This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2025.3602213

DARWIN: Digital Twin Assisted Robot Navigation and WIreless Network Management

Batool Salehi*, Debashri Roy†, Mark Eisen‡, Amit Baxi‡, Dave Cavalcanti‡, and Kaushik Chowdhury§
*Northeastern University, †University of Texas Arlington, ‡Intel Corporation, and §University of Texas Austin
Email: *bsalehihikouei@ece.neu.edu, †debashri.roy@uta.edu, ‡mark.eisen@ieee.org
‡{amit.s.baxi, dave.cavalcanti}@intel.com, §kaushik@utexas.edu

Abstract—Automated warehouses involve robots that move across the floor, avoiding obstacles while remaining connected via an access point (AP) to a central controller that instructs the robots. The complex propagation environment and presence of metallic surfaces results in spotty coverage, which changes over time as the location of stored products and machinery changes. Thus, maintaining an assured connectivity to APs while performing navigation is a challenge, although it is needed to relay local sensor data from the robots to the controller and receive directions from the latter. DARWIN, involves creating a digital twin of the warehouse for training the robots by jointly optimizing the navigation and avoiding wireless dead-spots. DARWIN has three key capabilities: First, it captures the features of both physical and RF environments in the digital world. Second, it allows real-time updating of the digital twin if significant disparity is detected compared to the physical environment. Finally, it includes a reinforcement learning algorithm that jointly optimizes navigation and network resource management, while accounting for handover and outage. We validate DARWIN on an emulation environment consisting of Robot Operating System and Gazebo platforms along with real-world RF measurements. Results reveal that DARWIN reduces the number of steps by 43% compared to choosing the closest AP, while detecting environmental changes with maximum 96% accuracy to maintain a high-fidelity digital twin.

Index Terms—Deep reinforcement learning, digital twin, network resource management, robot navigation.

1 Introduction

The ready access to data, enhanced communications technology and networked robotics are poised to usher in a transformation of the manufacturing sector. For example, an automated warehouse uses robotic systems to lower operation costs, improve warehouse productivity, and eliminate safety risks [1]. In many instances, such robots utilize Artificial Intelligence (AI) algorithms to navigate through the warehouse floor, while safely avoiding obstacles [2]. In this work, we assume robots are passive and are instructed by a central controller [3], which we refer to as an *autonomous edge*. The latter receives sensor information from the robots using an uplink channel, which is then provided as input to the AI algorithm to generate movement instructions, linear and angular velocity in our case. As a result, the performance of a network of robots depends on the ability of the wireless links to deliver the sensor data in uplink and return instruction to the robots in downlink [4] by associating with the appropriate Access Points (AP).

In this paper, we address these challenges by constructing a digital twin of the physical warehouse environment and training AI models to obtain a path planning policy that minimizes the number of robot steps towards its target, while maintaining connectivity with the autonomous edge. A digital twin is a true-to-reality emulation of the real world [5], [6]. This is especially useful when trial-and-error methods are infeasible due to possible adverse impact to life or property, such as in the case of Autonomous Robot Navigation (ARN) systems [7], [8]. Different from digital replicas (i.e., simulation only), our approach to using digital twin considers the impact of (a) near real-time response and (b) bidirectional interaction with the physical world.

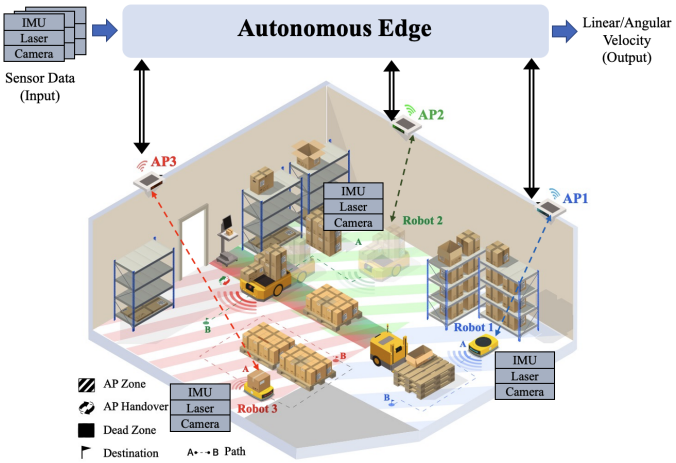


Fig. 1: Overview of the computation and communication in DARWIN. The robots transmit local sensor information to the autonomous edge in the uplink and receive instructions in the downlink. DARWIN adapts to the changes in the environment, handles AP handover and outage, and jointly optimizes navigation and network management.

1.1 Challenges in Connected Robot Navigation

Fig. 1 shows our scenario of interest where multiple robots navigate to their targets with minimum number of steps, while safely avoiding obstacles and maintaining the connection with the autonomous edge. Several key challenges remain to be addressed to realize this vision.

• **C1. Dynamic Environments:** One approach to path planning involves training a Reinforcement Learning (RL) agent that explores the environment over multiple trials. To avoid

human or material loss, a digital replica of the real world can be utilized to generate training scenarios. However, this digital replica must be continuously updated to mimic the dynamic nature of the real world. For example, in Fig. 1, an unanticipated obstacle in the form of a machine suddenly blocks the path of Robot 1. In this case, the autonomous edge must respond to the changes in the environment and update the path planning policy to avoid collision. Formally, a collision occurs when there is physical contact between the robot and another object in its environment.

Moreover, while some of the state-of-the-art techniques such as Integrated Sensing and Communication (ISAC) [9] systems offer valuable capabilities in certain scenarios, our digital twin approach provides a more versatile and comprehensive solution for complex factory environments with challenging propagation characteristics, integrating diverse sensor data and network conditions to enable robust navigation across various frequency bands and dynamic settings.

• **C2. Real-time Generation of the Digital Twin:** Creating a digital twin is time consuming. For example, state-of-the-art Simultaneous Localization And Mapping (SLAM) [10] algorithm requires many closed loop trials of exploring the real world and also high quality sensors, which are not always available in robots. Ideally, digital twins should utilize instantaneous sensor data to create or update digital instances in near real-time.

• **C3. Cost of Re-training the Policy:** In a dynamic warehouse environments, the path planning policy must be updated to avoid the collision with the obstacles. However, updating path planning policy at the autonomous edge is computationally expensive. As a result, triggering such an update must be avoided as long as no severe performance degradation is observed.

• **C4. Modeling Handover and Outage:** Since there are typically multiple APs in the environment, there could be overlapping coverage areas for the robot (see Robot 2 in Fig. 1). Thus, the AP allocation policy must be designed to enable the best handover strategy. During handover, the robot loses connections with the APs (i.e., the autonomous edges consequently) for a fraction of time, which might affect the navigation performance. On the other hand, we need to model the outage scenarios where the signal to noise ratio (SNR) of the received signal is not enough to maintain connectivity with the autonomous edge and try to avoid them as much as possible.

• **C5. Dependency of Navigation on Network Management:** In wireless dead-zones, the robots may not be able to receive directives from the autonomous edge. This is a problem since navigation and network management are tightly coupled. For example, Robot 3 in Fig. 1 must maneuver around the obstacle to reach to the designated target. It may do so by turning around the obstacle in two possible directions. However, following the right path traps the robot in a wireless dead-zone, which results in connection loss between the autonomous edge and the robot. As a result, the robot should ideally select the left path, although it results in a higher number of steps.

1.2 DARWIN: Digital Twin at Autonomous Edge

In this paper, we propose DARWIN to mitigate the challenges in robotic navigation using interaction between the real world and a digital twin that is deployed at the autonomous edge. First, our proposed method, continuously monitors the real world, generates the most updated twin, and updates the path planning policy, if required, addressing (C1). Second, given the instantaneous local sensor data from robots, we propose an algorithm to create the latest digital twin of the real world in near real-time (C2). Third, we compare consecutive digital twin instances and only trigger the path planning update if a drastic change occurs in the environment. This avoids unnecessary computation cost at the autonomous edge (C3). Fourth, we model the wireless propagation pattern to reflect the handover and outage cost in the reinforcement learning algorithm (C4). Finally, we introduce a path planning policy that jointly optimizes navigation and network management to minimize the number of steps for the robot, while ensuring that the robots are maintaining the connection with the autonomous edge and not trapped in wireless dead-zones (C5).

1.3 Summary of Contributions

Formally, our contributions are as follows:

1) We propose a novel approach for constructing a high-fidelity twin of the real world at the autonomous edge. We use instantaneous laser and camera sensor data acquired by robots to identify the regions of obstacles and map them to the digital twin. Our approach demonstrates up to 96% and 61% accuracy in the detection of new objects with laser and camera sensors, respectively, and detection time of $\sim 1ms$.
2) Given the cost of retraining the RL policy, we design a tolerance threshold to prevent unnecessarily policy updates. Our algorithm compares consecutive instances of the digital twin, and only triggers policy update when a drastic change in the environment is detected.
3) We run a real world RF measurement campaign using the Turtlebot and multiple software defined radios as APs to generate a lab environment that closely mimics the warehouse floor setup with multiple obstacles and reflecting materials.
4) We propose a RL algorithm that jointly optimizes navigation and network management to account for both presence of obstacles and quality of connection with the autonomous edge. We use the state-of-the-art Deep Deterministic Policy Gradient (DDPG) [11] as the reinforcement learning algorithm. Moreover, we validate our solution on the real world SNR measurements with penalties associated with AP handover and outage. Our observation indicates that by carefully allocating the robots to the access points, the average number of steps decreases by 43%, compared to always choosing the nearest AP.
5) In summary, we propose the first-of-its-kind framework for creating an interactive digital twin in emulation environment consisting of Robot Operating System and Gazebo platforms along with real-world RF measurements that maps a real world warehouse floor setup with multiple robots participating in a joint task through an autonomous edge.

2 Related Works

2.1 Autonomous Robot Navigation

Krell *et al.* [12] present a swarm intelligence method, namely Particle Swarm Optimization (PSO), to navigate in an unknown environment with static obstacles, while minimize the robot traveling distance. Similarly, Chen *et al.* [13] use a stochastic PSO with high exploration ability, that ensures smooth path planning. Almanza *et al.* [14] present an algorithm to continuously generate the occupancy map of the environment using ultrasonic sensor. The path planning is updated using potential field algorithm [15], accordingly. Hu *et al.* [16] adapt the DDGP algorithm [11] for robotic navigation and present a collision avoidance algorithm with deep reinforcement learning to navigate the robot towards the target. In the proposed approach, the instantaneous sensor data are used to predict the linear and angular velocity of the robots and a utility function is designed that takes into account the path cost and the target distance.

2.2 Network Resource Management

Elbamby *et al.* [17] discuss edge networking services as key enablers to achieve low-latency and high-reliability networking in mission-critical applications such as Virtual Reality (VR), Vehicle-to-Everything (V2X), and edge Artificial Intelligence (AI). Ahmadi *et al.* [18] propose a method for mobility management and handover at the edge network for a seamless connection by predicting the future locations of a moving receiver on a vehicle. Ho *et al.* [19] propose a deep reinforcement learning algorithm for joint server selection, cooperative off-loading, and handover in multi-access edge wireless networks in dynamic environment. Regarding the robotic systems, Mohanti *et al.* [20] exploit DDPG algorithm for robot navigation in a static environment. The authors then propose a strategy for network management given the designated path to reach to the target. Eisen *et al.* [21] develop a communications-control co-design paradigm for adapting the QoS and control inputs provided to robotic agents controlled via a wireless edge.

Novelty of DARWIN: In summary, the state-of-the-art work on navigation and network management has the following shortcomings. First, they are only validated on static environments ([12], [13], [16], [20], [21]) and fail to adapt to the changes in the environment. Second, the wireless aspect is ignored or glossed over ([12], [13], [14], [16]). Third, only one of the navigation ([12], [13], [14], [16]) or network management ([17], [18], [19]) aspects are studied or, when considered together, solved as disjoint task or for static environments [20], [21]. Tab. 1 highlights the unique features of DARWIN compared to the state-of-the-art for navigation and network resource management.

Paper	Navigation	Network Management	Dynamic Environment	Joint Optimization
Krell *et al.* [12]	✓	✗	✗	✗
Chen *et al.* [13]	✓	✗	✗	✗
Almanza *et al.* [14]	✓	✗	✓	✗
Hu *et al.* [16]	✓	✗	✗	✗
Elbamby *et al.* [17]	✗	✓		✗
Ahmadi *et al.* [18]	✗	✓		✗
Ho *et al.* [19]	✗	✓		✗
Mohanti *et al.* [20]	✓	✓	✗	✗
Eisen *et al.* [21]	✗	✓	✗	✓
DARWIN	✓	✓	✓	✓

TABLE 1: Advantages of DARWIN over the state-of-the-art for navigation and network resource management. The filled cell denote that the target feature is not applicable.

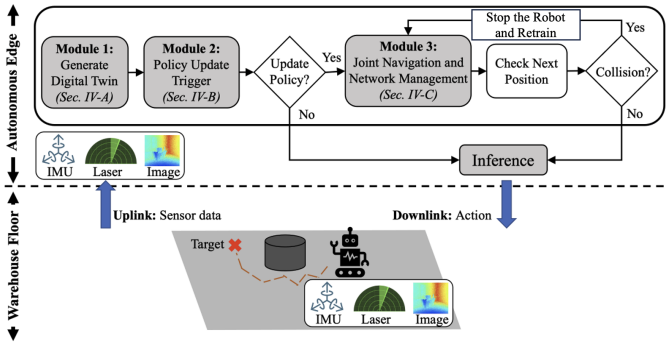


Fig. 2: Overview of the proposed system architecture. DARWIN continuously monitors the environment. In *Module 1*, we use the instantaneous sensor information to generate the digital twin. If a significant change in environment is detected (*Module 2*), we update the path planning policy by jointly optimizing the navigation and network resource management (*Module 3*).

3 System Model

In this section, we first formally state the problem. We then present the system architecture of DARWIN.

3.1 Problem Statement

We consider $\{R_i\}_{i=1}^{K}$ robots and $\{A_j\}_{j=1}^{M}$ access points in the environment with O obstacles. In our setting, the robots are equipped with Inertial Measurement Unit (IMU), laser, and camera sensors. Moreover, they are passive and communicate the sensor information in the uplink to the autonomous edge using the APs. The autonomous edge runs a decision algorithm and sends the instructions in the downlink, linear and angular velocity, and corresponding access point for each robot (see Fig. 1). Crucially, the selection of the corresponding access point for each robot at time step t is not a separate, disjoint allocation task, but rather an integral part of the reinforcement learning algorithm's action space, jointly optimized alongside the robot's linear and angular velocities. This ensures that network connectivity considerations are inherently coupled with navigation decisions, preventing any sub-optimal, disconnected planning. The path planning policy is targeting to minimize the number of steps reaching to the targets, while avoiding the obstacles and maintaining connection with the autonomous edge.

3.2 System Architecture in DARWIN

An overall view of our framework is shown in Fig. 2 and consists of three main modules as follows:

- **Real-time Digital Twin Generation** *(Module 1)*: The autonomous edge receives the instantaneous local sensor data from the robots in the uplink. We present a detection algorithm that exploits laser and camera data to detect the obstacles in close and distant

ranges, respectively. Our method generates the digital twin in near real-time, unlike SLAM that requires offline exploration of the environment (details in Sec. 4.1).

- **Policy Update Trigger** *(Module 2)*: We design an algorithm to compare the different instances of the digital twins and update the path planning policy, if a major change in the environment is observed. As a result, we avoid unnecessary updates to ensure computation efficiency at the autonomous edge (details in Sec. 4.2).
- **Joint Navigation and Network Management** *(Module 3)*: We design a deep reinforcement learning algorithm tailored to jointly optimize navigation and network management. Apart for navigation, our reward function takes into account the SNR of the received signal and handover cost. Moreover, we model wireless link outage, when the robots are not able to send the sensor data in the uplink or receive the actions in downlink, due to low link quality (details in Sec. 4.3). Such wireless link outage is modeled by incorporating penalties associated with outage events directly into the reinforcement learning reward function, compelling the policy to actively avoid scenarios leading to connection loss and thereby ensuring robust connectivity as part of the overall optimization.

Symbol	Type	Description
$\{R_i\}_{i=1}^{K}$	Set	Set of all robots
K	Integer	Number of robots, $K = \lvert\{R_i\}_{i=1}^{K}\rvert$
$\{A_j\}_{j=1}^{M}$	Set	Set of all Access Points (APs)
M	Integer	Number of Access Points, $M = \lvert\{A_j\}_{j=1}^{M}\rvert$
$\mathcal{O}$	Function	Big O Notation
O	Integer	Number of obstacles
$\mathcal{S}$	Space	State space for the reinforcement learning algorithm
s_t	Vector	State vector at time step t
a_t	Vector	Action vector at time step t
lv_t^i	Scalar	Linear velocity of robot i at time t (e.g., $lv_t^i \in [0, v_{max}]$)
av_t^i	Scalar	Angular velocity of robot i at time t (e.g., $av_t^i \in [\alpha_{min}, \alpha_{max}]$)
ρ_t^i	Integer	Index of the associated Access Point for robot i at time t ($\rho_t^i \in \{1, \ldots, M\}$)
$f_R^\theta(.)$	Function	Overall reward function
$f_{NV}^\theta(.)$	Function	Navigation reward component
$f_{NM}^\theta(.)$	Function	Network management reward component
ω	Scalar	Weighting factor for the network management reward component
κ	Scalar	Control parameter for handover management

TABLE 2: List of important notations.

4 Detailed Description of DARWIN Design

In this section, we present our design for three modules in the DARWIN framework (see Fig. 2). The list of notations is presented in Tab. 2.

4.1 Module 1: Real-Time Digital Twin Generation

Our proposed system architecture continuously monitors the environment to detect the changes and update the path planning policy, if required. As a result, it is important to generate the latest digital twin in near real-time, in order to provide rapid response from the autonomous edge. We propose algorithms to exploit the *laser* (details in Sec. 4.1.1) and *camera* (details in Sec. 4.1.2) sensors, acquired locally by the robots, to generate the digital twin. We assume that the general boundaries of the environment, such as walls and static obstacles are known, pre-generated by SLAM, for example. We use the *instantaneous* robot sensor information to detect the dynamic obstacles and update the digital twin, accordingly. For simplicity, in this section, we present our method for rectangular obstacles only to establish a baseline for our algorithm's performance under controlled conditions [22]. To that end, we focus on estimating the width and length of the obstacles, as it has the main effect on the performance. Nevertheless, our methodology can be extended as a future study to estimate the more complex shapes and the height of diverse obstacles in complex environments.

4.1.1 Detection using Laser

The laser sensor scans the surrounding by emitting laser light at different angles and computing the distance. In our setting, we consider a Laser Distance Sensor (LDS), which is a 2D laser capable of sensing 360degrees with the resolution of one degree (see Fig. 3a). The laser recordings include the scanning angles and associated distance of the objects from the sensor, within the detection range ($3.5\,m$ in our setting). We note that the laser data resembles a polar coordinates system. Fig. 3b shows a laser sample for the scenario shown in Fig. 3a with two symmetric rectangular obstacles. In order to generate the digital twin, we convert the laser data to the Cartesian coordinates system as:

$$x = rcos(\theta),\ y = rsin(\theta), \tag{1}$$

where r denotes the distance from the detected obstacle at angle θ. Fig. 3c shows the output of this transformation. The detected objects by laser are computed based on the current coordinates of the robot. Thus, we use the IMU sensor information to obtain the location of the robot R_i as (X_{R_i}, Y_{R_i}). Hence, for 360 degree scanning at robot R_i, the generated laser vector is defined as: $\mathfrak{L}^i = \{(X_{R_i} + r_1 cos(\theta_1), Y_{R_i} + r_1 sin(\theta_1)), \cdots, (X_{R_i} + r_{360} cos(\theta_{360}), Y_{R_i} + r_{360} sin(\theta_{360}))\}$. We then compute the shape (d_1, d_2) and centroid (c_x, c_y) of the detected objects by computing the difference and average of the minimum and maximum along each axis, respectively. Formally denoted as:

$$d_1 = \max_x \mathfrak{L}^i - \min_x \mathfrak{L}^i$$
$$d_2 = \max_y \mathfrak{L}^i - \min_y \mathfrak{L}^i$$
$$c_x = \frac{1}{2}(\max_x \mathfrak{L}^i + \min_x \mathfrak{L}^i)$$
$$c_y = \frac{1}{2}(\max_y \mathfrak{L}^i + \min_y \mathfrak{L}^i)$$

The overall laser processing pipeline is presented in Fig. 4.

4.1.2 Detection using Camera

The accuracy of the laser data degrades in far distances, where the laser light reflections cannot reach back to the

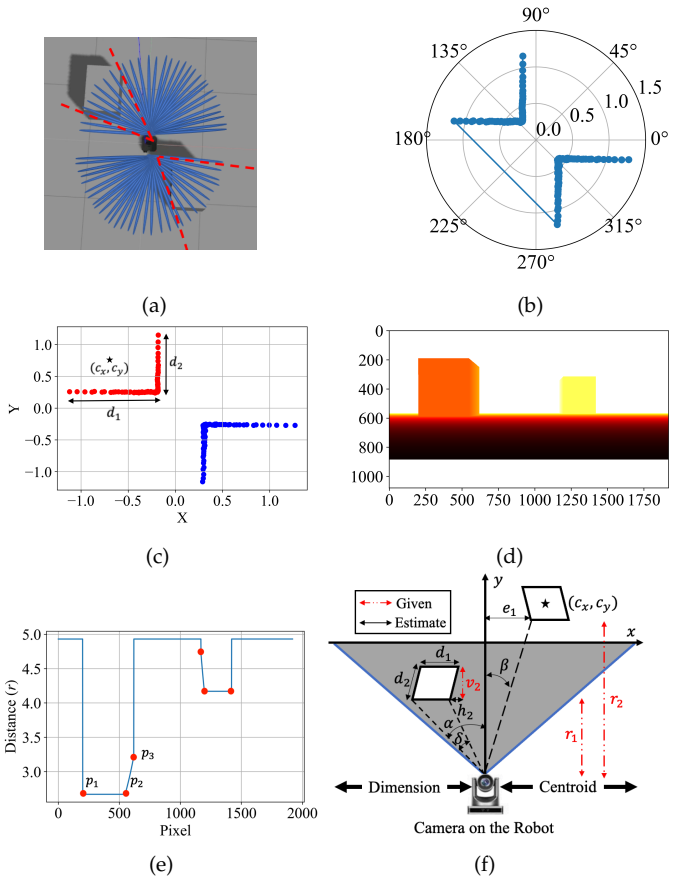


Fig. 3: (a) Scanning range of laser; (b) laser data represented in polar coordinates; (c) laser data represented in Cartesian coordinates; (d) A camera image sample; (e) Camera image sample after computing the maximum value of the pixels, horizontally; (f) Proposed detection algorithm using camera sensor, estimating dimension (left) and centroid (right).

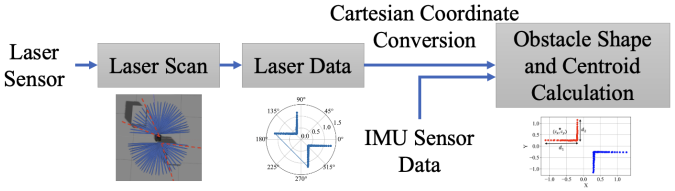


Fig. 4: Overview of the laser processing pipeline.

sensor. In this case, using the camera sensor is an alternative solution to monitor the changes in the environment in the state-of-the-art [23], [24] . However, the camera sensor does not include the detection angles, which makes it challenging to extract the precise location of the obstacles. Moreover, the flattening feature of camera sensor affects the accuracy of estimating the shape and centroid of the obstacles. To address these challenges, we propose to incorporate the Field of View (FoV) of the camera for our detection algorithm. In our setting, we consider an Intel stereovision 3D imaging RealSense R200 camera [25], with FoV of 1.3439 radian ($\sim$ 77°). The RealSense R200 is a depth camera, capturing images with height (H) and width (W) of 1920 and 1080 pixels, respectively, and maximum scanning range of $100m$.

Consider $\mathbf{C} = [c_{i,j}]$ as the matrix representing a camera sensor sample, where $1 \leq i \leq H, 1 \leq j \leq W$. Each element $c_{i,j}$ denotes the distance of the objects from the camera. In the first step, we compute the maximum value of the pixels over matrix $\mathbf{C}$ horizontally and generate array $\mathcal{C}$ defined as: $\mathcal{C} = [\underset{i}{max}(c_{i,j})]$. Fig. 3d and Fig. 3e show an example of image sensor data and array $\mathcal{C}$ after computing the maximum horizontally, respectively. This process is equivalent to an upper view of the camera sample, where the height is discarded. We then identify the number of pixels associated to each of the edges (p_1 and p_2 for perpendicular edge in Fig. 3e for example). We use the number of pixels and FoV of the camera to estimate the dimensions (left in Fig. 3f) and centriods (right in Fig. 3f) as follows.

Detecting Dimension: We use the number of pixels associated to each edge to compute the angles δ and α as:

$$\delta = \frac{p_2 - p_1}{W} \times FoV, \; \alpha = \frac{p_3 - p_1}{W} \times FoV. \tag{2}$$

We then compute the perpendicular dimension as:

$$d_1 = 2\, r_1\, [tan(\frac{\alpha}{2}) - tan(\frac{\delta}{2})] \tag{3}$$

In order to estimate the oblique dimension (between p_2 and p_3), we use the Pythagorean theorem. We use a similar analysis as above to estimate the horizontal distance (h_2). On the other hand, the vertical distance of the second edge (v_2) is given by the camera data and computed by comparing the distance at pixel p_3 and p_2 (see Fig. 3e). As a result, we compute the second dimension as $d_2 = \sqrt{h_2^2 + v_2^2}$.

Detecting Centroid: For detecting the centroids (c_x, c_y), we compute the perpendicular distance of the object from the origin (e_1 in Fig. 3f). In this regard, we first estimate the angle β by using the number of pixels similar to Eq. 2. We then compute e_1 as: $e_1 = r_2\ tan(\beta)$. On the other hand, given the dimension estimation from above (d_1, d_2) and robot coordinates (X_{R_i}, Y_{R_i}), the centroid along each axis is computed as:

$$c_x = e_1 - X_{R_i} + \frac{d_1}{2}, \;\; c_y = r_2 + Y_{R_i} + \frac{d_2}{2}, \tag{4}$$

where r_2 denotes the distance of the obstacles from the camera sensor (see Fig. 3f).

4.1.3 *Laser versus Camera and Detection Time*

The camera sensor does not provide a 360° detection angle and has a limited Field of View (FoV). As a result, the obstacles are not captured with camera sensor, when the robot is close to them or pointing to another direction. However, it provides a higher detection range. The laser sensor, on the other hand, has a 360° angle; however, it has lower detection range.

Strategic Approach: Taken together, we propose to use the laser and camera sensor data at close and distant ranges, respectively. We provide quantitative values for detection range of each sensor in Sec. 5.3.

4.2 Module 2: Policy Update Trigger

The real-time digital twin generation module (presented in Sec. 4.1), continuously monitors the environment and generates the digital twin. We denote the different instances of the digital twin at each step as D_t. Each twin instance includes the centroids $(c_o = (c_x^o, c_y^o))$ and dimensions $(d_o = (d_1^o, d_2^o))$ of the detected objects for O obstacles in the environment, $D_t = \{(c_o, d_o)\}_{o=1}^{O}$. If there is no significant change in the environment, DARWIN framework runs the inference directly without updating the Module 3 policy. In particular, the local sensor data of the robots (received in uplink) are fed to the Module 3 RL model to obtain the linear and angular velocity, which are shared with the robots in the downlink. On the other hand, upon detection of a change in the environment, the path planning Module 3 policy θ must be updated to avoid the collisions. We note that accuracy of the proposed detection algorithm (see Sec. 4.1) degrades in far distances and is prone to errors like any other detection algorithm. On the other hand, minor changes in the environment, caused by a slight change in the position of the obstacles or errors with the detection algorithm, will not significantly affect the Module 3 path planning policy. The autonomous edges typically have multiple powerful computation resources; however, updating the policy is still expensive and must be triggered if it significantly improves the performance. As a result, we define a level of tolerance to avoid unnecessary updates of the Module 3 path planning policy. In particular, we compare the boundary of the detected objects at different instances of the D_t and compute the similarity between two consecutive digital twin instances as:

$$S = \frac{1}{O} \frac{\sum_{o=1}^{O} Area\{B_{t-1}^o \cap B_t^o\}}{\sum_{o=1}^{O} Area\{B_{t-1}^o \cup B_t^o\}}, \tag{5}$$

where B_t^o denotes the area of the object o at the digital twin instance D_t. The space complexity of storing multiple such instance of digital twin D_t depends on the complexity of the environment, i.e., the number of obstacles in the environment, which we derive as: $\mathcal{O}(D_t) = \mathcal{O}(O \times |(c_o, d_o)|) = \mathcal{O}(O \times 2) = \mathcal{O}(O)$.

In DARWIN, we trigger the Module 3 policy update when a significant change in similarity metric (Eq. 5) of the twin instances is observed. In such cases, DARWIN updates the Module 3 policy for one trial and runs the inference (see Fig. 2). As result, instead of waiting for the model to converge to final optimal, which results in interrupting the path planning by the autonomous edge, DARWIN updates the Module 3 policy as the robot explores the environment and improves the realization of the digital twin. Nevertheless, to ensure avoiding collisions, we consider a safety mechanism, when the robot gets extremely close to the obstacles. In this regard, we first compute the next location of the robot, given the linear and angular velocity of the robot. If the next movement results in a potential collision, the autonomous edge instructs the robots to stop and trains the path planning policy for a few more trials (see Fig. 2). We consider this safety limit to be $25\,cm$ in our experiments.

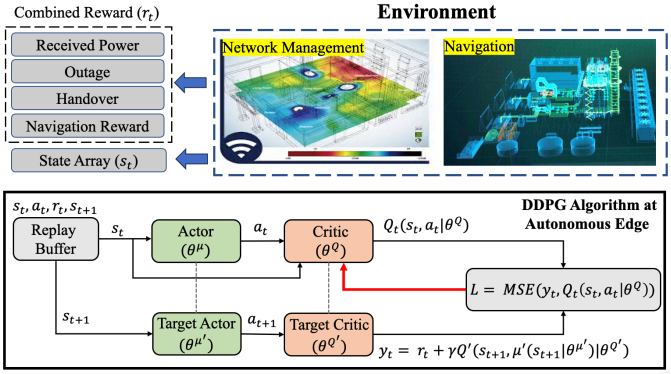


Fig. 5: Overview of the DDPG algorithm with Actor-Critic networks. We design the state and reward space to jointly optimize navigation and network management.

4.3 Module 3: Joint Navigation and Network Management

In this section, we present our proposed solution to jointly optimize navigation and network management in DARWIN. We propose a reinforcement learning algorithm as a solution which will work as a global planner for robot path at the autonomous edge. We discuss various components of the proposed reinforcement learning algorithm in the rest of the section.

4.3.1 State Space

We define the state s_t^i of the robot R_i at step t as: $s_t^i = \{\mathfrak{R}_t^i, \mathcal{C}_t^i, \mathfrak{L}_t^i, \mathfrak{d}_t^i, \mathfrak{A}_t^i, lv_{t-1}^i, av_{t-1}^i, p_t^{i,j}\}$. To construct the state array, we use the laser samples from $[-8°, 0]$ $(\mathfrak{R}_t^i)$ and $[0, 8°]$ $(\mathfrak{L}_t^i)$. Moreover, for the image samples, we first compute the maximum pixel value horizontally and upsample it by factor 32, resulting in 60 elements for $\mathcal{C}_t^i$ (details in Sec. 4.1.2). The remaining elements $\mathfrak{d}_t^i \in \mathbb{R}, \mathfrak{A}_t^i \in \mathbb{R}, \in \mathbb{R}, lv_{t-1} \in \mathbb{R}, av_{t-1} \in \mathbb{R}$, and $p_t^{i,j} \in \mathbb{R}$ denote the distance to the target, angular difference with the target, linear velocity and angular velocity at previous time step $(t-1)$, and normalized SNR of the received signal from the access point A_j, respectively. The overall state array has 81 elements, encapsulating both navigation and network management elements. The state array s_t for all the robots at time step t is: $s_t = \{(\mathfrak{R}_t^i, \mathcal{C}_t^i, \mathfrak{L}_t^i, \mathfrak{d}_t^i, \mathfrak{A}_t^i, lv_{t-1}^i, av_{t-1}^i, p_t^{i,j})\}_{i=1}^{K}$.

4.3.2 Action Space

Given the state array at the time t, the actor model (θ^μ, details in Sec. 4.3.4) predicts the next actions for all robots: $a_t = \theta^\mu(s_t)$. The action a_t includes the linear $(lv_t^i \in \mathbb{R})$ and angular $(av_t^i \in \mathbb{R})$ velocity, and associated access point $\varrho_t^i \in \mathbb{N}$ for all the robots $\{R_i\}_{i=1}^{K}$ at step t, stated as: $a_t = \{(lv_t^i, av_t^i, \varrho_t^i)\}_{i=1}^{K}$. This unified action space directly addresses the joint optimization objective, where the DDPG algorithm learns to simultaneously determine the robot's movement and its optimal AP association, ensuring a holistic navigation and network management strategy.

4.3.3 Reward Space

We define the overall reward as the combination of the navigation and network management rewards as:

$$f_{\mathcal{R}}^{\theta}(s_t, a_t) = f_{NV}^{\theta}(s_t, a_t) + \omega f_{NM}^{\theta}(\{p_t^j\}_{j=1}^{M}). \tag{6}$$

Symbol	Reward Description	Reward Value
Navigation		
$r^{i,t}_{d,d'}$	Relative distance to target	$(d^{'} - d) \times 420$
$r^{i,t}_{arr}$	Reaching target	100
$r^{i,t}_{obs}$	Distance to obstacle (μ^t)[m]	-200 (if $\mu^t < 0.2$) -80 (if $\mu^t < 0.4$) 0 (if $\mu^t > 0.4$)
$r^{i,t}_{L}$	Laser reward	$\sum\{\mathfrak{R}^i_t, \mathcal{C}^i_t, \mathfrak{L}^i_t\} - \lvert\{\mathfrak{R}^i_t, \mathcal{C}^i_t, \mathfrak{L}^i_t\}\rvert$
$r^{i,t}_{lv^i_t}$	Linear velocity [m/s]	-2 (if $lv^i_t \leq 0.2$) 0 (if $lv^i_t > 0.2$)
$r^{i,t}_{av^i_t}$	Angular velocity [rad/s]	0 (if $-0.8 < av^i_t < 0.8$) -1 (if $av^i_t \leq -0.8$ or $av^i_t \geq 0.8$)
Network Management		
$\Omega(\{p^{i,j}_t\}^M_{j=1})$	SNR	Eq. 8
$\Phi(\{p^{i,j}_t\}^M_{j=1})$	Handover	Eq. 9

TABLE 3: Robot reward details.

Here, s_t and a_t denote the state array and action for all the robots $\{R_i\}^K_{i=1}$ at step t, respectively. The reward is computed under policy θ, encapsulating both actor (θ^μ) and critic (θ^Q) polices (see. Fig. 5). Moreover, p^j_t denotes that SNR of the received signal from access point j for all the robots $\{R_i\}^K_{i=1}$ at step t. Finally, the control parameter ω adjusts the contribution of navigation and network management rewards in the overall reward. The navigation reward component, $f^\theta_{NV}(.)$, implicitly carries a weight of 1. This formulation ensures that the primary objective of minimizing robot steps towards its target (navigation) serves as the baseline for the overall reward. The weight ω is specifically applied to the network management component,$f^\theta_{NM}(.)$, allowing for a tunable balance between navigation efficiency and maintaining robust wireless connectivity. By adjusting ω, we can control the relative importance of network management, ranging from solely prioritizing navigation (when $\omega = 0$) to increasing the emphasis on optimizing AP associations and avoiding wireless dead-zones. This design choice provides a clear mechanism to manage the inherent trade-off between movement efficiency and network performance requirements.

Navigation Reward: We define the navigation reward as:

$$f^\theta_{NV}(s_t, a_t) = \sum_{i=1}^{K}(r^{i,t}_{d,d'} + r^{i,t}_{arr} + r^{i,t}_{obs} + r^{i,t}_{L} + r^{i,t}_{lv^i_t} + r^{i,t}_{av^i_t}). \quad (7)$$

The individual reward descriptions for each robot R_i are given in Tab. 3. Intuitively, the relative distance reward ($r^{i,t}_{d,d'}$) for each robot R_i is positive when the robot gets closer to the target and negative when the distance is increased over two consecutive steps. We set a high reward of 420 for each meter it moves towards the target and a high negative reward or *penalty* of 420 for each meter it moves away from the target, to ensure faster convergence to the target. The robot gets a high reward of 200 for reaching the target ($r^{i,t}_{obs}$). On the other hand, the distance to obstacles reward ($r^{i,t}_{obs}$) for each robot R_i punishes the robot when getting closer to the obstacles than a threshold. We give a high penalty of 200 when it comes within 0.2 meter of an obstacle, and relatively lower penalty of 80 when comes to 0.4 meter of an obstacle. The laser reward ($r^{i,t}_{L}$) for each robot R_i is the summation of the upsampled multimodal sensor data $\{\mathfrak{R}^i_t, \mathcal{C}^i_t, \mathfrak{L}^i_t\}$ minus the number of elements in this set. The linear and angular velocity rewards ($r^{i,t}_{lv^i_t}$ and $r^{i,t}_{av^i_t}$) for each robot R_i are designed such that to avoid low linear speed (resulting in increased number of steps) and high angular speed (sudden turning of the robot). We provide relatively lower penalty of -1 when the robot has a very low linear and angular velocity as well as very high angular velocity. The specific thresholds employed in this study were derived from empirical observations of robot navigation in indoor scenarios. While these thresholds have demonstrated validity across various indoor environments for the same application [26], however, it may require adjustment for different applications or significantly different environmental contexts.

Network Management Reward: The network management reward includes the SNR ($\Omega(\{p^j_t\}^M_{j=1})$) and handover ($\Phi(\{p^j_t\}^M_{j=1})$) rewards. The SNR reward is a function of access point allocation strategy Ω. We define two access point allocation strategies as follows:

$$\Omega(\{p^j_t\}^M_{j=1}) = \sum_{i=1}^{K} \begin{cases} \{p^{i,j}_t | \underset{1\leq j\leq M}{\arg\max}\ p^{i,j}_t\} & \text{if } \textit{Best AP} \\ \{p^{i,j}_t | \underset{1\leq j\leq M}{\arg\min}\ d^{i,j}\} & \text{if } \textit{Closest AP} \end{cases} \quad (8)$$

where $d^{i,j}$ denotes the distance of the robot R_i to the access point j. On the other hand, when handover happens, the robot loses connection for a fraction of time, resulting in the SNR of the received signal to be lesser than a threshold. As a result, in the case of handover, we punish the robots by a function of the maximum SNR of the received signal over all access point. Formally,

$$\Phi(\{p^j_t\}^M_{j=1}) = -\frac{\kappa}{2}\sum_{i=1}^{K}[max\ (\{p^{i,j}_t\}^M_{j=1})] \quad (9)$$

where κ is a control scalar. Overall, the network management rewards for is defined as:

$$f^\theta_{NM}(\{p^j_t\}^M_{j=1}) = \Omega(\{p^{i,j}_t\}^M_{j=1}) + \Phi(\{p^j_t\}^M_{j=1}) \quad (10)$$

The network management reward component, $f^\theta_{NM}(.)$, directly incorporates the costs associated with wireless link quality, including penalties for outage and handover. By penalizing outages (defined as insufficient SNR for reliable communication), the reinforcement learning agent is incentivized to choose navigation paths and and AP associations that proactively maintain strong connectivity and avoid dead-zones, thus ensuring the reliability of control and sensor data exchange.

Optimization Problem: Finally, the path planning policy θ is formulated as:

$$\underset{\theta}{\text{Maximize:}} \quad \mathbb{E}_t\Big[f^\theta_{\mathcal{R}}(s_t, a_t)\Big], \quad (11a)$$

$$\text{s.t} \quad a_t = \theta(s_t), \quad (11b)$$

$$\varrho^i_t \in \{A_j\}^M_{j=1}, \forall i \quad (11c)$$

$$\sum_{i=1}^{K}\sum_{t=1}^{\mathcal{T}} \varphi^i_t = 0. \quad (11d)$$

Here, s_t and a_t denote the state and action for all robots $\{R_i\}^K_{i=1}$ at step t. The action space includes linear velocity, angular velocity, and associated access point of robot:

$a_t = \{(lv_t^i, av_t^i, \varrho_t^i)\}_{i=1}^K$. The $f_{\mathcal{R}}^{\theta}(.)$ denotes a reward function, which is averaged over robots and time and is a function of state, action, as well as the allocated access point. Each robot R_i chooses the access point ϱ_t^i from access point set $\{A_j\}_{j=1}^M$ (as noted in Sec. 3.1) using allocation strategy Ω. Finally, φ_t^i is a Boolean predicate, with φ_t^i to be 1 if collision happens, and 0 otherwise, that ensures avoiding collisions.

4.3.4 Actor-Critic Networks in DDPG Algorithm

We use the state-of-the-art Deep Deterministic Policy Gradient (DDPG) [11], [20] algorithm to solve Eq. 11 and identify the optimum path planning policy. The DDPG algorithm is a deep reinforcement algorithm inspired by Deep Q-Learning [27] and is based on actor-critic neural networks as shown in Fig. 5.

In our actor-critic model, the actor network learns the optimal policy over time to generate the actions from the state space. Therefore, the optimal policy is determined by the parameters of the trained actor network. To develop the training method for policy θ which corresponds to solving the problem stated in Eq. 11b, we parameterize the actor network as θ^{μ} where $\theta = \theta^{\mu}$. The actor model (θ^{μ}) takes as input the state array (s_t) and outputs the action (a_t). Next, we formulate the actor network as, $a_t = \theta^{\mu}(s_t)$, where the s_t and a_t are state and action (details in Secs. 4.3.1 and 4.3.2) for all robots $\{R_i\}_{i=1}^K$at time slot t.

On the other hand, the critic network takes the state and action as input to estimate the expected reward $Q_t(s_t, a_t|\theta^Q)$, while following the policy θ^Q. Unlike standard Q-Learning, the DDPG algorithm employs two sets of actor-critic neural networks (see Fig. 5). The critic network takes both the state and the generated action from the actor as inputs and determines the quality of the generated action. The critic model is formulated as, $Q^t = \theta^Q(s_t, a_t)$, where Q^t is the Q value generated by the critic at time slot t evaluating the success of the generated action a_t, and critic network is parameterized with θ^Q. Here Q is a function in the DDPG algorithm, which generates the expected rewards for an action taken in a given state. Q-learning finds an optimal policy in the sense of maximizing the expected value of the total reward over any and all successive steps, starting from the current state [27].

Training: At the training phase, the agent interacts with the environment by predicting an action given the current state and stores the observations in a reply buffer. The observation includes the current state (s_t), action (a_t), reward (r_t), as well as the next state (s_{t+1}). The agent then samples a mini-batch from the replay buffer and uses the data to update the Actor-Critic network as follows. The critic model update is designed to minimize the difference between the loss (mean square error) of the original and target critic models. On the other hand, the actor model update is policy gradient based. Finally, the target Actor-Critic models are updated. Instead of updating the target network weights by directly copying them from the Actor-Critic network, the DDPG algorithm slowly updates the target networks through a process known as Soft Target [28] updates, which is shown to improve the stability of learning. Upon convergence, at the inference phase, the agents feed the state array to the actor network (θ^{μ}) and drive the actions. We use the state-of-the-art DDPG algorithm described above. However, we use state and reward functions that are tailored to jointly optimize navigation and network management.

4.3.5 Modeling Outage in DARWIN

In case of outage, the wireless coverage is not enough to deliver the sensor data in the uplink or receive the actions in the downlink, which results in losing the connection with the autonomous edge. We define the outage as SNR of the received signal being lower than a threshold. Note that the outage is a function of access point strategy (Eq. 8) as well. Formally for each robot R_i,

$$Outage = thresh(\Omega(\{p_t^{i,j}\}_{j=1}^M)) = \begin{cases} 1 & \text{if } \Omega(\{p_t^{i,j}\}_{j=1}^M) \geq \lambda \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

We model outage, i.e. $Outage = 1$, by instructing the robot to follow the previous action ($a_t^i = a_{t+1}^i$). As a result, the robot follows the latest action instructed by the autonomous edge until it reaches to a region, where the SNR of the received signal is higher than threshold λ and the connection with the autonomous edge is retrieved.

5 Evaluating DARWIN

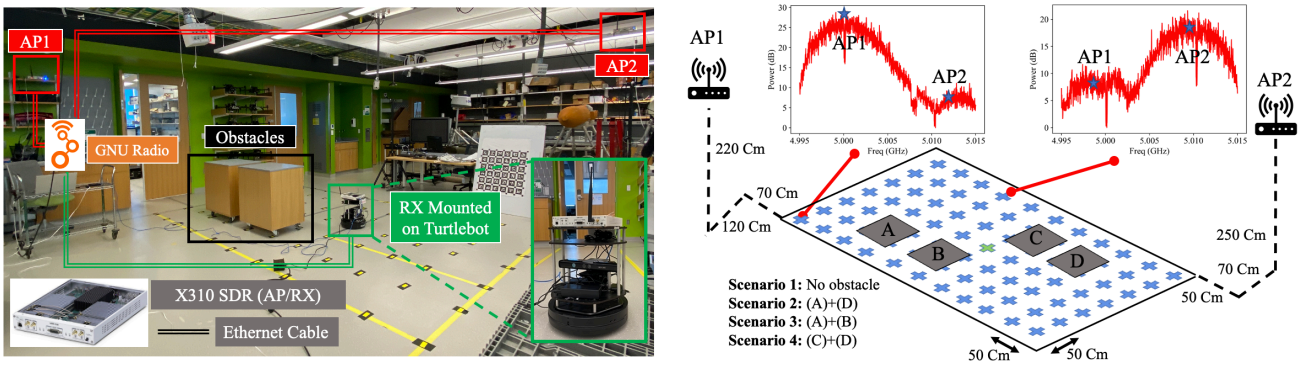
In this section, we present the implementation and evaluation of DARWIN, our proposed framework for joint optimization of navigation and network management. Our approach leverages a digital twin environment mimicking a warehouse floor environments to validate robotic navigation with network considerations. While the core principles of DARWIN are platform-agnostic, our current implementation utilizes Robot Operating System (ROS), Gazebo, and Python environments. Due to limitations in integrating radio modules within ROS, we conduct separate real-world RF data collection campaign for network measurements, which were then incorporated into our digital replica. This methodology allows us to demonstrate the feasibility of our joint optimization approach in a controlled, yet realistic setting.

5.1 RF Data Collection Campaign

We validate DARWIN with experimental data collected from a testbed that mimics a warehouse floor environments with Turtlebot robots and two USRP X310 software defined radios as access points.

5.1.1 Testbed Environment

We demonstrate DARWIN in a setting that resembles warehouse floor environments. We setup our testbed in a lab environment that covers a $5m$ by $3m$ area (see Fig. 6a). We consider discrete points with spacing of $50cm$ in the testbed area, resulting in 7 and 11 points along each axis. The testbed includes multiple reflective surfaces, including metal, glass and composite walls, that are typically present in factory environments as well.



(a) (b)

Fig. 6: (a) Testbed setup with two X310 radios as access points, a Turtlebot with a X310 radio mounted on it, and rectangular obstacles; (b) A diagram of the testbed representing the positioning of the access point, discrete points in the testbed, and location of the obstacles in four data collection scenarios.

5.1.2 Software Defined Radios

We use USRP X310 radios as both access point and receiver. In our setting, we consider two access points and one receiver mounted on the Turtlebot (see Fig. 6a). The access points are located at the boundary regions of the testbed at the height of $220\,cm$ and $250\,cm$ (the exact placement of the access points is shown in Fig. 6b). All the radios are connected to a central control unit that runs GNU Radio [29]. The access points emit frames generated via MATLAB WLAN System toolbox. The frames are modulated by Modulation and Coding Scheme (MCS) index of 5, which corresponds to 64 QAM modulation with coding rate of $2/3$. The first access point (AP1 in Fig. 6a) is configured to operate at $5GHz$ band, whereas the second access point is set at the center frequency of $5.01GHz$. We use these center frequencies to avoid interference by commercial devices that operate at ISM band ($2.4\ GHz$). Both access points have the bandwidth of $10MHz$. On the other hand, the receiver SDR samples the incoming signals at $15.36\ MS/s$ sampling rate and scans the spectrum from $4.995 - 5.015\ GHz$.

5.1.3 Data Collection Scenarios

In order to study the effect of different obstacle positioning, we consider four different scenarios in our testbed. Fig. 6b shows a diagram of our testbed, where the obstacle positioning are reported for each scenario. The obstacles are rectangular with dimensions $77\,cm \times 56\,cm \times 88\,cm$. First, we consider a simple scenario where there is no obstacle present in the testbed (*Scenario 1*). The second scenario (*Scenario 2*), describes a symmetric obstacle positioning in the testbed, where the obstacles are located at (A) and (D) markers in Fig. 6b. The third (*Scenario 3*) and fourth (*Scenario 4*) scenarios correspond to having extreme blockage for AP1 and AP2, respectively, where the obstacles are located at (A, B) and (C, D) markers.

5.1.4 Data Collection Strategy

We put the Turtlebot on each of the 77 discrete point in the testbed and sweep the spectrum from $4.995\ GHz - 5.01\ GHz$ with $10\ KHz$ step and record the SNR of the received signals from both the access points. The primary focus of this research is on wireless network management, specifically ensuring continuous connectivity for networked robots. To this end, our experimental methodology concentrates on measuring the downlink SNR of the received signals from both access points to the robot. Upon association with a specific access point, uplink communication is conducted in accordance with the corresponding protocol standards, although this is not explicitly measured in our current study. The plots in Fig. 6b show the spectrum over two points in the testbed for *Scenario 2* (symmetric obstacle positioning). We observe that for the point close to AP1, the SNR of the received signal from AP2 is much less than AP1, due to presence of an obstacle at marker A. On the other hand, the second point exhibits a better performance for AP2 as a result of the line of sight path from this access point. In order to account for the time-varying effect of wireless channel, we record the SNR for one minute (equivalent to 120 rounds of scanning) at each point for both access points. In Fig. 7, we demonstrate the average SNR for both access points and four scenarios in our testbed. Moreover, Tab. 4 summarizes the statistics of the SNR for both access points with respect to mean, standard deviation, minimum, and maximum over all discrete points in the testbed.

Scenario	Mean	Standard Deviation	Minimum	Maximum
1	(11.47, 12.02)	(0.05, 0.06)	(5.32, 6.18)	(19.94, 18.33)
2	(11.52, 11.90)	(0.06, 0.04)	(4.66, 4.74)	(23.16, 21.38)
3	(10.86, 11.59)	(0.05, 0.041)	(4.52, 6.29)	(24.15, 21.61)
4	(11.12, 11.97)	(0.06, 0.04)	(5.52, 5.17)	(19.78, 20.86)

TABLE 4: Statistics of the SNR of received signal (in dB) for two access point in the testbed. The first and second number in parenthesis are associated with AP1 and AP2, respectively.

5.2 Implementation Details and Evaluation Metrics

Fig. 8 demonstrates our methodology for implementing DARWIN that works with coordination between Gazebo [30], Robotic Operating System [31], and Keras 2.3.1 with Tensorflow backend. Using Gazebo, we create the digital twin for each of four scenarios described in Sec. 5.1.3. On the other hand, we import Turtlebots in ROS with IMU, laser, and camera sensors. The sensor information from ROS and measurements from data collection campaign (see Sec. 5.1) are fed to the DDPG algorithm to jointly optimize navigation and network management. The output of DDPG

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2025.3602213

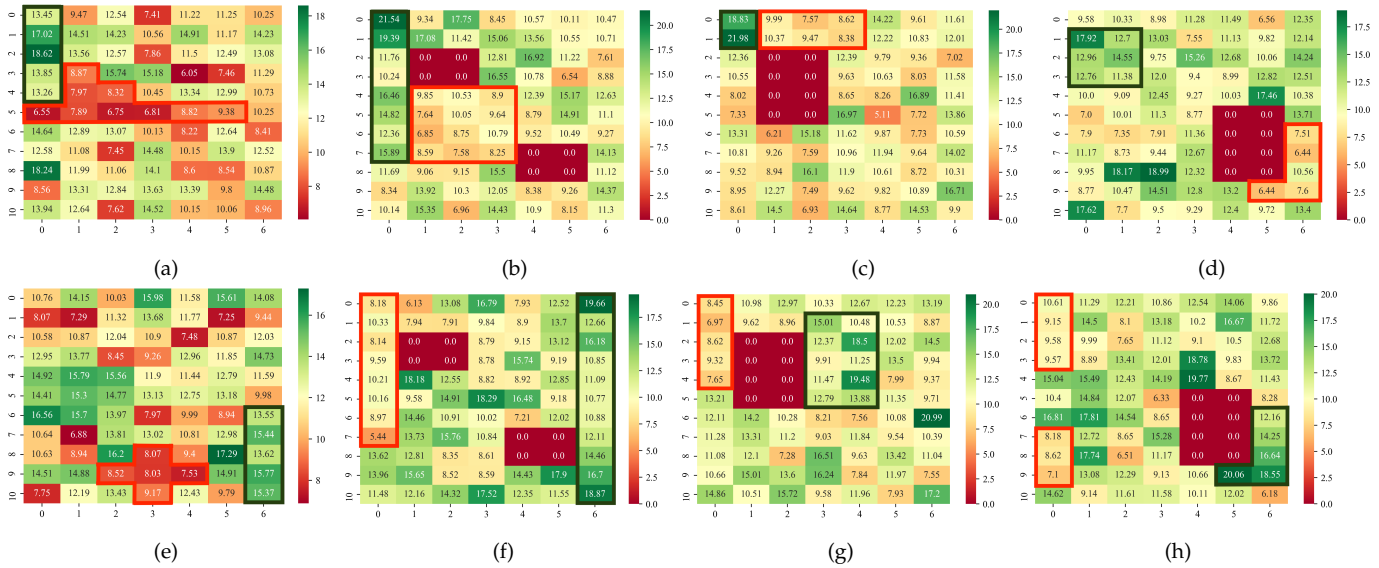


Fig. 7: Average SNR of the received signal at each discrete point in our testbed for AP1 (a, b, c, d) and AP2 (e, f, g, h) at each of the four data collection scenarios. The number in each square denotes the average SNR. The sold red squares show the position of obstacles in each scenario. Selective regions with low and high received SNR are marked with red and green lines.

algorithm (next robot's movement) are fed back to ROS and Gazebo using a Gym Wrapper [32] to update the location of the robot in the Gazebo (i.e. digital twin). The neural networks are trained using Adam optimizer [33] with learning rate of 0.0001. We consider 100 and 5 trials for training and testing, respectively, with maximum 2000 steps per trial.

We compare different methods and settings with respect to the average number of steps to reach to the target and average reward. Moreover, from network management aspect, we report the access point allocation ratio as well as outage ratio. The access point allocation ratio denotes the ratio of the times, where the robot is allocated to either of AP1 and AP2 in our setting. On the other hand, outage ratio characterizes the ratio of losing connection with the autonomous edge due to low SNR of received signal, while following the path to the target. Finally, for *Module 1*, we use Intersection over Union (IOU) metric to assess the accuracy of detection defined as:

$$IOU = \frac{Area\{B_p \cap B_{gt}\}}{Area\{B_p \cup B_{gt}\}}, \tag{13}$$

where B_{gt} and B_p correspond to the ground-truth and predicted areas by the proposed method in Sec. 4.1.

5.3 Performance of Module 1 in DARWIN

In this module, the goal is to generate the digital twin using the instantaneous sensor data, recorded locally at the robots. Our proposed approach in Sec. 4.1 exploits the camera and laser sensor information to detect the presence of the objects in the environment. In Fig. 9, we report the IOU (Eq. 13) for *Scenario 2* (symmetric obstacle positioning), as the robot moves in the environment. In this scenario, also shown in Fig. 3a, the first obstacle is located at quadrant 2 and the second one is placed at quadrant 4. In Fig. 9, the x-axis denotes the distance of the robot from the origin (starting point of Turtlebot). We report the IOU for using the laser and image sensor data for both obstacles.

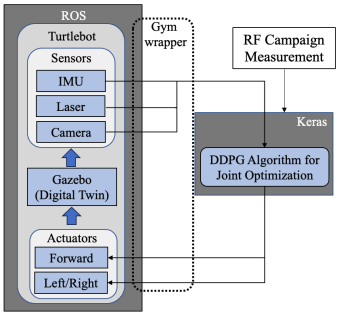


Fig. 8: We implement DARWIN using Gazebo, Robotic Operating System (ROS), and Keras with Tensorflow backend.

Broader Observations: We detail the main findings below:

• Laser provides high ($> 75\%$) detection accuracy in short ranges (distance less than $3.75m$), which gradually drops as the robot gets far from the obstacles. The IOU with laser exhibits fluctuation which are caused by the errors in the data, specially around the edges of the obstacles.

• We observe a significant drop in IOU to 0.0192 for obstacle 2, when the distance is $\sim 0.4315m$. We note that around this region, the robot is located perpendicular to the second obstacle, at coordinates of $(0.75, -1.25)$. As a result, only one angle of the obstacle is captured in the laser data. In this case, the output of detection algorithm is a slim rectangle, where the visible angle is accurately estimated, while the hidden angle is estimated to be smaller than the ground-truth. We observe that the IOU increases after the robot passes this region and both edges are captured in the laser data again.

• The IOU with camera sensor data is zero when the distance from the origin is less then $0.6568m$ for both obstacles. In this region, the obstacles are not captured in the images due to limited FoV of $77°$ for the camera sensor. However, we note as the robot gets far from the origin, the first and

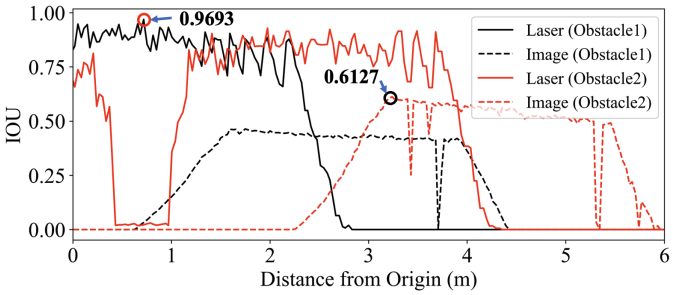


Fig. 9: Intersection over Union (Eq. 13) with laser and camera sensors for *scenario 2* (symmetric obstacle positioning), when the robot starts from the origin and follows a trajectory (details in Sec. 5.4, *validates Contribution 1*).

then the second obstacles gradually appear in the images. Once the entire obstacle is captured in the images, the IOU does not change, drastically.

• We note that the dimensions of obstacles are detected with high accuracy using the camera sensor and proposed method in Sec. 4.2. However, estimating the centroids is affected by perspective distortion and flattening effect. For example at the distance of $3.80\,m$ from the origin, the dimensions of the obstacles are detected as $(0.5662\,m, 0.6811\,m)$ and $(0.5492\,m, 0.6434\,m)$ for two obstacles. Comparing with the ground-truth dimensions of the obstacle, i.e. $(0.56\,m, 0.77\,m)$, we observe that the dimension estimations are close. However, due to errors in estimating the centroids, the IOU is calculated as 0.5822 and 0.4003 for first and second obstacles, respectively. As a result, we conclude detecting the dimensions is more robust than centroids with camera sensor.

• We estimate the detection time by computing the average time to run the algorithms presented in Sec. 4.1 as $\sim 1ms$ on a Dell Alienware desktop. Considering the availability of much more powerful computation resources at autonomous edge, we conclude that our proposed detection algorithm in Sec. 4.2 performs in near real-time.

Remark 1. *DARWIN shows maximum detection accuracy of 61.27% and 96.93% (see Fig. 9) with camera and laser for far and close ranges, respectively. The detection time of the proposed method is $\sim 1ms$ which suggests that DARWIN updates the digital twin in near real-time, (see Fig. 9, validates Contribution 1).*

5.4 Performance of Module 2 in DARWIN

In this set of experiments, we consider a scenario where there is discrepancy between the digital twin at the autonomous edge and real world placement of the obstacles. The discrepancy might be caused by the low detection accuracy of the sensors at far distances (see Sec. 4.2) or errors in the sensor measurement itself due to usage of cheap LDS laser sensor used in the Turtlebot 3.

5.4.1 Effect of Discrepancy in Centroid

We consider a scenario where the dimensions of the obstacles match the ground-truth, i.e. $(0.56m, 0.77m)$. We then move the centroids of obstacles along each axis with the factor ϵ. Formally, $c_x^{'} = (1 \pm \epsilon)c_x$ and $c_y^{'} = (1 \pm \epsilon)c_y$, where $c_x^{'}$ and $c_y^{'}$ denote the mismatched centroids. We consider a setting where the centroids of the both obstacles are moved in the same direction. Thus, we consider four settings overall, including moving to the right, left, up, and down. In Fig. 10a, we report the average reward for four moving directions and $\epsilon = \{0.01, 0.05, 0.1, 0.2\}$. The red dotted bar in Fig. 10a denotes the reward at ground-truth locations. From this figure, we observe that the average reward decreases as the centroids are moved from the ground-truth placement (i.e. increasing ϵ). In particular, we observe up to 16.63% drop in reward in the most extreme case, equivalent of IOU being 0.5774. Moreover, we observe that moving the robot to the right has slightly lower effect on the reward. Interestingly, the detection algorithm in Sec. 4.2, achieves minimum $\sim 50\%$ accuracy while the obstacles are within the detection range. As a result, we can expect up to 16.63% drop in the average reward, when the accuracy of the detection algorithm with DARWIN is affected by the robot being far from the obstacles.

5.4.2 Effect of Discrepancy in Dimensions

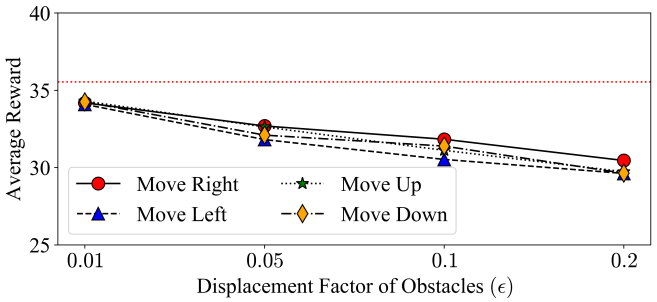
We scale the obstacles with factor $\rho = \{0.25, 0.5, 1, 1.25, 1.5\}$ and report the average reward in Fig. 10b. For example, when $\rho = 0.25$, the length and width of the obstacles are set as $(0.28m, 0.385m)$, and $\rho = 1$ is the ground-truth dimensions of the obstacles. We observe that the average reward stays within $25 - 35$ limit. Interestingly, we observe that scaling down the obstacles might also reduce the average reward ($\rho = 0.5$ for example). On the other hand, scaling up the dimensions also decreases the reward as expected. Overall, we observe that errors in detecting the dimensions will have a more intense effect on the average reward than errors in estimating the centroids.

Remark 2. *DARWIN handles slight discrepancies in detecting the centroids and dimension of the obstacles that are caused by errors in the data or affected by the detection accuracy at distant ranges (see Fig. 10, validates Contribution 2).*

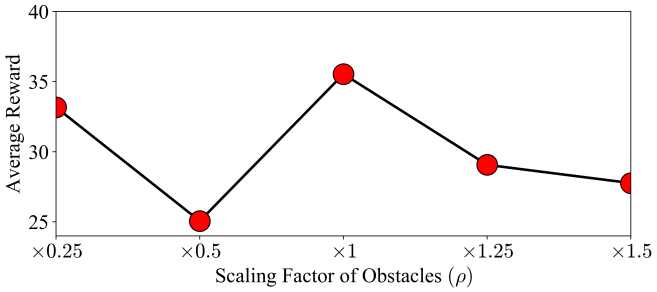
5.5 Performance of Module 3 in DARWIN

In this section, we study the effect of each parameter in module 3 (see Sec. 4.3), where the goal is to jointly optimize robot navigation and network management. In Tab. 5, we compare the performance over four scenarios, each having different obstacle positioning (see Fig. 6b). In this experiment, we use the best access point allocation strategy (Eq. 8), $\omega = 1$, and $\kappa = 1$ in Eq. 9 equivalent to the handover reward of -20. We observe that *Scenario 2* (symmetric obstacle positioning) has the highest number of steps. As a result, in the remaining set of experiments, we consider *Scenario 2*, which is the most challenging one, according to Tab. 5.

In Fig. 11a, we compare the average number of steps and reward for different parameters, including, access point allocation strategy (Eq. 8), RF reward weight (ω in Eq. 6), handover reward (Eq. 9) and outage limit (Eq. 12). Since the navigation reward parameters are extensively studied in the literature, we focus on the network management reward only. In particular, we study the effect of below parameters.



(a)



(b)

Fig. 10: Effect of discrepancy in centroid (a) and dimension (b) detection in DARWIN on average reward. The dotted line denotes the reward with the centroids and dimensions of the obstacles matching the ground-truth (details in Sec. 5.4, *validates Contribution 2*).

Scenario	Average Step	Average Reward	AP1	AP2	Outage
1	656.96	36.86	0.3623	0.6377	0.0407
2	**1099.88**	34.18	0.5816	0.4183	0.0269
3	751.34	35.39	0.2999	0.7	0.0248
4	809.25	32.69	0.3601	0.6398	0.0246

TABLE 5: Comparing the performance of four scenarios with different positioning of obstacles. We use the best access point allocation strategy (Eq. 8), $\omega = 1$, and $\kappa = 1$ in Eq. 9 equivalent to the handover reward of -20.

5.5.1 Effect of AP Allocation Strategy

We set the access point allocation strategy as choosing the closest versus the best access point as per Eq. 8. Recall that in our design we consider a stochastic channel profile. As a result, at each robot step, we randomly sample from one of 120 SNR measurements at each location (see Sec. 5.1.4). We then compare the SNR of received signal for both access points. In best access point strategy, we select the AP with the highest SNR of received signal, whereas with closest access point, we compute the distance of the robot with two access points and choose the closest one. In the first and third bars in Fig. 11a, we compare these two AP allocation strategies, while the rest of the parameters are fixed. We observe that the average number of steps decreases by 43% with best access point allocation compared to choosing the closest access point. This is intuitive as the wireless propagation patterns are complex and might be affected by the presence of obstacles (see Fig. 7). As a result, choosing the closest access point might not result in the optimal performance.

5.5.2 Effect of RF Reward Weight

In third and fourth bars in Fig. 11a, we compare the effect of the weight ω in Eq. 6. We observe that, by setting $\omega = 2$, the overall reward increases due to scaling of the network management reward as expected; however, we observe an increase in the number of steps by 12%. This is intuitive as rewarding the robot to prioritize following the path with better wireless coverage might result in an increased number of steps. As a result, it is important to find a harmonic combination of rewards to ensure that both goals of minimizing the number of steps and maintaining the connection with autonomous edge are achieved.

5.5.3 Effect of Handover Reward

According to Tab. 4, the maximum SNR of received signal of access points over all scenarios is 21.6dB. We consider $\kappa = \{1, 2\}$ in Eq. 9, which is equivalent to the handover reward being -10 and -20, respectively. We observe that by setting the handover reward as -20, the number of steps is 11% less. That is because by setting the handover punishment as $\kappa = 1$, the SNRs of received signal are modeled to be close to zero that resembles a realistic handover scenario where the robot loses the connection for a while. However, punishing the robots less than that (i.e., $\kappa = 2$) results in the combination of the SNR of received signal (Eq. 9) and handover (Eq. 9) rewards to be positive, which might result in encouraging the robot to switch access point more often.

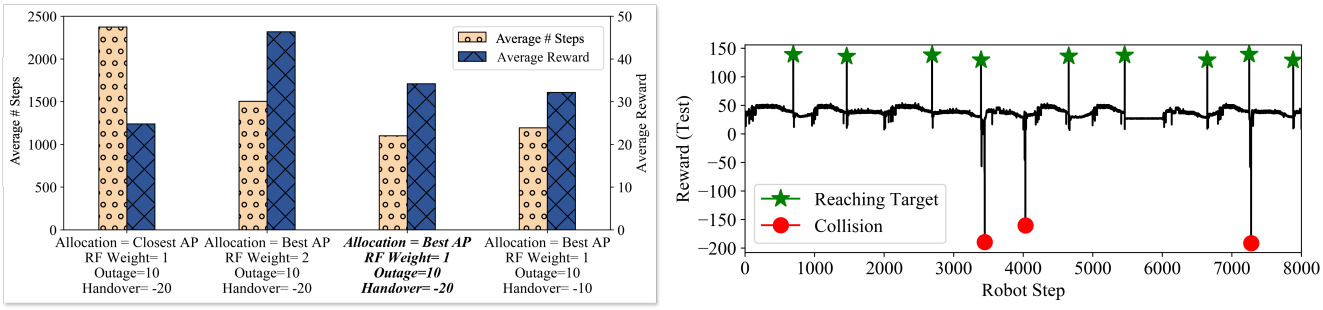
5.5.4 Targets Located at the Dead-Zones

Fig. 11b demonstrates the total reward (Eq. 7) over robot steps at inference phase for the best performing setting in Fig. 11a (third bar plot). In this figure, green star markers correspond to arrival at the target and red markers show the collisions. We note that the collision happens only when the target of the robot is located at a wireless dead-zone. As a result, while collisions are not frequent, they are inevitable when the robot must enter a wireless dead-zone to reach to its target. The solution here is to provide more access points to provide better coverage and eliminate wireless dead-zones.

Remark 3. *DARWIN carefully assign access points which results in 43% drop in the average number of steps compared to choosing the closest access point (see Fig. 11a, validates Contribution 4).*

5.6 Modeling Outage in DARWIN

In Tab. 6, we compare the performance when the outage SNR limit in Eq. 12 is set as $\lambda = \{5dB, 10dB, 15dB\}$. We compare the performance with respect to the average steps size and reward, as well as the access point allocation and outage ratios. From this table, we observe that the number of steps increases as the outage limit is uplifted, denoting a more strict constraint on the acceptable SNR of received signal. We note that the low outage limit ($\lambda = 5dB$ for example) is an optimistic scenario where the majority of the SNRs of received signal are higher that this threshold (see Tab. 4 for statistics of SNR of received signal in each scenario). This is reflected in the outage ratio of 0 for $\lambda = 5dB$, indicating that the outage never happens in this case. On the other hand, with $\lambda = 10dB$, the robot experiences outage in 2.6%



(a) (b)

Fig. 11: (a) Comparison of four settings with respect to access point allocation strategy (first and third columns), RF reward weight (second and third columns), and handover reward (third and fourth columns) (b) total reward (Eq. 7) over robot steps at inference phase for third bar in Fig. 11a. The green star markers correspond to arrival at the target, whereas the red markers show collision and are caused only when the target is placed in a wireless dead-zone (details in Sec. 5.5, *validates Contribution 4*).

Outage SNR (λ)	Scenario 2, Allocation = Best AP, ω=1, Handover =−20				
	Average Step	Average Reward	AP1	AP2	Outage
5	852.82	37.0180	0.39385	0.60615	0.0
10	1099.88	34.18	0.5816	0.4183	0.0269
15	6833	27.47	0.54795	0.45205	0.5992

TABLE 6: The effect of outage threshold (λ in Eq. 12) on *scenario 2*. The outage ratio increase as λ increases, depicting more strict constraints on the communication link (details in Sec. 5.5, *validates Contribution 4*).

of the time and this value increases to 59% in extreme case of $\lambda = 15dB$.

Remark 4. *By incorporating outage in DARWIN, we have a more realistic modeling of network management in warehouse floor environments. DARWIN shows a 80% increase in the number of steps where the outage limits is increased from $\lambda = 5$ to $\lambda = 10$ (see Tab. 6, validates Contribution 4).*

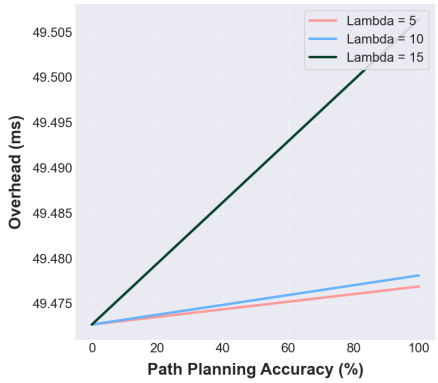


Fig. 12: The trade-off between path planning accuracy and end-to-end overhead for the three outage thresholds (λ) specified in Tab. 6.

5.7 Trade-off Analysis for Path Planning Accuracy vs Overhead

We analyze the trade-off between path planning accuracy and the associated overhead in retraining and communication. The total maximum end-to-end overhead is formulated as the sum of four components: uplink communication time from the robot, retraining time (triggered optionally for executing Module 2), inference time, and downlink communication to the robot. As described in Sections 4.3.1 and 4.3.2, the state array comprises 81 elements, while the action is represented by 3 elements. Our analysis considers a 20MHz channel with a 2.4 GHz WiFi link for both uplink and downlink communications. Fig. 12 illustrates the trade-off between path planning accuracy and end-to-end overhead for the three outage thresholds presented in Tab. 6. In our experiments, we define 100% accurate path planning as the average number of steps outlined in Tab. 6. Our findings demonstrate that for each threshold, higher accuracy corresponds to increased overhead, highlighting the inherent trade-off between precision and computational resources in robotic path planning systems.

5.8 Discussion: Computational Considerations and Edge Deployment

The deployment of DARWIN in real-world large-scale warehouse environments necessitates a thorough consideration of its computational complexity and scalability, particularly given the crucial requirement for real-time updates. Our approach is designed to explicitly manage these challenges. While the process of updating the RL policy in Module 3 can be computationally intensive, especially as the number of robots and environmental dynamics increase, DARWIN significantly mitigates this through its intelligent policy update trigger (Module 2). By defining a tolerance threshold for environmental changes and only initiating policy retraining when a substantial disparity is detected (as quantified by the similarity metric in Eq. 5), DARWIN drastically reduces the frequency of these expensive computations. This selective update mechanism ensures that unnecessary computational load is avoided, thereby enhancing the overall system's scalability and responsiveness in highly dynamic and expansive settings. Furthermore, the rapid detection time of approximately $1ms$ for digital twin generation (Module 1), as observed on a standard desktop, underscores the efficiency of the initial data processing, which is critical for supporting real-time operations.

The architectural decision to deploy DARWIN's core modules at an "autonomous edge" is fundamental to its feasibility and efficiency in practical, resource-constrained

environments. This edge computing paradigm positions powerful computational resources closer to the data sources (robots), enabling low-latency processing of sensor information and swift decision-making. By centralizing the heavy computational tasks of digital twin generation, policy update triggering, and joint navigation/network management at the edge, DARWIN effectively offloads these demands from individual robots, which are typically constrained by size, power, and processing capabilities. This distributed intelligence model not only ensures near real-time digital twin updates but also provides a robust and efficient platform for managing the complex interplay between robot navigation and wireless network conditions, even when deployed across expansive and challenging warehouse floor layouts. The autonomous edge thus serves as a critical enabler for DARWIN's real-time performance and scalability.

6 Conclusions

In this paper, we present DARWIN, a digital twin based approach for robotic systems in warehouse floor environments. The proposed method continuously monitors the environment to detect the changes and generates the digital twin in near real-time, and updates the path planning policy, if a substantial change in the environment is observed. Our proposed method detects the changes in the environment with up to 96% accuracy. In DARWIN, we design a reinforcement learning algorithm to jointly optimize navigation and network resource management while accounting for SNR, access point allocation strategy, handover, and outage. To validate DARWIN, we design a testbed with Turtlebot robots and X310 radios. The results demonstrate that by choosing the access point meticulously in DARWIN the average number of steps decreases by 43%. Our future attention is to integrate DARWIN for enabling joint sensing and communication and integrating more sensors (e.g., radar and ultra wide band sensors) for next-generation robotic and autonomous system. We will also pursue further exploration of real-world testing to enhance the validation and refinement of our algorithm.

References

[1] C. Bai, P. Dallasega, G. Orzes, and J. Sarkis, "Industry 4.0 technologies assessment: A sustainability perspective," *International journal of production economics*, vol. 229, p. 107776, 2020.

[2] M. Radmanesh, M. Kumar, P. H. Guentert, and M. Sarim, "Overview of path-planning and obstacle avoidance algorithms for uavs: A comparative study," *Unmanned systems*, vol. 6, no. 02, pp. 95–118, 2018.

[3] A. Baxi, M. Eisen, S. Sudhakaran, F. Oboril, G. S. Murthy, V. S. Mageshkumar, M. Paulitsch, and M. Huang, "Towards factory-scale edge robotic systems: Challenges and research directions," *IEEE Internet of Things Magazine*, vol. 5, no. 3, pp. 26–31, 2022.

[4] B. Holfeld, D. Wieruch, T. Wirth, L. Thiele, S. A. Ashraf, J. Huschke, I. Aktas, and J. Ansari, "Wireless communication for factory automation: An opportunity for lte and 5g systems," *IEEE Communications Magazine*, vol. 54, no. 6, pp. 36–43, 2016.

[5] H. X. Nguyen, R. Trestian, D. To, and M. Tatipamula, "Digital Twin for 5G and Beyond," *IEEE Communications Magazine*, vol. 59, no. 2, pp. 10–15, 2021.

[6] A. Fuller, Z. Fan, C. Day, and C. Barlow, "Digital twin: Enabling technologies, challenges and open research," *IEEE access*, vol. 8, pp. 108 952–108 971, 2020.

[7] A. Padovano, F. Longo, L. Nicoletti, and G. Mirabelli, "A digital twin based service oriented application for a 4.0 knowledge navigation in the smart factory," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 631–636, 2018.

[8] T. Delbrügger, L. T. Lenz, D. Losch, and J. Roßmann, "A navigation framework for digital twins of factories based on building information modeling," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2017, pp. 1–4.

[9] S. Lu, F. Liu, Y. Li, K. Zhang, H. Huang, J. Zou, X. Li, Y. Dong, F. Dong, J. Zhu *et al.*, "Integrated sensing and communications: Recent advances and ten open challenges," *IEEE Internet of Things Journal*, 2024.

[10] A. A. B. Pritsker, *Introduction to Simulation and SLAM II*. Halsted Press, 1984.

[11] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[12] E. Krell, A. Sheta, A. P. R. Balasubramanian, and S. A. King, "Collision-free autonomous robot navigation in unknown environments utilizing pso for path planning," *Journal of Artificial Intelligence and Soft Computing Research*, vol. 9, no. 4, pp. 267–282, 2019.

[13] X. Chen and Y. Li, "Smooth path planning of a mobile robot using stochastic particle swarm optimization," in *2006 International conference on mechatronics and automation*. IEEE, 2006, pp. 1722–1727.

[14] D.-L. Almanza-Ojeda, Y. Gomar-Vera, and M.-A. Ibarra-Manzano, "Occupancy map construction for indoor robot navigation," *Robot Control*, 2016.

[15] Y. K. Hwang, N. Ahuja *et al.*, "A potential field approach to path planning." *IEEE transactions on robotics and automation*, vol. 8, no. 1, pp. 23–32, 1992.

[16] J. Hu, H. Niu, J. Carrasco, B. Lennox, and F. Arvin, "Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 14 413–14 423, 2020.

[17] M. S. Elbamby, C. Perfecto, C.-F. Liu, J. Park, S. Samarakoon, X. Chen, and M. Bennis, "Wireless edge computing with latency and reliability guarantees," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1717–1737, 2019.

[18] K. Ahmadi, S. P. Miralavy, and M. Ghassemian, "Software-defined networking to improve handover in mobile edge networks," *International Journal of Communication Systems*, vol. 33, no. 14, p. e4510, 2020.

[19] T. M. Ho and K.-K. Nguyen, "Joint server selection, cooperative offloading and handover in multi-access edge computing wireless network: A deep reinforcement learning approach," *IEEE Transactions on Mobile Computing*, vol. 21, no. 7, pp. 2421–2435, 2020.

[20] S. Mohanti, D. Roy, M. Eisen, D. Cavalcanti, and K. Chowdhury, "L-norm: Learning and network orchestration at the edge for robot connectivity and mobility in factory floor environments," *IEEE Transactions on Mobile Computing*, 2023.

[21] M. Eisen, S. Shukla, D. Cavalcanti, and A. S. Baxi, "Communication-control co-design in wireless edge industrial systems," in *2022 IEEE 18th International Conference on Factory Communication Systems (WFCS)*. IEEE, 2022, pp. 1–8.

[22] C. Chen, J. Gao, Y. Guo, and Y. Chen, "Deep reinforcement learning for indoor mobile robot path planning," *Sensors*, vol. 20, no. 19, p. 5493, 2020.

[23] M. Mancini, G. Costante, P. Valigi, and T. A. Ciarfuglia, "Fast robust monocular depth estimation for obstacle detection with fully convolutional networks," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4296–4303.

[24] F. H. Ajeil, I. K. Ibraheem, A. T. Azar, and A. J. Humaidi, "Autonomous navigation and obstacle avoidance of an omnidirectional mobile robot using swarm optimization and sensors deployment," *International Journal of Advanced Robotic Systems*, vol. 17, no. 3, p. 1729881420929498, 2020.

[25] "Intel realsense depth and tracking cameras products, howpublished = https://www.intelrealsense.com,."

[26] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 31–36.

[27] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International conference on machine learning*. PMLR, 2016, pp. 2829–2838.
[28] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.
[29] "Gnu radio, howpublished = https://https://www.gnuradio.org,."
[30] "Gazebo, howpublished = https://gazebosim.org/home,."
[31] "Ros (robotic operating system), howpublished = https://www.ros.org,."
[32] "Gym wrappers, howpublished = https://alexandervandekleut.github.io/gym-wrappers/,."
[33] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

Batool Salehi received her Ph.D. degree in computer engineering at Northeastern University under the supervision of Prof. K. Chowdhury. Her current research focuses on mmWave beamforming, Internet of Things, and the application of machine learning in the domain of wireless communication.

Debashri Roy is currently an Assistant Professor at the University of Texas Arlington. Prior to this, she was Associate Research Scientist at the Northeastern University. She received her MS (2018) and PhD (2020) degrees in Computer Science from University of Central Florida, USA. She was an experiential AI postdoctoral fellow at Northeastern University (2020-2021). Her research interests are in the areas of AI/ML enabled technologies in wireless communication, multimodal data fusion, network orchestration and networked robotics.

Mark Eisen received the Ph.D in electrical engineering and Masters in Statistics from the University of Pennsylvania in 2019. He is currently working as a research scientist at Intel Labs in Hillsboro, OR. His research interests include machine learning, wireless communications, networked control systems, and statistical optimization. Dr. Eisen was a recipient of the Outstanding Student Presentation at the 2014 Joint Mathematics Meeting, as well as the recipient of the 2016 Penn Outstanding Undergraduate Research Mentor Award. In 2022, Dr. Eisen received the Best Paper Award at the IEEE International Conference on Factory Communication Systems.

Amit Baxi holds M.S. degree in Digital Design and Embedded Systems from Manipal University, India. He is currently a Research Scientist with Intel Labs in Bengaluru. His research interests include sensing systems, wear- able health technologies, IoT and Edge robotics.

Dave Cavalcanti is Principal Engineer at Intel Corporation where he develops next generation wireless connectivity and distributed computing technologies to enable autonomous, time-sensitive systems and applications. He received his PhD in computer science and engineering in 2006 from the University of Cincinnati. He leads Intel Lab's research on Wireless Time-Sensitive Networking (TSN) and industry activities to enable time-critical systems and applications of wireless technologies, including WiFi and beyond 5G systems. He is Senior Member of the IEEE and serves as the chair of the Wireless TSN working group in the Avnu Alliance, an industry group facilitating an ecosystem of interoperable TSN devices and deterministic networking across Ethernet, Wi-Fi and 5G technologies.

Kaushik Chowdhury is a professor in the Department of Electrical & Computer Engineering at The University of Texas at Austin and co-PI on The Ohio State University-led NSF AI-Edge Institute from Northeastern University. He is the 2023 finalist for the U.S. Blavatnik National award and previous winner of the U.S. Presidential Early Career Award for Scientists and Engineers (PECASE) in 2017, the Defense Advanced Research Projects Agency Young Faculty Award in 2017, the Office of Naval Research Director of Research Early Career Award in 2016, and the National Science Foundation (NSF) CAREER award in 2015. He is the recipient of best paper awards at IEEE GLOBECOM'19, DySPAN'19, INFOCOM'17, ICC'13,'12,'09, and ICNC'13. He was co-director of the operations of Colosseum RF/network emulator, as well as the Platforms for Advanced Wireless Research project office. Prof. Chowdhury has served in several leadership roles, including Chair of the IEEE Technical Committee on Simulation, and as Technical Program Chair for IEEE MILCOM 2023, IEEE INFOCOM 2021, IEEE CCNC 2021, IEEE DySPAN 2021, and ACM MobiHoc 2022. His research interests are in large-scale experimentation, applied machine learning for wireless communications, data-centric IoT architectures, and networked robotics.