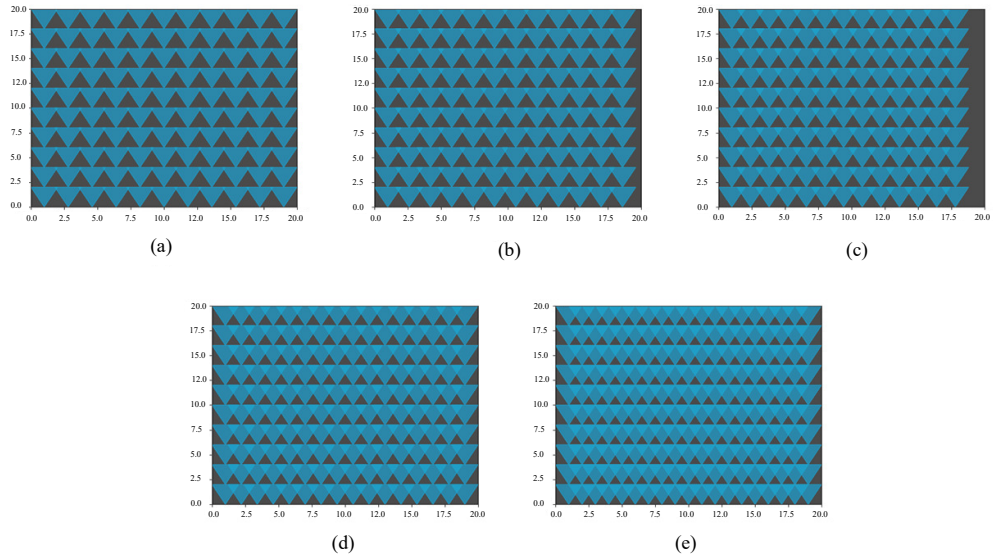

Figure 12. Overlapping camera distribution scenario.

Table 5. Overlapping camera distribution scenario results.

Trial	Distance	ω	No. Cameras	Coverage Area
a	1.8	0.1	110	54.55%
b	1.6	0.2	120	58.20%
c	1.4	0.3	130	60.95%
d	1.2	0.4	160	72.8%
e	1.0	0.5	190	83.75%

As mentioned before, to overcome the fact that with overlapping cameras the dark area will increase as long as ω is less than 0.5, we conducted another trial in which we added cameras in the opposite direction, as illustrated in Figure 13, and measured the coverage area, which was 83% for an ω of 0.3 and a 20 $\times$ 20 area with a FoV area of 2 and a CCD of 2.

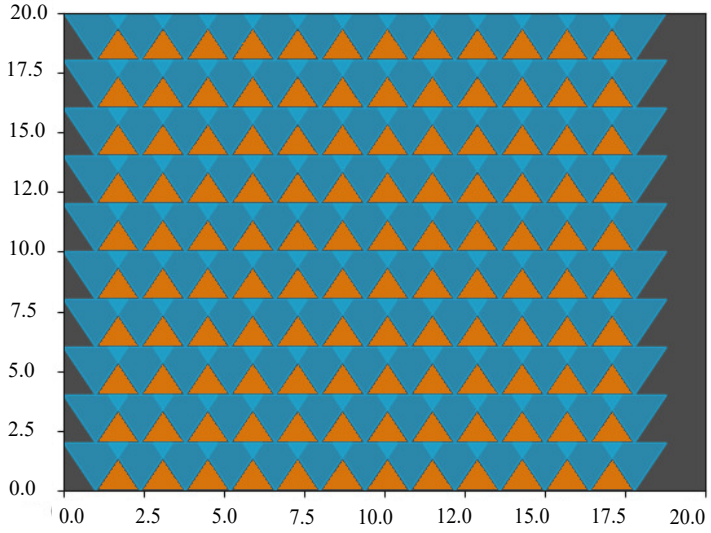

Figure 13. Overlapped scenario plot: added cameras in opposite direction.

5.3. Reinforcement Agent

We conducted a series of comprehensive trials and highlighted six of them, where we systematically altered the agent's hyperparameters in each trial according to the specifications outlined in Table 6. In this table, Trial (f) has the best performance among the others, with the lowest average loss value of 0.053. This means that the RL agent has learned the optimal policy in camera distribution that gives the maximum coverage. It is worth mentioning that our proposed model follows the Monte Carlo methodology [38], which leverages random sampling to approximate numerical results. When applying the Monte Carlo method in reinforcement learning, the number of steps or episodes becomes relevant because it affects the accuracy of the value estimation. The Monte Carlo method relies on collecting sample episodes and averaging their returns to estimate the value of a state–action pair or a policy. The more episodes or steps the agent takes, the better the estimate tends to be. Specifically, our approach involved performing 1000 steps and 10,000 episodes. The objective of these trials is to identify the hyperparameters that minimize fluctuations in Q-values as computed in Equation (6), indicating that the agent has successfully learned an optimal policy with the lowest achievable average loss.

Table 6. Reinforcement agent hyperparameters.

Trial	ϵ	α	γ	ϵ Decay	Average Loss
A	1.0	0.4	0.5	0.99	0.591
B	0.7	0.4	0.5	0.99	0.264
C	0.7	0.2	0.8	0.99	0.101
D	0.7	0.2	0.8	0.01	0.367
E	0.7	0.8	0.2	0.99	0.613
F	0.5	0.2	0.9	0.99	0.053

For each trial in Figure 14, we applied the moving average to the Q-values with a window size of 100 to visualize the learning process, and we meticulously evaluated the loss value, calculated using Equation (12), where Target_Q is the estimated optimal Q-value for the next state. Subsequently, we computed the average loss value using Equation (13). In Trial (a), we observed a significant number of fluctuations in the Q-values, indicating that the agent had not yet converged to an optimal policy. As a result, the average loss value in this trial was relatively high, measuring 0.591. This behavior can be attributed to the utilization of an ϵ value of 1.0, which introduced substantial randomness into the agent's decision-making process.

$$\text{Total Loss} = (\text{Target\_Q} - \text{Q})^2 \tag{12}$$

$$\text{Average Loss} = \frac{\text{Total Loss}}{\text{Number of Steps}} \tag{13}$$

During Trial (b), we noticed the agent's ability to acquire an optimal policy by reducing the value of ϵ to 0.7. However, the average loss value recorded in this trial was 0.264. In this case, there was a balance between the values of α and γ.

Likewise, in Trial (c), the agent also learned an optimal policy with an ϵ value of 0.7. Nevertheless, an impressive improvement occurred as the average loss value dropped significantly to a mere 0.101. This accomplishment was made possible by adjusting the γ value to be higher than the α value.

Motivated by the promising results of Trial (c), we attempted to replicate its success in Trial (d). However, this time, we modified by reducing the ϵ decay to 0.01. Unfortunately, this alteration yielded numerous fluctuations in Q-values, indicating that the agent strug-

gled to learn an optimal policy. Consequently, Trial (d) reported a relatively high average loss value of 0.367.

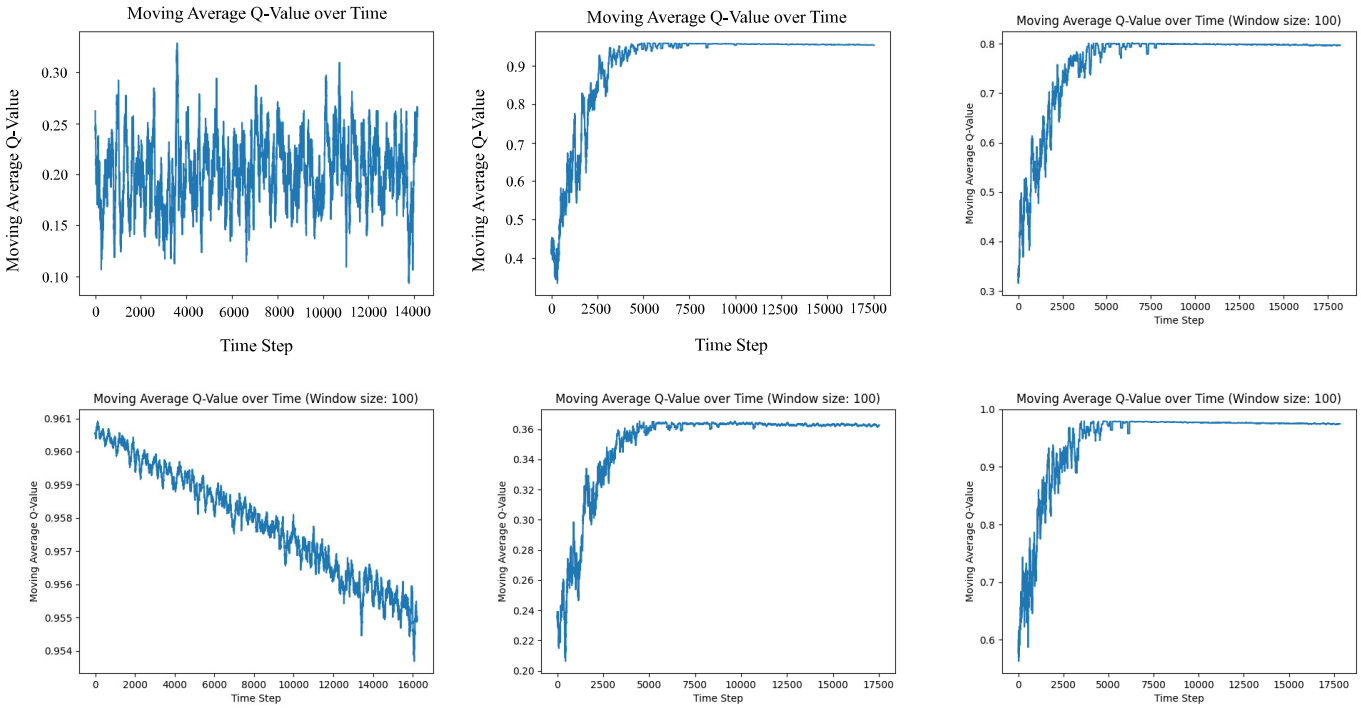

Figure 14. Reinforcement agent performance with window size (100).

In Trial (e), we aimed to explore the impact of setting a higher value for α compared with γ. Despite the agent successfully learning an optimal policy and demonstrating stability in the Q-values, the average loss value remained considerably high, measuring 0.613. Additionally, the Q-values exhibited stability at a level of 0.36, which is comparatively lower than those observed in other successful trials.

Finally, Trial (f) showcased the most exceptional performance among all the trials. We achieved remarkable outcomes by establishing a configuration where γ surpassed α, and ϵ was set to 0.5. The average loss value plummeted to an impressive 0.053, indicating that the agent had effectively learned an optimal policy with minimal fluctuations. The Q-values were stable at 1, which is higher than other successful trials. Furthermore, the selection of an epsilon value of 0.5 strikingly struck a fine balance between exploration and exploitation, contributing to the favorable results observed in Trial (f).

In summary, through our meticulous experimentation and analysis, we gained valuable insights into the influence of hyperparameters on the agent's performance. The diverse trials allowed us to identify the optimal combination of hyperparameters that led to the agent's effective learning, culminating in the attainment of an optimal policy with minimal fluctuations and a significantly reduced average loss value.

Regarding DQN RL agent performance, Figure 15 shows the agent's total reward improving rapidly during the early episodes, stabilizing near the maximum as training progresses. Fluctuations are due to exploration, but overall, the agent consistently learns an effective policy. Figure 16 shows the loss decreasing over time during the training process. Initially, the loss is high as the DQN agent's policy network is untrained, but as the DQN agent interacts with the environment and learns from experiences, the loss decreases significantly. By the later episodes, the loss approaches near zero, indicating that the Q-network is making accurate predictions and the DQN agent's policy has converged. This reflects successful learning and stabilization of the Q-values over time.

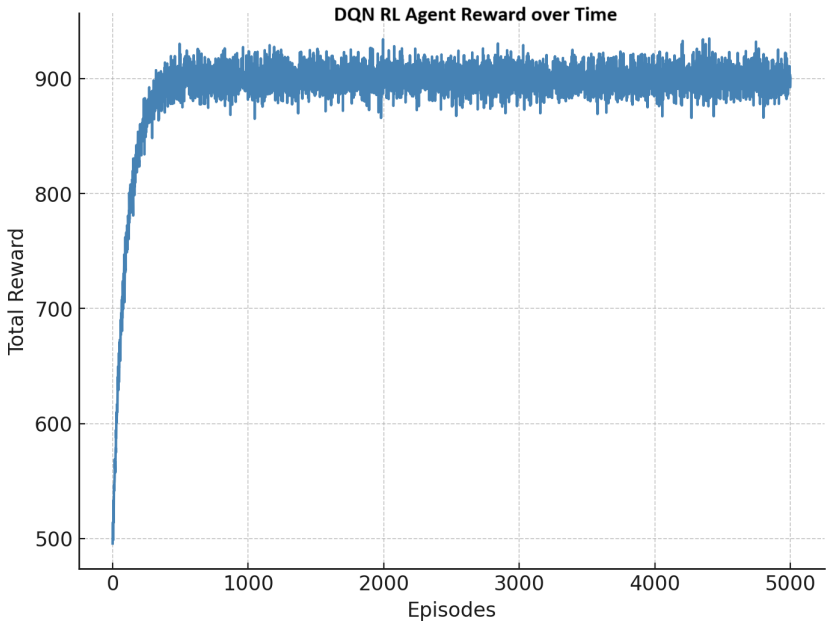

Figure 15. DQN RL agent total reward.

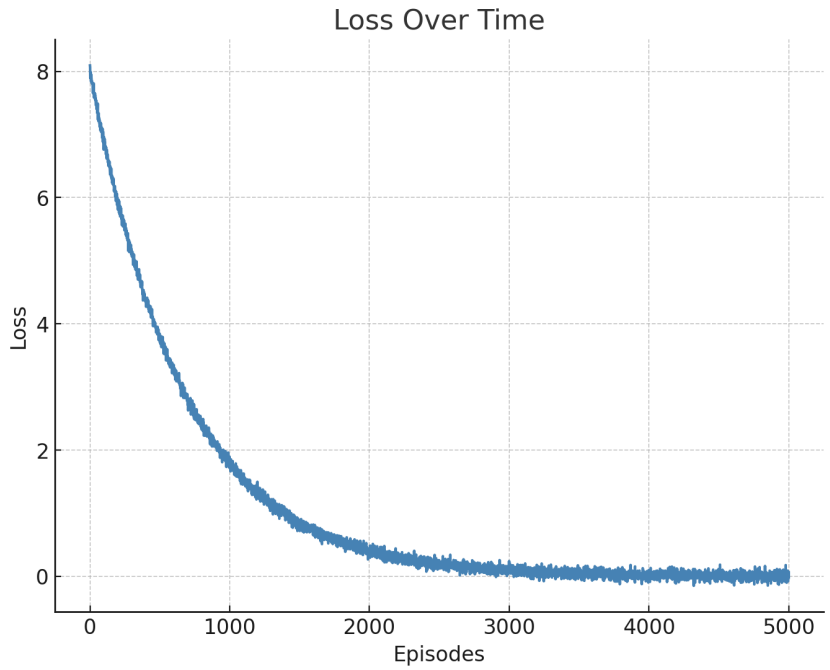

Figure 16. DQN RL agent loss function.

5.4. Complexity Analysis

To analyze the complexity of the algorithm, we consider the most time-consuming parts of the algorithm. The most relevant part of the complexity analysis is the loop that draws cameras in sections:

```
for i in range(num_sections)
...
for j in range(num_cameras_in_section)
...
```

In terms of triangle geometry (non-overlapping and overlapping) pseudocode, let S be the number of sections and C be the number of cameras in a section. The complexity of nested loops is O(S * C). However, both S and C depend on the input parameters and are not directly proportional to the input size. The complexity of the code therefore depends on the specific configuration of the problem and cannot be expressed as a function of a single variable (e.g., n).

Other parts of the code perform basic arithmetic operations and their complexity is O(1). In summary, the code complexity is mainly determined by nested loops with O(S * C) complexity depending on the number of sections and the number of cameras in the sections. Complexity is not directly proportional to a single input variable, making it more difficult to express in standard Big-O notation.

The time complexity of Q-Learning algorithms mainly depends on the number of episodes, the number of steps per episode, and the size of the Q-table.

To break down the complexity analysis, we can consider the following:

- Number of episodes: E.
- Maximum number of steps per sequence: T.
- Number of states: S.
- Number of actions: A.

The episode's loop is executed E times, and the inner loop of steps within the episode is executed at most T times. At each step, the algorithm updates the Q-table entries for state–action pairs. This will take some time. The overall time complexity of the Q-Learning algorithm is $O(ET)$. Note that this complexity analysis does not take into account the time required to explore the environment, perform actions, or observe new states and rewards that may depend on the particular problem and implementation. For space complexity, the Q table stores the value for each state–action pair, so the space complexity is $O(SA)$.

5.5. Comparison with Previous Approaches

Our proposed solutions were compared with existing work in the field, specifically [15,18,27,28]. Our comparison focused on evaluating the impact of various factors on coverage area, including the number of cameras, terrain in square meters, fairness index, and number of targets. The fairness index is given by Jain's fairness index, as described in Equation (14).

In Figures 17–20, we present plots that demonstrate the performance of each approach concerning the metrics of interest. Upon careful examination of these figures, we observe that our approach consistently performs better than the other approaches across all of the selected metrics. The superiority of our approach can be attributed to its unique design and novel features that enable it to effectively address the challenges associated with the problem at hand. The results of this study provide strong evidence for the efficacy of our approach and highlight its potential to significantly enhance the performance of similar systems in real-world applications.

The Hierarchical policy results in [18] lack coordination among cameras on different clusters' boundaries. This leads to a loss of coverage, especially when more cameras are deployed in a fixed-sized terrain. As more sensors are added (beyond 40–50) in [15], having fewer time slots (1-δt, 2-δt, 3-δt) reduces coverage instead of increasing it, because of overlapping redundancy. In [28], HTOH may have a lack of coverage because it relies on a combination of the Target-Oriented Heuristic (TOH) and the Centralized Greedy Algorithm (CGA). The TOH focuses on covering targets, while the CGA selects cameras and orientations to maximize target coverage. However, the CGA does not take into account the distribution of targets and may end up selecting cameras that do not cover important areas. In the HTOH algorithm, the TOH is used to first cover the most critical

targets, and then the CGA is used to select cameras and orientations to cover the remaining targets. However, if the critical targets are not properly covered by the TOH, the CGA may not be able to compensate for the lack of coverage.

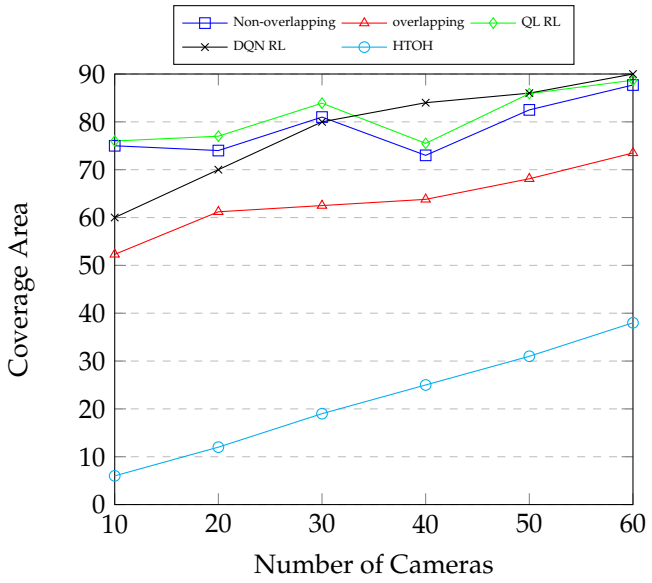

Figure 17. Coverage area vs. number of cameras.

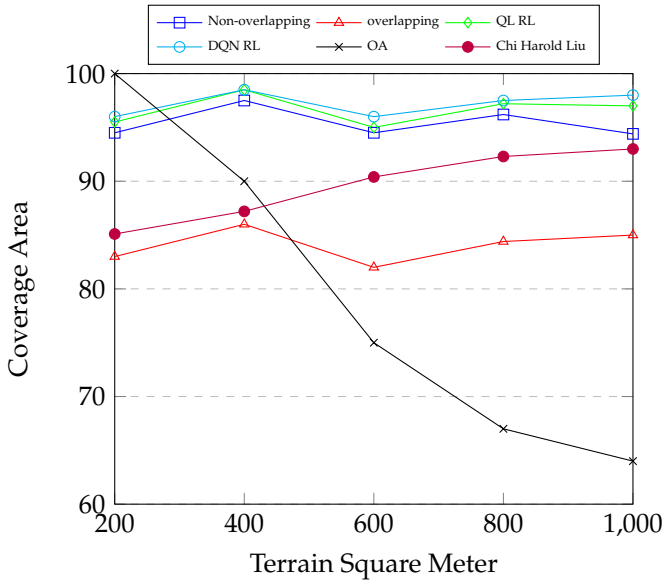

Figure 18. Coverage area vs. terrain.

Regarding our investigation of the correlation between cluster size and coverage area, the FoV in the first proposal remained constant, with a variable cluster size at 95%, which we achieved using non-overlapping cameras, and a higher percentage of 96% with overlapping cameras, provided that cameras are added: 95% without overlapping and 84.4% with overlapping.

In terms of the fairness index, as calculated in Formula (14), where n represents the number of different geographic regions or neighborhoods where cameras are installed, and x is the number of cameras installed in each geographic region, our approach maintains a near 100% fairness index, because the minimum and maximum number of cameras covering any subsection are the same. Maintaining a high fairness index is desirable because it ensures that all parts of the monitored area receive the same level of surveillance, and there are no blind spots or areas with less coverage. It can also help to ensure that any

incidents or activities in the monitored area are detected and recorded with equal accuracy and detail.

$$JFI = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2} \tag{14}$$

Our agent achieves complete target coverage by dynamically adjusting the overlap distance and camera's FoV in response to environmental changes, ensuring that the traffic is fully covered at all times maintaining the curve at 100%.

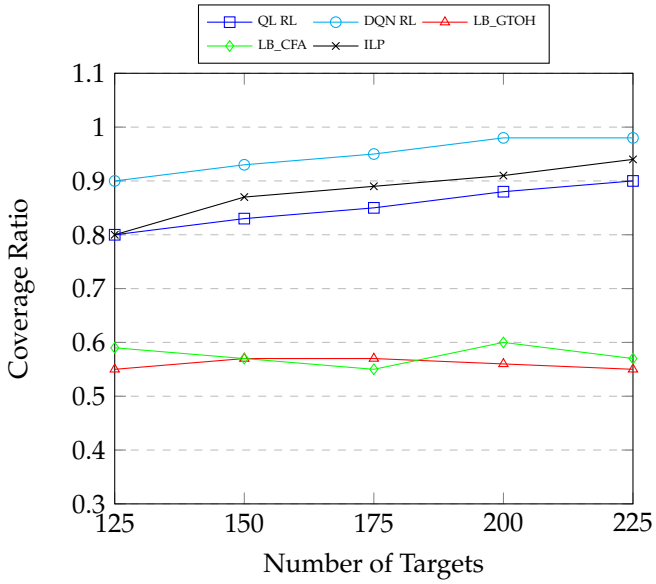

Figure 19. Coverage ratio vs. number of targets.

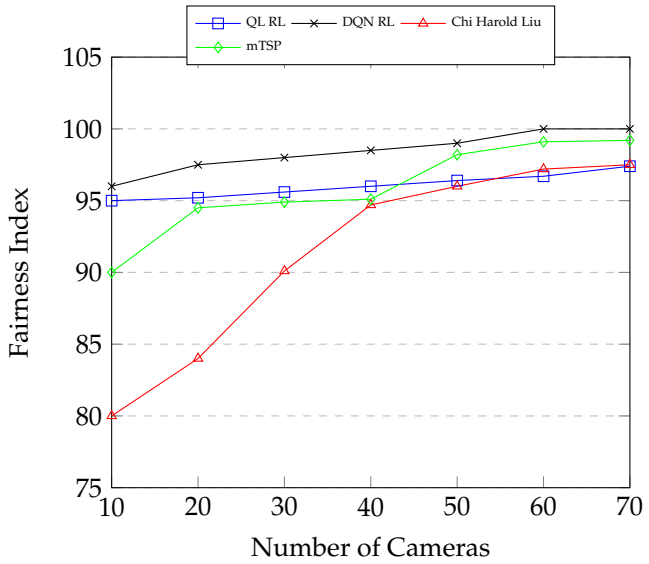

Figure 20. Fairness index vs. number of cameras.

6. Conclusions

Advances in affordable communication devices and cameras have paved the way for Internet of Things (IoT) systems to provide effective solutions for a variety of applications, such as surveillance systems that monitor specific areas. However, current surveillance systems often incur additional communication costs and have redundancy in the detection area. To address these issues, we propose an intelligent IoT-based surveillance system using two reinforcement learning (RL) methods, Q-Learning and Deep Q-Learning. The proposed

schemes aim to optimize camera placement based on camera FoV for maximum target coverage with minimal camera sensors. The RL agent dynamically adjusts camera overlap distance and FoV based on changing traffic distribution and conditions in the monitored area. Simulation results show that the proposed camera distribution algorithm outperforms existing methods in terms of the required number of cameras, coverage percentage, and traffic coverage.

While our proposed intelligent RL-based approach offers significant performance results in improving camera distribution, it is important to acknowledge certain limitations. The training phase of the RL agent can be computationally intensive, especially for large-scale environments. Additionally, scaling the approach to accommodate rapidly changing environments or large-scale deployments may require efficient optimization techniques and distributed computing. Also, adapting to rapidly changing environments, such as frequent object movement or network topology changes, may pose challenges to the RL agent's performance. Future research directions include handling heterogeneous camera networks, exploring more efficient RL algorithms, incorporating advanced techniques for handling dynamic environments, and investigating the integration of our approach with real-world scenarios.

Author Contributions: Conceptualization, I.T.A.; Methodology, I.T.A.; Software, L.A.-A. and M.A.; Validation, R.H. and A.A. (Alaa Alghazo); Formal analysis, I.T.A.; Investigation, M.F.A.-H. and A.A. (Ayoub Alsarhan); Resources, A.A. (Alaa Alghazo); Data curation, M.F.A.-H., L.A.-A. and M.A.; Writing—original draft, I.T.A.; Writing—review & editing, R.H., M.F.A.-H., A.A. (Alaa Alghazo), A.A. (Ayoub Alsarhan) and S.A.A.; Visualization, R.H.; Project administration, I.T.A.; Funding acquisition, S.A.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported and funded by Hashemite University. The authors also extend their appreciation to the Deanship of Scientific Research at Northern Border University, Arar, KSA for funding this research work through the project number NBU-FFR-2024-2119-01.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Please contact the corresponding author by email: eslam.malkawi@hu.edu.jo for any request regarding the setup or the code of our simulation work.

Acknowledgments: The authors extend their appreciation to the Deanship of Scientific Research at Northern Border University, Arar, KSA for funding this research work through the project number NBU-FFR-2024-2119-01.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analysis or interpretation of the data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Nižetić, S.; Šolić, P.; González-de-Artaza, D.L.d.I.; Patrono, L. Internet of Things (IoT): Opportunities, issues and challenges towards a smart and sustainable future. *J. Clean. Prod.* **2020**, *274*, 122877. [CrossRef] [PubMed]
2. Cui, Y.; Liu, F.; Jing, X.; Mu, J. Integrating sensing and communications for ubiquitous IoT: Applications, trends, and challenges. *IEEE Netw.* **2021**, *35*, 158–167. [CrossRef]
3. Almalkawi, I.T.; Al-Abed, L.; Al-Karaki, J.N.; Zapata, M.G. IoT-Based Surveillance Camera Distribution Using Triangle Geometry. In Proceedings of the 2023 9th International Conference on Control, Decision and Information Technologies (CoDIT), Rome, Italy, 3–6 July 2023; pp. 2193–2198.
4. Akhter, F.; Khadivizand, S.; Siddiquei, H.R.; Alahi, M.E.E.; Mukhopadhyay, S. IoT Enabled Intelligent Sensor Node for Smart City: Pedestrian Counting and Ambient Monitoring. *Sensors* **2019**, *19*, 3374. [CrossRef]
5. Watkins, C.J.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]

6. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
7. Andrew, A.M. *Reinforcement Learning: An Introduction by Richard S. Sutton and Andrew G. Barto, Adaptive Computation and Machine Learning Series*; MIT Press (Bradford Book): Cambridge, MA, USA, 1999.
8. Pan, X.; Liu, M.; Zhong, F.; Yang, Y.; Zhu, S.C.; Wang, Y. MATE: Benchmarking Multi-Agent Reinforcement Learning in Distributed Target Coverage Control. In *Advances in Neural Information Processing Systems*; Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., Oh, A., Eds.; MIT Press: Cambridge, MA, USA, 2022; Volume 35, pp. 27862–27879.
9. Su, Y.H.; Huang, K.; Hannaford, B. Multicamera 3D Viewpoint Adjustment for Robotic Surgery via Deep Reinforcement Learning. *J. Med. Robot. Res.* **2021**, *6*, 2140003. [CrossRef]
10. Jin, Z.; Wu, J.; Liu, A.; Zhang, W.A.; Yu, L. Policy-Based Deep Reinforcement Learning for Visual Servoing Control of Mobile Robots with Visibility Constraints. *IEEE Trans. Ind. Electron.* **2022**, *69*, 1898–1908. [CrossRef]
11. Chen, Y.; Tsukada, M.; Esaki, H. Reinforcement Learning Based Optimal Camera Placement for Depth Observation of Indoor Scenes. In Proceedings of the 2021 IEEE International Conference on Networking, Sensing and Control (ICNSC), Xiamen, China, 3–5 December 2021; Volume 1, pp. 1–6.
12. Alaei, M.; Barcelo-Ordinas, J.M. A hybrid cooperative design for energy-efficient surveillance in Wireless Multimedia Sensor Networks. In Proceedings of the 18th European Wireless Conference, Poznan, Poland, 18–20 April 2012; pp. 1–7.
13. Zhang, C.; Jia, Q.S. An occupancy distribution estimation method using the surveillance cameras in buildings. In Proceedings of the 13th IEEE Conference on Automation Science and Engineering (CASE), Xi'an, China, 20–23 August 2017; pp. 894–899.
14. Piciarelli, C.; Foresti, G.L. Drone Patrolling with Reinforcement Learning. In Proceedings of the 13th International Conference on Distributed Smart Cameras, New York, NY, USA, 9–11 September 2019; pp. 1–6.
15. Esmaeilzadeh, R.; Abbaspour, M. Optimum Temporal Coverage with Rotating Directional Sensors. *Wirel. Pers. Commun.* **2019**, *105*, 369–386. [CrossRef]
16. Bisagno, N.; Xamin, A.; De Natale, F.; Conci, N.; Rinner, B. Dynamic Camera Reconfiguration with Reinforcement Learning and Stochastic Methods for Crowd Surveillance. *Sensors* **2020**, *20*, 4691. [CrossRef] [PubMed]
17. Gonen, B.; Akkaya, K.; Senel, F. Efficient camera selection for maximized target coverage in underwater acoustic sensor networks. In Proceedings of the 2015 IEEE 40th Conference on Local Computer Networks (LCN), Clearwater Beach, FL, USA, 26–29 October 2015; pp. 470–473.
18. Munishwar, V.P.; Abu-Ghazaleh, N.B. Scalable Target Coverage in Smart Camera Networks. In Proceedings of the Fourth ACM/IEEE International Conference on Distributed Smart Cameras, New York, NY, USA, 31 August–4 September 2010; pp. 206–213.
19. Kheirkhah, M.M.; Khansari, M. Clustering wireless camera sensor networks based on overlapped region detection. In Proceedings of the 7'th International Symposium on Telecommunications (IST'2014), Tehran, Iran, 9–11 September 2014; pp. 712–719.
20. Pham, C. Low-Cost, Low-Power and Long-Range Image Sensor for Visual Surveillance. In Proceedings of the 2nd Workshop on Experiences in the Design and Implementation of Smart Objects, New York, NY, USA, 3 October 2016; pp. 35–40.
21. Rab, R.; Jahan, M.; Mridha, M.S.H.; Olee, A.; Nusrat, S.; Rahman, A. On Efficient Selection and Orientation of Directional Sensors in Visual Sensor Networks. In Proceedings of the 2019 International Conference on Electrical, Computer and Communication Engineering (ECCE), Cox's Bazar, Bangladesh, 7–9 February 2019; pp. 1–6.
22. Altahir, A.A.; Asirvadam, V.S.; Hamid, N.H.B.; Sebastian, P.; Saad, N.B.; Ibrahim, R.B.; Dass, S.C. Optimizing Visual Surveillance Sensor Coverage Using Dynamic Programming. *IEEE Sens. J.* **2017**, *17*, 3398–3405. [CrossRef]
23. Esterle, L. Centralised, Decentralised, and Self-Organised Coverage Maximisation in Smart Camera Networks. In Proceedings of the IEEE 11th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), Tucson, AZ, USA, 18–22 September 2017; pp. 1–10.
24. Han, Z.; Li, S.; Cui, C.; Song, H.; Kong, Y.; Qin, F. Camera planning for area surveillance: A new method for coverage inference and optimization using Location-based Service data. *Comput. Environ. Urban Syst.* **2019**, *78*, 101396. [CrossRef]
25. Jiang, X.; Fang, Z.; Xiong, N.N.; Gao, Y.; Huang, B.; Zhang, J.; Yu, L.; Harrington, P. Data Fusion-Based Multi-Object Tracking for Unconstrained Visual Sensor Networks. *IEEE Access* **2018**, *6*, 13716–13728. [CrossRef]
26. Kritter, J.; Brévilliers, M.; Lepagnot, J.; Idoumghar, L. On the real-world applicability of state-of-the-art algorithms for the optimal camera placement problem. In Proceedings of the 6th International Conference on Control, Decision and Information Technologies (CoDIT), Paris, France, 23–26 April 2019; pp. 1103–1108.
27. Liu, C.H.; Ma, X.; Gao, X.; Tang, J. Distributed Energy-Efficient Multi-UAV Navigation for Long-Term Communication Coverage by Deep Reinforcement Learning. *IEEE Trans. Mob. Comput.* **2019**, *19*, 1274–1285. [CrossRef]
28. Zannat, H.; Akter, T.; Tasnim, M.; Rahman, A. The coverage problem in visual sensor networks: A target oriented approach. *J. Netw. Comput. Appl.* 2016, *75, 1–15.* [CrossRef]
29. Guzmán, J.A.; Pizarro, G.; Núñez, F. A reinforcement learning-based distributed control scheme for cooperative intersection traffic control. *IEEE Access*, 2023, *11*, 57037–57045.

30. Awaisi, K.S.; Abbas, A.; Khattak, H.A.; Ahmad, A.; Ali, M.; Khalid, A. Deep reinforcement learning approach towards a smart parking architecture. *Clust. Comput.* **2023**, *26*, 255–266. [CrossRef]
31. Wang, Y.; Peng, T.; Wang, W.; Luo, M. High-efficient view planning for surface inspection based on parallel deep reinforcement learning. *Adv. Eng. Inform.* **2023**, *55*, 101849. [CrossRef]
32. Ze, Y.; Hansen, N.; Chen, Y.; Jain, M.; Wang, X. Visual Reinforcement Learning With Self-Supervised 3D Representations. *IEEE Robot. Autom. Lett.* **2023**, *8*, 2890–2897. [CrossRef]
33. Ou, Q.; Xie, Q.; Chen, F.; Peng, J.; Xiong, B. Reinforcement learning-based calibration method for cameras with large FOV. *Measurement* **2022**, *202*, 111732. [CrossRef]
34. Almalkawi, I.T.; Guerrero Zapata, M.; Al-Karaki, J.N.; Morillo-Pozo, J. Wireless Multimedia Sensor Networks: Current Trends and Future Directions. *Sensors* **2010**, *10*, 6662–6717. [CrossRef] [PubMed]
35. TurtleBot 2. Available online: http://wiki.ros.org/Robots/TurtleBot (accessed on 19 April 2023).
36. Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y. ROS: An open-source Robot Operating System. In Proceedings of the ICRA Workshop on Open Source Software, Kobe, Japan, 12–17 May 2009, Volume 3.
37. Elhoseny, M. Multi-object Detection and Tracking (MODT) Machine Learning Model for Real-Time Video Surveillance Systems. *Circuits Syst Signal Process* **2020**, *39*, 611–630. [CrossRef]
38. Yoo, H.; Kim, B.; Kim, J.W.; Lee, J.H. Reinforcement learning based optimal control of batch processes using Monte-Carlo deep deterministic policy gradient with phase segmentation. *Comput. Chem. Eng.* **2021**, *144*, 107133. [CrossRef]