Fall 2020

Deep Reinforcement Learning based Path-Planning for Multi-Agent Systems in Advection-Diffusion Field Reconstruction Tasks

Deepak Talwar
San Jose State University

Recommended Citation
Talwar, Deepak, "Deep Reinforcement Learning based Path-Planning for Multi-Agent Systems in Advection-Diffusion Field Reconstruction Tasks" (2020). *Master's Theses*. 5164.
DOI: https://doi.org/10.31979/etd.e36p-dxjt
https://scholarworks.sjsu.edu/etd_theses/5164

DEEP REINFORCEMENT LEARNING BASED PATH-PLANNING FOR MULTI-AGENT SYSTEMS IN ADVECTION-DIFFUSION FIELD RECONSTRUCTION TASKS

A Thesis

Presented to

The Faculty of the Department of Computer Engineering

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Deepak Talwar

December 2020

The Designated Thesis Committee Approves the Thesis Titled

DEEP REINFORCEMENT LEARNING BASED PATH-PLANNING FOR MULTI-AGENT SYSTEMS IN ADVECTION-DIFFUSION FIELD RECONSTRUCTION TASKS

by

Deepak Talwar

APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING

SAN JOSÉ STATE UNIVERSITY

December 2020

Wencen Wu, Ph.D.	Department of Computer Engineering
Kaikai Liu, Ph.D.	Department of Computer Engineering
Magdalini Eirinaki, Ph.D.	Department of Computer Engineering

ABSTRACT

DEEP REINFORCEMENT LEARNING BASED PATH-PLANNING FOR MULTI-AGENT SYSTEMS IN ADVECTION-DIFFUSION FIELD RECONSTRUCTION TASKS

by Deepak Talwar

Many environmental processes can be represented mathematically using spatial-temporal varying partial-differential equations. Timely estimation and prediction of processes such as wildfires is critical for disaster management response, but is difficult to accomplish without the availability of a dense network of stationary sensors. In this work, we propose a deep reinforcement learning-based real-time path-planning algorithm for mobile sensor networks traveling in a formation through a spatial-temporal varying advection-diffusion field for the task of field reconstruction. A deep Q-network (DQN) agent is trained on simulated advection-diffusion fields to direct the mobile sensor network to travel along information-rich trajectories. The field measurements made by the mobile sensor network along their trajectories enable identification of field advection parameters, which are required for field reconstruction. A cooperative Kalman filter developed in previous works is employed to receive estimates of the field values and gradients, which are essential for reconstruction as well as for the estimation of the diffusion parameter. A mechanism is provided that encourages exploration in the field domain once a stationary state is reached, which allows the algorithm to identify other information-rich trajectories that may exist in the field improving reconstruction performance significantly. Two simulation environments of different fidelities are provided to test the feasibility of the proposed algorithm. The low-fidelity simulation environment is used for training of the DQN agent. The high-fidelity simulation environment is based on Robot Operating System (ROS) and simulates real robots. We provide results of running sample test episodes in both environments which demonstrate the effectiveness and feasibility of the proposed algorithm.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my thesis advisor, Dr. Wencen Wu, professor in the Computer Engineering department at San José State University for her guidance throughout my time at SJSU. She has always been ready to answer my questions, provide me with feedback, and help me understand concepts related to and unrelated to this research. Her encouragement, positivity and belief in me has helped me tremendously in successfully completing this work.

Also, I would like to express my sincere gratitude to Dr. Kaikai Liu and Dr. Magdalini Eirinaki who took their precious time to participate in my thesis defense. I am grateful for their valuable comments on my thesis.

I would also like to thank my parents and grandparents for their unconditional love and constant support throughout my life. I can never repay for all the sacrifices that my mother has made in order to get me where I am today. I will forever be grateful. I would also like to thank my girlfriend, Qin Bian, for her continuous encouragement throughout my study and research, and her unique ways of pushing me to make myself better. Finally, I would like to thank my friends Sachin Guruswamy and Seung Won Lee, for brainstorming with me and being my soundboards.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

A number of environmental, physical, chemical, and biological processes are often called distributed parameter systems (DPSs) as their states vary both temporally and spatially. Examples of such spatial-temporal varying fields include pollution concentration, temperature, salinity, etc. These DPSs are often modeled mathematically as partial differential equations varying over space and time (PDEs) [1], [2]. In this work, we focus on a special form of such spatial-temporal PDE called the advection-diffusion equation, which for example may be used to describe the dynamics of a smoke plume in a given region over time. The advection term describes the movement of the smoke plume due to the velocity of the carrier wind, and the diffusion term describes the spatial spreading of the smoke from regions of higher concentration to regions of lower concentration. In such environmental monitoring and pollution management tasks, timely estimation, prediction and reconstruction of advection-diffusion fields become very important and can assist disaster management response. For instance, prediction of the propagation of a wildfire can assist with firefighting, and identifying the source of oil leakage in an ocean can be important for pollution containment.

Networks of static sensors are often used to provide distributed measurements across a spatial domain [3]. However, several challenges exist in the estimation and reconstruction of spatial-temporal fields using static sensor networks, such as low spatial resolution or complete lack of sensor measurements in the regions of interest, and high computational cost of solving PDEs over both spatial and temporal domains. Tasks that require exploring unknown fields are often distributed across a large area in scenarios like ocean science and wildfires, which makes installing and maintaining sensor networks infeasible. It takes a large number of stationary sensors to obtain snapshots of the concentration field in such areas. To overcome this issue, it is ideal to use a small number of mobile sensing agents to scout the large area by taking measurements along the motion trajectories [4]–[6]. Such mobile sensing agents may consist of intelligent robots that can collaboratively accomplish exploration tasks by exploiting their on-board sensing, computing, mapping and locomotion capabilities. In this work, we explore how intelligent paths can be planned for such mobile sensing robots for the advection-diffusion field reconstruction task.

The field reconstruction task involves two major sub-tasks: (1) identification of the advection-diffusion parameters governing the unknown field, and (2) state estimation of the concentration field in the chosen spatial domain. The performance of the state estimation depends heavily on the paths that the mobile sensing agents travel on. Thus, there is a need to efficiently solve real-time path planning problems to guide agents to move along information-rich trajectories. In recent years, various techniques have emerged as solutions to this problem. In works [5] and [7], the authors incorporate the dynamics of the mobile robot agents into the dynamics of the spatial-temporal varying field, which allows them to compute trajectories for the mobile robot agents deployed in a spatial field domain. However, the cost functions used for path optimization in works [5] and [7] only aim to reduce the mapping error, which can make the solutions get stuck in local minima and fail to account for fields with multiple high-concentration areas or sources and limit their ability to reconstruct more complicated fields. In literature [8], the authors propose using a geometric reinforcement learning based approach for path-planning in fields that may contain multiple high-concentration regions within the spatial domain. With this approach, the authors encode the mapping error and the length of the robots' trajectories into a time-varying reward matrix, which enables them to convert the problem into a shortest-path finding problem in a weighted graph that can be solved using dynamic programming [8]. This approach, however, requires the user of the algorithm to provide a destination location that can be used to search for in the graph, and the quality of the field reconstruction depends heavily on this chosen destination. In unknown fields with complicated concentration surfaces, choosing this destination may not be trivial. In this research, we propose a deep reinforcement learning-based algorithm that does not require the user to provide a destination for the formation, and encourages exploration in situations when a local minimum for the mapping error is reached, thus enabling the capability to reconstruct fields with complicated concentration surfaces and multiple high-concentration zones.

Over the years, a variety of different methods have been developed for estimation of advection-diffusion parameters. With the identification of advection-diffusion parameters in the advection-diffusion PDE, we are able to obtain some preliminary knowledge on the evolution of the field, which makes it possible to use only a small number of mobile sensor robots to reconstruct a large dynamic field. In this research, we use the cooperative filtering scheme developed for online identification of the diffusion parameter in dynamic diffusion field in works [9] and [10]. In [9], authors use the data collected by a mobile sensor network moving in a diffusion field to develop a cooperative Kalman filter that provides estimates of field values, the gradient and temporal variations of the field values along trajectories of the mobile robot formation. Furthermore, the authors develop a recursive least squares (RLS) based method for estimation of the unknown diffusion coefficient of the field. In this work, we use the estimated field values and the gradient provided by the cooperative Kalman filter for training the deep reinforcement learning models, as well as for field reconstruction. Additionally, the estimates of the diffusion coefficient are used to reconstruct the field as the mobile robot formation travels through the field domain. Moreover, we are able to extend these results by also providing a method that uses the trained deep reinforcement learning model to plan trajectories for the mobile robot formation that allow for online estimation of advection parameters for the advection-diffusion field as well.

Finally, in this work, we develop and present results in two simulation environments of varying fidelities that show satisfactory performance. The Low-Fidelity Simulation

Environment is a 2D environment that simulates advection-diffusion fields in spatially discretized grids and represents mobile robots as omni-directional particles, and is used for training of the deep reinforcement learning models, as well as for testing and validation of the proposed algorithms. The High-Fidelity Simulation Environment is a Robot Operating System (ROS) [11] based 3D environment that employs simulated differential drive robots, and is used to validate the developed path-planning algorithms.

The key contributions of this research include:

1) A deep reinforcement learning based path-planning algorithm for mobile sensing robots traveling in a formation through an advection-diffusion field for the task of field reconstruction.
2) Procedure for estimation of advection coefficients as the formation travels through the advection-diffusion field following the paths planned by the deep reinforcement learning based algorithm.
3) A Low-Fidelity Simulation Environment, that is used for training the deep reinforcement learning models, as well as to validate and test the performance of the proposed algorithms.
4) A Robot Operating System (ROS) [11] based High-Fidelity Simulation Environment that is used to test the proposed algorithm with realistic simulated robot agents.

The problem is formulated in Chapter 3. Chapter 5 presents the proposed algorithm, Chapter 6 elaborates on the deep reinforcement training process and Chapter 7 shows simulation results in the Low-Fidelity Simulation Environment. Chapter 9 presents the High-Fidelity Simulation Environment and Chapter 10 presents simulation results in the High-Fidelity Simulation Environment.

2 LITERATURE REVIEW

A large body of work concerning with state estimation and parameter identification of distributed parameter systems (DPSs) using sensor networks has been produced. Many approaches address this problem using static sensor networks [12]–[14]. However, in large spatial domains, it is often impractical to set up stationary sensors, and instead mobile sensor networks are deployed to perform a variety of tasks including parameter identification, state estimation, coordinated exploration and multi-target tracking [1], [6], [15], [16]. One popular approach for parameter estimation, which is applicable to both static as well as mobile sensor networks, is to first identify optimal locations of stationary sensors, or trajectories for mobile sensors offline, and then formulate a least squares problem to find parameters that minimize the errors between the state estimations and true states at the locations or trajectories chosen offline [17], [18]. However, in many realistic environmental monitoring tasks, there is no prior knowledge of the field and it is desirable to obtain parameter identification while the mobile sensor network is traveling through the field domain. As mentioned previously, in works [9], [10], authors develop an online parameter identification algorithm that iteratively estimates the diffusion coefficient as the mobile robots travel through the field domain in a formation. In this work, we extend this result to produce a field reconstruction algorithm that can also estimate the advection coefficients in an advection-diffusion field.

Reinforcement learning, and specifically deep reinforcement learning, has recently been used for a variety of decision-making tasks. Some of the most popular use of reinforcement learning has been done in the realm of computer gaming. In [19], a convolutional neural network trained using a variant of Q-learning is shown to successfully play many Atari computer games. Path-planning for robots is another decision-making domain where reinforcement learning has been increasingly used. In [20], the authors introduce Globally Guided Reinforcement Learning framework

(G2RL), which is a hierarchical path-planning algorithm that combines global guidance (using a graph-based search method such as A*) and a local RL-based planner that uses local observations to plan obstacle free paths. In [21], the authors demonstrate the use of double Q-network (DDQN) in solving local path-planning for robots in unknown dynamic environments with LiDAR (Light Detection and Ranging) signal. Some work using reinforcement learning has also emerged in path-planning of mobile sensor networks in the problem of reconstructing spatial-temporal varying fields. As mentioned previously, in [8], the authors propose using a geometric reinforcement learning based approach for path-planning in fields that may contain multiple high-concentration regions within the spatial domain. In this approach, the authors build a time-varying reward matrix, which encode the quality of the field reconstruction and the length of the robots' trajectories, and enables them to convert the problem into a shortest-path finding problem in a weighted graph that can be solved using dynamic programming [8]. This approach uses the user-supplied global destination as a way for the formation to not get stuck in local minima of the mapping error function, which may assist in discovery of other information-rich regions. However, the quality of the field reconstruction depends heavily on this chosen destination and in unknown fields with complicated concentration surfaces, choosing this destination may not be trivial. In this work, we explore how deep reinforcement learning could be applied to plan paths for a collection of mobile robots moving in a formation to reconstruct spatially and temporally varying advection-diffusion fields. In most works mentioned here, the user is responsible for providing a global destination to the planner, and the main task is to reach the destination avoiding collisions. However, in this work, the main task is to reconstruct the field as accurately as possible and there is no need for the user to supply a destination. The algorithm developed identifies information-rich paths that improve the reconstruction of the field, and is also capable of exiting local minima when they are detected.

Over the last two decades, several middleware message-passing software have been developed and popularized which are useful for robotics applications. These middleware software provide an abstraction layer above the underlying operating system and provide convenient inter-process communication methods, which are useful in a robotics application that may have many asynchronous processes running that need to share data. Lightweight Communications and Marshalling (LCM) [22] is a library for data passing and marshalling specifically designed for robotics research applications, and was first developed during the DARPA Urban Challenge. LCM is language independent and processes can share data through publish/subscribe message-passing system.
ZeroMQ [23] is a generic open-source message-passing library that can be used for in-process, inter-process, TCP and multicast data sharing, and has been used in a number of robotics projects. The most popular robotics framework, however, is Robot Operating System (ROS) [11]. ROS is fully open-source and not only provides TCP/IP based message-passing, but also comes with a rich ecosystem of useful tools and robotics algorithms. In this work, we use ROS as our middleware of choice in the development of the in-lab testbed (Chapter 8) and the High-Fidelity Simulation Environment (Chapter 9).

Simulation has become an integral part of robotics development and testing, and the fidelity of simulation environments has improved substantially over the last few decades. Most robotics algorithms are developed and iterated upon in simulation before they are deployed on real robots, as it is much easier, faster, cheaper and safer to test in simulation environments. As the graphics quality of 3D simulations have improved, simulators are also being used as sources of cheap training data for machine learning and deep learning models. As a result, a number of commercial, as well as, open-source simulators have emerged focusing on various robotics domains. Several open-source simulators such as the LGSVL Simulator [24], CARLA [25] and Voyage Deepdrive [26] are focusing on the Autonomous Driving domain, and provide vehicles with simulated sensors that can be driven in simulated 3D environments. Other simulators such as Webots [27], Gazebo [28] and CoppeliaSim [29] provide simulation for all kinds of robotic applications including drones, wheeled robots and robotic arms. Several companies such as Applied Intuition [30], rFpro [31] and Cognata [32] have commercialized simulators and are providing them as a service. For this work, our simulation platform of choice is Gazebo [28] as it has deep integration with ROS, is open-source, a large community of users, and provides good simulation support for wheeled robots.

Over the last few years, a large number of open-source robotics platforms have been introduced focusing on a variety of robotic research applications. The Multi-agent System for non-Holonomic Racing (MuSHR) [33] project provides an open-source platform designed for autonomous robotics racing which supports ROS [11]. The Khepera IV Mobile Robot [34] aims at providing a robotics platform that can bring robotics system research from simulation to the real world, and is specifically designed for indoor robotics experiments. The TurtleBot [35] is a low-cost robotics kit that is expandable and comes tightly integrated with ROS [11]. In this work, we use the NVIDIA Jetbot AI Robot kits [36] as our choice of robotic platform, for its small size, low cost and upgradeability. Chapter 8 elaborates on the modifications we made to this kit for this research.

3 PROBLEM FORMULATION

In this chapter, we formulate the problem of reconstructing an unknown spatial-temporal varying field obeying the advection-diffusion equation by planning paths for mobile sensing robots moving in a formation through the field.

3.1 Advection-Diffusion Fields

Many spatial-temporal varying processes such as atmospheric and waterborne pollution transport processes can be modelled as two-dimensional (2D) partial differential equations in some domain Ω of the following form:

$$\frac{\partial z}{\partial t}(r,t) = \mathscr{F}(z(r,t), \nabla z(r,t), \nabla^2 z(r,t)), \quad r \in \Omega, \tag{1}$$

where Ω is the domain that the robots are operating in, $z(r,t)$ is the field concentration function that is spatial-temporal varying, ∇ represents the gradient operator, ∇^2 represents the Laplacian operator, and $\mathscr{F}(\cdot)$ is an unknown non-linear function.

The 2D advection-diffusion partial differential equation is a specific version of Equation (1) which incorporates advection (spatial movement of concentration values) and diffusion (movement of concentration values from high-concentration zones to low-concentration zones) processes. We are interested in the reconstruction of fields obeying the advection-diffusion equation, which can be written as follows:

$$\frac{\partial z}{\partial t}(r,t) = \theta \nabla^2 z(r,t) + \mathbf{v} \nabla z(r,t), \quad r \in \Omega, \tag{2}$$

where $\theta > 0$ is the constant diffusion coefficient and the $\mathbf{v}$ is the constant 2D velocity vector. Therefore, given a field Ω with $z(r,t), r \in \Omega$ defining the field values, Equation (2) defines how the field evolves over time. Additionally, in many practical environmental monitoring applications, the field domain Ω is much larger than the robots, and so the boundary can be modeled as a flat surface. This allows us to assume initial and Dirichlet boundary conditions on the boundary $\partial\Omega$ [6],

$$\begin{aligned} z(r,0) &= z_0(r), \\ z(r,t) &= 0,\ \ r \in \partial\Omega. \end{aligned} \tag{3}$$

The field concentration function $z_0(r,t=0)$ may consist of a number of high-concentration regions forming a non-linear surface over the field domain Ω. The task of field reconstruction, then, involves the following two sub tasks:

1) Identification of advection-diffusion coefficients in Equation (2). These include the advection coefficients $\mathbf{v} = [v_x, v_y]$, the velocity of the field, and θ, the diffusion coefficient. While $\mathbf{v}$ specifies how fast the field is moving and in what direction, θ specifies how quickly the concentration values of the field are diffusing.
2) State estimation of the concentration field $z(r,t)$ in the field domain Ω to reproduce the field map.

3.2 Mobile Sensor Robots

In this work, we consider mobile sensor robots to be modeled as points with the ability to move omni-directionally as commanded. The algorithm proposed in this work commands a formation of such mobile sensing robots to travel on information-rich paths to efficiently reconstruct advection-diffusion fields. We make certain assumptions regarding these mobile sensing agents.

Assumption 3.1. *Robots follow single-integrator dynamics.*

The robots are controlled to move in the field domain Ω in a coordinated formation. Consider a formation of N mobile sensing robots traveling in the field; these robots follow single-integrator dynamics as follows:

$$\dot{r}_i(t) = u_i(t),\ \ i = 1,2,3...N, \tag{4}$$

where, $r_i(t) \subseteq \mathbb{R}^2$ is the location of the ith robot in Ω and $u_i(t) \subseteq \mathbb{R}^2$ is the velocity command for the ith robot at time t. The single-integrator dynamics in Equation (4) allow the robots to follow the commanded velocity exactly. While this assumption does not hold well for wheeled robots that are generally not omni-directional, it is a reasonable motion assumption for drones flying in a plane. Furthermore, in Chapter 9, we demonstrate how this assumption can be shown to work for wheeled robots following differential-drive dynamics. Additionally, there exist several results for formation control of mobile robotic agents in [37], [38] which we apply here such that the robots stay in a desired formation.

Assumption 3.2. *Each sensing robot has the ability to localize itself in Ω and share its location with other robots.*

The ability for a mobile robot to localize itself is critical for the ability to associate sensor measurements to physical locations, as well as for the ability to maintain formations. Our assumption here is that each robot i is equipped with sensors to localize itself in field domain Ω at discrete time step k and share its location r_i^k with the other robots.

Additionally, using the locations of all the robots in the formation at time step k, we can determine the location of the formation center r_c^k at time step k using the following equation:

$$r_c^k = \frac{1}{N} \sum_i^N r_i^k. \tag{5}$$

Assumption 3.3. *Each sensing robot is equipped with a sensor to measure the field concentration value at its current location.*

At each discrete timestep k, each robot has the ability to measure and report the concentration value of the field at its location. The measurement of the ith sensing robot at time step k is modeled as follows:

$$p(r_i^k, k) = z(r_i^k, k) + n_i, \tag{6}$$

where, n_i is assumed to be i.i.d Gaussian noise.

3.3 View-Scope of Mobile Sensing Robots

Provided that the robots will coordinate and move in a formation, it is important to specify the benefits of this coordination. Here, we define the time-varying view-scope $\Gamma(t)$ of the formation. The view-scope Γ at time t is the area of the field domain Ω that falls within the polygon formed by the locations of sensing robots. The shaded region in Fig. 1 illustrates the time-varying view-scope $\Gamma(t)$, in which the blue circles represent the four mobile robots at the current time step in a formation, and the red circle represents the formation center at the current time step. While the mobile sensing robots can only measure and share the concentration values at their locations at any given time, the field values $z(r,t),\ r \in \Gamma(t)$ can be obtained through interpolating the values measured by the robots. Thus, it is reasonable to assume that the estimated field values, $z(r,t), r \in \Gamma(t)$ are available to us at all times. Additionally, as stated previously, a cooperative Kalman filter developed in [9], [10] and explained further in Chapter 4 is employed to output estimates of concentration, $z(r_c,t)$ and gradients, $\nabla z(r_c,t)$, at the formation center r_c. These estimated values will play a major role in the developed algorithm described in Chapter 5.

The cooperative Kalman Filter implementation requires that the robots stay relatively close to each other to collect measurements in the field [9], [10]. Therefore, it is important to have formation control for the mobile sensor robots as they move through the field domain. Since the robots are controlled to stay in a formation, the problem reduces to planning the path for formation center r_c instead of planning paths for each robot in the formation. We will revisit the topic of formation control when discussing the

High-Fidelity 3D Simulation Environment in Chapter 9, where formation control needs to be implemented for testing of the proposed algorithm.

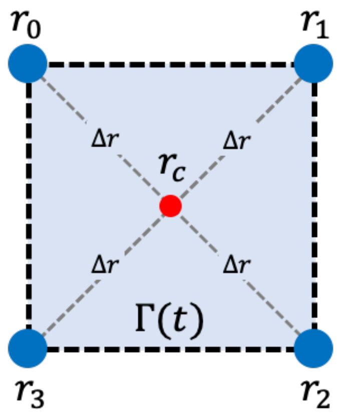


Fig. 1. A symmetric formation composed of four mobile robots $r_i,\ i = 0,1,2,3$ shown in blue. The formation center r_c is shown in red. The distance between each robot and the formation center is Δr. The shaded region is the time-varying view-scope $\Gamma(t)$.

4 PRELIMINARIES

In this chapter, we describe the results of works [9], [10] that are critical to the development of the algorithm in this research. In [9], [10], the authors consider the task of online parameter identification of 2D diffusion processes using data collected by a mobile sensor network in a diffusion field. To realize this task, the authors incorporate the diffusion equation into the information dynamics associated with the trajectories of mobile sensor networks and develop a cooperative Kalman Filter to provide estimates of field values, gradients and the temporal variations in the field along the trajectories. Then, utilizing the estimates from the cooperative Kalman Filter, they develop a recursive least-squares (RLS) based algorithm to iteratively estimate the diffusion coefficient of the field. The outputs of the cooperative Kalman Filter are used in this research for the training and inference of the deep reinforcement learning models which are explained in Chapter 6. The estimated diffusion coefficient is used in the reconstruction of the advection-diffusion field.

Since the main focus of this work is in development of the path-planning algorithm, we will not reproduce the mathematical derivation and proof of convergence for the cooperative Kalman Filter. Interested readers can refer to [9], [10] for the complete derivation and proof of convergence. The developed cooperative Kalman Filter provides estimates of the information state at each time step k when the measurements from the mobile sensors in the formation are available. The information vector $X(k)$ is defined as follows:

$$X(k) = [z(r_c^k, k), \nabla z(r_c^k, k), z(r_c^k, k-1), \nabla z(r_c^k, k-1)]^T, \tag{7}$$

where, r_c refers to the center of the mobile robot formation and k is the current time step. Fig. 1 illustrates four mobile robots traveling in a symmetric formation through the field domain. Therefore, given that the mobile robots maintain a symmetric formation, and are close to the formation center, the cooperative Kalman Filter can provide estimates of the concentration ($z(r_c^k, k)$) and gradients ($\nabla z(r_c^k, k)$) at the current formation center at the current time step, as well as the estimates of the concentration ($z(r_c^k, k-1)$) and gradients ($\nabla z(r_c^k, k-1)$) at the current formation center at the previous time step. As elaborated in Section 5.2.2 and shown in Equation (16), the state used for training of the deep reinforcement learning models contains the concentration and gradient estimates at the formation center produced by the cooperative Kalman Filter.

Similarly, we will not reproduce the mathematical derivation and proof of convergence for the recursive least-squares (RLS) algorithm developed for the estimation of the diffusion coefficient. Interested readers can refer to [9], [10] for the complete derivation and proof of convergence. To develop the RLS algorithm, the authors use the values of the information state at time steps k and $k+1$ from the filter and calculate the temporal variations of the field along the trajectory of the formation center, given by $\frac{z(r_c^k, k+1) - z(r_c^k, k)}{t_s}$, where t_s is the sampling frequency. Then, given an initial estimate of the diffusion coefficient, RLS can be applied to update that estimate according to the following equation:

$$\begin{aligned}
\hat{\theta}_k &= \hat{\theta}_{k-1} + g(k)\left(\frac{z(r_c^k, k+1) - z(r_c^k, k)}{t_s} - \nabla^2 z(r_c^k, k)\hat{\theta}_{k-1}\right) \\
g(k) &= \eta(k-1)(\nabla^2 z(r_c^k, k))^T \left[\nabla^2 z(r_c^k, k)\eta(k-1)(\nabla^2 z(r_c^k, k))^T + R_e\right]^{-1} \\
\eta(k) &= \left(I - g(k)\nabla^2 z(r_c^k, k)\right)\eta(k-1),
\end{aligned} \tag{8}$$

where $g(k)$ is the estimator gain matrix, $\eta(k)$ is the estimation error covariance matrix, and R_e is the noise covariance. The discrete Laplacian $\nabla^2 z(r_c^k, k)$ is calculated by using the sensor measurements directly from the mobile sensors using the following equation:

$$\nabla^2 z(r_c^k, k) \approx \frac{p(r_0^k, k) + p(r_1^k, k) + p(r_2^k, k) + p(r_3^k, k) - 4z(r_c^k, k)}{\Delta r^2}, \tag{9}$$

where $p(r_i^k, k),\ i = 0, 1, 2, 3$ are the sensor measurements from each of the four robots in the formation at time step k, and Δr is the distance between each of the robots to the formation center. The estimated diffusion coefficient is used in the discretized advection-diffusion Equation (30) for the reconstruction.

5 PROPOSED ALGORITHM

In this chapter, we propose an algorithm that provides a solution to the problem formulated in Chapter 3. The key idea of the proposed algorithm is to use learning assisted path-planning for the mobile sensor robot formation that enables it to take information-rich paths that enable efficient reconstruction of the evolution of the field, as well as identification of its advection-diffusion parameters.

Fig. 2 provides an overview of all the major components of this algorithm and how they are used. The blocks in purple are the components of the proposed algorithm that are explained in this chapter, while the blocks in gray are the preliminaries that are introduced in Chapter 4. At the beginning of the episode, a starting location is provided for the mobile robot formation. The robots measure the field values at their respective locations which are sent to the cooperative Kalman Filter (Chapter 4) and the recursive least squares (RLS) algorithm (Chapter 4). The measurements are interpolated to obtain estimates of field values inside the view-scope. Components from the output of the cooperative Kalman filter are used as the input state for the trained Deep Q-Network (introduced in Section 5.2), which then outputs the optimal action for the formation to take. The same state is also used by an algorithm that triggers identification of advection parameters (introduced in Section 5.3) when a stationary state is reached. Now, the advection parameters can be identified using the algorithm introduced in Section 5.4. Once the advection parameters are identified, a destination selector algorithm (introduced in Section 5.5) chooses a destination location for the formation to encourage exploration. Then, a controller transports the formation to the chosen destination. Once the destination is reached, the control of the formation is transferred back to the trained Deep Q-Network and the algorithm continues until the episode is terminated. The estimated field values and advection-diffusion parameters are used to reconstruct the field (elaborated in Section 5.6).

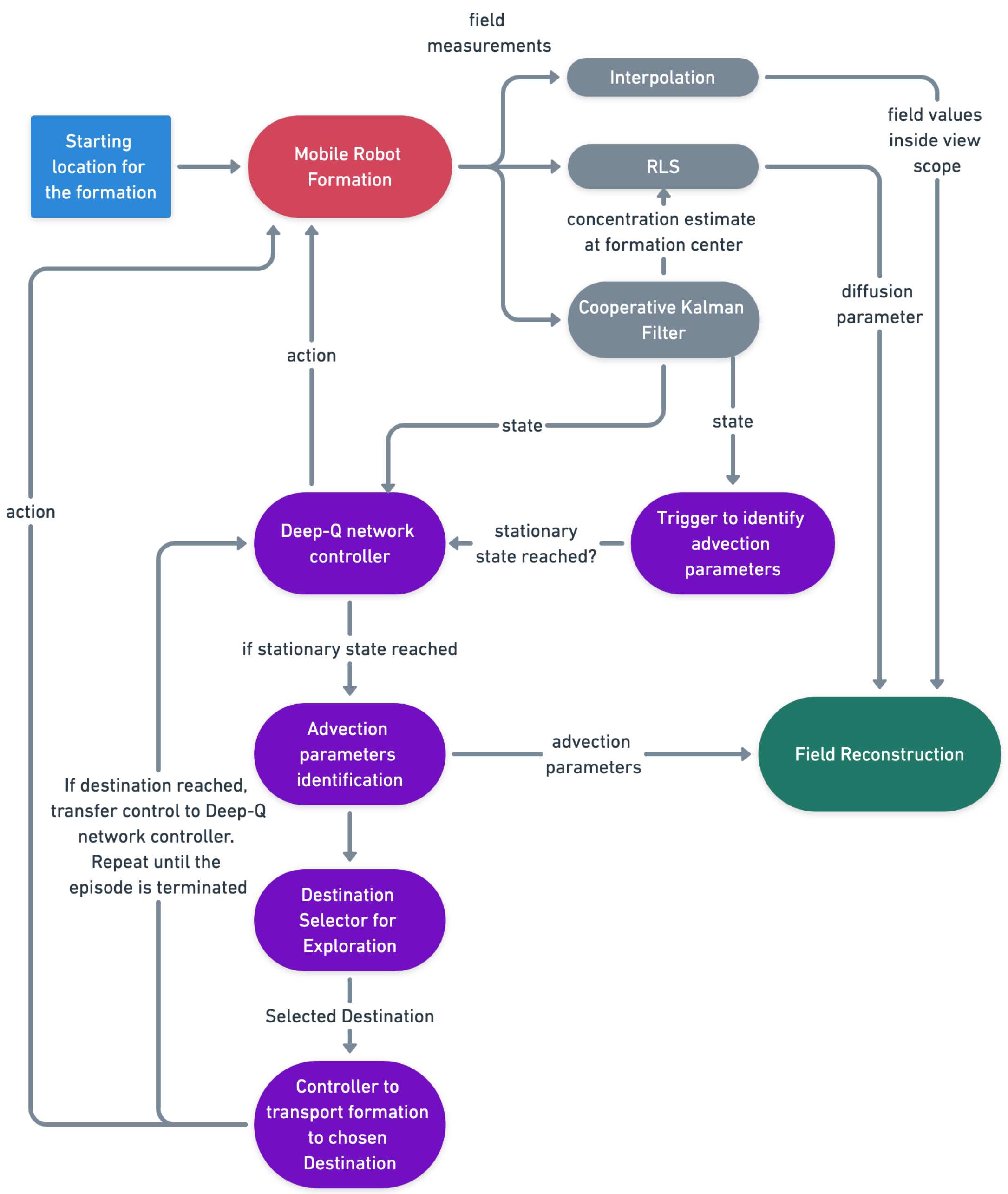

field measurements
Interpolation
Starting location for the formation
Mobile Robot Formation
RLS
field values inside view scope
concentration estimate at formation center
diffusion parameter
Cooperative Kalman Filter
action
state
state
action
Deep-Q network controller
stationary state reached?
Trigger to identify advection parameters
if stationary state reached
Advection parameters identification
advection parameters
Field Reconstruction
If destination reached, transfer control to Deep-Q network controller. Repeat until the episode is terminated
Destination Selector for Exploration
Selected Destination
Controller to transport formation to chosen Destination

Fig. 2. Flowchart providing an overview of the proposed algorithm. Inputs to the algorithm are marked in blue, the components of the proposed algorithm are marked in purple, and the blocks in gray are the preliminaries. The mobile robot formation being controlled is marked in red, while the output reconstructed field is marked in green.

5.1 Inputs to the Proposed Algorithm

The proposed algorithm is designed such that a minimum number of inputs are required from the user. The proposed algorithm requires the following two inputs for functioning in an advection-diffusion field domain:

1) Starting location for the center of the formation: The users of this algorithm need to provide a starting location for the center of the formation, $r_c^0 \in \Omega$. Since most practical applications would require the mobile sensor robots to start the field reconstruction task at some known starting location, this is a reasonable requirement. Importantly, the algorithm does not require the starting locations of the mobile robots and instead only requires the starting location of the center.
2) Maximum duration (T_{max}) of an episode: This is the maximum time for which the advection-diffusion field reconstruction task must run for.

5.2 Deep Reinforcement Learning-Based Path-Planning for Field Reconstruction

Reinforcement learning has been shown to solve a large variety of problems that involve optimal decision making. In general, these problems involve an agent acting in an environment which in return provides rewards as well as some observations for taking those actions. The agent's goal is to choose actions at each time step such that it can maximize the cumulative reward it receives. Since the agent does not know what reward it may receive from the environment on taking a particular action, it needs to learn the quality of taking that action through trial and error. The reward, which is a scalar value, is a way to incentivize the robot to learn what we want it to learn.

In this section, we develop a reinforcement learning based algorithm for controlling our mobile sensor robot formation to travel along information-rich trajectories that lead to accurate field reconstruction. It is important to note that the field of reinforcement learning is a collection of large number of algorithms and problem formulations; however, in this work we develop our algorithm using the family of Q-learning algorithms. First, we will introduce the idea behind tabular Q-learning, then we will elaborate on the components of our reinforcement learning problem formulation, show why tabular Q-learning is intractable for this problem, and then show how it can be extended to Deep Q-learning.

5.2.1 Tabular Q-Learning

Before introducing the idea of Q-learning, it is important to understand the ideas of state (s), state values ($V(s)$) and state-action values ($Q(s,a)$). The observations that the agent receives from the environment after taking an action is referred to as the state, and the set of all possible states for an environment is called its state space. The value of a state $V(s)$ can be thought of as the expected total reward that the agent can obtain from that state [39]. Formally,

$$V(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \tag{10}$$

where r_t is the award obtained at current time step t of the episode, $\mathbb{E}$ is the expectation operator and γ is a discount factor. While the problem can be formulated as non-episodic (or continuous), in this work we formulate it as an episodic learning problem in which each episode has a starting state and many possible termination states. In general, the goal of any reinforcement learning problem is to learn a policy π that maximizes the state value function $V(s)$ for all states. Formally, the policy is defined as a probability distribution of actions over all possible states in the state space, [39]

$$\pi(a|s) = P[A_t = a|S_t = s], \tag{11}$$

which can be read as the probability of choosing action a, when in state s at time step t. A good policy would assign a higher probability to an action that leads to a state with a large value, and in-turn a large reward.

Additionally, we can define a new quantity called state-action value, known as $Q(s,a)$. This value is the total reward that can be obtained by taking action a in state s and can be defined in terms of state values $V(s)$ as follows:

$$Q(s,a) = \mathbb{E}_{s'}[r(s,a) + \gamma V(s')], \tag{12}$$

where s' is the resulting state after taking action a in state s. Additionally, Equation (10) can be re-written as the following:

$$V(s) = \max_{a \in A} Q(s,a), \tag{13}$$

where A is the set of all actions. Now, combining Equations (12) and (13), we obtain a recursive definition for $Q(s,a)$ which can be used for learning:

$$Q(s,a) = r(s,a) + \gamma \max_{a' \in A} Q(s',a'). \tag{14}$$

The application of Equation 14 to learn state-action values for all sets of states and actions is known as Tabular Q-learning. It is important to note, however, that Equation (14) is not used directly in practice as replacing values at each step can make the learning process unstable. Instead, a blending approach is used where the previous value is updated towards the new value based on some learning rate α:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r(s,a) + \gamma \max_{a' \in A} Q(s',a')). \tag{15}$$

We do not include the entire Tabular Q-learning algorithm here since we do not use it in this work. The entire algorithm with pseudo-code is provided in [39].

5.2.2 States, Actions, Reward function and Termination Criteria

With the Q-learning framework defined, we can now develop the algorithm for learning information-rich paths in an advection-diffusion field. First, we need to define a representation of the field domain Ω. For the purpose of this training, we consider the field domain to be an $E \times F$ grid, which each grid cell representing a single location, r

and having a single concentration value at a given time, $z(r,t)$. Given K mobile sensor robots moving in a formation, the location of the centroid of the formation is represented by r_c and the area enclosed by the polygon formed by the formation represents the view-scope $\Gamma(t)$. Additionally, recall that we are employing the cooperative Kalman filter introduced in chapter 4 which provides us with estimates of concentration at formation center $z(r_c,t)$ as well as the gradients at formation center $\nabla z(r_c,t)$ at all time steps concentration measurements from mobile sensor robots are available.

We can now specify the definition of state $s(t)$ chosen for this algorithm:

$$s(t) = [z(r_c,t), \nabla z_x(r_c,t), \nabla z_y(r_c,t)], \tag{16}$$

where $\nabla z_x(r_c,t), \nabla z_y(r_c,t)$ are the concentration gradient estimates at formation center r_c at time t in the x and y directions respectively. The definition of the input state vector is critical to successful learning of the state-value function. Here, $z(r_c,t)$ provides an estimate of the value of the current location at the current time step – a high $z(r_c,t)$ value indicates that the formation is currently in a highly "polluted" environment and contributes a large amount to the reconstruction, whereas a small value indicates that the current location is not as "valuable" and contributes less to the reconstruction. The gradients, $\nabla z_x(r_c,t), \nabla z_y(r_c,t)$ provide an estimate of the direction of largest concentration value change at r_c, which can be useful in determining which action would result in obtaining the most information from the field. It is important to note that the formation center r_c, or the location of any of the mobile robots is not part of the state vector s. This is important for generalizability of the algorithm since we do not want our learned model to associate the value of an action based on the location of the formation in Ω. Specifically, an ideal action to take at the same location $r \in \Omega$ might be different for different advection-diffusion fields. Similarly, it is important to note that time step t is not part of the state vector s. This follows the same reasoning that the ideal action to take at the same time step in different fields may be different. Therefore, our definition of the state vector s constraints the model to learn state-action values based only on the field characteristics.

With the state vector s defined, we can now define the set of actions A that the robots can take to move in the field domain Ω. As mentioned previously, the problem formulation allows us to plan paths solely for the formation center r_c instead of planning paths individually for each of the robots. Thus, it is reasonable to think of actions being taken by the formation center r_c instead of each of the robots in the formation, and that the robots replicate the action applied to the formation center to maintain that formation. The action space A consists of 9 actions and is defined as follows:

$$A = \{\text{“up”, “down”, “left”, “right”, “up-left”, “up-right”, “down-left”, “down-right”, “stay”}\}. \tag{17}$$

As evident from Equation (17), in a single time step, robots can move to any of their adjacent cells (including diagonals) or choose to stay in the current cell. This action-space allows the reinforcement learning model to move the formation as flexibly as possible.

As mentioned previously, we formulate the reinforcement learning problem as an episodic task. Thus, it is important to define termination criteria for each of the episodes. In our formulation, we have the following two termination criteria:

1) We impose a max time limit on the duration of an episode specified by T_{max}. This limit is important since advection-diffusion fields eventually diffuse away and the concentration values within the field domain Ω converge to 0. Thus, it is not important to continue the task until time reaches infinity. Additionally, mobile robots have limited access to power source and may only be able to operate for a certain length of time.
2) If any of the mobile robots in the formation move outside of the field domain Ω before T_{max} is reached, the episode terminates.

Therefore, we want the mobile robots to operate for the maximum time possible (T_{max}) without any of the robots in the formation exiting the field domain Ω. These incentives are encoded in the reward function.

One of the most important components of a reinforcement learning problem is the definition of a reward function as it can severely impact the quality of the training. In this work, we want the model to incentivize moving the formation on information-rich trajectories while running for the entire episode length of T_{max}, and ensuring that all robots in the formation remain inside the field domain Ω for the duration of the episode. Thus, to meet these requirements the following piece-wise reward function is developed:

$$R(t) = \begin{cases} a\sum_{r\in\Gamma(t)} z(r,t) & t < T_{max} \\ -R_{max} & r_k(t) \notin \Omega \ \forall k = 1,2,3...N \\ +R_{max} & t = T_{max}, \end{cases} \tag{18}$$

where, a is scalar coefficient to weigh the reward, R_{max} is some large scalar value, N is the number of mobile robots in the formation and $\Gamma(t)$ is the view-scope of the formation. Here, a large negative reward of $-R_{max}$ is provided if any of the mobile robots $r_k \ \forall k = 1,2,3...N$ leaves the field domain Ω before the episode is over, incentivizing the model to run the episode till T_{max} is reached. A large positive reward of $+R_{max}$ is provided when $t = T_{max}$ is reached. This reward indicates to the model that we want it to collect field values for the entire length of the episode. For the duration of the episode before termination, a reward proportional to the sum of all concentrations inside the view-scope $\Gamma(t)$ is returned. Recall that we are interpolating robot sensor measurements at each time step to obtain estimates of the concentration values $z(r,t), r \in \Gamma(t)$. Since regions with large concentration values contribute more to the field reconstruction, a reward proportional to the sum of concentration values inside the view-scope encourages the model to learn to move towards areas with large concentration values which reduce the error in reconstruction (Equation (28)) to a greater extent, and thus are information-rich.

5.2.3 Deep Q-Network (DQN) Learning

In this section, we first discuss why tabular Q-learning is insufficient for solving this problem, then we introduce how neural networks can be used as state-value function approximators and finally show how we adapt this problem to be solved using Deep Q-Network (DQN) Learning. As explained in the previous section, the state s is composed of three floating-point values, however, the tabular Q-learning approach expects discrete states and actions and stores the state-action values in a tabular form with states and actions as keys. While the combination of all values states can take is finite as they're represented by a finite number of bits, this number is extremely large and can lead us into memory constraints [39]. Therefore, storing the Q-values for each state-action pair quickly becomes intractable. To solve this issue, we can frame the learning problem as a regression task of approximating the state-action value function with a deep neural-network.

Fig. 3 shows the architecture of the neural-network we chose for this purpose. This network consists of three hidden layers of size 512 each, an input layer of size 3 which takes the state s as input, and an output layer of size 9, which outputs the estimate of state-action value for each of the 9 actions. The layers are fully-connected with random dropouts to help with generalizability, and the non-linear activation function of choice is rectified linear unit (ReLU). This network is trained using the stochastic gradient descent (SGD) algorithm [40] using interactions from the advection-diffusion field environment. Training implementation details are provided in Chapter 6.

Finally, we can define the loss function that we want to minimize during the training of this neural network. Since we want the network to provide state-action value estimates for all possible actions from a state, our target y for the neural-network loss should be defined as follows,

$$y = \begin{cases} r & \text{if episode has ended} \\ r + \gamma \max_{a' \in A} \hat{Q}(s', a') & \text{otherwise,} \end{cases} \tag{19}$$

where r is the immediate reward, s' is the next state received from interaction with the environment, γ is the discount factor and $\hat{Q}$ is our current best estimate of the state-action value function. Then, we can define the loss function to be minimized as follows,

$$\mathscr{L} = (Q(s,a) - y)^2, \tag{20}$$

where $Q(s,a)$ is the output of the neural-network being trained. The neural-network is trained until the loss function value converges.

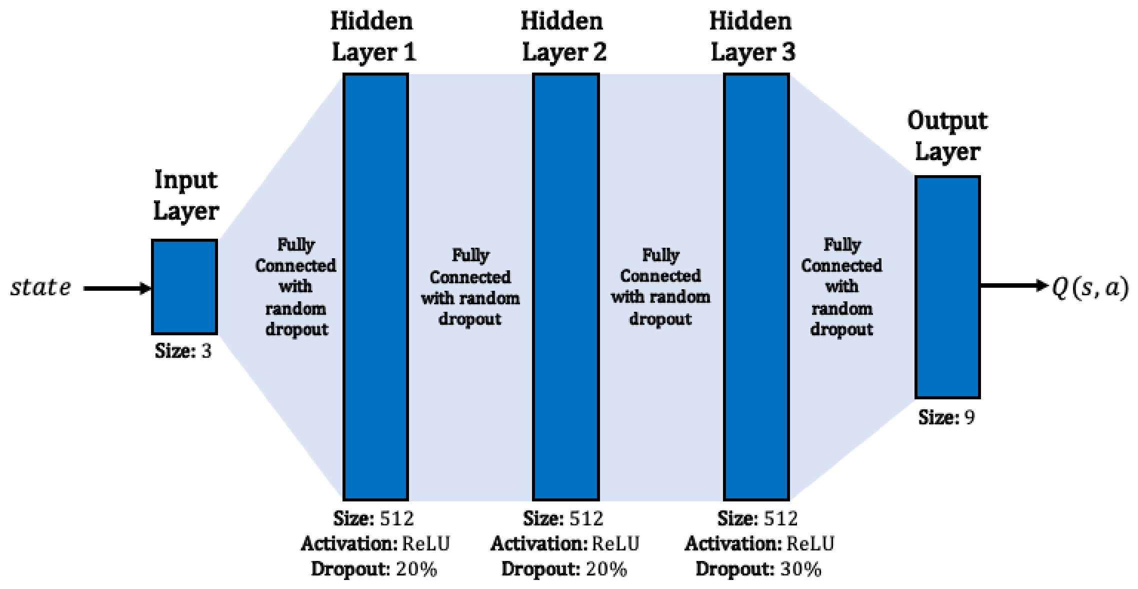


Fig. 3. Neural-Network used for Deep-Q Learning function approximation.

5.3 Procedure for Triggering Identification of Advection Parameters

As explained in the previous section, the Deep Q-Network based control is designed to move the robot formation along information-rich trajectories that minimize the field reconstruction error (Equation (28)). At the beginning of the task, the DQN controller is responsible for directing the mobile robot formation, and it will make the formation travel on information rich paths until a stationary state is reached. A stationary state is characterized by flat estimated gradients at the formation center, that is, $\nabla z(r_c,t)$ tends towards 0, while the estimated concentration at the formation center $z(r_c,t)$ stays high. At this time, the formation has reached a stationary state in the advection-diffusion field and is traveling at approximately the same velocity as the local field, making it an ideal state to estimate the field's advection parameters at. However, the DQN controller itself cannot identify when such a stationary state is reached, and an additional procedure is required to identify such a state and trigger identification of advection parameters.

Given that both the estimated concentration value at the formation center $z(r_c,t)$, and the gradients at the formation center $\nabla z(r_c,t)$ are provided to us at all times t by the cooperative Kalman Filter (introduced in Chapter 4), we can develop an averaging-window based algorithm to detect when a stationary point is reached, and the formation must be directed to search for other information-rich trajectories.

Let W specify the length of the averaging-window we desire, then buffers of length W for each of the state variables $(z(r_c,t), \nabla z_x(r_c,t), \nabla z_y(r_c,t))$ are created. We use double-ended queues (also known as deque) as the choice of data structure for these buffers. These buffers contain only the most recent W values inserted into them, which makes them a good choice for moving-averages. At each time step when the state output is available from the advection-diffusion field, the buffers are populated with their corresponding state values. Then, the arithmetic means for the values in the buffers are calculated. Let $\mu(z(r_c,t)), \mu(\nabla z_x(r_c,t))$ and $\mu(\nabla z_y(r_c,t))$ denote the moving averages for each of the state variables respectively, then, a stationary state is reached when $\mu(z(r_c,t)) \geq z_{min}$, $|\mu(\nabla z_x(r_c,t))| \leq \varepsilon$ and $|\mu(\nabla z_y(r_c,t))| \leq \varepsilon$, for some minimum field concentration $z_{min} > 0$ and some small $\varepsilon > 0$. The procedure for this algorithm is described below in Algorithm 1.

Algorithm 1 Procedure for detecting when Stationary State has been reached and identification of advection coefficients can begin

```
$\mathrm{buf}_z = \mathrm{deque}(W)$        ▷ Initialize buffers for all state variables of max length $W$
$\mathrm{buf}_{\nabla z_x} = \mathrm{deque}(W)$
$\mathrm{buf}_{\nabla z_y} = \mathrm{deque}(W)$
**procedure** StationaryStateReached($s$, $z_{min}$, $\varepsilon$)
    $\mathrm{buf}_z.\mathrm{insert}(s[0])$        ▷ Insert state variables into their buffers
    $\mathrm{buf}_{\nabla z_x}.\mathrm{insert}(s[1])$
    $\mathrm{buf}_{\nabla z_y}.\mathrm{insert}(s[2])$
    $\mu(z(r_c,t)) \leftarrow \frac{1}{|\mathrm{buf}_z|}\sum_{i=0}^{|\mathrm{buf}_z|}\mathrm{buf}_z[i]$        ▷ Calculate mean values
    $\mu(\nabla z_x(r_c,t)) \leftarrow \frac{1}{|\mathrm{buf}_{\nabla z_x}|}\sum_{i=0}^{|\mathrm{buf}_{\nabla z_x}|}\mathrm{buf}_{\nabla z_x}[i]$
    $\mu(\nabla z_y(r_c,t)) \leftarrow \frac{1}{|\mathrm{buf}_{\nabla z_y}|}\sum_{i=0}^{|\mathrm{buf}_{\nabla z_y}|}\mathrm{buf}_{\nabla z_y}[i]$
                                                        ▷ If conditions are met, return true
    **if** $\mu(z(r_c,t)) \geq z_{min}$ and $|\mu(\nabla z_x(r_c,t))| \leq \varepsilon$ and $|\mu(\nabla z_y(r_c,t))| \leq \varepsilon$ **then**
        **return** true
    **return** false
```

Once the above described algorithm returns true, the velocity of the formation is expected to match the velocity of the field and thus, we can begin estimating the advection coefficients of the field.

5.4 Identification of Advection Coefficients

As explained in the previous section, Algorithm 1 allows us to detect when the formation has reached a stationary state in the advection-diffusion field where the velocity of the movement of the formation matches the velocity of the local field. Since stationary states are characterized with very small gradients and high concentration values, and the DQN controller is designed to direct the formation towards states with such values, at such states the velocity of the formation as controlled by the DQN controller must represent the velocity of the advection-diffusion field. This allows us to estimate the advection coefficients $\hat{\mathbf{v}} = [\hat{v}_x, \hat{v}_y]$ as the average velocity of the mobile robot formation for M time steps after a stationary point has been found.

Let $k_S > 0$ be a time step at which a stationary state has been identified using Algorithm 1, and $M > 0$ be the number of time steps we want to estimate the formation velocity for, then the following equations provide us with estimates of the advection coefficients:

$$\begin{aligned} \hat{v}_x &= \frac{r_{c,x}(t = k_S + M) - r_{c,x}(t = k_S)}{M} \\ \hat{v}_y &= \frac{r_{c,y}(t = k_S + M) - r_{c,y}(t = k_S)}{M}, \end{aligned} \tag{21}$$

where, $r_c \in \Omega$ is the location of the formation center as controlled by the DQN controller. With these advection coefficient estimates, and the diffusion coefficient estimate obtained as explained in Chapter 4, we have all the information to begin reconstructing the advection-diffusion field and is explained in Section 5.6.

5.5 Destination Selection for Further Exploration

Once the mobile robot formation has reached a stationary state and identified advection coefficients, it is important to encourage the formation to explore other information-rich trajectories in the field domain. Many environmental processes may contain multiple high-concentration zones that may lead to complex concentration surfaces, all of which will need to be measured by the formation in order to produce an accurate reconstruction. Since the DQN controller is designed to solely follow information-rich trajectories, it will not encourage the formation to move away from local information-rich paths to search for other trajectories that may improve the accuracy of the reconstruction further. In that case, the formation is likely to never leave a local information-rich path and the reconstructed field will be highly inaccurate. In this section, we develop an algorithm to encourage exploration in search for other information-rich trajectories away from visited areas in the field domain Ω. The output of this algorithm would be a destination $r_d \in \Omega$ that the formation will travel to, and then recommence following information-rich paths using the DQN controller.

The destination selection algorithm must select locations as destinations that follow the following criteria:

1) The selected destination must be away from already explored locations, if any.
2) The algorithm must be biased towards choosing unexplored areas or regions within field domain Ω.
3) The selected destination must be far away enough such that the DQN controller can identify previously unidentified information-rich paths if they exist.

To develop this algorithm, let's first define a distance d that we want the chosen destination to be far away from the current formation center location (r_c). Then, given this distance d, we build a set D of candidate destinations. Among the candidate destinations, any locations that lie outside of the field domain Ω or are previously visited will be removed. Let D_u denote this pruned set of unvisited candidate locations. The next task is to choose a destination among these candidate locations that follows our criteria. To do this, for ith candidate c_i in D_u, we assign a weight w_i that will then be used as the weight for this candidate for weighted random sampling using the following equation:

$$w_i = \text{similarity}(r_c - \mathbf{C}_u, c_i - \mathbf{C}_u)\|c_i - r_{v,i}\|_2, \ \forall c_i \in D_u, \tag{22}$$

where $r_{v,i}$ is the closest visited location to the candidate c_i in the direction of r_c, $\mathbf{C}_u$ is the centroid of the unvisited regions U in the field domain Ω and similarity is the cosine similarity operator which performs the following operation given two n-dimensional vectors $\mathbf{A}$ and $\mathbf{B}$:

$$\text{similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}, \tag{23}$$

and determines cosine of the angle between the two vectors. Thus, the similarity$(r_c - \mathbf{C}_u, c_i - \mathbf{C}_u)$ factor in Equation (22) measures the cosine of the angle formed between the $r_c - \mathbf{C}_u$ vector (vector between formation center's current location and the centroid of the unvisited regions) and the $c_i - \mathbf{C}_u$ vector (vector between the *i*th candidate and the centroid of the unvisited regions). Therefore, the candidates in the direction towards $\mathbf{C}_u$ are weighted higher than candidates in directions away from it. The $||c_i - r_{v,i}||_2$ term in Equation 22 measures the Euclidean distance between the *i*th candidate c_i and the closest visited location to it in the direction of r_c. Thus, among candidates that are towards the centroid of unvisited regions, candidates farther from the visited regions are weighted higher. Fig. 4 illustrates an example scenario for destination selection.

To convert the assigned weights into a probability distribution that can be used to randomly draw a destination, we exponentiate and normalize the weights using the following equation:

$$\Pr(c_i) = \frac{e^{w_i}}{\sum_{i=0}^{|D_u|} e^{w_i}}. \tag{24}$$

Here, the exponentiation operation helps with assigning a higher weightage to better candidates and reducing weightage from unfavorable candidates leading to better randomly drawn locations. The resulting probability values are then used as probability distribution for drawing of the chosen destination. The procedure for this algorithm is described below in Algorithm 2.

Once the destination is selected, the formation must be controlled to reach that destination. Recall that the mobile sensor robots follow single-integrator dynamics as explained in Equation (4). Therefore, constant velocity commands $u(t)$ are given to each of the robots in the formation until the formation center r_c reaches the chosen destination. Once the chosen destination is reached, the control of the formation movement is transferred back to the DQN controller so that information-rich paths can continue to be tracked.

Algorithm 2 Procedure for selecting destination for further exploration

```
procedure CHOOSEEXPLORATIONDESTINATION($r_c, d, U$)    ▷ Given current location, distance and unvisited regions, select destination
    $D \leftarrow$ candidate destinations $d$ distance away from $r_c$
    $D_u = []$
    for $c_i \in D$ do
        if $c_i \in \Omega$ and $c_i \in U$ then                ▷ Prune set of candidates
            $D_u$.insert($c_i$)
    $\mathbf{C}_u = \frac{1}{|U|} \sum_{i=0}^{|U|} r_i$          ▷ Calculate centroid of unvisited regions
    weights = []                                          ▷ Initialize empty weights array
    for $c_i \in D_u$ do
        $r_{v,i} \leftarrow$ trace ray from $c_i$ to $r_c$ to find first visited location
        $w = \text{similarity}(r_c - \mathbf{C}_u, c_i - \mathbf{C}_u) \|c_i - r_{v,i}\|_2$
        weights.insert($w$)
    probabilities = []                                    ▷ Initialize probabilities array
    for $w_i \in$ weights do
        $p = \frac{e^{w_i}}{\sum_{i=0}^{|D_u|} e^{w_i}}$
        probabilities.insert($p$)
    $idx \leftarrow$ randomly drawn index with probabilities as probability distribution
    return $D_u[idx]$
```

5.6 Advection-Diffusion Field Reconstruction

As stated previously, and as shown in Fig. 2, the task of field reconstruction consists of two major sub-tasks: (1) identification of the advection-diffusion parameters governing the unknown field, and (2) state estimation of the concentration field in the chosen spatial domain. In this section, we describe how we use the algorithms developed in this chapter, and the preliminaries described in Chapter 4 to reconstruct the unknown advection-diffusion field.

As the mobile sensor robots travel through the field domain in a formation, they collect measurements along their trajectories. At any time step, as mentioned previously, the field measurements from the sensors are interpolated to provide estimates of the state of the field within the view-scope. Since the field is a spatial-temporal varying dynamic field, these interpolated estimates, are only valid for the current time step. However, these collected field estimates can be evolved over time using the advection-diffusion field Equation (2) if we had the knowledge of the advection parameters **v** and diffusion parameter θ, which makes the task of reconstruction possible by using only a few mobile robots. Thus, the field measurements are also used by the Recursive Least Squares (RLS) method introduced in Chapter 4 to estimate the diffusion parameter, θ. The cooperative Kalman Filter (introduced in Chapter 4) also uses the measurements to provide estimates of the information state in Equation (7), which is further used by the Deep Q-Network to provide the action for the formation to take. As the formation reaches a stationary state in the field, the process for identification of advection parameters is triggered. The advection parameters, **v**, are then identified using the process explained in Section 5.4.

Applying Equation (2) for the task of reconstruction with estimated advection-diffusion parameters takes the following form:

$$\frac{\partial \hat{z}}{\partial t}(r,t) = \hat{\theta}\nabla^2 \hat{z}(r,t) + \hat{\mathbf{v}}\nabla \hat{z}(r,t), \quad r \in \Omega, \tag{25}$$

where, $\hat{\theta}$ is the estimated constant diffusion coefficient, $\hat{\mathbf{v}}$ is the estimated constant 2D advection parameter vector, and $\hat{z}(r,t)$ is the reconstructed field concentration function. To be able to apply Equation (25) for propagating the field, we need to have the initial and boundary conditions defined. Assuming that we have no knowledge of the field at the start of the experiment, the initial condition can be stated as:

$$\hat{z}(r,0) = 0, \; r \in \Omega, \tag{26}$$

over the entire field domain. Additionally, as stated earlier, in many practical environmental monitoring applications, the field domain Ω is much larger than the robots' size, and so the boundary can be modeled as a flat surface. This allows us to assume

Dirichlet boundary conditions on the boundary $\partial\Omega$ [6],

$$\hat{z}(r,t) = 0,\ \ r \in \partial\Omega. \tag{27}$$

Now, at each time step k, at which robots measure the concentration field, we obtain estimates of the field concentration function inside the view-scope, that is, $\hat{z}(r,k)$ $r \in \Gamma(k)$. Therefore, at each such time step k, we can populate the estimated concentration values at the current view-scope and apply Equation (25) to propagate the field. This process allows us to reconstruct the field with only sparse measurements along the robots' trajectories. Please note that in practice, we discretize Equation (25) over space and time to evolve the field spatially and temporally. Please refer to Section 7.1 which elaborates on how the field is reconstructed in practice as it requires the introduction of the Low-Fidelity Simulation Environment (Section 6.1). Equations (31) and (32) show how Equation (25) is discretized spatially and temporally for the reconstruction process.

As evident from Equation (25), the accuracy of identifying advection-diffusion parameters is crucial in the quality of the reconstruction over time. Inaccurate advection-diffusion parameters will cause the reconstructed field to diverge from the true field over time and deteriorate the quality of the reconstruction. To evaluate the quality of the reconstruction, we use the mapping error, also known as field reconstruction error, as a metric. The mapping error at time step k is defined as follows:

$$e_M(k) = \sum_{r\in\Omega} |z(r,k) - \hat{z}(r,k)|, \tag{28}$$

where, $z(r,k)$ is the true field concentration function at time step k and the $\hat{z}(r,k)$ is the concentration function for the reconstructed field at time step k. Therefore, in a successful reconstruction, the mapping error $e_M(k)$ should decrease over the course of the episode.

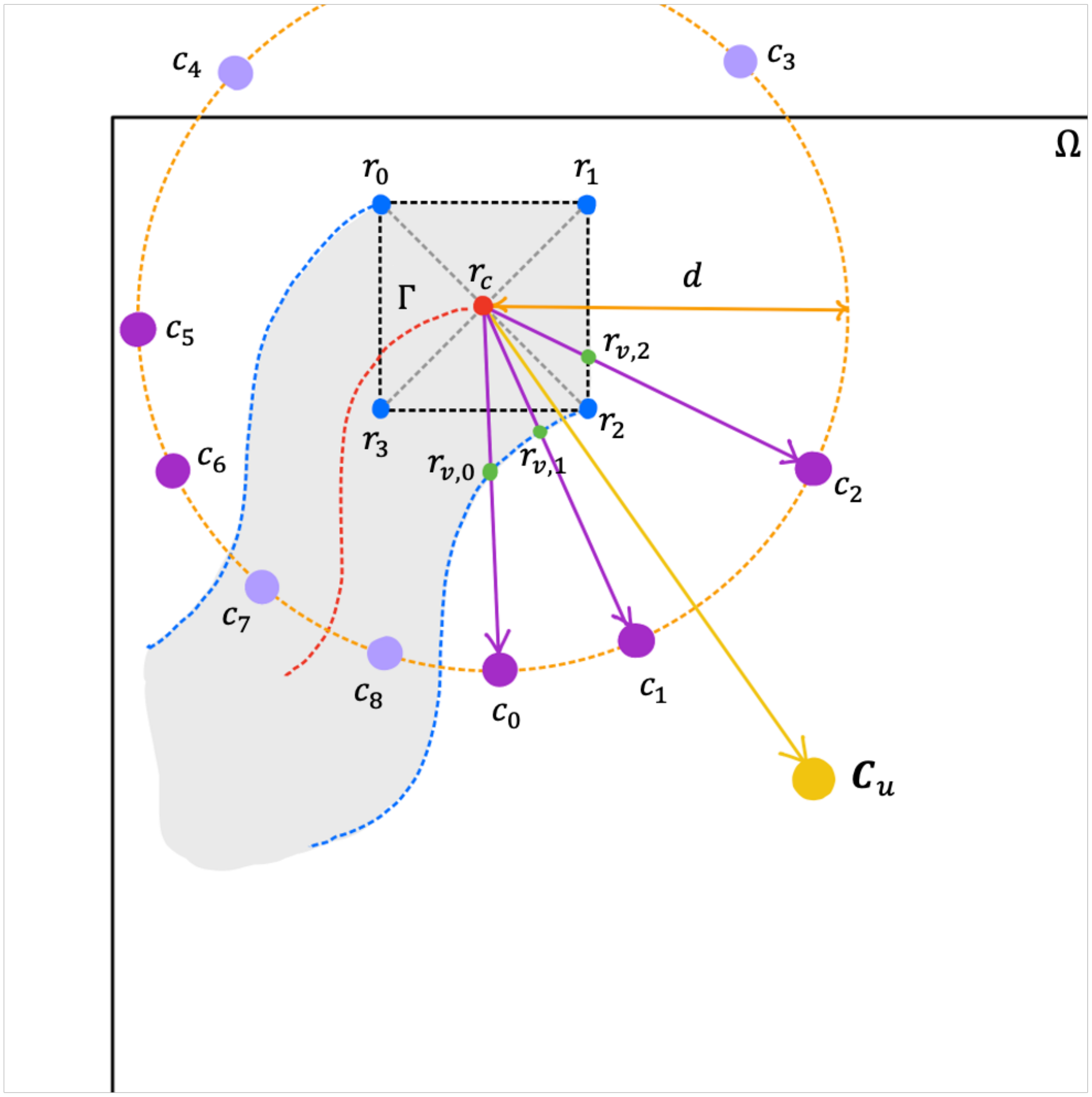


Fig. 4. Illustration demonstrating the destination selection algorithm. In this figure, r_0, r_1, r_2 and r_3 are the four mobile sensor robots making the formation, while r_c is the formation center. The area inside the black dashed line marks the view-scope Γ. The yellow location $\mathbf{C}_u$ denotes the centroid of the unvisited regions within Ω. The region marked in gray has been previously visited by the formation. The dashed orange circle denotes all locations d distance away from r_c which form the candidate set D. A few sample candidates from this set are labeled with $c_i, i = 0, 1, 2...8$. Among the sample candidates, c_3 and c_4 are removed from consideration since they lie outside the field domain Ω, while c_7 and c_8 are removed from consideration since they lie within the visited region. Eventually, c_0, c_1, c_2, c_5 and c_6 are added to set D_u and considered as candidate destinations. Locations $r_{v,0}, r_{v,1}$ and $r_{v,2}$ denote the closest visited locations in the direction of r_c for candidates c_0, c_1 and c_2 respectively. For this example, candidate c_1 would achieve the largest probability for being selected since it is towards $\mathbf{C}_u$.

6 TRAINING IN LOW-FIDELITY SIMULATION ENVIRONMENT

Having proposed the algorithm in Chapter 5, we can now move to the experimentation and implementation of the algorithm. In this chapter, we first introduce the *Low-Fidelity Simulation Environment* and then show how we set it up to be used for training of the deep reinforcement learning model proposed in Section 5.2. Then we provide training as well as testing results if we were to use the trained deep reinforcement learning model solely for path-planning. The Low-Fidelity Simulation Environment introduced here will also be used for testing of the complete proposed algorithm, and the results are presented in Chapter 7.

6.1 Low-Fidelity Simulation Environment

The Low-Fidelity Simulation Environment is a 2D environment that represents the field domain Ω as a discretized $E \times F$ matrix with $E = F = 100$, with each grid cell $r \in \Omega$ holding a concentration value $z(r,t)$ at time t. Since one of the major purposes of this environment is to enable training of the deep reinforcement learning model, it is set up as an OpenAI Gym [41] environment, which allows us to use a standard interaction interface during the learning task.

6.1.1 Advection-Diffusion Field Representation

As introduced in Chapter 3, an advection-diffusion field is a special spatial-temporal varying field expressed by the partial differential Equation (2). Therefore, given the initial state of the field $z(r,0) \forall r \in \Omega$, Equation (2) provides us with how the field evolves over space and time for certain advection and diffusion coefficients $\mathbf{v}$ and θ, respectively. Since the field representation is discretized spatially, the Equation (2) will need to be discretized spatially and temporally to be applied to this field. Consider the 3×3 section of the discretized advection-diffusion field representation in Fig. 5. Then, using the finite difference method, we can discretize Equation (2) as follows:

$$\frac{z(r_0,k+1)-z(r_0,k)}{t_s} = \theta\left[\frac{z(r_2,k)+z(r_4,k)-2z(r_0,k)}{\Delta r_x^2}+\frac{z(r_1,k)+z(r_3,k)-2z(r_0,k)}{\Delta r_y^2}\right]+ \mathbf{v}^T\nabla z(r_0,k)+e(r_0,k), \tag{29}$$

where k is the discretized time stamp, t_s is the sampling interval and $e(r_0,k)$ accounts for the approximation error. Assuming square grid cells, i.e., $\Delta r_x = \Delta r_y$, Equation (29) simplifies to the following.

$$\frac{z(r_0,k+1)-z(r_0,k)}{t_s} = \theta\frac{\sum_{i=1}^{4} z(r_i,k)-4z(r_0,k)}{\Delta r_x^2}+\mathbf{v}^T\nabla z(r_0,k)+e(r_0,k). \tag{30}$$

Thus, by choosing advection and diffusion coefficients $\mathbf{v}$ and θ we can simulate the evolution of an advection-diffusion field applying Equation (30) for each grid cell $r_0 \in \Omega$. Fig. 7 shows the evolution of an example advection-diffusion field over 300 time steps.

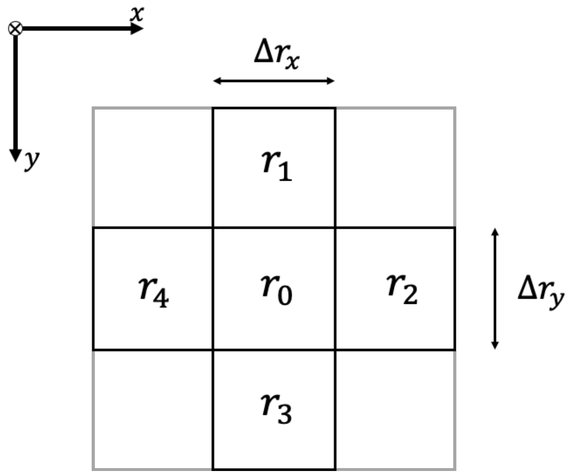


Fig. 5. A small 3×3 section of the discretized advection-diffusion field representation.

6.1.2 Setup as OpenAI Gym Environment

As introduced earlier, the advection-diffusion field is implemented as an OpenAI Gym [41] environment in Python 3. Using OpenAI Gym provides a number of benefits and conveniences for the reinforcement learning task. First, it provides standard interaction patterns between the agent and the environment. These patterns make it easy to develop and test different algorithms on the same environment or the same algorithm on different environments and quickly compare performance and results. Second, OpenAI gym environments provide concept of `Space` which is used for defining state and action spaces called `observation_space` and `action_space`, respectively. These spaces define the valid format of states and actions, respectively. We use the derived `Discrete` space for defining the action space. This space allows us to map the 9 actions in Equation (17) as non-negative integers that can then be sampled from the `action_space` object. For the `observation_space` we use the derived `Box` space. We define `observation_space` as 3-dimensional bounded box of floats corresponding to our definition of the state in Equation (16). This object allows us to sample a random valid state and check the validity of a state with ease.

6.2 Deep Q-Network (DQN) Training Setup

Having described the setup of the advection-diffusion field environment, we can now begin the training of the Deep Q-Network (DQN) based on the formulation described in Section 5.2. In this section, we first address some implementation issues with DQN training and then provide the full training algorithm.

6.2.1 Exploration using ε-greedy algorithm

For the purposes of training the neural network, we need to interact with the environment and collect rewards. Fig. 6 shows the steps involved in a single interaction with the environment. Therefore, for an episode to proceed, the agent needs to make a decision on which action it must take. If we had a good policy $\pi(a|s)$, we could simply use this policy to make this decision, however, since the goal of training is to obtain a good policy, we cannot assume the existence of such a policy [39]. An alternative approach is to sample a random action from the `action_space` at each time step to interact with the environment, however, it is likely that the agent will spend time exploring states that are not very useful and the training may take a long time to converge. While random exploration is important for the discovery of good actions to take from a state at the beginning of training, it is also important to use what the model has learned thus far during the later stages in the training.

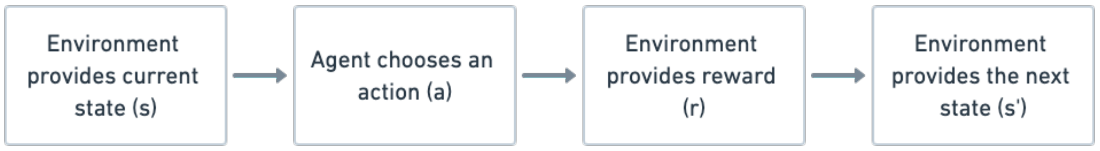

Environment provides current state (s)
Agent chooses an action (a)
Environment provides reward (r)
Environment provides the next state (s')

Fig. 6. A single agent-environment interaction.

A way to combine the two approaches is to use the ε-greedy method for action selection, where ε is a training hyperparameter that is reduced over the duration of the training. In this method, during each interaction a random floating point number α is drawn from a uniform distribution. If $\alpha < \varepsilon$, a random action is sampled from the `action_space` and used for the interaction. Otherwise, the trained neural network is queried using the current state s to produce an estimate of the state-action values $(Q(s,a))$, and the action corresponding to the largest Q-value is chosen. The ε training parameter is reduced linearly over the course of the training which encourage exploration by enabling usage of random behavior at the beginning of training, while exploiting the previously learned behavior more as the training progresses [39].

6.2.2 Experience Replay Buffer

As noted in Section 5.2, we will be using the stochastic gradient descent (SGD) algorithm [40] for training the neural network, with Equation (20) serving as the loss function to minimize, effectively converting the DQN training problem into a supervized learning problem. However, the usage of SGD algorithm requires that the training data be independent and identically distributed (i.i.d.), meaning that a batch of training samples must be independent from each other and must represent the overall distribution of training data. Therefore, the way we build batches of training data for SGD is important. It is important to note that the requirement of i.i.d. is not fulfilled if we use interactions from a single episode in a batch due to the following reasons:

1) Consecutive interactions in an episode are not independent as the resulting state for the first interaction is the input state for the next.
2) Distribution of the training data will be different from the optimal policy we wish to learn [39]. Instead the data will be based on the ε-greedy policy introduced in the previous section.

To counter this issue, we create a large buffer called the *Experience Replay Buffer* using a double-ended queue (also known as deque) data structure that stores tuples of agent-environment interactions in the form (s, a, r, s') of a given length B. This buffer holds the B most recent interactions from multiple recent episodes. Then, to build our batch of training data, we sample random interactions from this buffer, which gives us interactions that are more likely to be independent from each other, while providing us with fresh experience to learn from.

6.2.3 Issue with Bootstrapping

As evident from the Q-learning Equation (14), the $Q(s,a)$ (value of the current state-action pair) is improved using $Q(s',a')$ (value of the next state-action pair). This process is known as bootstrapping since the improvement of the current state-action pair is dependent on the estimate of the next state-action pair [39]. The issue with using bootstrapping with neural-networks is that the states s and s' are only a single step apart and very similar to each other [39]. Since neural networks are unable to discriminate between states that are very similar [42], updating the weights of the neural network to make $Q(s,a)$ closer to our target value, may inadvertently also change the state-action values of s' and other nearby states. It is important to note that this is a special property of using neural networks for function approximation, where update in the weights impacts multiple state-action values, as opposed to Tabular Q-learning where updating one state-action value does not impact any other state-action pairs. This issue with neural-networks can make the training very unstable where in an effort to improve one state-action value we worsen nearby state-action values.

To counter this issue, we need to separate the model that is being trained and the model where we extract the target state-action value $Q(s',a')$ from, called the *target network* [39]. This network is a copy of the network being trained and the weights between the two networks are synchronized every T_{sync} interactions with the environment. By doing so, updating the weights in our training network to train for $Q(s,a)$ does not impact the value of the target state-action pair and helps with the stability of the training.

6.2.4 DQN Training Pseudo-code

With all the sub components of the training defined, we are now ready to provide the entire DQN training algorithm. This algorithm is a modified version of the algorithm provided in [39] and is provided in Algorithm 3 below.

6.3 Training Results

In this section, we provide results of training the DQN model with a variety of advection-diffusion field simulations as performed in the Low-Fidelity Simulation Environment introduced in Section 6.1. The advection-diffusion field simulations will differ in their initial state $z(r,0)$ $r \in \Omega$ and their advection and diffusion parameters $\mathbf{v}$ and

Algorithm 3 DQN Training algorithm

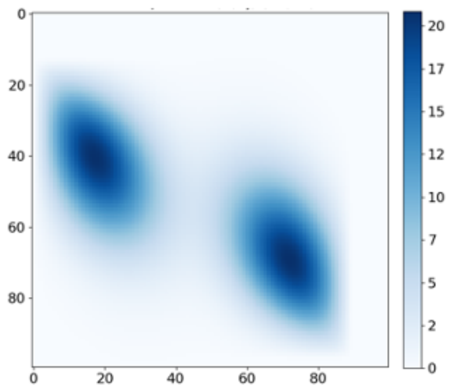
```
procedure DQNTraining(buf_len, max_steps, batch_size, $T_{sync}$)
    train_net $\leftarrow$ DQN()    ▷ Initialize training network with random weights
    target_net $\leftarrow$ DQN()    ▷ Initialize target network with random weights
    env $\leftarrow$ AdvectionDiffusionEnvironment()    ▷ Initialize the environment
    $\varepsilon \leftarrow 1.0$    ▷ Initialize $\varepsilon$
    $\text{buf}_{replay}$ = deque(buf_len)    ▷ Initialize experience replay buffer
    step $\leftarrow 0$
    $s \leftarrow$ env.reset()    ▷ Get initial state
    while not converged or step < max_steps do
        if $rand() < \varepsilon$ then    ▷ Choose a random action
            $a \leftarrow$ sample(env.action_space)
        else
            $Q(s,a) \leftarrow$ train_net($s$)    ▷ Query trained network for $Q(s,a)$
            $a \leftarrow \text{argmax}_a Q(s,a)$    ▷ Choose best current action

        $r, s'$, done $\leftarrow$ env($a$)    ▷ Take a step and receive reward and next state
        $\text{buf}_{replay}$.insert($(s,a,r,s')$)    ▷ Store interaction in the replay buffer
        $s \leftarrow s'$
        batch $\leftarrow$ sample($\text{buf}_{replay}$, batch_size)    ▷ Sample a random batch of data

        if done then    ▷ If episode has ended
            $y \leftarrow r$    ▷ Calculate target for each interaction in the batch
            $s \leftarrow$ env.reset()
        else
            $\hat{Q}(s',a') \leftarrow$ target_net($s'$)    ▷ Query target network for $\hat{Q}(s',a')$
            $y \leftarrow r + \gamma \max_{a'} \hat{Q}(s',a')$    ▷ Calculate target for each interaction in the batch

        $L \leftarrow (Q(s,a) - y)^2$    ▷ Calculate loss value for each interaction in the batch
        $SGD$(train_net, $L$)    ▷ Apply SGD algorithm to minimize loss
        $\varepsilon \leftarrow 1.0 - \frac{\texttt{step}}{\texttt{max\_steps}}$    ▷ Decay $\varepsilon$ linearly

        if step % $T_{sync}$ == 0 then    ▷ Synchronize weights every $T_{sync}$ steps
            target_net.weights $\leftarrow$ train_net.weights

        step++
```

θ respectively. Fig. 7, 8 and 9 show the initial states of the three different simulated fields with increasing complexity and their evolution over time using their respective advection-diffusion parameters. To easily refer to these fields, we name them *Field-1*, *Field-2* and *Field-3* respectively. Additionally, Table 1 summarizes the advection-diffusion parameters for each of the simulated fields.

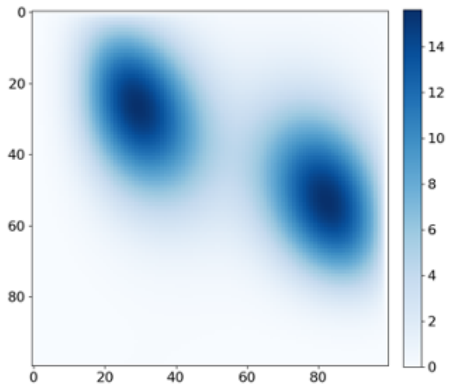
Table 1
Summary of Advection-Diffusion Parameters Chosen for the Simulated Fields

Field Name	θ	$\mathbf{v}$
Field-1	1.0	[0.6, -0.8]
Field-2	1.0	[0.7, 0.3]
Field-3	1.0	[-0.4, 0.7]

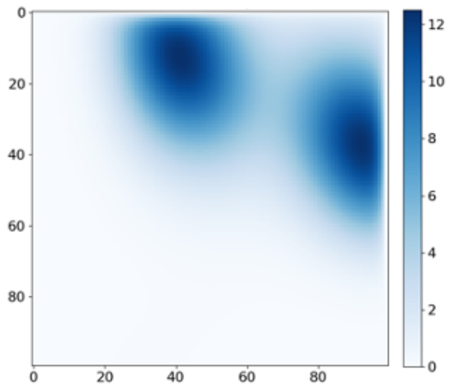
Please take note of the color bars drawn in Fig. 7, 8 and 9. The values representing the darkest color in the color map decrease as time increases demonstrating the impact of the diffusion in the simulated fields.



(a)



(b)



(c)

Fig. 7. These figures show the state of the *Field-1* simulated advection-diffusion field at the following time steps (a) $k = 0$, (b), $k = 150$, (c) $k = 300$.

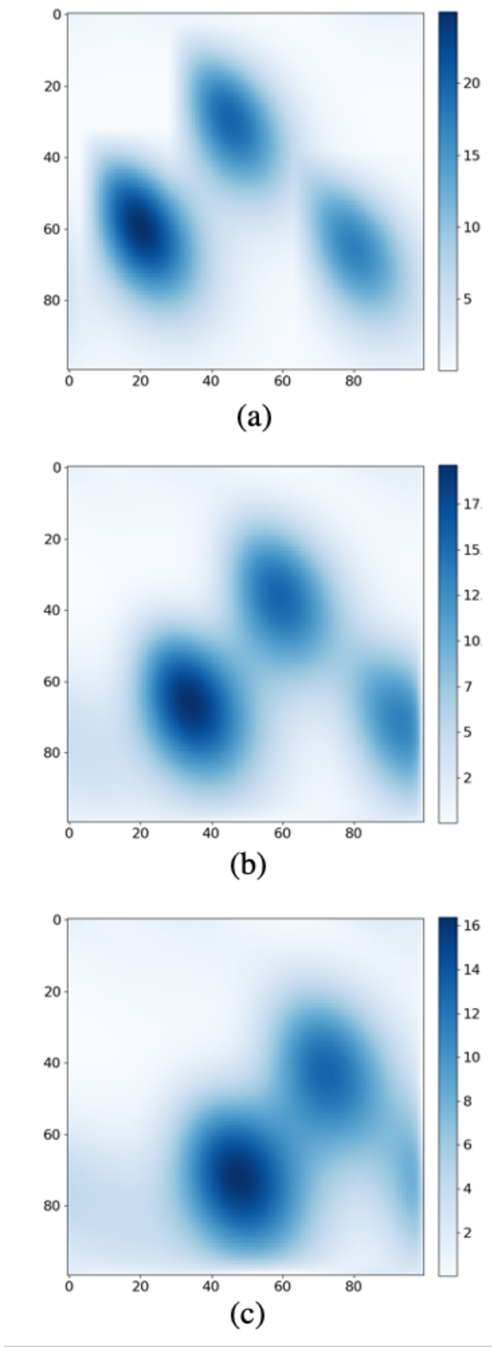


Fig. 8. These figures show the state of the *Field-2* simulated advection-diffusion field at the following time steps (a) $k = 0$, (b), $k = 150$, (c) $k = 300$

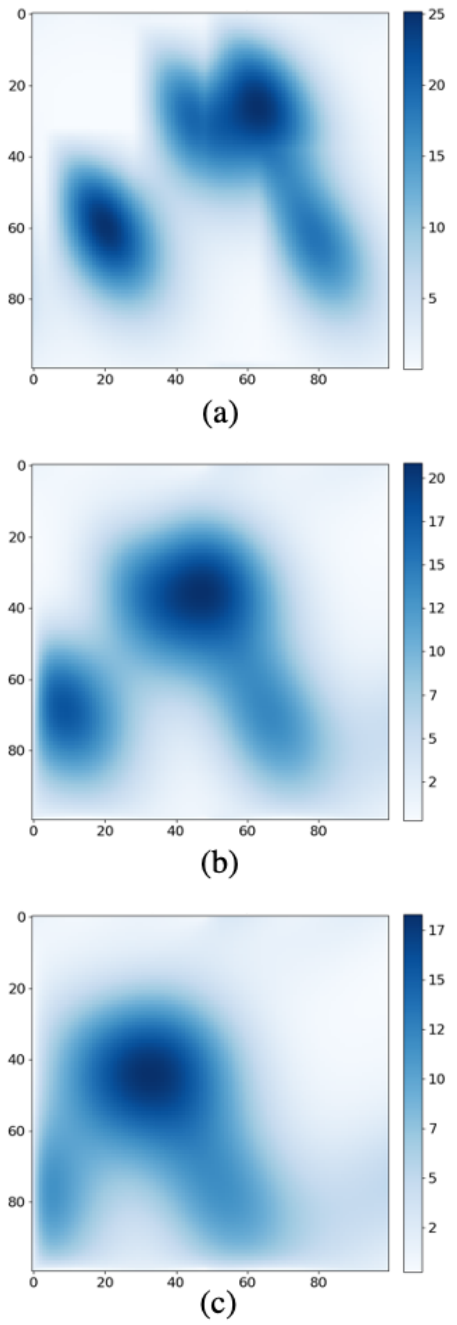


Fig. 9. These figures show the state of the *Field-3* simulated advection-diffusion field at the following time steps (a) $k = 0$, (b), $k = 150$, (c) $k = 300$

6.3.1 DQN Training on Field-1

The neural-network model described in Fig. 3 was trained using Algorithm 3 with *Field-1* as the advection-diffusion environment. It is important to note that unlike most path-finding algorithms that require a fixed starting location and destination, each episode in our training starts at a random location $r_c \in \Omega$. Some of the important hyper-parameters used for this training are as follows:

- Number of training interactions (`max_steps`): 4000000
- Optimizer used: Adam [43]
- Initial Learning rate: 0.0001
- Discount factor (γ): 0.99
- Batch size (`batch_size`): 64
- Experience replay buffer size (`buf_len`): 200000
- Time steps to synchronize training and target networks (T_{sync}): 900

Fig. 10 shows the average accumulated reward per episode over the course of the network training. According to Equation (18), the absolute value of the reward accumulated per episode depends on the field values and the path taken by the mobile robot formation. As evident from Fig. 10, the training converged at an average reward per episode of 9815 showing that the network learns to accumulate roughly the same reward each episode regardless of the formation's starting location.

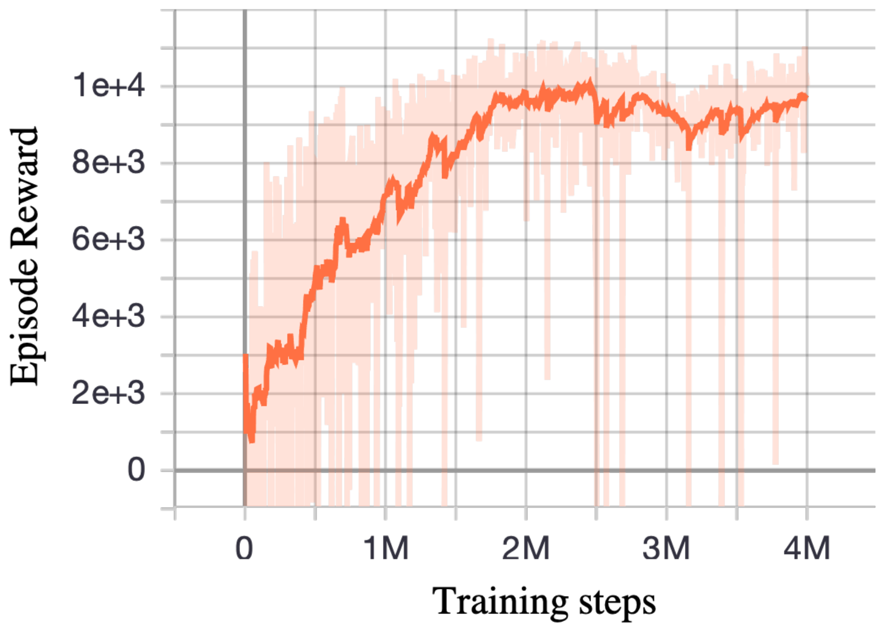


Fig. 10. Accumulated episode rewards over time steps during training on *Field-1*

6.3.2 DQN Training on Field-2

The neural-network model described in Fig. 3 was trained using Algorithm 3 with *Field-2* as the advection-diffusion environment. It is important to note that unlike most path-finding algorithms that require a fixed starting location and destination, each episode in our training starts at a random location $r_c \in \Omega$. Some of the important hyper-parameters used for this training are as follows:

- Number of training interactions (`max_steps`): 4000000
- Optimizer used: Adam [43]
- Initial Learning rate: 0.0001
- Discount factor (γ): 0.99
- Batch size (`batch_size`): 64
- Experience replay buffer size (`buf_len`): 200000
- Time steps to synchronize training and target networks (T_{sync}): 900

Fig. 11 shows the average accumulated reward per episode over the course of the network training. According to Equation (18), the absolute value of the reward accumulated per episode depends on the field values and the path taken by the mobile robot formation. As evident from Fig. 11, the training converged at an average reward per episode of 9044 showing that the network learns to accumulate roughly the same reward each episode regardless of the formation's starting location.

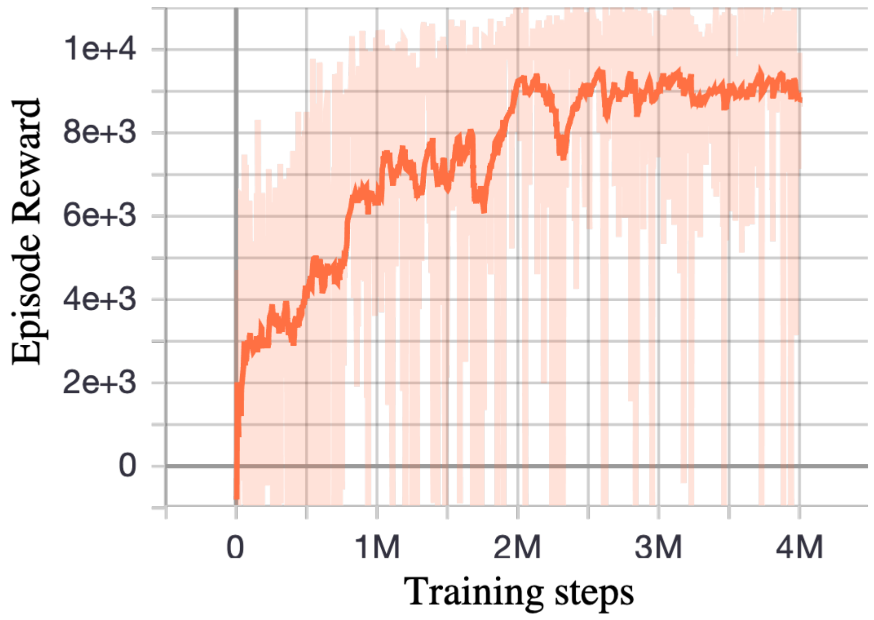


Fig. 11. Accumulated episode rewards over time steps during training on *Field-2*

6.3.3 DQN Training on Field-3

The neural-network model described in Fig. 3 was trained using Algorithm 3 with *Field-3* as the advection-diffusion environment. It is important to note that unlike most path-finding algorithms that require a fixed starting location and destination, each episode in our training starts at a random location $r_c \in \Omega$. Some of the important hyper-parameters used for this training are as follows:

- Number of training interactions (`max_steps`): 4000000

- Optimizer used: Adam [43]
- Initial Learning rate: 0.0001
- Discount factor (γ): 0.99
- Batch size (`batch_size`): 64
- Experience replay buffer size (`buf_len`): 200000
- Time steps to synchronize training and target networks (T_{sync}): 900

Fig. 12 shows the average accumulated reward per episode over the course of the network training. According to Equation (18), the absolute value of the reward accumulated per episode depends on the field values and the path taken by the mobile robot formation. As evident from Fig. 12, the training converged at an average reward per episode of 10084 showing that the network learns to accumulate roughly the same reward each episode regardless of the formation's starting location.

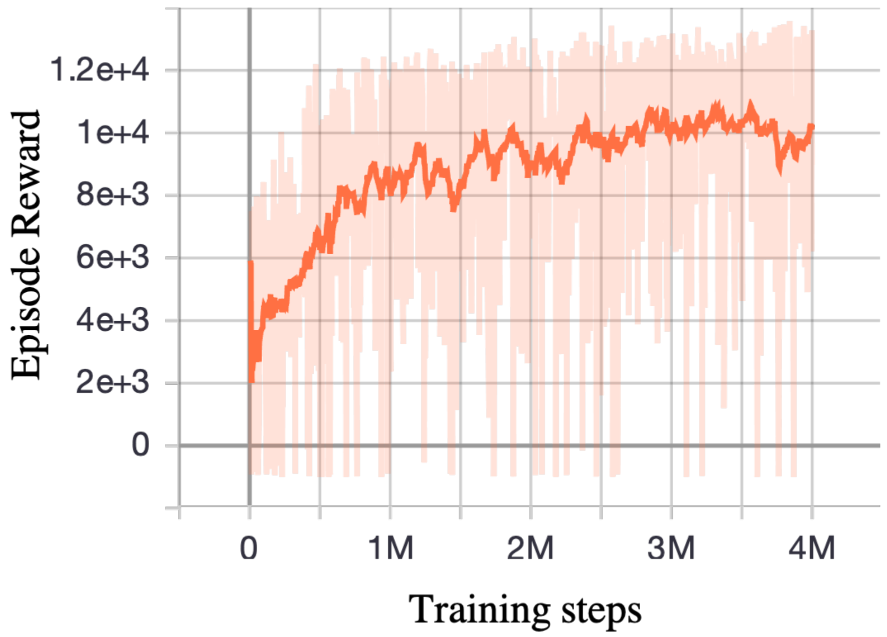


Fig. 12. Accumulated episode rewards over time steps during training on *Field-3*

6.4 Testing Results

In this section, we use the trained models described in the previous section to perform inference and generate sample test episodes. Please note that we do not show results of running the full algorithm proposed in Chapter 5 here, those are provided in Chapter 7. Please note the following remarks for figures showing the field domain Ω in this section:

- The number of mobile robots in these results is 4 i.e., $N = 4$. These robots will travel in a square shaped formation. The square formed by the mobile robots is the time varying view-scope $\Gamma(k)$ and is represented by a red colored square.
- The initial location of the formation center $r_c(k = 0)$ is denoted by a red colored circle.
- The trajectory followed by the formation center $r_c(k)$ is denoted by filled black colored circles.

Fig. 13 and 14 show a sample episode each from two different starting locations if the DQN model trained on *Field-1* is followed for the entirety of the episode. In both figures, the trained DQN model commands the formation to move over information-rich trajectories to reach high-concentration zones which provide the most information to reduce the mapping error. After reaching the high-concentration zones, the DQN model commands the formation to track the high-concentration zones matching the local velocity of the advection-diffusion field. This feature of the DQN model allows us to identify the advection coefficients of the field using the algorithm described in Algorithm 1.

Fig. 15 and 16 show a sample episode each from two different starting locations if the DQN model trained on *Field-2* is followed for the entirety of the episode. Both these figures show that the DQN model commands the formation to move on information-rich trajectories, and then track high-concentration zones similar to how it did on *Field-1* as shown in Fig. 13 and 14. This shows that the training works as expected over different initial field concentrations and advection-diffusion parameters.

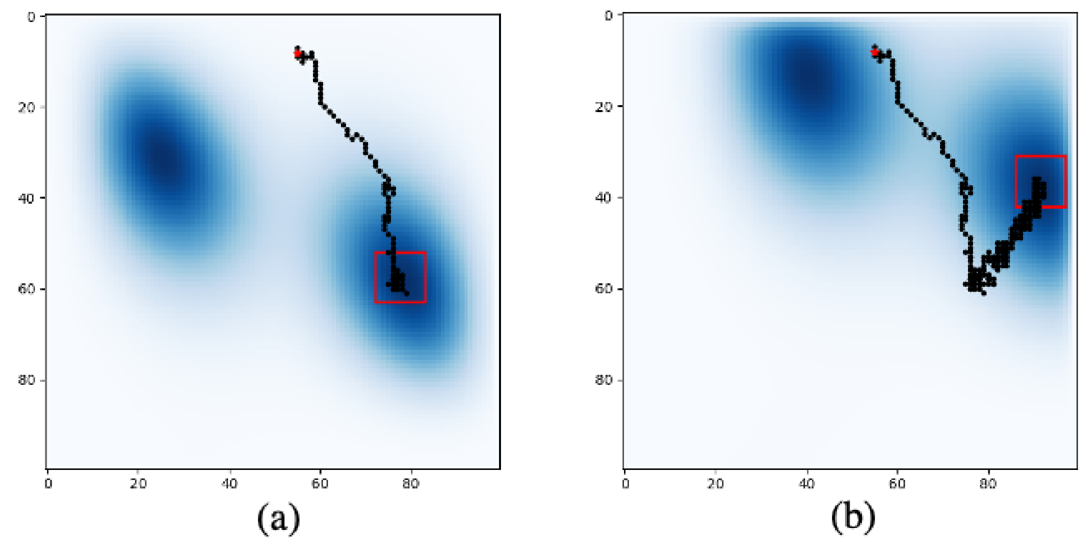


Fig. 13. Sample test episode 1 using trained DQN model on *Field-1* at following time steps (a) $k = 100$, (b) $k = 300$

Similar results for test episodes run using DQN model trained on *Field-3* are obtained, as shown in Fig. 17 and 18. In Fig. 18(a), it is important to note that while the DQN model tracked a high-concentration zone for some time steps, as soon as it discovered a trajectory that reduces the mapping error to a higher degree, it was able to control the formation to zone with higher concentration, as apparent in Fig. 18(b). This demonstrates that the DQN model is able to command the formation towards the areas that reduce the mapping error the most from the current location, at each time step k.

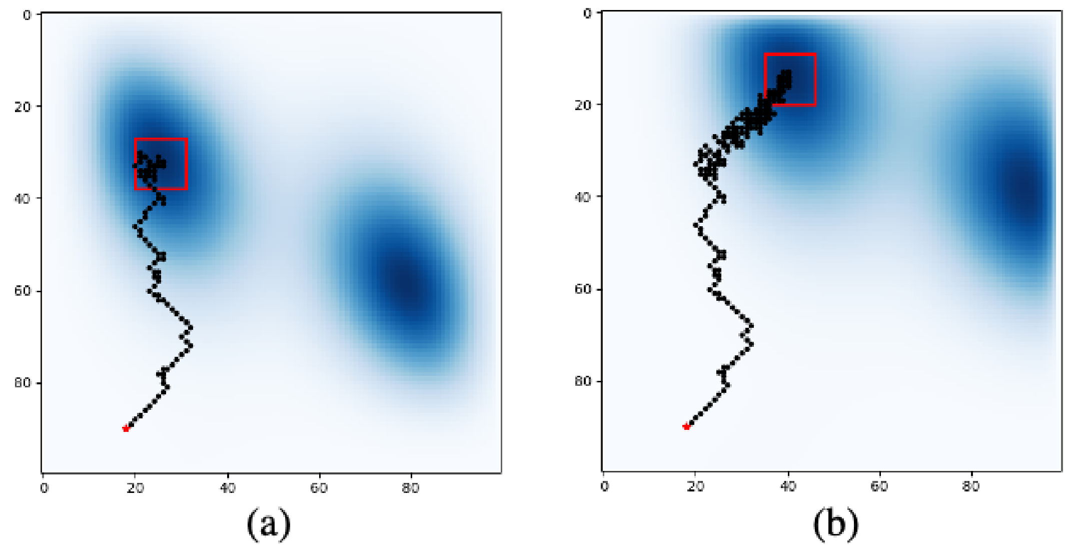


Fig. 14. Sample test episode 2 using trained DQN model on *Field-1* at following time steps (a) $k = 100$, (b) $k = 300$

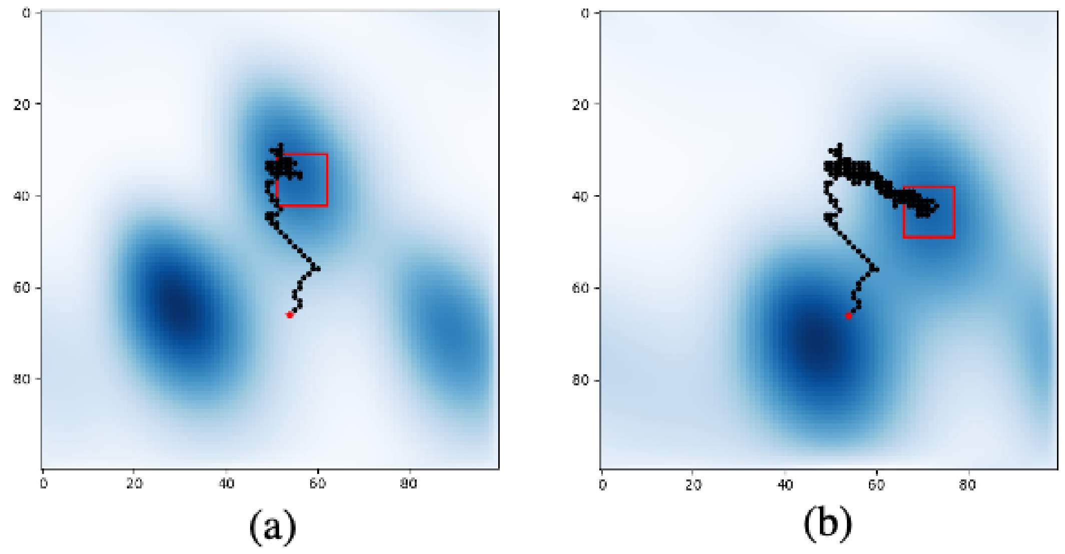


Fig. 15. Sample test episode 1 using trained DQN model on *Field-2* at following time steps (a) $k = 100$, (b) $k = 300$

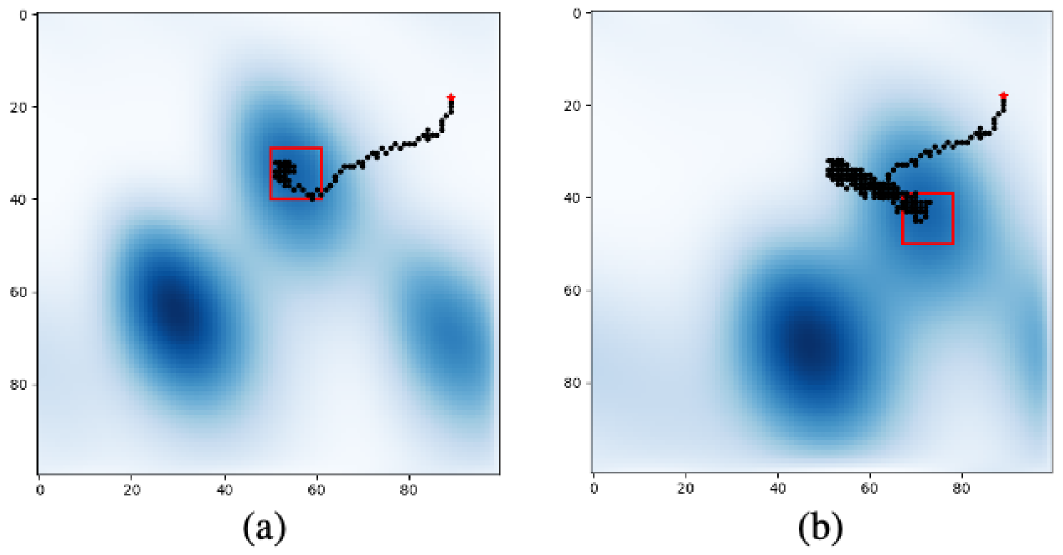


Fig. 16. Sample test episode 2 using trained DQN model on *Field-2* at following time steps (a) $k = 100$, (b) $k = 300$

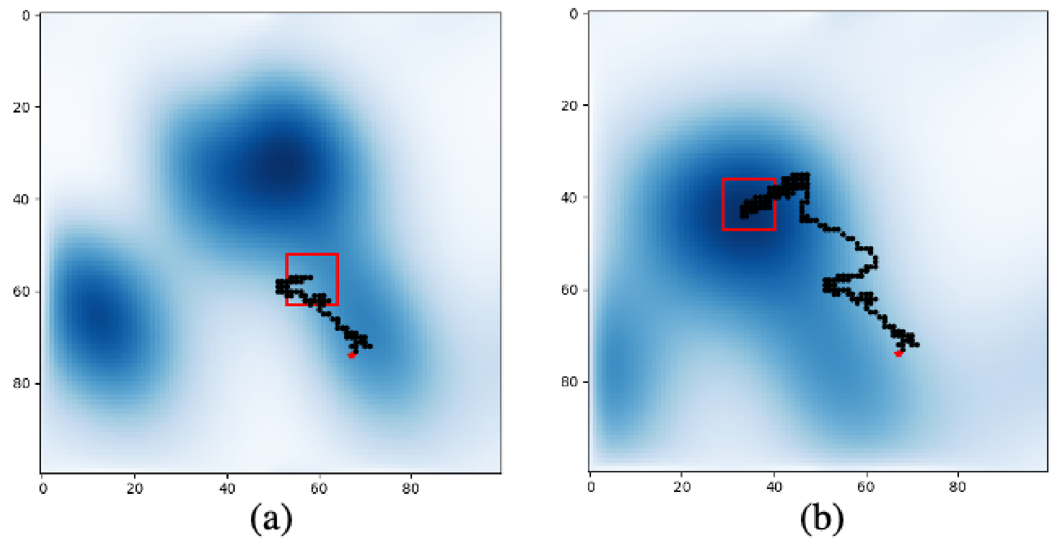


Fig. 17. Sample test episode 1 using trained DQN model on *Field-3* at following time steps (a) $k = 100$, (b) $k = 300$

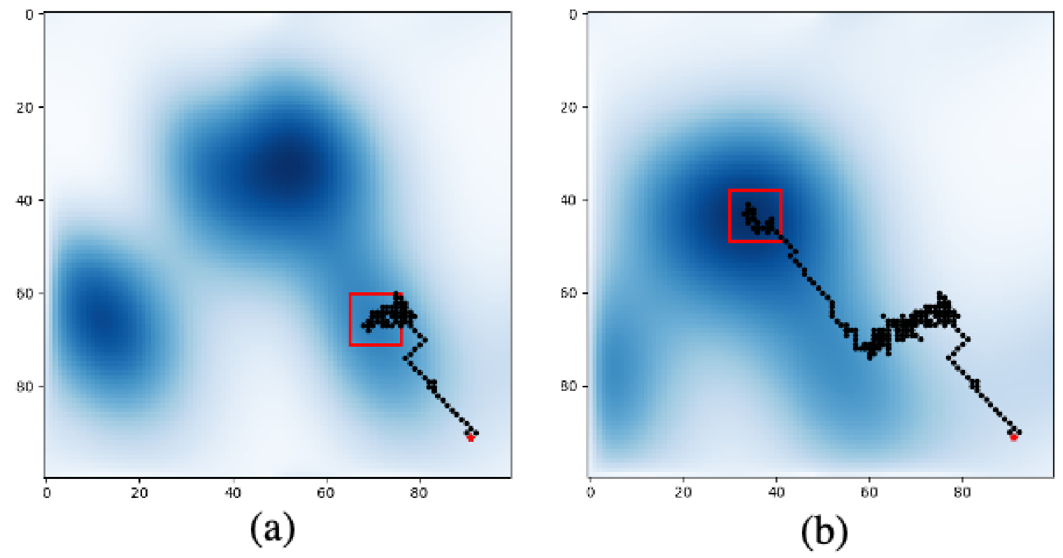


Fig. 18. Sample test episode 2 using trained DQN model on *Field-3* at following time steps (a) $k = 100$, (b) $k = 300$

7 FIELD RECONSTRUCTION RESULTS IN LOW-FIDELITY SIMULATION ENVIRONMENT

In this chapter, we first elaborate on the field reconstruction process (Section 5.6) as it applies to the Low-Fidelity Simulation Environment and then provide testing results of using the algorithm proposed in Chapter 5 on the three simulated fields described in Table 1. The trained models described in Section 6.3 are used for the DQN based control portions of the proposed algorithm in Fig. 2. Additionally, we introduce two new test fields that were not used in the training process and demonstrate the performance of the proposed algorithm on these test fields.

7.1 Advection-Diffusion Field Reconstruction in Low-Fidelity Simulation Environment

In this section, we elaborate on how the advection-diffusion PDE for field reconstruction (Equation (25)) is discretized using the finite-difference method, and used to reconstruct the field in the Low-Fidelity Simulation Environment. Similar to the simulated field described in Section 6.1, the reconstructed field domain Ω is represented as a discretized $E \times F$ matrix with $E = F = 100$, with each grid cell $r \in \Omega$ holding a concentration value $\hat{z}(r,t)$ at time t. Please refer to Fig. 5 as a 3×3 section of the discretized field representation, as it applies here as well. Therefore, to apply Equation (25) on this discretized field, we use the finite difference method to spatially and temporally discretize this equation as follows:

$$\begin{aligned}\frac{\hat{z}(r_0,k+1)-\hat{z}(r_0,k)}{t_s} =& \hat{\theta}\left[\frac{\hat{z}(r_2,k)+\hat{z}(r_4,k)-2\hat{z}(r_0,k)}{\Delta r_x^2}+\frac{\hat{z}(r_1,k)+\hat{z}(r_3,k)-2\hat{z}(r_0,k)}{\Delta r_y^2}\right]+ \\ & \hat{\mathbf{v}}^T\nabla\hat{z}(r_0,k)+e(r_0,k),\end{aligned} \tag{31}$$

where k is the discretized time stamp, t_s is the sampling interval, $e(r_0,k)$ accounts for the approximation error, $\hat{\theta}$ is the diffusion parameter estimated by the RLS algorithm (Chapter 4), $\hat{\mathbf{v}}$ are the advection parameters estimated by the proposed algorithm and $\hat{z}(r,t)$ is the reconstructed concentration function. Assuming square grid cells, i.e., $\Delta r_x = \Delta r_y$, Equation (31) simplifies to the following.

$$\frac{\hat{z}(r_0,k+1)-\hat{z}(r_0,k)}{t_s} = \hat{\theta}\frac{\sum_{i=1}^{4}\hat{z}(r_i,k)-4\hat{z}(r_0,k)}{\Delta r_x^2} + \hat{\mathbf{v}}^T\nabla\hat{z}(r_0,k)+e(r_0,k). \tag{32}$$

Thus, at each time step k, when the estimated field values $\hat{z}(r,k),\ r\in\Gamma(k)$ are available, we populate these values at the current view-scope in the reconstructed field and apply the discretized Equation (32) to propagate the field. This process allows us to reconstruct the field with only sparse measurements along the robots' trajectories. Figures in the next section demonstrate episodes showing the process of reconstruction.

7.2 Reconstruction Results

To demonstrate the efficacy of the proposed algorithm, we first provide two sample test episodes each for running the proposed algorithm on each of the three simulated advection-diffusion fields in Table 1. Second, we provide plots of mapping errors over the course of 30 episodes with randomized starting locations for each of the simulated advection-diffusion fields showing the reduction in mapping error regardless of the formation's starting location. Third, we create two test fields named *Test Field-1* and *Test Field-2* described in Table 2 and provide two sample test episodes on these fields using the DQN model trained on *Field-3* to show the generalizability of the algorithm on unseen fields. Finally, we provide plots of mapping errors over the course of 30 test episodes with randomized starting locations for the two test fields, showing the reduction in mapping error regardless of the formation's starting location.

Please note the following remarks for figures showing the field domain Ω in this section:

- The number of mobile robots in these results is 4 i.e., $N = 4$. These robots will travel in a square shaped formation. The square formed by the mobile robots is the time varying view-scope $\Gamma(k)$ and is represented by a red colored square.
- The initial location of the formation center $r_c(k = 0)$ is denoted by a red colored circle.
- The trajectory followed by the formation center $r_c(k)$ while the formation is controlled by the DQN network is denoted by filled black colored circles.
- The trajectory colored in red followed by the formation center $r_c(k)$ is the most recent path followed to reach the destination provided by the destination selector algorithm described in Section 5.5.
- The figures are organized in 3×2 grids. Images in the first column show the state of the simulated field, while the images in the second column show the field reconstructed by the mobile robot formation. Images in the same row represent the same time step in the episode.

7.2.1 Testing on Field-1

In this section, we discuss the two sample test episodes performed on *Field-1* shown in Fig. 19 and 20. Fig. 19 shows the state of first sample episode at time steps $k = 100, k = 136$ and $k = 136$. At $k = 100$, Fig. 19(a) shows that the formation has reached a stationary state in the field, identified the advection coefficients, chosen a destination and planned the path to the destination. The identified advection parameters are $\hat{\mathbf{v}} = [0.57, -0.82]$, which are similar to the true advection parameters for *Field-1* of $\mathbf{v} = [0.6, -0.8]$ as stated in Table 1. Then, using the estimated advection parameters and the known diffusion parameters in equation 29, we retroactively reconstruct the field as explained in Section 5.6 and shown in Fig. 19(b). In Fig. 19(c) and 19(d), the state of the episode at $k = 136$ is shown. At this time step, the formation has reached the chosen destination, and the control is transferred back to the DQN model. The formation then reaches and tracks the second high-concentration zone until the end of the experiment. Fig. 19(f) shows the state of the reconstructed field at the end of the episode. Additionally, Fig. 21 shows the reduction in mapping error over the course of this test episode. As evident from the figure, the mapping error reduces monotonically through out the episode. It is interesting to note that the mapping error reduces at a higher rate when the formation is moving towards the high-concentration zones.

Fig. 20 shows us the state of the second sample episode at time steps $k = 118, k = 218$ and $k = 300$. Additionally, Fig. 22 shows the reduction in mapping error over the course of this test episode. We see similar results in the path planned in this episode as we did with episode 1. At $k = 118$, in Fig. 20(a) and 20(b), the formation has reached a stationary state in the field, identified the advection coefficients, chosen a destination and planned a path to the destination. The identified advection parameters are $\hat{\mathbf{v}} = [0.59, -0.85]$, which are similar to the true advection parameters for *Field-1* of $\mathbf{v} = [0.6, -0.8]$ as stated in Table 1. At $k = 218$, in Fig. 20(c) and 20(d), we see that the formation has reached another stationary state in the field, and thus chooses another destination to trigger exploration and plans a path to it. After reaching the destination, the control is transferred back to the DQN model and the formation tracks the second high-concentration zone until the end of the episode.

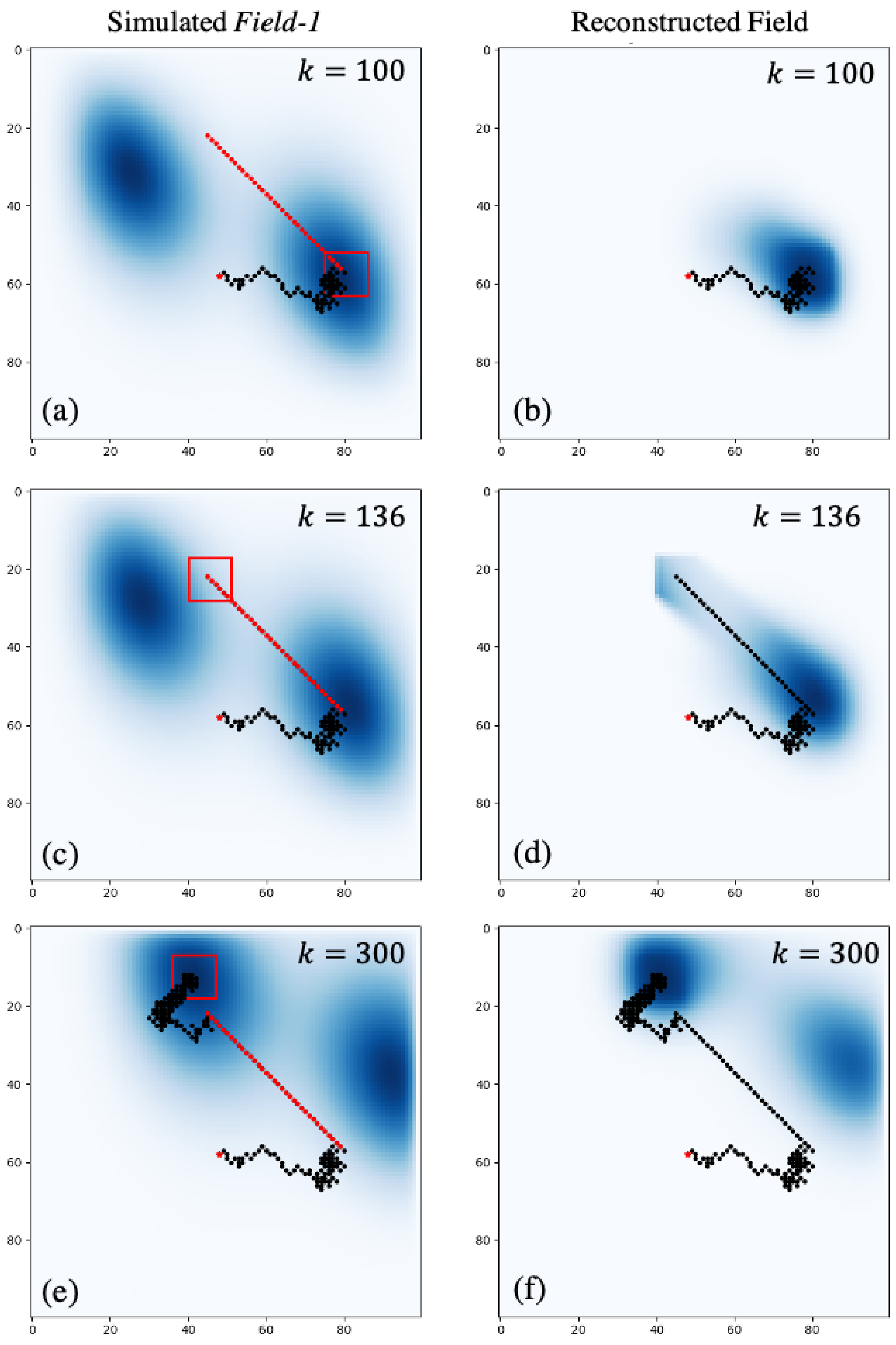


Fig. 19. Sample test episode 1 performed on *Field-1*

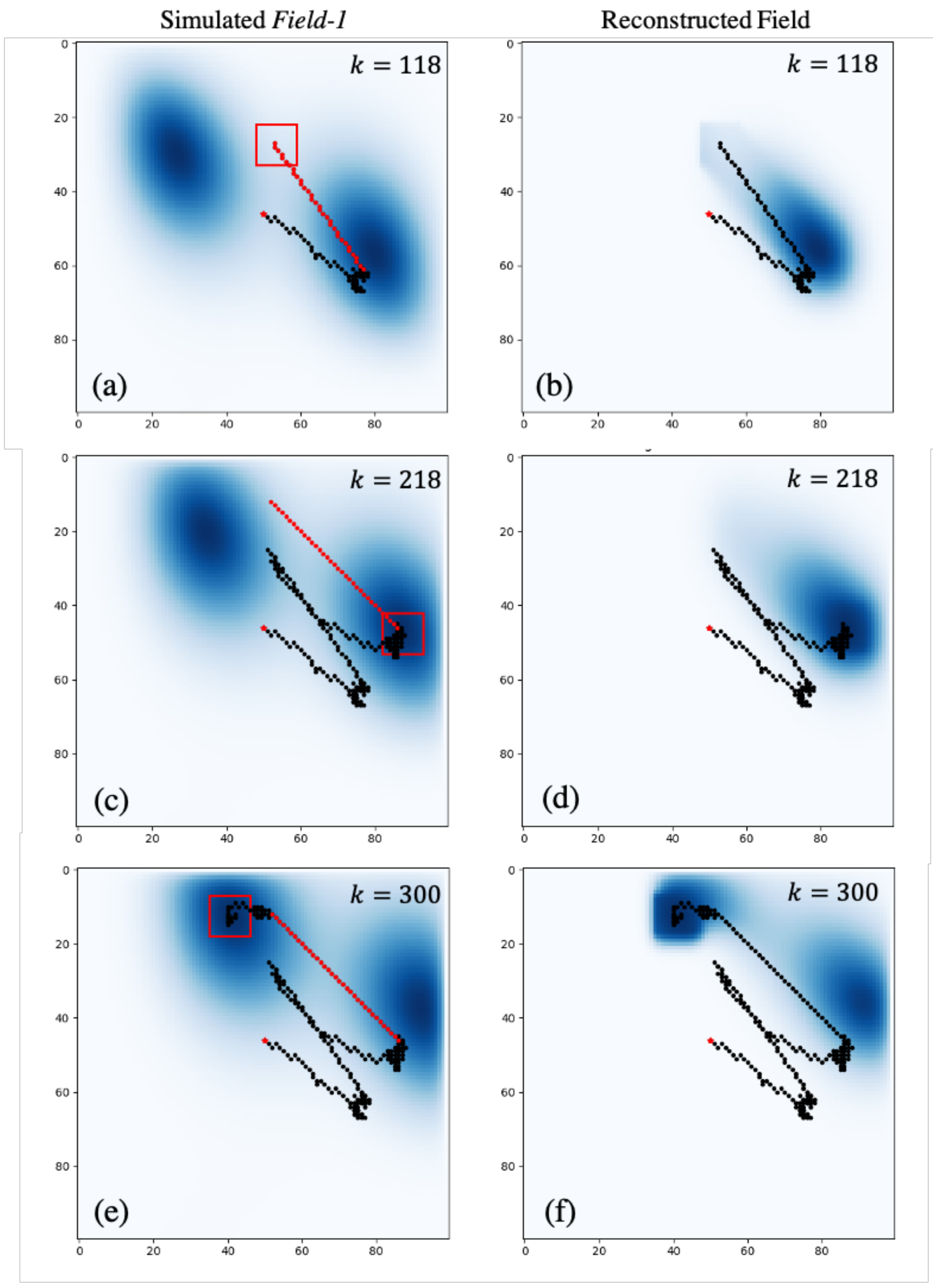


Fig. 20. Sample test episode 2 performed on *Field-1*

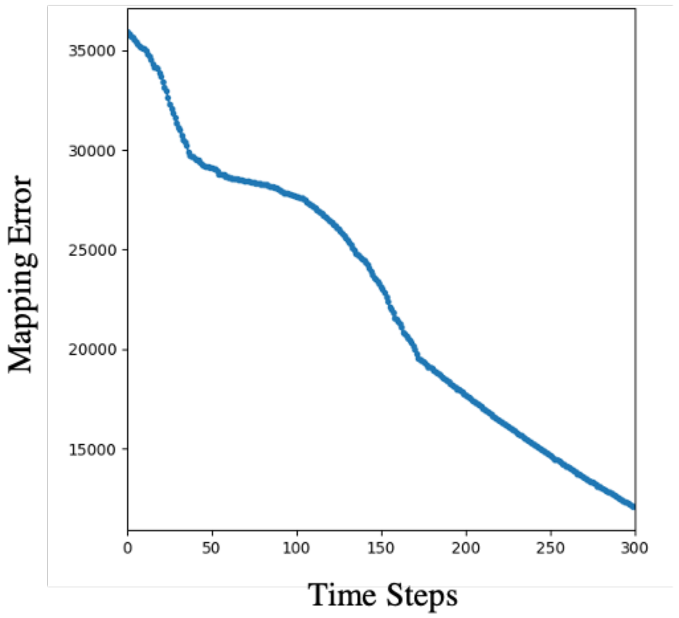


Fig. 21. Mapping error for sample test episode 1 performed on *Field-1*

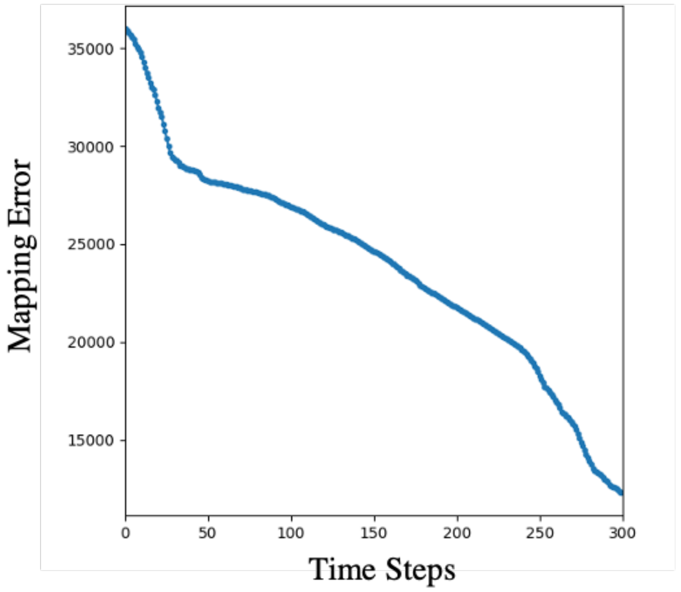


Fig. 22. Mapping error for sample test episode 2 performed on *Field-1*

7.2.2 *Testing on Field-2*

In this section, we discuss the two sample test episodes performed on *Field-2*. As shown in Fig. 8, *Field-2* has 3 high-concentration zones making the initial field surface more complicated than for *Field-1*, making the reconstruction task more challenging.

Fig. 23 shows the state of the first sample episode at time steps $k = 101, k = 202$ and $k = 300$. Additionally, Fig. 25 shows the reduction in mapping error over the course of this test episode. At $k = 101$, in Fig. 23(a) and 23(b), the formation has reached a stationary state in the field, identified the advection coefficients, chosen a destination and planned a path to the destination. The identified advection parameters are $\hat{\mathbf{v}} = [0.72, 0.34]$, which are similar to the true advection parameters for *Field-2* of $\mathbf{v} = [0.7, 0.3]$ as stated in Table 1. At $k = 202$, in Fig. 23(c) and 23(d), we see that the formation has reached another stationary state in the field, and thus chooses another destination to trigger exploration and moves to the destination. After reaching the destination, the control is transferred back to the DQN model and the formation tracks the second high-concentration zone until the end of the episode. As observed in previous episodes, the mapping error reduces at a higher rate when new high-concentration zones are found.

Fig. 24 shows the state of the second sample episode at time steps $k = 107, k = 227$ and $k = 300$. Additionally, Fig. 26 shows the reduction in mapping error over the course of this test episode. At $k = 107$, in Fig. 24(a) and 24(b), the formation has reached a stationary state in the field, identified the advection coefficients, chosen a destination and planned a path to the destination. The identified advection parameters are $\hat{\mathbf{v}} = [0.68, 0.28]$, which are similar to the true advection parameters for *Field-2* of $\mathbf{v} = [0.7, 0.3]$ as stated in Table 1. At $k = 227$, in Fig. 24(c) and 24(d), we see that the formation has reached another stationary state in the field, and thus chooses another destination to trigger exploration and moves to the destination. After reaching the destination, the control is transferred back to the DQN model and the formation tracks the second high-concentration zone until the end of the episode. As observed in previous episodes, the mapping error reduces at a higher rate when new high-concentration zones are found.

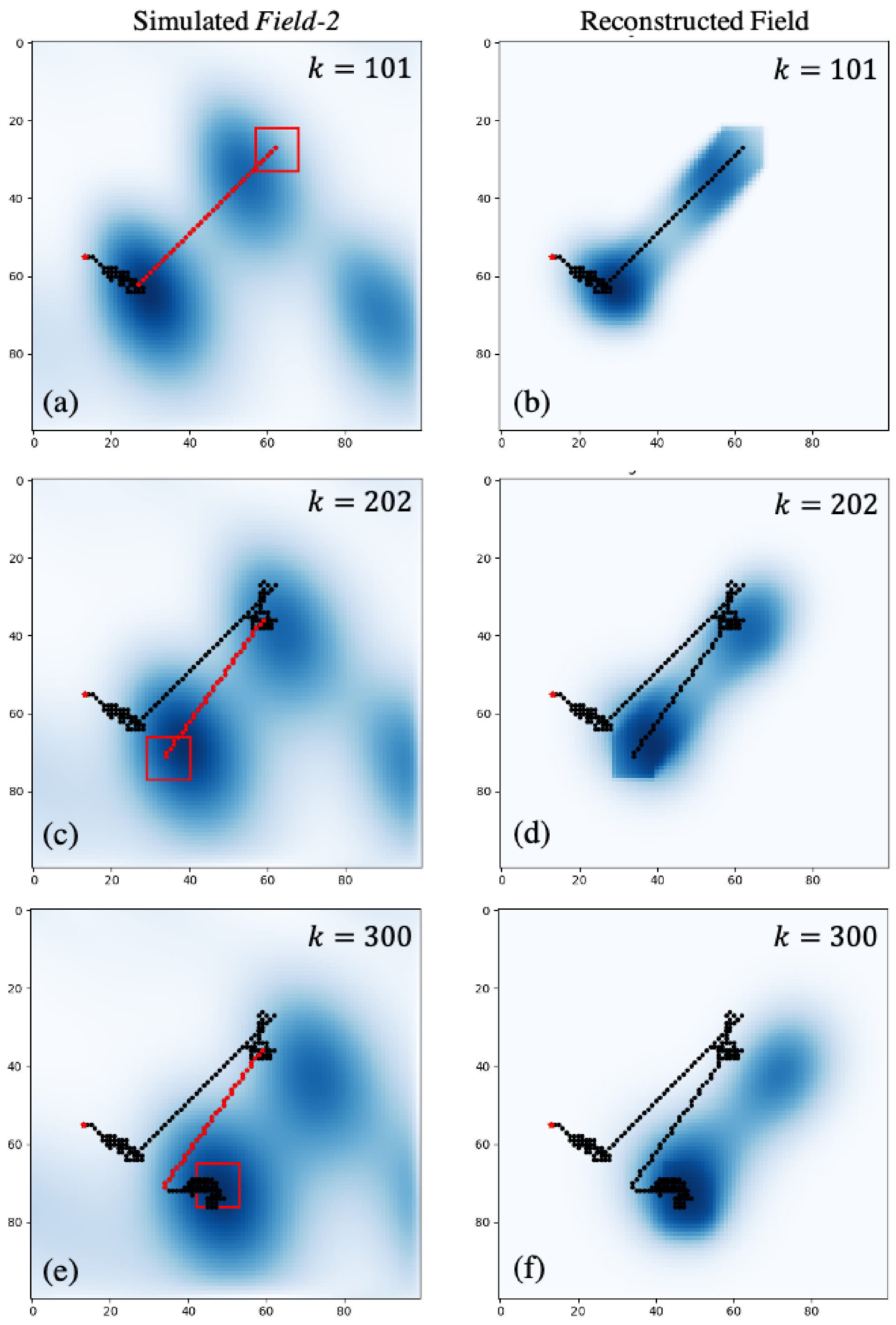


Fig. 23. Sample test episode 1 performed on *Field-2*

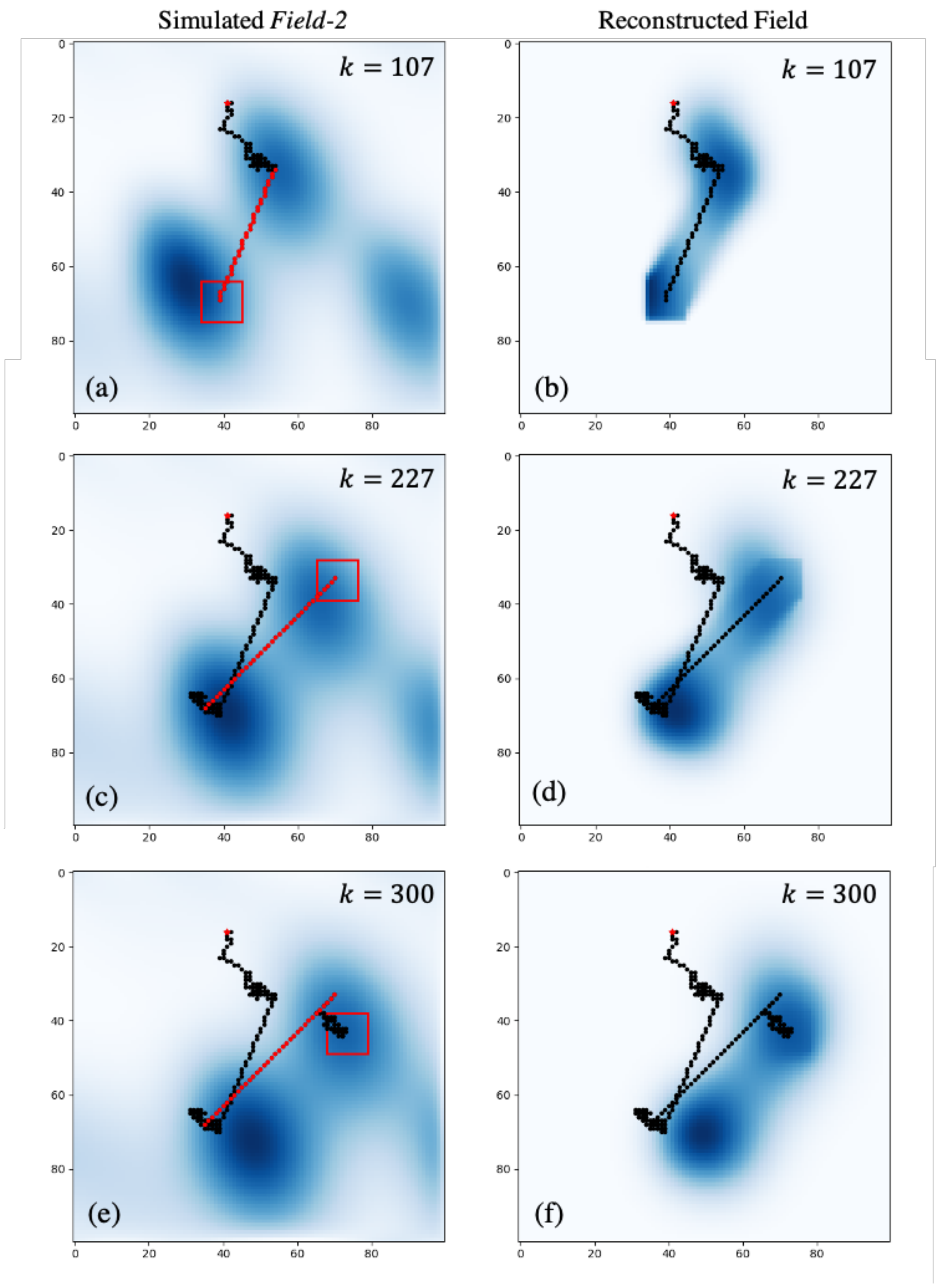


Fig. 24. Sample test episode 2 performed on *Field-2*

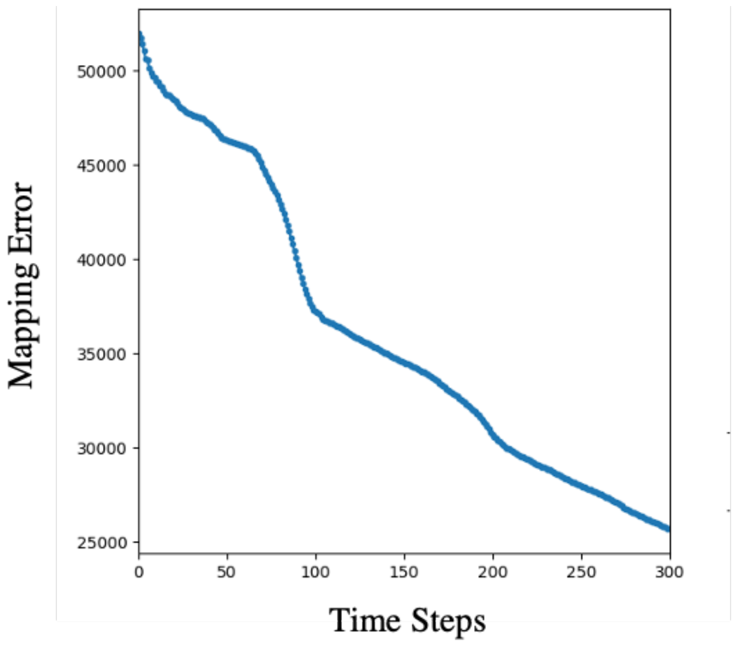


Fig. 25. Mapping error for sample test episode 1 performed on *Field-2*

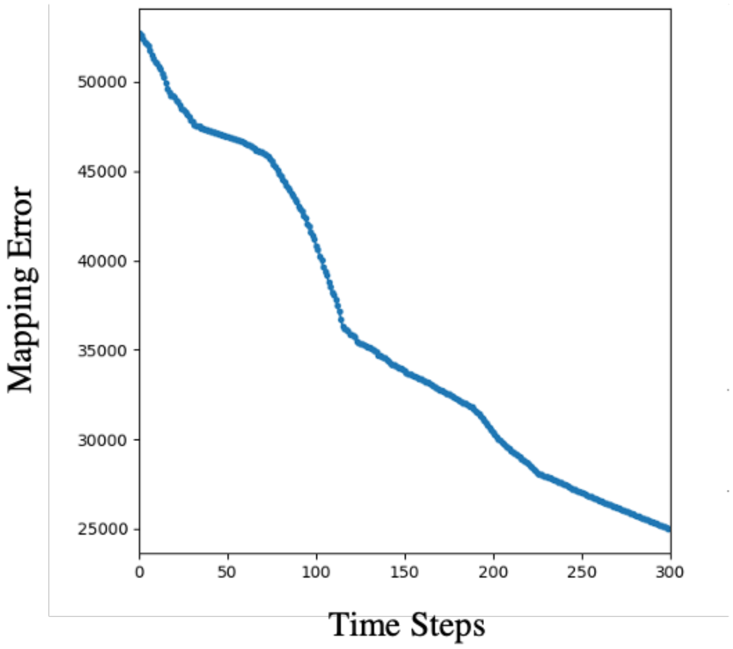


Fig. 26. Mapping error for sample test episode 2 performed on *Field-2*

7.2.3 Testing on Field-3

In this section, we discuss the two sample test episodes performed on *Field-3*. As shown in Fig. 9, *Field-3* has 4 high-concentration zones making the initial field surface more complicated than for *Field-1* and *Field-2*, making the reconstruction task even more challenging.

Fig. 27 shows the state of the first sample episode at time steps $k = 139, k = 238$ and $k = 300$. Additionally, Fig. 29 shows the reduction in mapping error over the course of this test episode. At $k = 139$, in Fig. 27(a) and 27(b), the formation has reached a stationary state in the field, identified the advection coefficients, chosen a destination and planned a path to the destination. The identified advection parameters are $\hat{\mathbf{v}} = [-0.39, 0.71]$, which are similar to the true advection parameters for *Field-3* of $\mathbf{v} = [-0.4, 0.7]$ as stated in Table 1. At $k = 238$, in Fig. 27(c) and 27(d), we see that the formation has reached another stationary state in the field, and thus chooses another destination to trigger exploration and moves to the destination. After reaching the destination, the control is transferred back to the DQN model and the formation tracks the third high-concentration zone until the end of the episode. As observed in previous episodes, the mapping error reduces at a higher rate when new high-concentration zones are found.

Fig. 28 shows the state of the second sample episode at time steps $k = 127, k = 226$ and $k = 300$. Additionally, Fig. 30 shows the reduction in mapping error over the course of this test episode. At $k = 127$, in Fig. 28(a) and 28(b), the formation has reached a stationary state in the field, identified the advection coefficients, chosen a destination and planned a path to the destination. The identified advection parameters are $\hat{\mathbf{v}} = [-0.43, 0.74]$, which are similar to the true advection parameters for *Field-3* of $\mathbf{v} = [-0.4, 0.7]$ as stated in Table 1. At $k = 226$, in Fig. 28(c) and 28(d), we see that the formation has reached another stationary state in the field, and thus chooses another destination to trigger exploration and moves to the destination. After reaching the destination, the control is transferred back to the DQN model and the formation tracks the third high-concentration zone until the end of the episode. As observed in previous episodes, the mapping error reduces at a higher rate when new high-concentration zones are found.

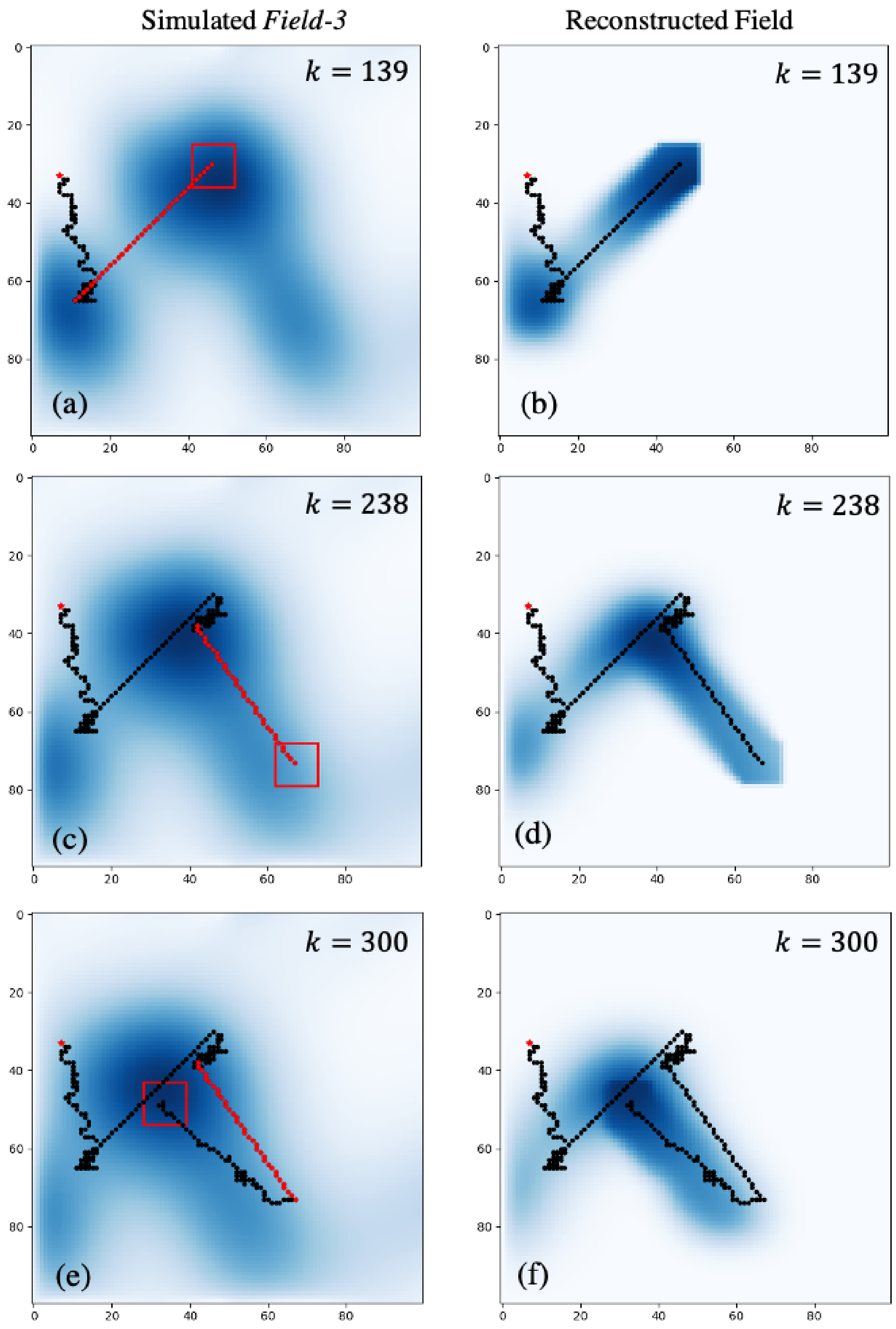


Fig. 27. Sample test episode 1 performed on *Field-3*

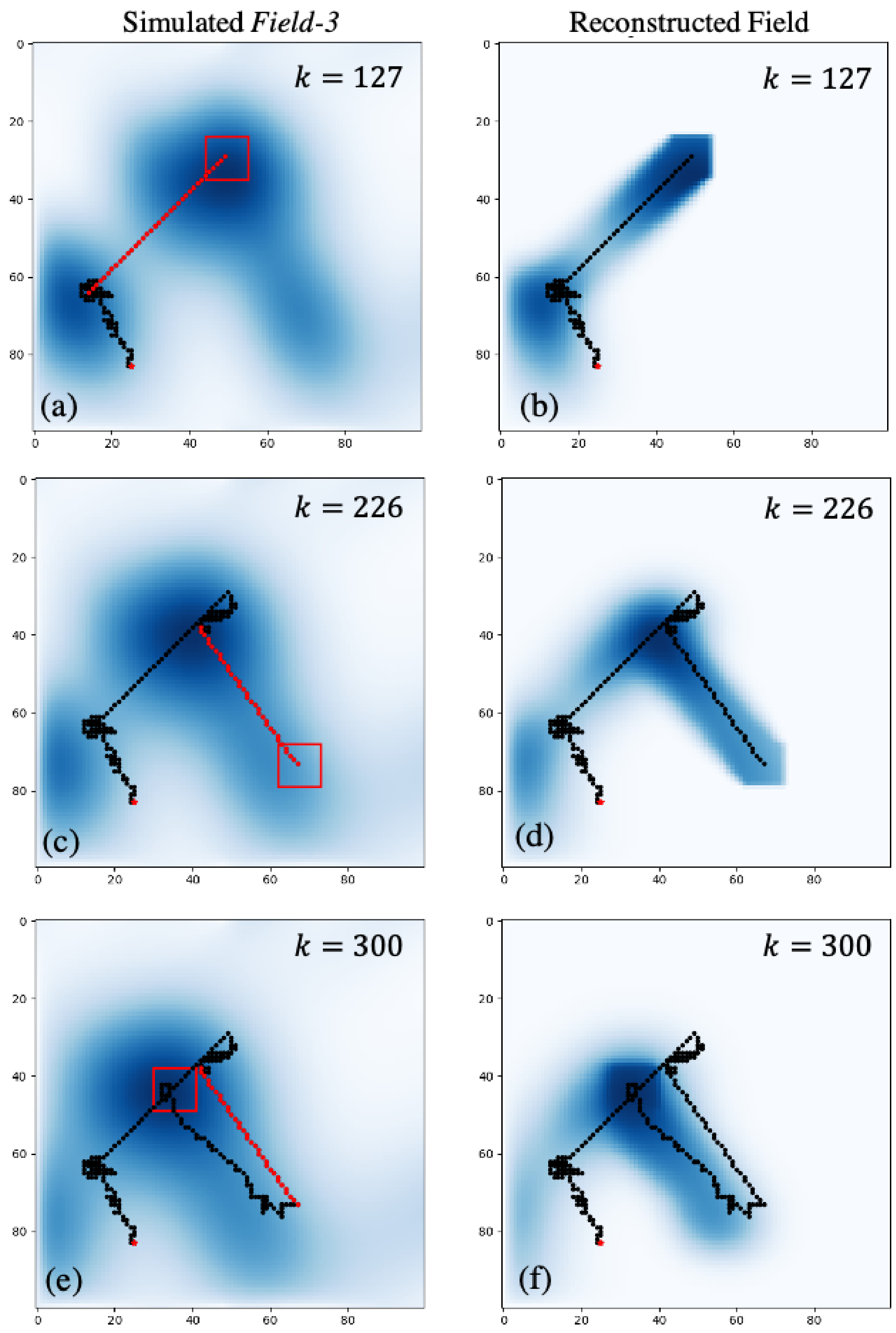


Fig. 28. Sample test episode 2 performed on *Field-3*

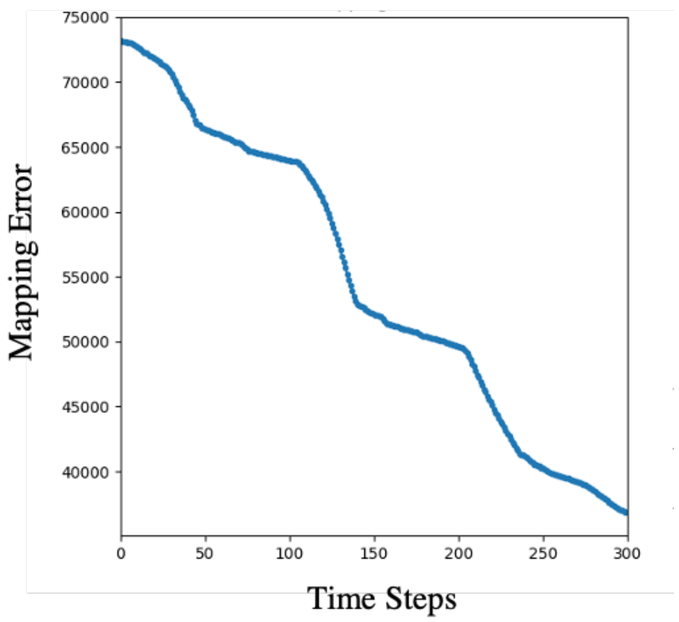


Fig. 29. Mapping error for sample test episode 1 performed on *Field-3*

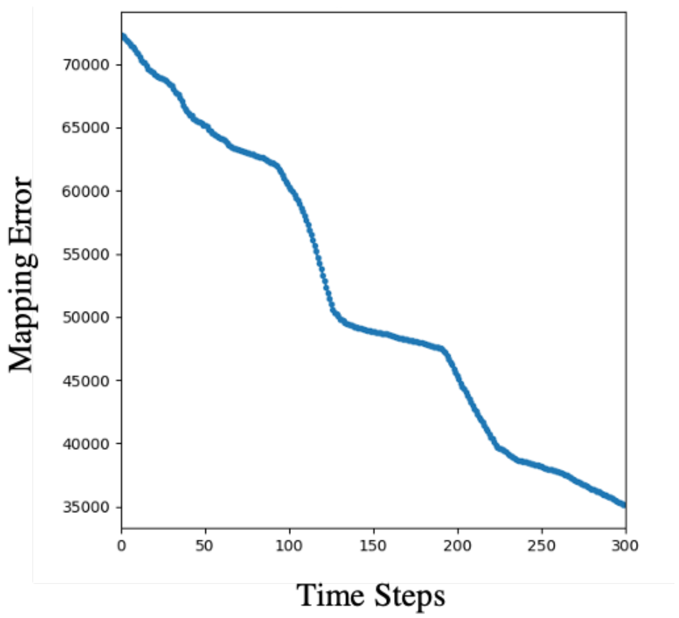


Fig. 30. Mapping error for sample test episode 2 performed on *Field-3*

7.3 Aggregated Mapping Errors with Random Starting Locations

In the previous section, we demonstrated a few test episodes for each of the simulated fields and explained the behavior and performance of the proposed algorithm. In this section, we will run a number of test episodes using the proposed algorithm on each of the the simulated fields and provide the mapping errors for these episodes in aggregate.

We run 30 test episodes each for *Field-1*, *Field-2* and *Field-3* with randomized formation center starting locations and aggregate the mapping errors for each of the episodes. Fig. 31, 32 and 33 plot the aggregated mapping errors for fields *Field-1*, *Field-2* and *Field-3* respectively. The mean mapping error is plotted in a dark blue color, and the blue shaded region shows the minimum and maximum bounds over the 30 episodes. As evident from the figures, the mapping error reduces monotonically in aggregate showing that the proposed algorithm is able to command the formation to travel along information-rich paths, trigger exploration as stationary states are reached regardless of the starting location of the formation center.

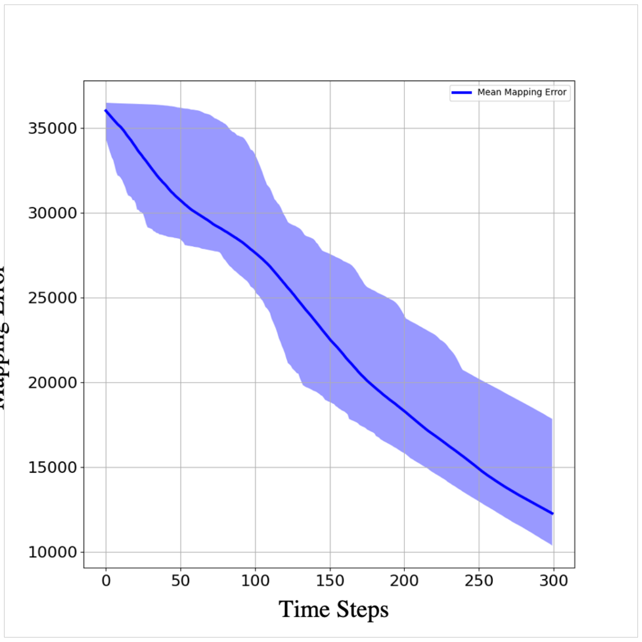


Fig. 31. Mean mapping errors over 30 episodes with random starting locations on *Field-1*. The shaded region shows the minimum and maximum bounds.

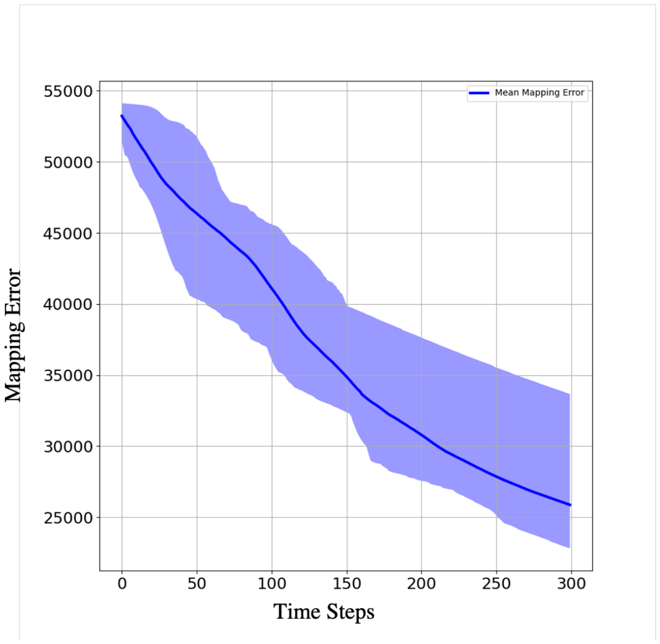


Fig. 32. Mean mapping errors over 30 episodes with random starting locations on *Field-2*. The shaded region shows the minimum and maximum bounds.

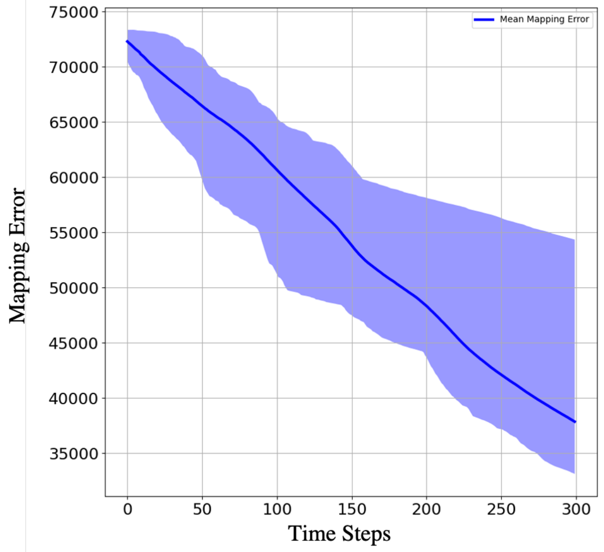


Fig. 33. Mean mapping errors over 30 episodes with random starting locations on *Field-3*. The shaded region shows the minimum and maximum bounds.

7.4 Testing Results on Unseen Test Fields

Thus far, we have demonstrated the performance of the algorithm on simulated advection-diffusion fields that the DQN models had previously been trained on. While it is shown that the algorithm performs well on the fields described in Table 1, it is important to show the algorithm can also be used on fields that are unseen by the DQN model, and can generalize well regardless of the initial concentration surface $z(r,0) \forall r \in \Omega$, and advection-diffusion coefficients. Therefore, in this section, we create two test fields named *Test Field-1* and *Test Field-2* and using the DQN model trained on *Field-3*, we perform two sample test episodes of the proposed algorithm. Table 2 summarizes the advection-diffusion coefficients used for the simulated test fields, and Fig. 34 and 35 show the initial states of *Test Field-1* and *Test Field-2* and their evolution over time using their respective advection-diffusion parameters. Please note that to be able to easily differentiate testing fields from training fields, a green color map is used in the figures showing testing fields.

Table 2
Summary of Advection-Diffusion Parameters Chosen for the Simulated Test Fields

Field Name	θ	$\mathbf{v}$
Test Field-1	1.0	[-0.65, 0.45]
Test Field-2	1.0	[0.8, -0.4]

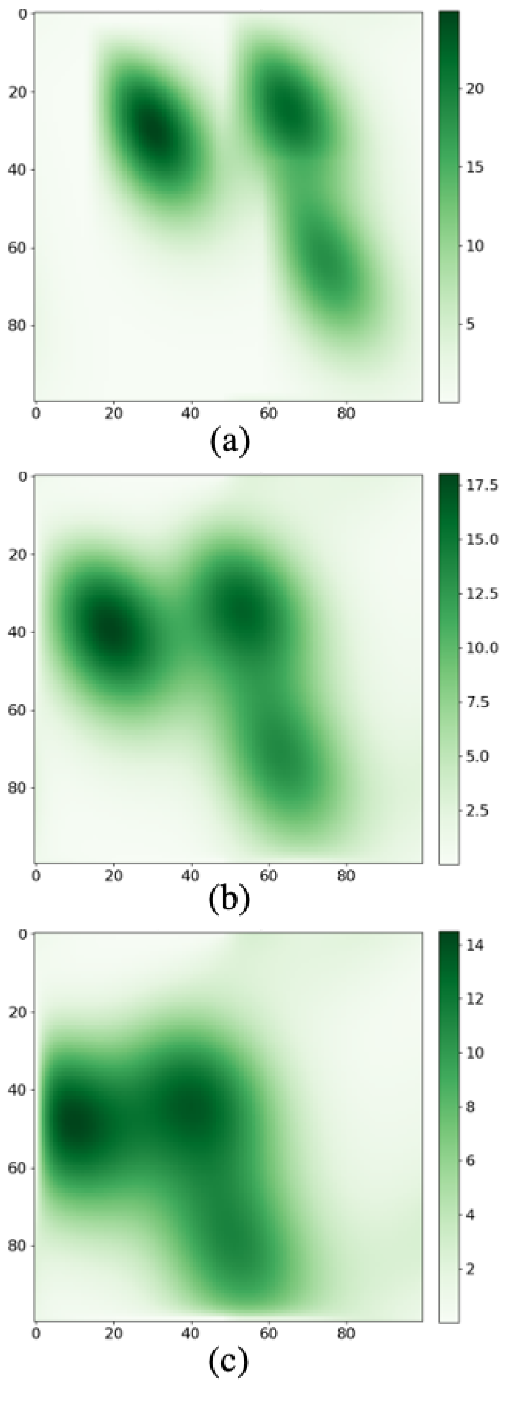


Fig. 34. These figures show the state of the *Test Field-1* simulated advection-diffusion field at the following time steps (a) $k = 0$, (b), $k = 150$, (c) $k = 300$

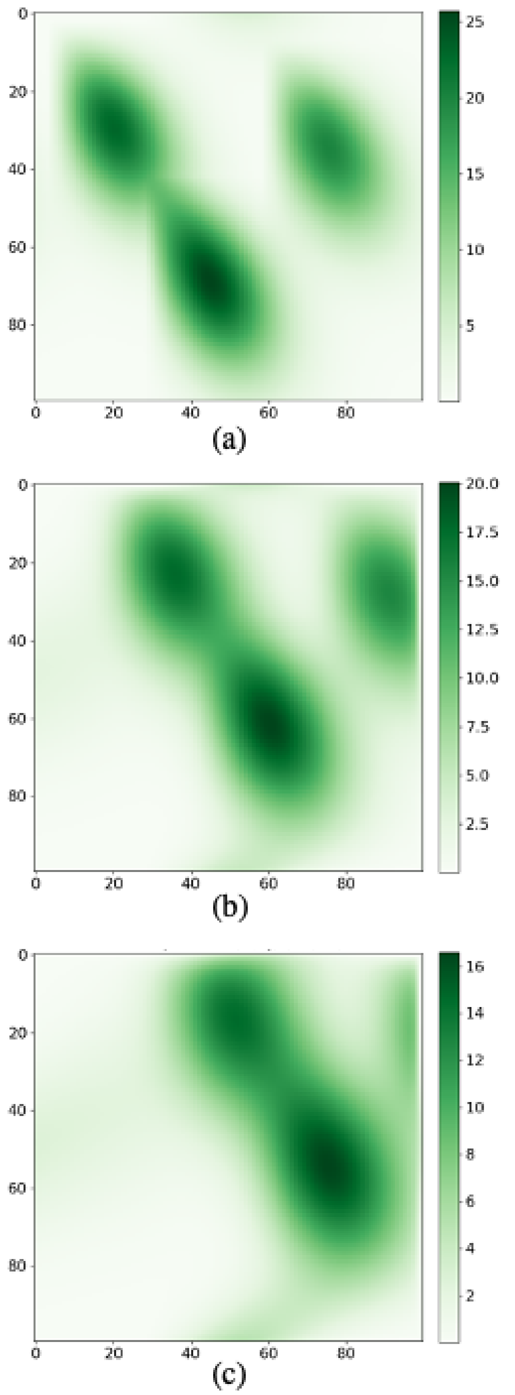


Fig. 35. These figures show the state of the *Test Field-2* simulated advection-diffusion field at the following time steps (a) $k = 0$, (b), $k = 150$, (c) $k = 300$

Now, using the DQN model trained on *Field-3*, we perform one sample test episode each on *Test Field-1* and *Test Field-2*. Fig. 36 shows the state of the test episode performed on *Test Field-1* at time steps $k = 129, k = 214$ and $k = 300$. Additionally, Fig. 38 shows the reduction in mapping error over the course of this test episode. At $k = 129$, in Fig. 36(a) and 36(b), the formation has reached a stationary state in the field, identified the advection coefficients, chosen a destination and planned a path to the destination. The identified advection parameters are $\hat{\mathbf{v}} = [-0.67, 0.44]$, which are similar to the true advection parameters for *Test Field-1* of $\mathbf{v} = [-0.65, 0.45]$ as stated in Table 2. At $k = 214$, in Fig. 36(c) and 36(d), we see that the formation has reached another stationary state in the field, and thus chooses another destination to trigger exploration and moves to the destination. After reaching the destination, the control is transferred back to the DQN model and the formation tracks the third high-concentration zone until the end of the episode. As observed in previous episodes performed on training fields, the mapping error reduces at a higher rate when new high-concentration zones are found.

Fig. 37 shows the state of the test episode performed on *Test Field-2* at time steps $k = 106, k = 244$ and $k = 300$. Additionally, Fig. 39 shows the reduction in mapping error over the course of this test episode. At $k = 106$, in Fig. 37(a) and 37(b), the formation has reached a stationary state in the field, identified the advection coefficients, chosen a destination and planned a path to the destination. The identified advection parameters are $\hat{\mathbf{v}} = [0.84, -0.37]$, which are similar to the true advection parameters for *Test Field-2* of $\mathbf{v} = [0.8, -0.4]$ as stated in Table 2. At $k = 244$, in Fig. 37(c) and 37(d), we see that the formation has reached another stationary state in the field, and thus chooses another destination to trigger exploration and moves to the destination. After reaching the destination, the control is transferred back to the DQN model and the formation tracks the third high-concentration zone for a short duration before re-triggering exploration. The episode ends just as the formation reaches the chosen destination. As observed in previous episodes performed on training fields, the mapping error reduces at a higher rate when new high-concentration zones are found.

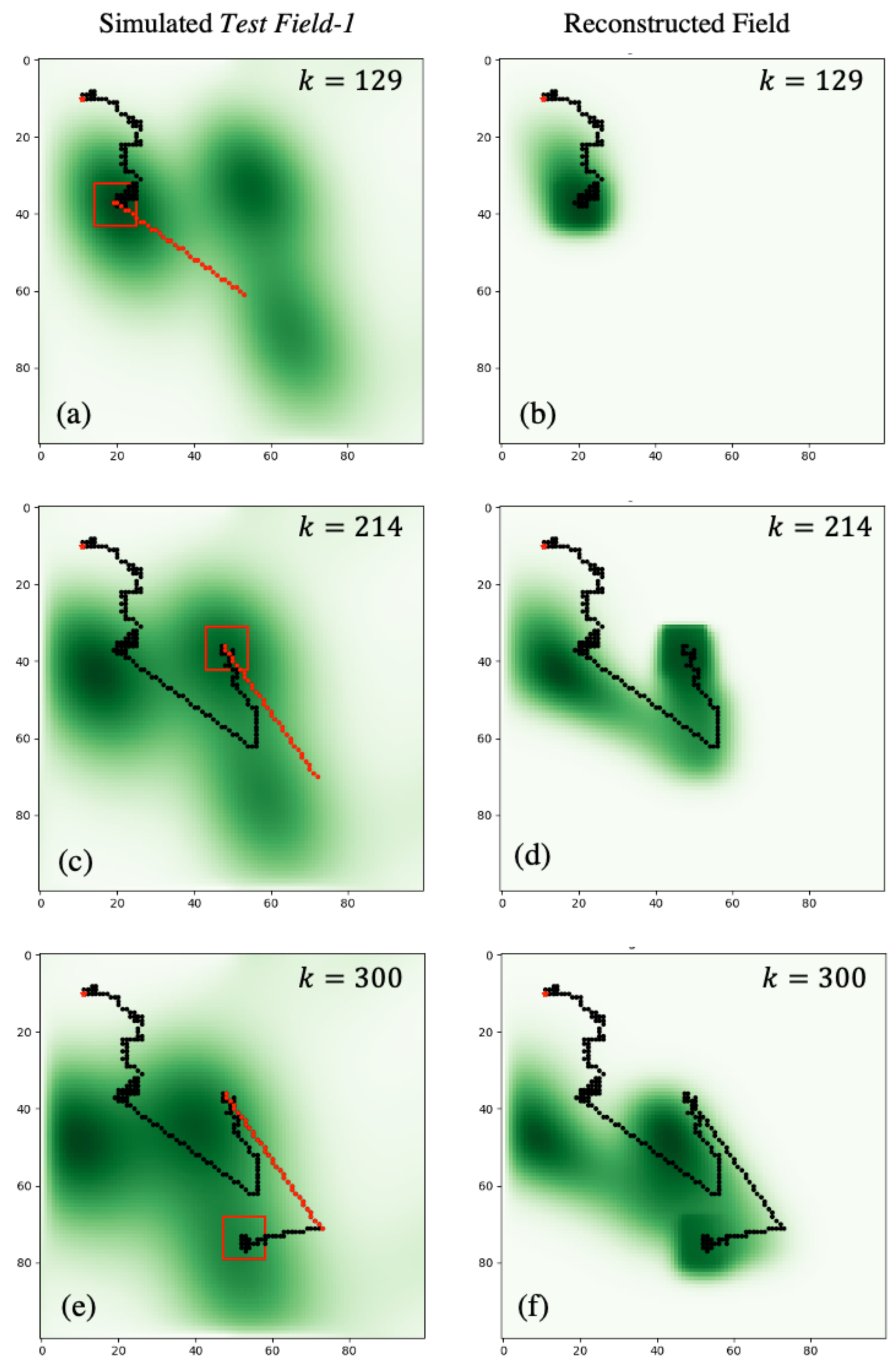


Fig. 36. Sample test episode performed on *Test Field-1*

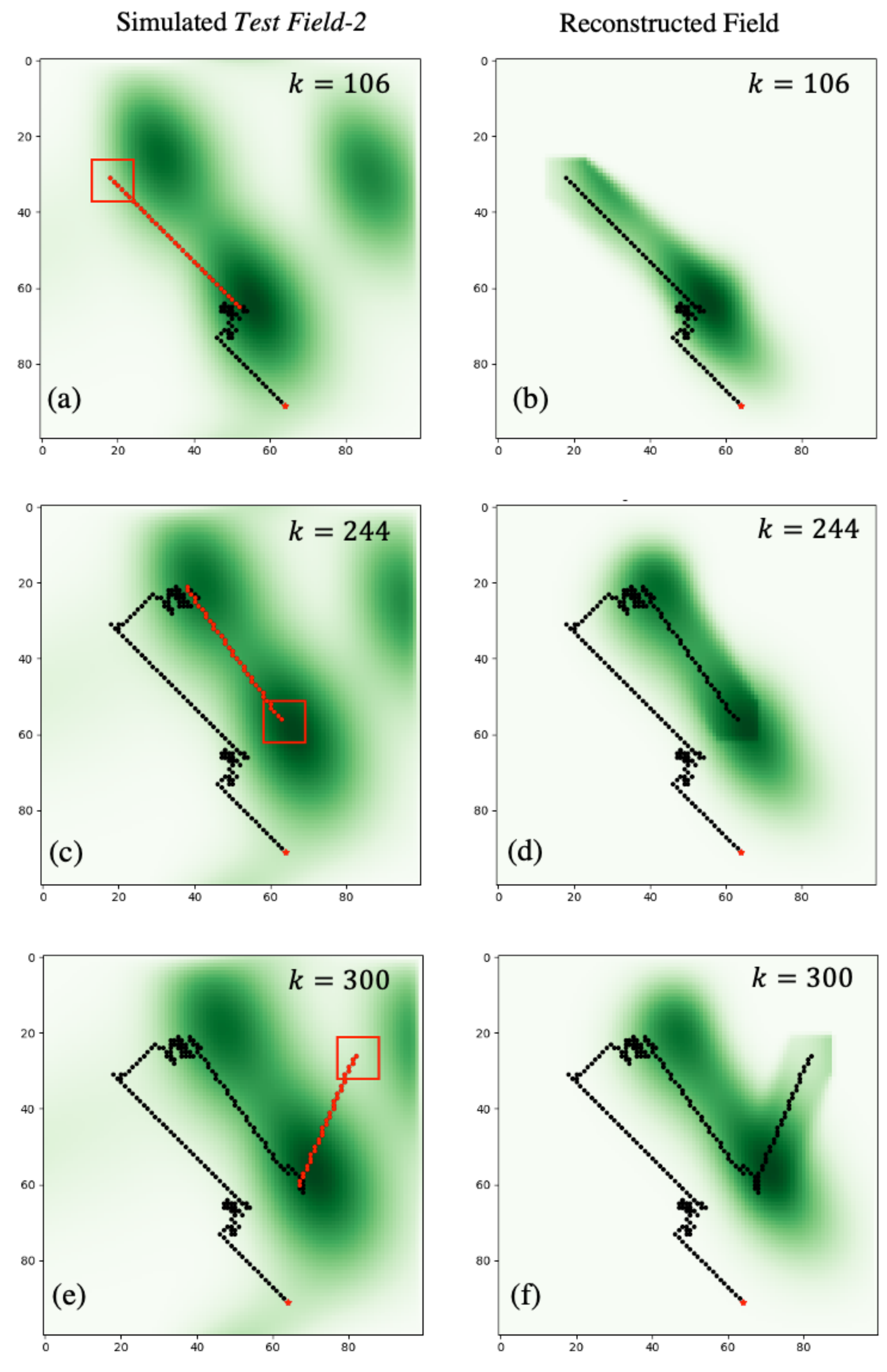


Fig. 37. Sample test episode performed on *Test Field-2*

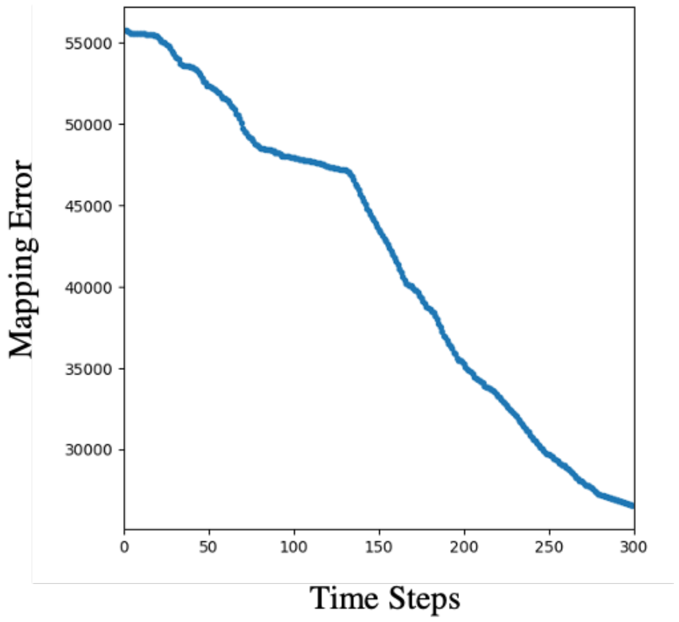


Fig. 38. Mapping error for test episode performed on *Test Field-1*

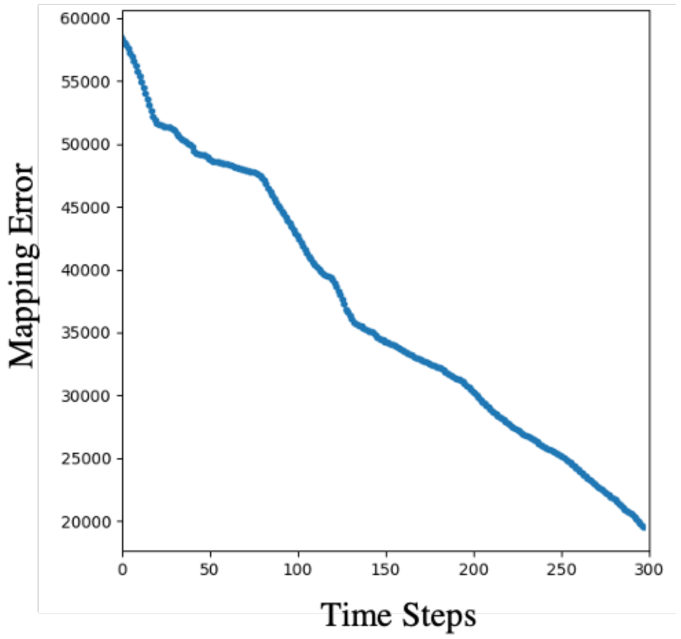


Fig. 39. Mapping error for test episode performed on *Test Field-2*

7.5 Aggregated Mapping Errors with Random Starting Locations on Unseen Fields

In the previous section, we demonstrated test episodes for each of the simulated test fields and explained the behavior and performance of the proposed algorithm on previously unseen fields. In this section, we will run a number of test episodes using the proposed algorithm on both simulated test fields and provide the mapping errors for these episodes in aggregate.

We run 30 test episodes each for *Test Field-1* and *Test Field-2* with randomized formation center starting locations and aggregate the mapping errors for each of the episodes. Fig. 40 and 41 plot the aggregated mapping errors for fields *Test Field-1* and *Test Field-2* respectively. The mean mapping error is plotted in a dark green color, and the green shaded region shows the minimum and maximum bounds over the 30 episodes. As evident from the figures, the mapping error reduces monotonically in aggregate showing that the proposed algorithm is able to command the formation to travel along information-rich paths, trigger exploration as stationary states are reached regardless of the starting location of the formation center on unseen test fields as well.

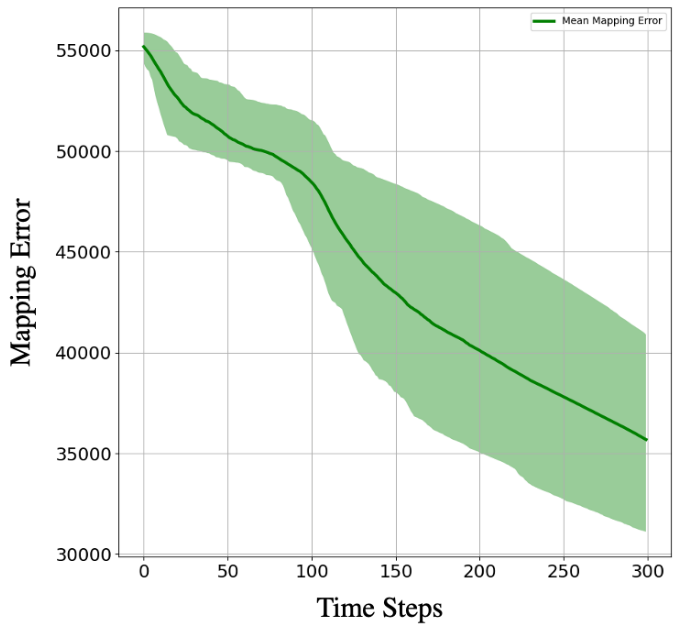


Fig. 40. Mean mapping errors over 30 episodes with random starting locations on *Test Field-1*. The shaded region shows the minimum and maximum bounds.

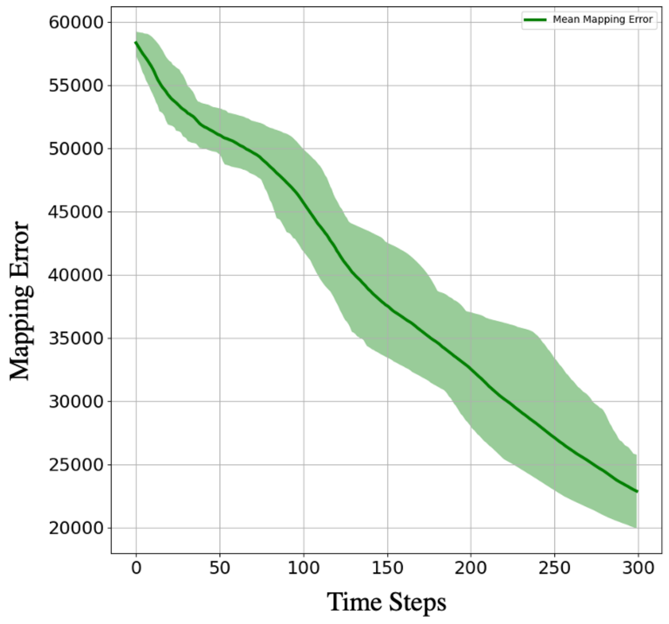


Fig. 41. Mean mapping errors over 30 episodes with random starting locations on *Test Field-2*. The shaded region shows the minimum and maximum bounds.

8 IN-LAB TESTBED

In this chapter, we briefly describe the temporary testbed that we built to develop and test the proposed algorithm. It is important to note that due to the COVID-19 campus closures, we were unable to complete building the testbed setup, and instead opted to perform experiments in a High-Fidelity Simulation Environment (described in Chapter 9) which closely mimics the in-lab testbed. Therefore, we do not include all the implementation details of the setup here. Please refer to Chapter 9 for details on how the High-Fidelity Simulation Environment is built and used.

The in-lab testbed consists of two major components, a field map that mimics the spatial domain Ω, and mobile robotic agents that can move in the said field map.

8.1 Robotic Platform

Fig. 42 shows an image of the modified NVIDIA Jetbot Robot AI kit [36] using the NVIDIA Jetson Nano Developer Kit [44] that we use as the Robotic platform for this work. The NVIDIA Jetson Nano Developer Kit [44] runs a version Ubuntu 18.04 [45] that comes with full Robot Operating System (ROS) [11] support, allowing us to use and build cross-compatible robotics packages. We modified the robotics platform in two main ways. First, we mounted a YDLIDAR G4 sensor [46], which is a 360° two-dimensional rangefinder to the top of the platform. Second, we mounted hall-effect wheel encoder sensors to the motors so that we could obtain wheel odometry.

The 2D LiDAR sensor allows us to measure depth in 360° around the robotic platform in the plane of the sensor. This sensor outputs data as ROS `LaserScan` messages [47], which are then converted to `PointCloud2` [48] type messages to be used for localization. Please refer to Section 9.4.1 for details on this conversion. We added a 3D printed base to the existing platform to mount this sensor.

To enable wheel odometry, we installed a DAGU wheel encoder kit [49] to each of the two motors on the robotic platform chassis. This encoder kit consists of neodymium

8-pole magnets with rubber hubs and two hall-effect sensors, which allow us to measure the forwards and backwards rotation of the wheels. Combining inputs from both wheels allows us to accurately predict the pose of the robot platform in the odometry frame. This input is crucial in the localization process as well, which is further explained in Section 9.5.

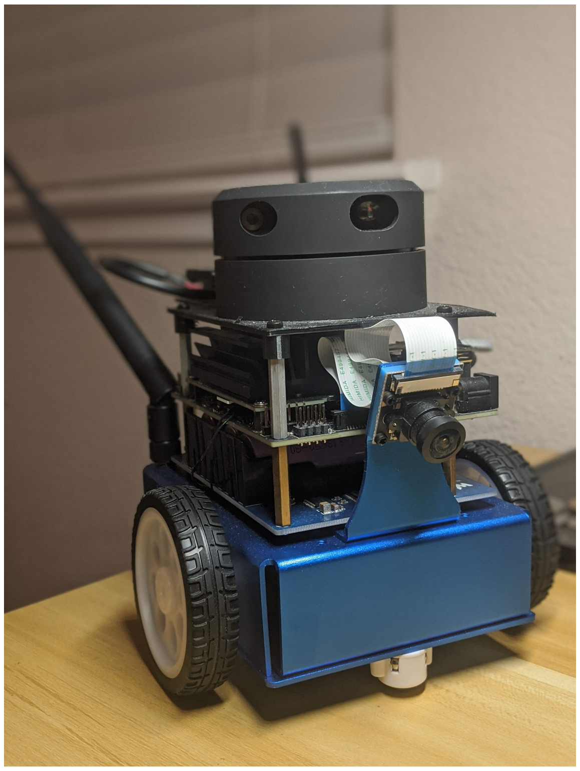


Fig. 42. Photo of the Robotic Platform based on NVIDIA Jetbot kit.

8.2 Field Map Setup

In order to perform experiments, we need a representation of the environment that the robotic platforms move in. Fig. 43 shows the temporary field map that was built using cardboard boxes. The irregular shape of the map provides local features that are useful for successful localization of the robots using point cloud matching algorithms. Fig. 44 shows the point cloud representation of the field map as captured by the 2D LiDAR sensor on a robot.

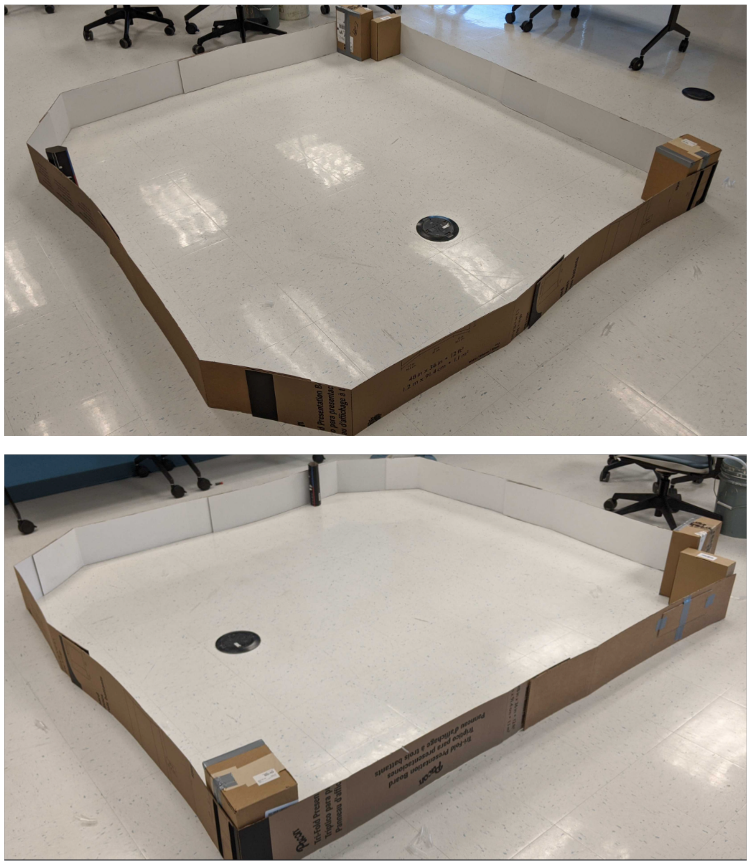


Fig. 43. Temporary field bed set up for in-lab testing.

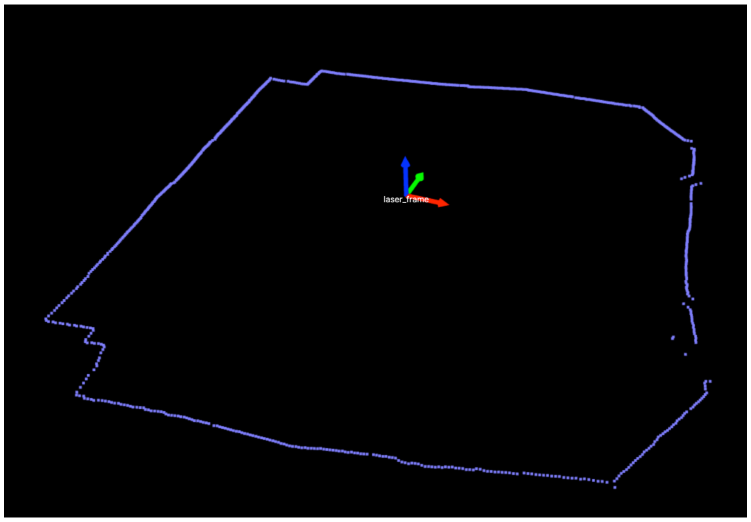


Fig. 44. The Field Map PointCloud generated by the 2D LiDAR on the Robotic Platform.

9 HIGH-FIDELITY SIMULATION TESTING ENVIRONMENT

A major portion of this work consists of extending the path-planning algorithms developed in the Low-Fidelity Simulation Environment to a system that more closely resembles the real-world. To that end, this section elaborates on the High-Fidelity Simulation Testing Environment that is developed to test, adapt and improve the developed algorithms in. Originally, it was planned to extend the algorithms to the in-lab testbed described in Chapter 8. However, due to lack of access to resources on campus during the COVID-19 pandemic, a High-Fidelity Simulation Environment was developed that mimics the in–lab testbed as closely as possible.

9.1 Robot Operating System (ROS) based Simulation Framework

As introduced earlier, Robot Operating System (ROS) [11] is a general-purpose robotics middleware that simplifies the task of setting up communication between various processes running on a robotics system. ROS is a natural fit for such a simulation system as it comes with a variety of simulation and sensor data visualization tools, and many community-created open-source packages. Additionally, using ROS allows us to build packages that can then directly be used in the in-lab testbed (Chapter 8).

9.1.1 Gazebo Simulator

Gazebo [28] is an open-source robotics simulator that has the ability to simulate multiple robots in a 3D environment. Gazebo is a good fit for this purpose for a variety of reasons:

1) Gazebo has a tight integration with ROS [11], which allows us to re-use ROS packages developed for in-lab testbed (Chapter 8).
2) It provides the ability to simulate a variety of sensors, including the sensors that are used on the Robotic Platform (Section 8.1).
3) Gazebo provides a robust physics-engine that simulates interactions appropriately.

4) NVIDIA [50] provides an open-sourced Gazebo model of the Jetbot [36], which our Robotic Platform is based on.

Fig. 45 shows the the simulated version of our Robotics Platform as it renders in Gazebo [28]. The modifications made to the Jetbot model are described in the following sections.

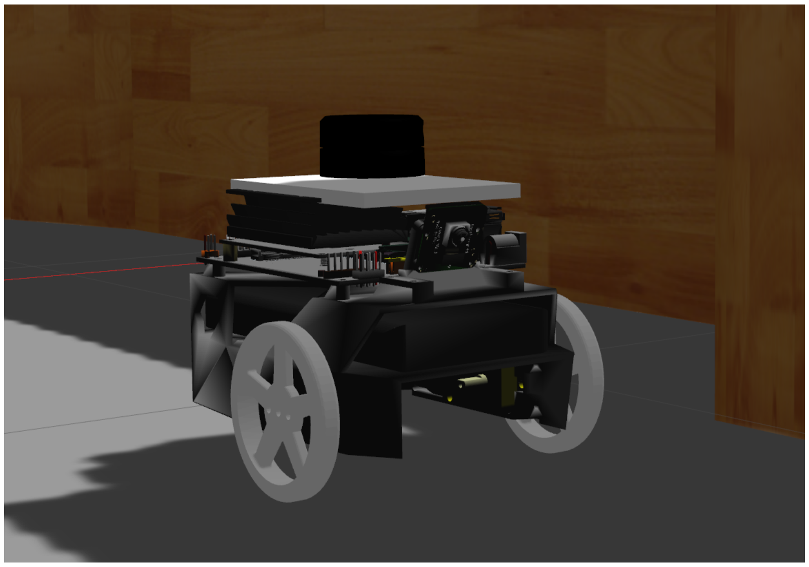


Fig. 45. Simulated version of the Robotics Platform derived from NVIDIA Jetbot kit as rendered in Gazebo.

9.1.2 RViz Visualizer

RViz [51] is a sensor data visualizer built for ROS [11]. It provides the capability to visualize a number of sensor data such as point clouds, images, depth maps, odometry, frames of reference etc., and is therefore extremely useful throughout the robotics development process. Moreover, RViz provides the capability of rendering 3D models which allows us to visualize our robotics platform as well. Fig. 46 shows the visualization of the simulated version our Robotics Platform as it appears in RViz [51].

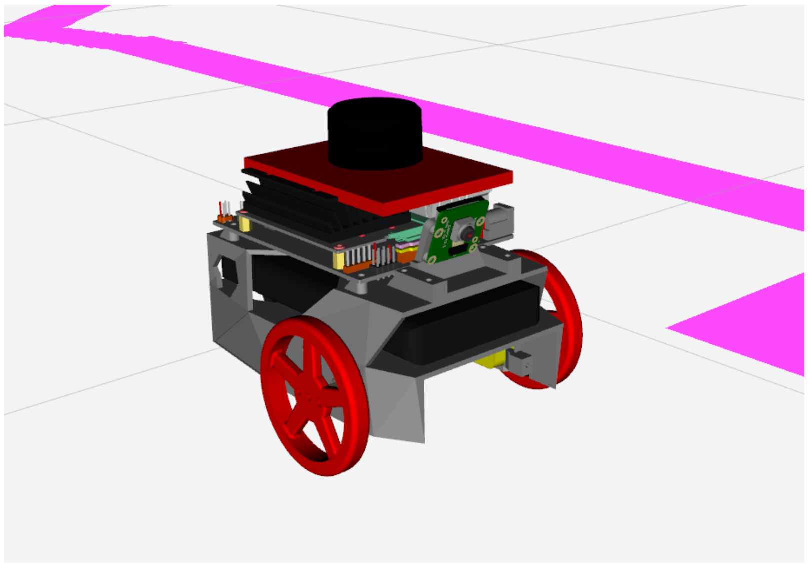


Fig. 46. Simulated version of the Robotics Platform derived from NVIDIA Jetbot kit as rendered in RViz.

9.2 Advection-Diffusion Field Representation

As with other components of the system described in Section 6.1, we need a way to represent the advection-diffusion field in high-fidelity simulation as well. Simulating advection-diffusion adequately is crucial for the transferability of algorithms developed in Low-Fidelity Simulation Environment to the High-Fidelity Simulation Environment. To achieve this, we need a 3D representation of a grid that can hold arbitrary float values. In addition, we need to be able to efficiently update and extract values in the field, be able to visualize the current values inside the field as well as their progression.

The `grid_map` package by ANYbotics [52] fulfills most of these requirements and is a good candidate for advection-diffusion field representation. While the `GridMap` data structure that this package provides was mainly developed for robots to model and visualize terrain, the functionality for accessing, updating and visualizing the `GridMap` makes it an excellent choice to represent the advection-diffusion field in our problem. Fig. 47 and 48 show the 2D and 3D representations of an advection-diffusion field at three separate time steps as visualized inside RViz [51]. Some of the important features of `GridMap` data structure include the following [52]:

1) ROS interface: The `grid_map` meta-package also provides a `grid_map_ros` package that provides the ability to publish and subscribe to `GridMap` messages over ROS topics. We use this capability to publish the updated state of the field during the duration of the experiment.
2) Based on Eigen C++ Matrix library [53]: `GridMap` data is stored as Eigen C++ library [53] data-types, which allows us to directly apply Eigen algorithms and manipulations to the data. This is especially beneficial as Eigen is used extensively in many parts of the stack and provides helpful functions for updating the field data at each time step.
3) Iterators for polygon regions: This package allows users to define polygonal regions and access data as iterators in that region. This is especially useful for accessing the field concentration values inside the view-scope. While the view-scope (Section 3.3) was assumed to be a square of fixed dimension in Low-Fidelity Simulation Environment, as explained in Section 3.3, making data access easy, in this environment, however, the view-scope is expected to be a quadrilateral which each of the vertices positioned by the location of the corresponding robot. Thus, these positions can no longer be assumed to always form a square, making the iterator accessor extremely useful. More details on robot formation are provided in Section 9.6.

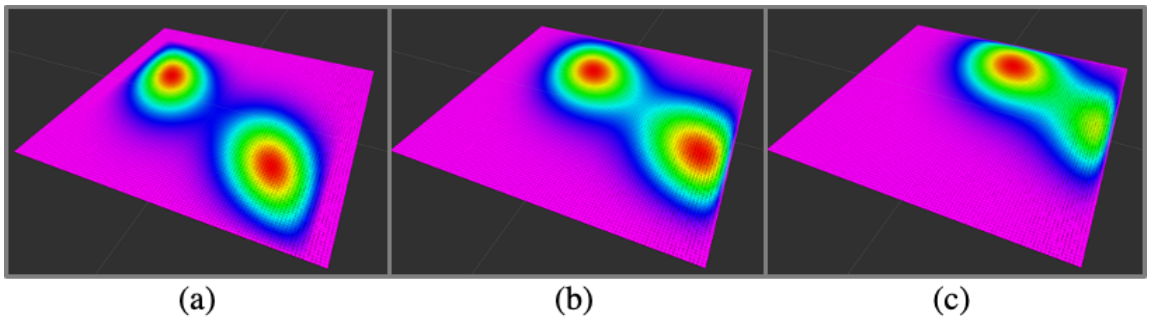


Fig. 47. Representation of an advection-diffusion field grid map in RViz at three different time steps – (a) $k = 0$, (b) $k = 150$, (c) $k = 300$.

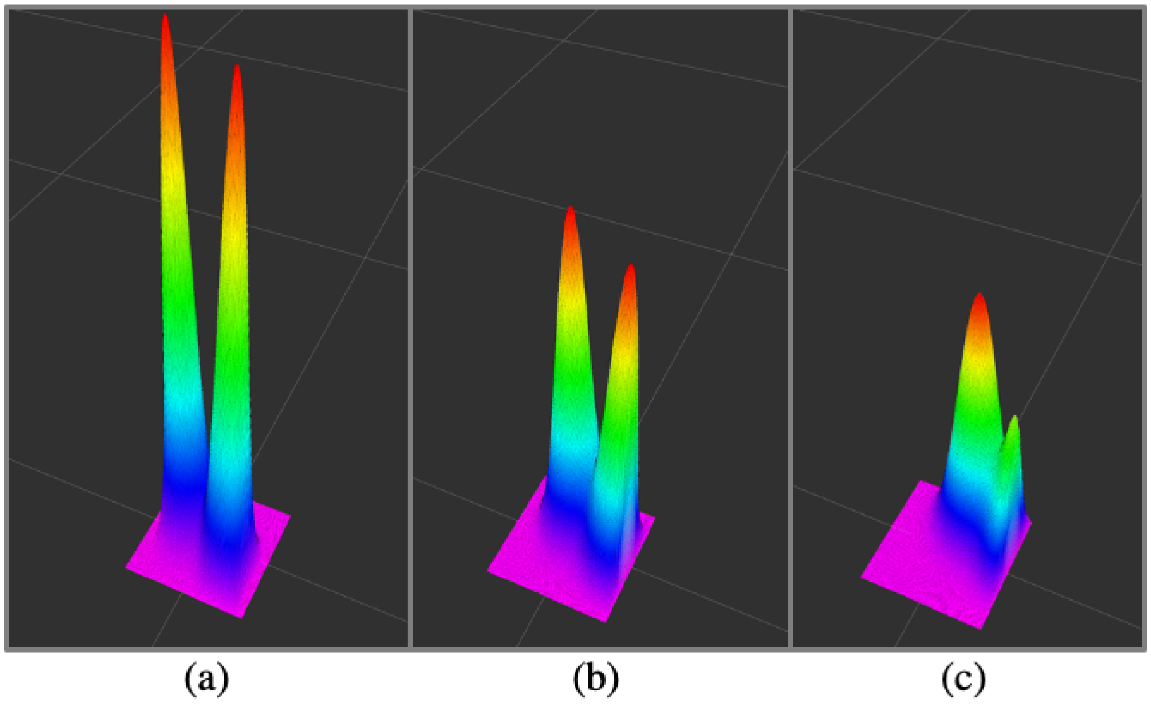


Fig. 48. 3D representation of an advection-diffusion field grid map in RViz at three different time steps – (a) $k = 0$, (b) $k = 150$, (c) $k = 300$. Field concentration values are used to visualize the heights of all the cells. The progression through time demonstrates the effect of diffusion.

9.3 Simulated Field Map

With the representation of the advection-diffusion field defined, we now need a way to anchor it to a fixed frame of reference. This is needed so that we can use the robots' locations as physical locations on the advection-diffusion field. To achieve this, we create a simulated version of the Field Map as explained in Section 8.2. The origin of the field map is named the `map` inertial frame of reference. This `map` frame of reference is aligned such that the transformation from the advection diffusion field's frame of reference, called `field`, to the `map` frame of reference is Identity – no translation and no rotation. This allows us to treat each grid cell in the advection-diffusion field as a 2D location on the XY plane of the `map` reference frame.

Additionally, the simulated field map has an irregular shape similar to the field map in the in-lab testbed, as shown in Fig. 43. This irregular shape provides many features that aid in performing point cloud matching based localization using the 2D LiDAR sensors on the robots. Fig. 49 and 50 shows the simulated field map as it is visualized in Gazebo [28] and RViz [51] respectively.

9.4 Simulated Robot Platform

As mentioned earlier, the robot platform used in High-Fidelity Simulation Environment is derived from the NVIDIA Jetbot [36] robotic platform, which consists of a differential drive robot base, NVIDIA Jetson Nano [44] as the computing platform and an RGB camera. The benefit of using this platform is that the NVIDIA Jetson Nano runs a version of Ubuntu 18.04 [45] operating system that provides full ROS [11] support, making integration into the overall system possible. Additionally, NVIDIA provides a Gazebo [28] model of the Jetbot platform [36] which we modified to suit the purpose of this simulation. Fig. 45 shows the simulated robot platform as it appears in Gazebo [28]. The features and modifications made to this platform are explained in the following sections.

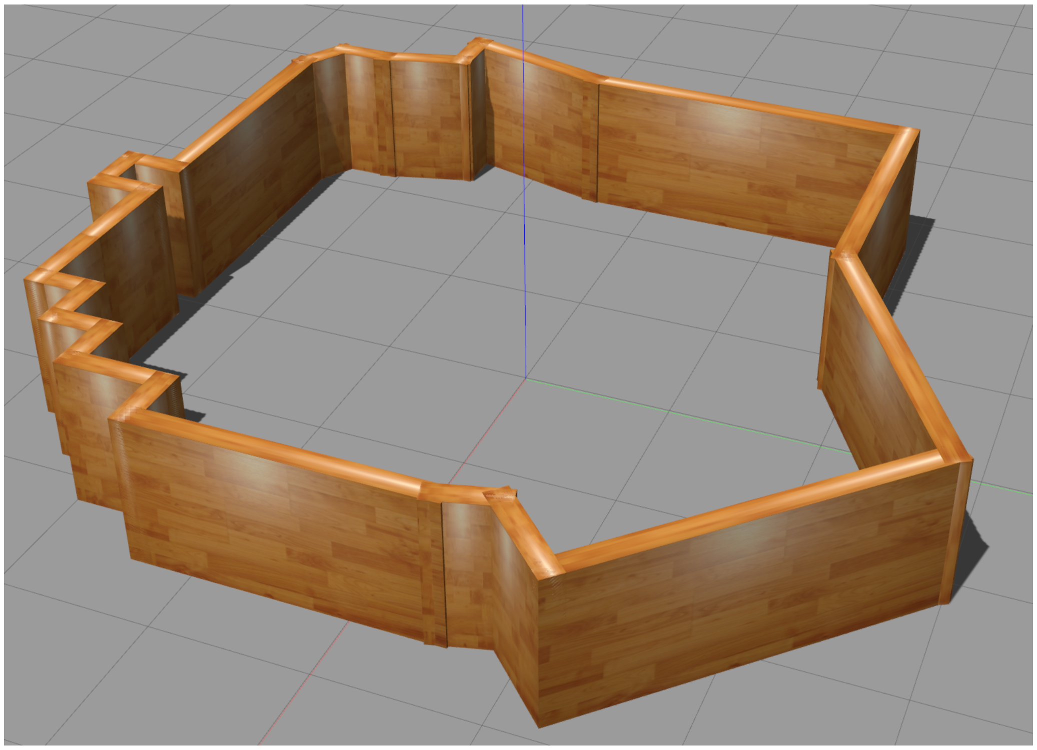


Fig. 49. Simulated Field Map as it appears in Gazebo.

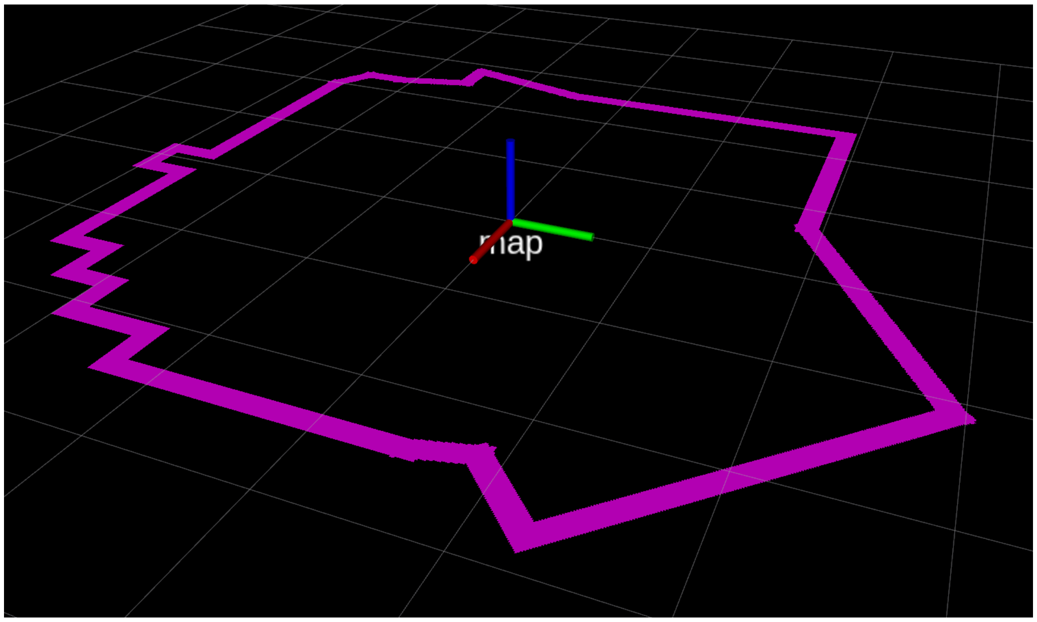


Fig. 50. Simulated Field Map as it is served in RViz by the `map_server` node.

9.4.1 Simulated 2D LiDAR Sensor

The most important addition made to the simulated robot platform is the addition of a simulated 2D LiDAR sensor. This sensor enables us to perceive distances around the robot in a single plane. A 2D LiDAR (Light Detection and Ranging) sensor consists of a single infrared light source and photo-detector mounted on a spinning head. The sensor shoots light beams in all directions as the head spins and the photo-detector detects the light as it is reflected back. The time taken by the light to reach back to the sensor in a particular direction is then used to compute the distance of an obstacle in that direction. Fig. 45 shows how the 2D LiDAR is mounted on the robot platform, and Fig. 51 shows an example of a scan generated by one revolution inside the Simulated Field Map.

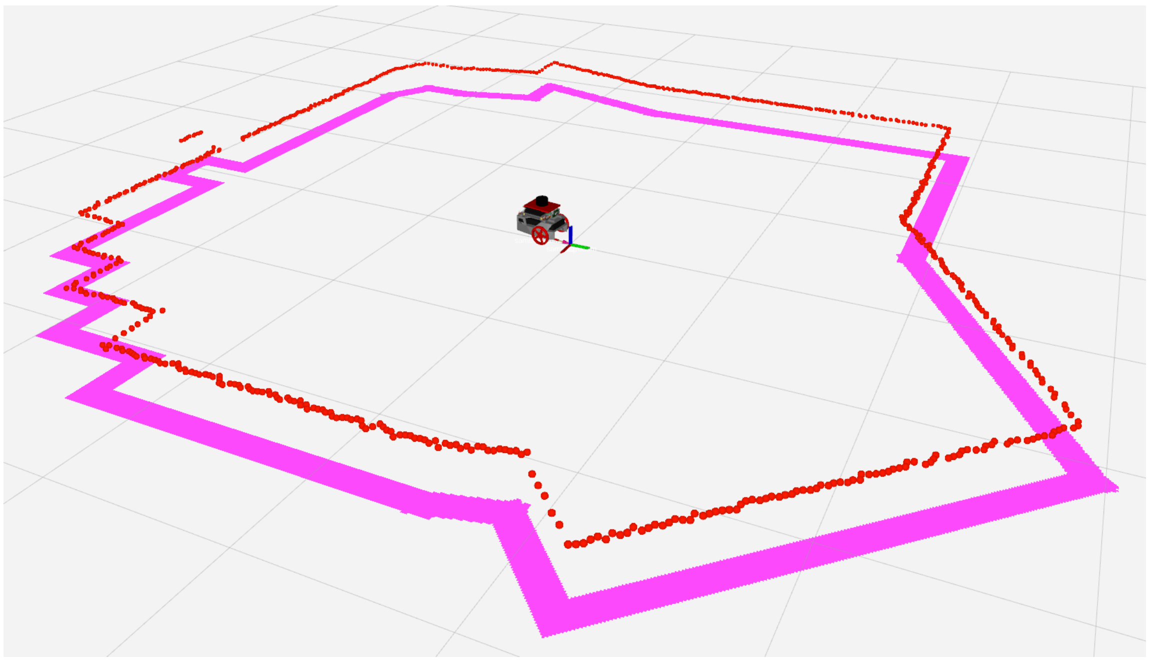


Fig. 51. Point cloud output (in red) from the simulated 2D LiDAR sensor (with Gaussian noise added) mounted on a single robot near the center of the Field Map as visualized in RViz.

One of the features that Gazebo provides is the ability to add simulated sensors. A 2D LiDAR is available as a plugin into Gazebo, and we employed this plugin to define the placement, configuration and settings to match the real LiDAR sensor deployed on the Robotic platform described in 8.1. Some of the important chosen settings include:

1) Scanning frequency: 10 Hz
2) Distance range: 0.1 m to 6 m
3) Radial resolution: 0.5°, which results in 720 points per scan

The default output of the 2D LiDAR is in the format of `sensor_msgs/LaserScan` [47], which encodes the distance values as radial distances. That is, distance values (in meters) are associated with yaw angles (in radians) in the LiDAR reference frame. While this is a natural way to encode distance information from a spinning sensor, encoding distances in point clouds is more useful in practice. Point clouds store 3D point locations with respect to the LiDAR's frame of reference. Thus, a ROS node is developed for converting incoming `sensor_msgs/LaserScan` [47] into `sensor_msgs/PointCloud2` [48] type messages. These point clouds will be used to localize robots in downstream ROS nodes. Details on localization are provided in Section 9.5.

9.4.2 Simulated Odometry

While Gazebo [28] does not provide the ability to simulate rotary encoders which are generally used to estimate odometry, it does provide a plugin to output odometry with respect to the robot's `odom` frame of reference directly. This estimate provides perfect localization of the robot in the `odom` frame, which is not representative of the real-world where odometry drifts over time. Thus, random noise is added to this odometry estimate, which accumulates over time to mimic real-world drift. This drift will be corrected by our localization algorithm described in Section 9.5.

9.4.3 Frames of Reference (TF-Tree)

Fig. 53 shows the frames of reference in the simulation of a single robot in a field. The relationships between these reference frames form a tree-like structure often called the *TF-Tree* (for transformations tree). The `tf2` ROS package [54] is responsible for maintaining this data structure, and providing access to the different frames and their relationships for the duration of the simulation. The tree can be traversed in both directions and transformations (represented by the edges of the tree) can be concatenated to get transformations between any two frames at a given time.

In the section, we elaborate on some of the important frames of references and their relationships. Fig. 53 is the output of the `rqt_tf_tree` package [55] provided by ROS. Please note that ”sambot” is the representative name of a robot in the High-Fidelity Simulated Environment. Fig. 52 shows the physical locations of the frames of reference for a single robot and labels the frames discussed below.

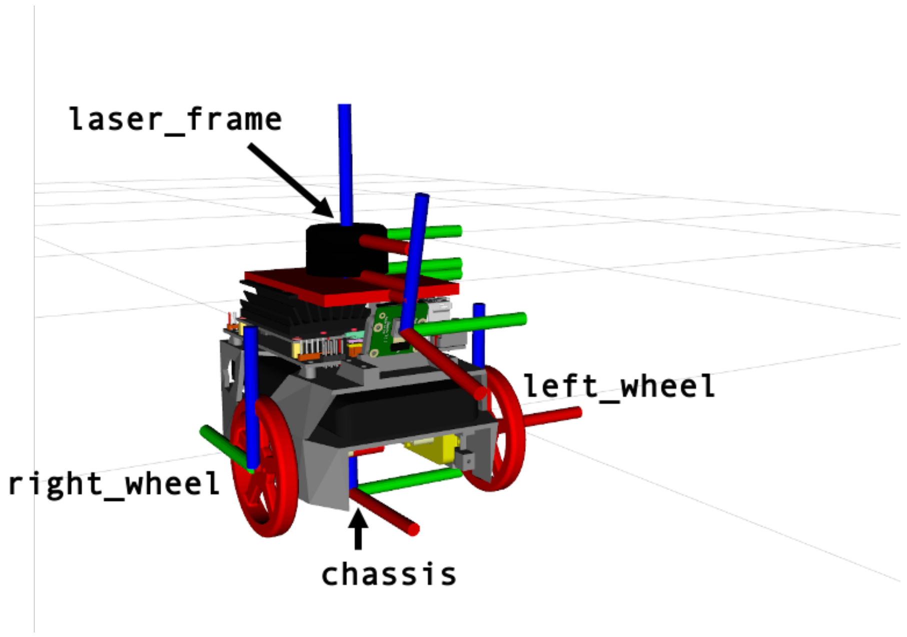


Fig. 52. Important frames of reference for a single robot as visualized in RViz.

- `map` frame: The `map` frame represents the origin of the local environment that the robots are operating in. In our case, it is the origin of the Simulated Field Map as shown in Fig. 50. The pose of a robot in the `map` frame of reference is referred to as its localization. Additionally, the `map` reference frame also acts as the origin of the simulated advection-diffusion field grid map (Section 9.2).

- `sambot1/odom` frame: The `odom` frame refers to the map-fixed frame of reference in which a robot's (here `sambot1`) pose, as calculated using wheel odometry, is defined. Each robot has its own `odom` frame, and it is usually chosen as the position of the robot at the beginning of the simulation. As per ROS convention, odometry is considered to be continuous and thus a good and accurate short-term reference of the robot's location. However, odometry accumulates errors and tends to drift over time making it a poor reference for the robot's location over a long-term.
- `sambot1/chassis` frame: The `chassis` frame represents the root frame for the *TF-Tree* of a single robot. The process of determining the pose of a robot's `chassis` frame in the `map` frame is referred to as the localization of that robot. All other frames of reference for each robot are linked either statically or dynamically to the robot's `chassis` frame.
- `sambot1/laser_frame` frame: The `laser_frame` is the frame of reference with respect to which all measurements from the 2D LiDAR scanner are defined. That is, all of the points in a single `sensor_msgs/PointCloud2` measurement from the 2D LiDAR scanner are defined with the `laser_frame` as their origin. For each robot, the `laser_frame` is assumed to be rigidly attached to its `chassis` frame, and the relative transformation from the `chassis` to `laser_frame` through `base_laser` frame is defined by the 3D model of the robot in Gazebo [28].

In a four-robot simulation, the sub-tree rooted at the `odom` frame will be replicated for each of the robots.

9.5 Localization

The process of localization of a robot refers to determining the location and orientation of the robot with respect to an external inertial frame of reference. For the purpose of this simulation, localization of a robot refers to the process of determining the pose of its `chassis` frame with respect to the `map` reference frame. Determining the location of each robot in the `map` reference frame allows us to locate the position of the grid cell that the robot is currently on top of, and thus, access its values. This allows the robot to ”sense” the concentration of the advection-diffusion field at that location as described above.

As explained in Section 9.4.3, the pose of a robot's `chassis` frame as expressed in the robot's `odom` frame is provided by simulated odometry in Gazebo. Thus, at a given time, the transformation from the robot's `chassis` frame to its `odom` frame is known. Let ${}^{odom}T_{chassis}$ denote this transformation, which can be read as

1) The pose of the `chassis` frame as expressed in the `odom` frame, or
2) as the transformation that transports a point defined in the `chassis` reference frame into the `odom` reference frame.

Additionally, the transform from the robot's `chassis` frame to the `laser_frame`, ${}^{chassis}T_{laser}$, is provided by the robot's CAD model and is assumed to be a static transform as the 2D LiDAR sensor is rigidly attached to the robot.

Thus, the task of the localization system reduces to determining the ${}^{map}T_{odom}$ transform, the pose of the `odom` reference frame as expressed in the `map` reference frame. With ${}^{map}T_{odom}$ determined, computing ${}^{map}T_{chassis}$ gives us the pose of the robot in the `map` reference frame. The overview of this process is provided in Fig. 54.

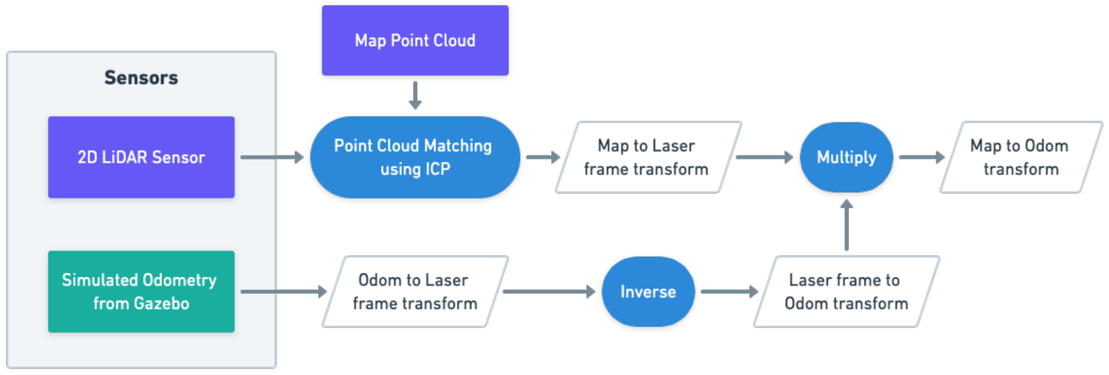

Sensors
2D LiDAR Sensor
Simulated Odometry from Gazebo
Map Point Cloud
Point Cloud Matching using ICP
Map to Laser frame transform
Multiply
Map to Odom transform
Odom to Laser frame transform
Inverse
Laser frame to Odom transform

Fig. 54. Overview of the localization process for a single robot.

9.5.1 Point Cloud Matching using Iterative Closest Point Algorithm

The Iterative Closest Point algorithm (ICP) [56] is a point cloud matching algorithm that is used to align two point clouds observing the same local environment but collected at different poses within that local environment. The result of this alignment provides the relative transformation between the two poses at which the point clouds are observed from. Given that each robot is equipped with a simulated 2D LiDAR sensor, and that the map that the robots are operating in is static and already known, ICP can be used to get the relative transformation between the `map` reference frame and the `laser_frame` reference frame, ${}^{map}T_{laser}$. The ICP algorithm implementation [57] provided by the Point Cloud Library (PCL) [58] is used for this procedure.

9.5.2 Procedure

With all the transformations, their sources and relationships defined, we can now develop the procedure to obtain the ${}^{map}T_{odom}$ transform. Since the transformations are expressed as 4×4 invertible matrices in homogeneous coordinates, we can use matrix algebra to obtain the expression for ${}^{map}T_{odom}$. First, traversing the *TF-Tree* downward, we obtain the following expression.

$$ {}^{map}T_{laser} = {}^{map}T_{odom}\ {}^{odom}T_{chassis}\ {}^{chassis}T_{base}\ {}^{base}T_{laser}, $$

which can be simplified and rewritten,

$$ \begin{aligned} \underbrace{{}^{map}T_{laser}}_{\text{ICP}} &= {}^{map}T_{odom}\quad \underbrace{{}^{odom}T_{chassis}}_{\text{Simulated Odometry}}\quad \underbrace{{}^{chassis}T_{base}\ {}^{base}T_{laser}}_{\text{Static Transforms}} \\ &= {}^{map}T_{odom}\ {}^{odom}T_{chassis}\ {}^{chassis}T_{laser} \\ &= {}^{map}T_{odom}\ {}^{odom}T_{laser}. \end{aligned} $$

Rearranging and right multiplying with $({}^{odom}T_{laser})^{-1}$ yields,

$$ \begin{aligned} {}^{map}T_{odom}\ {}^{odom}T_{laser} &= {}^{map}T_{laser} \\ {}^{map}T_{odom}\ {}^{odom}T_{laser}\ ({}^{odom}T_{laser})^{-1} &= {}^{map}T_{laser}\ ({}^{odom}T_{laser})^{-1} \\ {}^{map}T_{odom} &= {}^{map}T_{laser}\ ({}^{odom}T_{laser})^{-1}. \end{aligned} \tag{33} $$

Equation (33) provides us with an expression to determine ${}^{map}T_{odom}$. This transform is computed each time a 2D LiDAR point cloud is received and is then broadcast over the *TF-Tree*, which updates and corrects the ${}^{map}T_{chassis}$ transform that represents the location and orientation of the robot as expressed in the `map` reference frame.

9.6 Motion and Formation Control

In this section, we describe how the simulated robots are set up so that they can be controlled by sending velocity commands. As mentioned previously, the simulated robots follow a differential-drive scheme, which means that the robot's left and right wheels are controlled by separate motors, and can be driven forwards or backwards independently, and at different speeds. This allows the robots to move forwards, backwards, follow an arc, and even rotate in place. The differential-drive scheme is popular for many small robots for its simplicity and flexibility.

The Gazebo [28] simulator provides simulated motor actuators, as well as a plugin for a differential-drive controller that we employ. This plugin allows us to abstract away the low-level control and simply focus on providing velocity commands to the robot's wheels to achieve the motion that we desire. This plugin converts the supplied velocity commands into voltage signals that are sent to the motors to actuate. The Gazebo [28] server subscribes to the `cmd_vel` topic for each robot (for example `/sambot1/cmd_vel` for `sambot1`), which carries `geometry_msgs/Twist` [59] type messages. The `geometry_msgs/Twist` contains linear and angular velocity commands for the robots. Therefore, the task of moving the robots is simplified to providing velocity commands at each time step.

Recall that are our proposed algorithm is designed to provide discrete actions to move the formation center (Equation (17)). Knowing the formation's current location, and using the current action provided by the proposed algorithm, we can produce the target locations for the formation center to move to. However, computing velocity commands for the robots in order to move the formation center to the goal location is a non-trivial task and requires controller design. Fortunately, the ROS navigation ecosystem provides a useful package called `move_base` [60] that is designed specifically to command differential-drive robots to move from the current location to a goal location. Using this package allows us to abstract away the velocity commands, and simply provide the goal locations for each of the robots at each time step to direct them to move in a formation.

The `move_base` [60] node is set up for each of our robots such that given a goal location in the `map` frame, the `move_base` node publishes velocity commands in order to reach the specified goal location. Therefore, having accurate localization in the `map` frame is extremely important for the proper functioning of the `move_base` node. The `move_base` node links together a local and global planner to accomplish the navigation tasks, and provides a number of parameters that can be tuned to improve navigation performance.

Now, to complete the navigation and formation tasks, we create a ROS node called `goals_publisher` that runs the proposed algorithm at each time step and determines the action that the robot formation needs to take at each time step. Then, using the current location of the formation and the current action, it determines the next goal location for the formation. Since the robots are meant to travel in a fixed formation at fixed distances away from the formation center throughout the episode, we calculate the goal locations for each of the robots in the `map` frame and broadcast them over the *TF-Tree* and as `geometry_msgs/PoseStamped` [61] messages over the `move_base_simple/goal` topics for each robot. The `move_base` node for each robot then publishes velocity commands until the goal location is reached.

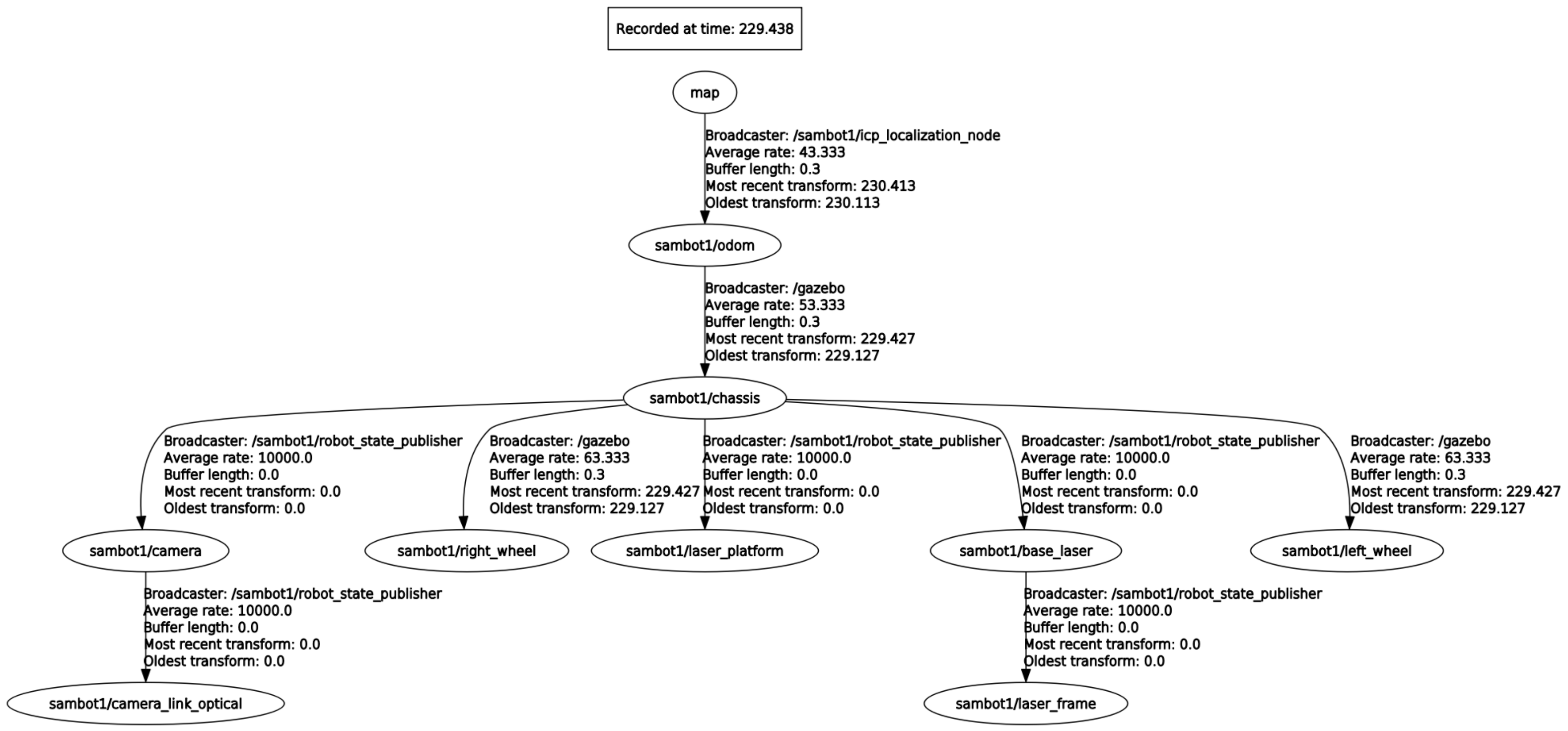

Recorded at time: 229.438
map
Broadcaster: /sambot1/icp_localization_node
Average rate: 43.333
Buffer length: 0.3
Most recent transform: 230.413
Oldest transform: 230.113
sambot1/odom
Broadcaster: /gazebo
Average rate: 53.333
Buffer length: 0.3
Most recent transform: 229.427
Oldest transform: 229.127
sambot1/chassis
Broadcaster: /sambot1/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /gazebo
Average rate: 63.333
Buffer length: 0.3
Most recent transform: 229.427
Oldest transform: 229.127
Broadcaster: /sambot1/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /sambot1/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /gazebo
Average rate: 63.333
Buffer length: 0.3
Most recent transform: 229.427
Oldest transform: 229.127
sambot1/camera
sambot1/right_wheel
sambot1/laser_platform
sambot1/base_laser
sambot1/left_wheel
Broadcaster: /sambot1/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /sambot1/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
sambot1/camera_link_optical
sambot1/laser_frame

Fig. 53. Frames of reference and their relationships in a single robot (known as sambot in simulation) simulation. This is the output of the `rqt_tf_tree` node provided by ROS. This representation of the frames of reference is commonly known as *TF-Tree*.

10 TESTING RESULTS IN HIGH-FIDELITY SIMULATION ENVIRONMENT

In this chapter, we show results of implementing and testing the proposed algorithm on fields simulated in the High-Fidelity Simulation testing environment elaborated in Chapter 9. To be able to perform test episodes in the High-Fidelity Simulation Environment, all major components of the proposed algorithm (Section 5) are implemented as ROS nodes that interact with each other to realize the algorithm. Fig. 55 shows the ROS computation graph showing all the ROS nodes and their connections for running the algorithm on a formation of four mobile robots. Since this graph is large and difficult to fully read and interpret, we provide Fig. 56 and 57 which show the two major subgraphs within the entire computation graph.

To demonstrate the efficacy of the proposed algorithm in the High-Fidelity Simulation Environment, we perform a test episode on simulated *Field-1*, described in Table 1 and shown in Fig. 7. We used the DQN model trained on *Field-3* to run this experiment, showing that a model trained on a complex simulated field can be generalized to a simpler simulated field. Fig. 58 shows the state of the test episode at time steps $k = 126, k = 234$ and $k = 300$. Additionally, Fig. 59 shows the mapping error over the course of this test episode. Fig. 60 shows a close-up snapshot of the four robots moving in a formation. It is important to note that the Simulated Field Map (described in Section 9.3) and the point cloud outputs of the 2D LiDAR sensors (shown in Fig. 51) are hidden in these figures to avoid clutter. Additionally, the path in red shows the trajectory followed by the formation center and the green polygon formed by the locations of the four mobile robots represents the view-scope. Figures in the first column show the simulated-advection diffusion field, while figures in the second column show the reconstructed field following the proposed algorithm.

At $k = 126$, in Fig. 58(a) and 58(b), the formation has reached a stationary state in the field, identified the advection coefficients, chosen a destination, planned a path to the destination and is moving towards the destination. The identified advection parameters are $\hat{\mathbf{v}} = [0.62, -0.83]$, which are similar to the true advection parameters for *Field-1* of $\mathbf{v} = [0.6, -0.8]$ as stated in Table 1. At $k = 234$, in Fig. 58(c) and 58(d), we see that the formation has reached another stationary state in the field, and thus chooses another destination to trigger exploration and moves to the destination. After reaching the destination, the control is transferred back to the DQN model and the formation tracks the second high-concentration zone until the end of the episode. As observed in the episodes run in the Low-Fidelity Simulation Environment, the mapping error reduces at a higher rate when new high-concentration zones are found. As evident from Fig. 58 and 38, the results of running episodes in the High-Fidelity Simulation Environment are very similar to the results achieved in the Low-Fidelity Simulation Environment, showing that the algorithm can be adapted to more realistic scenarios.

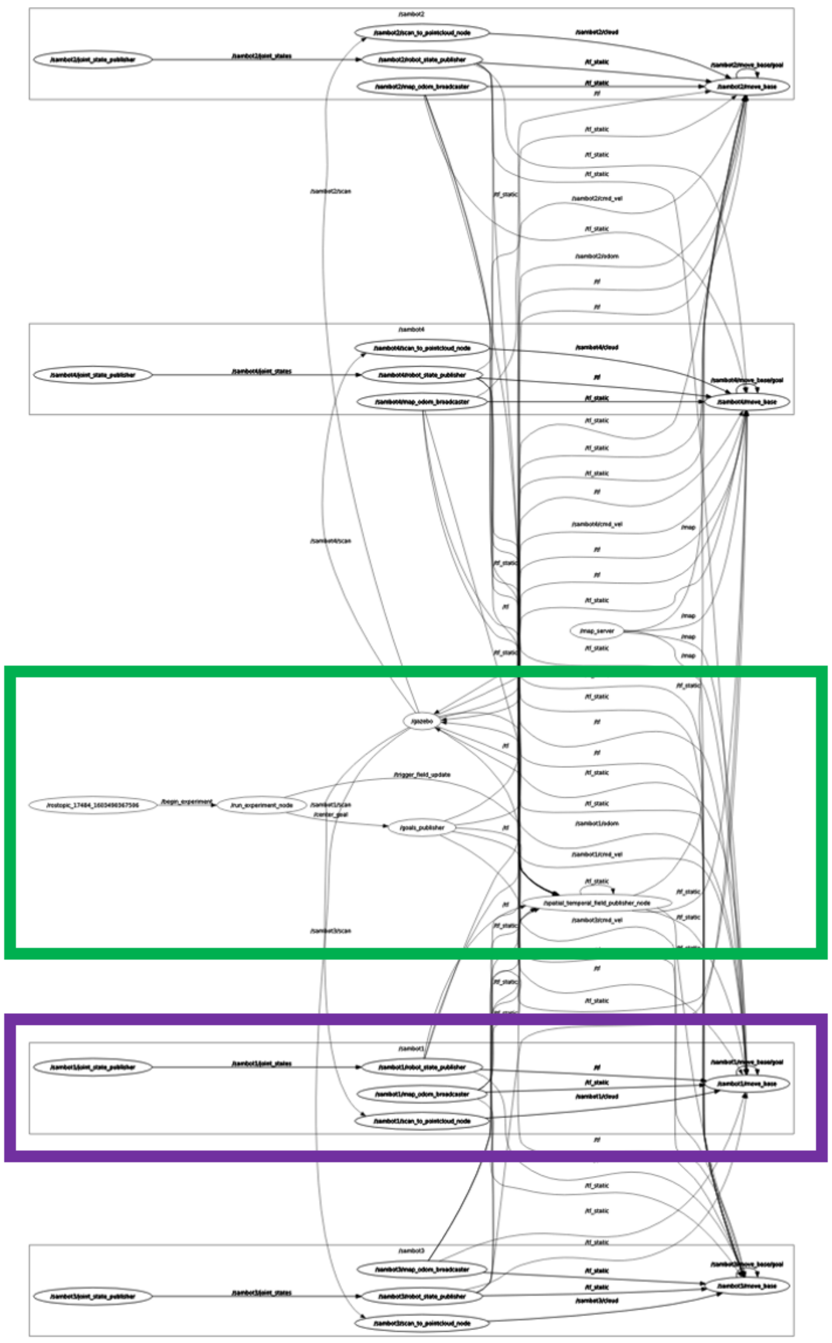


Fig. 55. ROS computation graph while running a test episode in the High-Fidelity Simulation Environment. This output is generated using the `rqt_graph` tool. Two important subgraphs are highlighted in this graph and zoomed-in images for these regions are provided below. The green highlighted section corresponds to all the nodes that are responsible for running the proposed algorithm and maintaining the advection-diffusion fields. Fig. 56 shows this subgraph zoomed in. The purple highlighted section shows all the nodes running within the `sambot1` namespace. Fig. 57 shows this subgraph zoomed in.

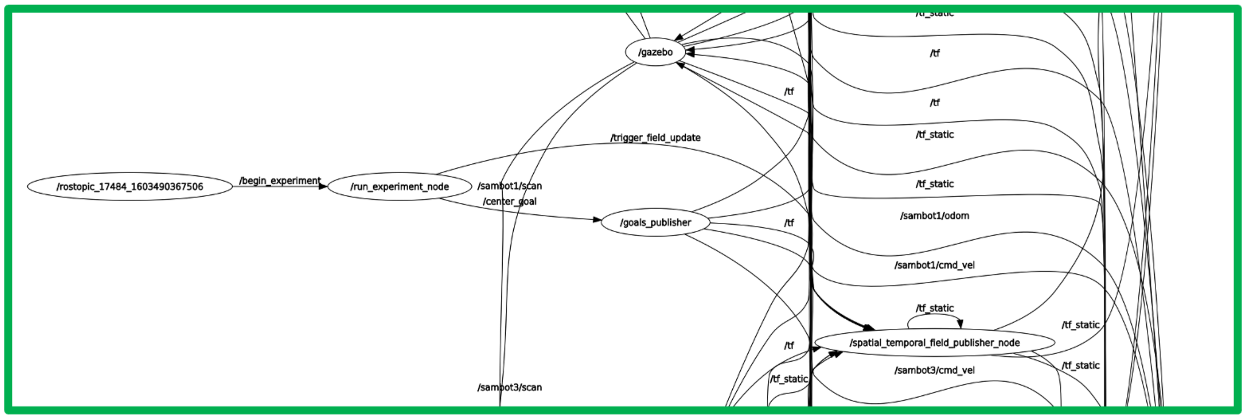


Fig. 56. This subgraph shows all the nodes required for running the episode with the proposed algorithm. `run_experiment_node` is responsible for managing the entire episode. It runs all major components of the proposed algorithm, decides the action that the formation must take and publishes the subsequent location on the `center_goal` topic. The `goals_publisher` node is responsible for computing and publishing goal locations for each of the four robots. The `spatial_temporal_field_publisher_node` is responsible for maintaining the simulated advection-diffusion field as well as evolving the reconstructed field as the formation travels through the field. The `gazebo` node runs Gazebo Simulator's server-side node responsible for running the physics engine and simulating the robots' movements.

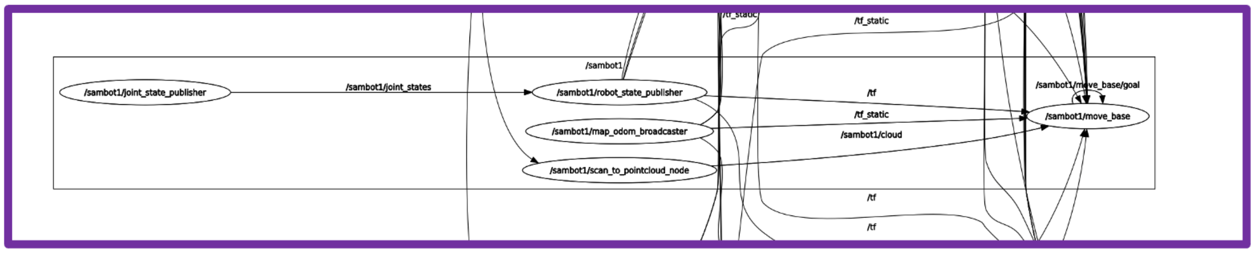


Fig. 57. This subgraph shows the nodes that need to be run for one simulated mobile robot. This figure shows the namespace for `sambot1`. All nodes in this namespace are replicated for each of the other three robots. The `joint_state_publisher` node is responsible for publishing the states of each of the joints in the robot which change as the robot moves in the field. The `robot_state_publisher` consumes the joint states and publishes them as transforms over the `tf` topic. The `map_to_odom_broadcaster` node publishes simulated odometry and the `scan_to_pointcloud` node converts `LaserScan` messages from the simulated LiDAR sensor to `PointCloud2` type messages and publishes them. The `move_base` node is responsible for sending velocity commands to the robot to reach the goal locations specified by the proposed algorithm at each time step.

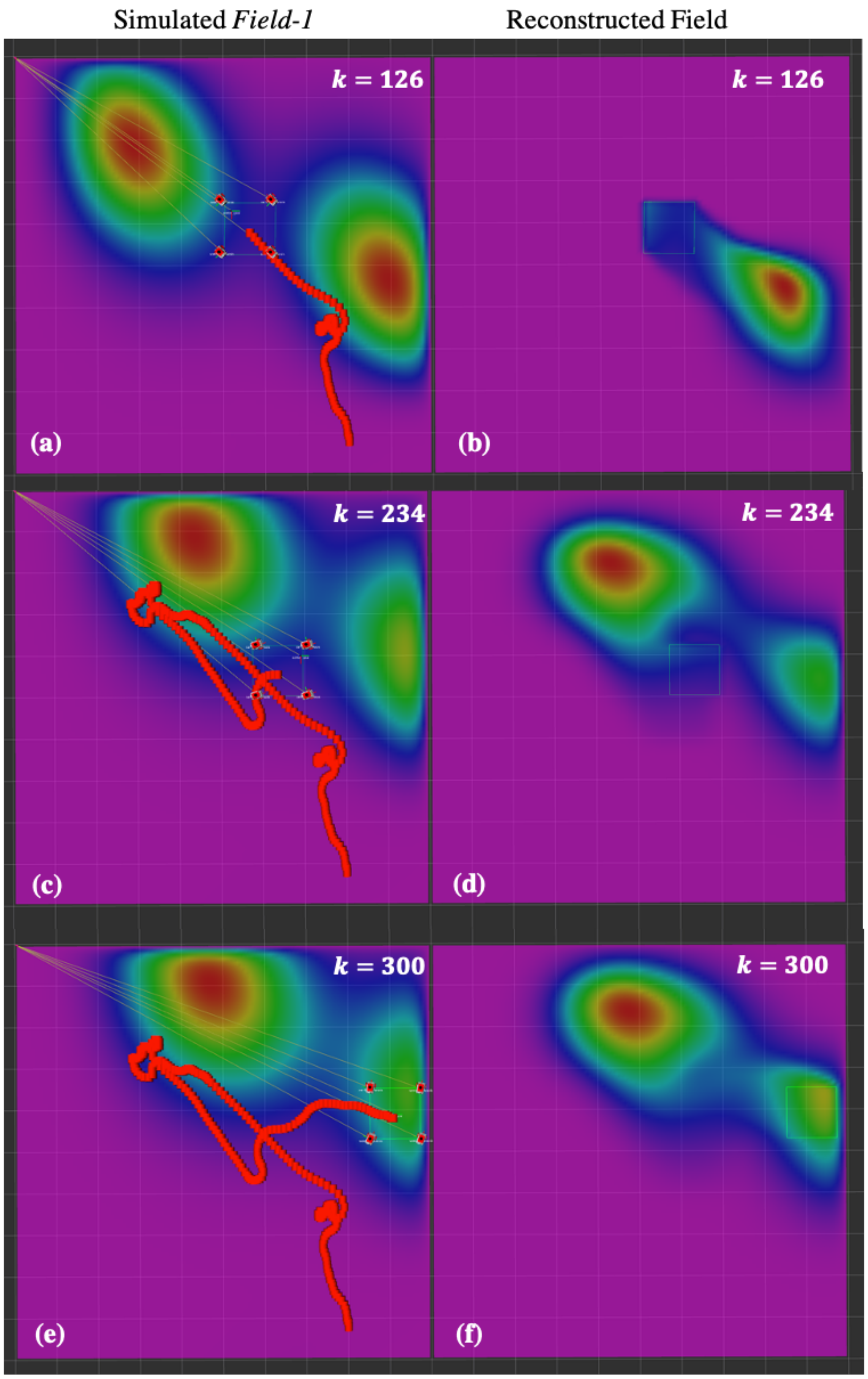


Fig. 58. Sample test episode performed on *Field-1* in High-fidelity Simulation Testing environment.

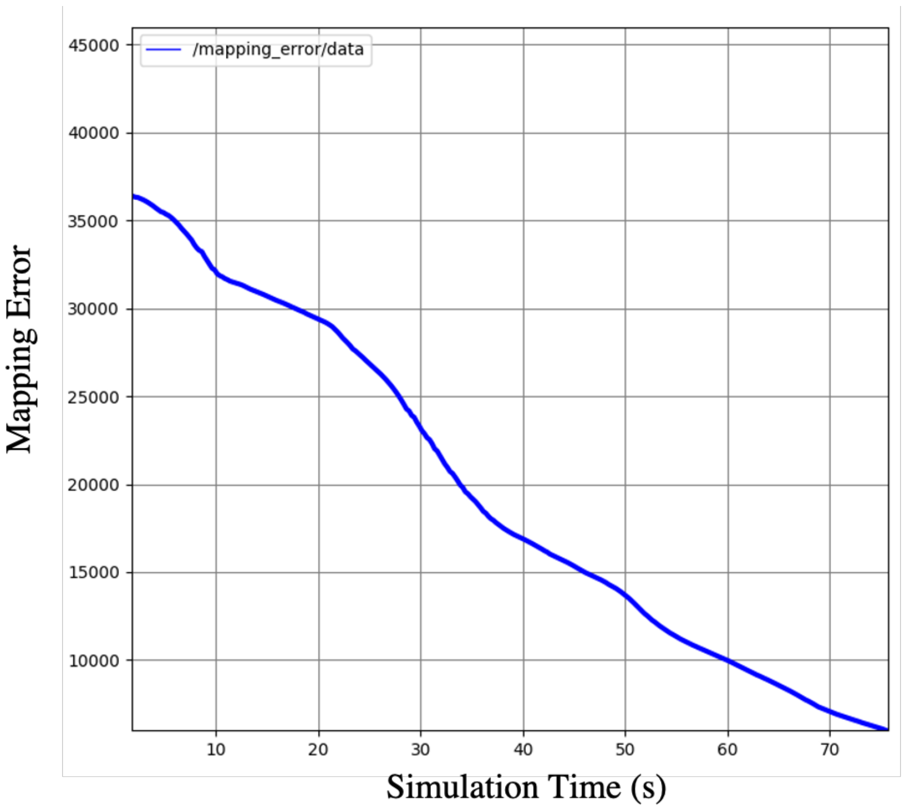


Fig. 59. Mapping error for test episode performed on simulated *Field-1* in High-fidelity testing environment. Please note that the units on the *x*-axis are seconds elapsed during the simulation in ROS and not the time steps (k) as in previous mapping error plots.

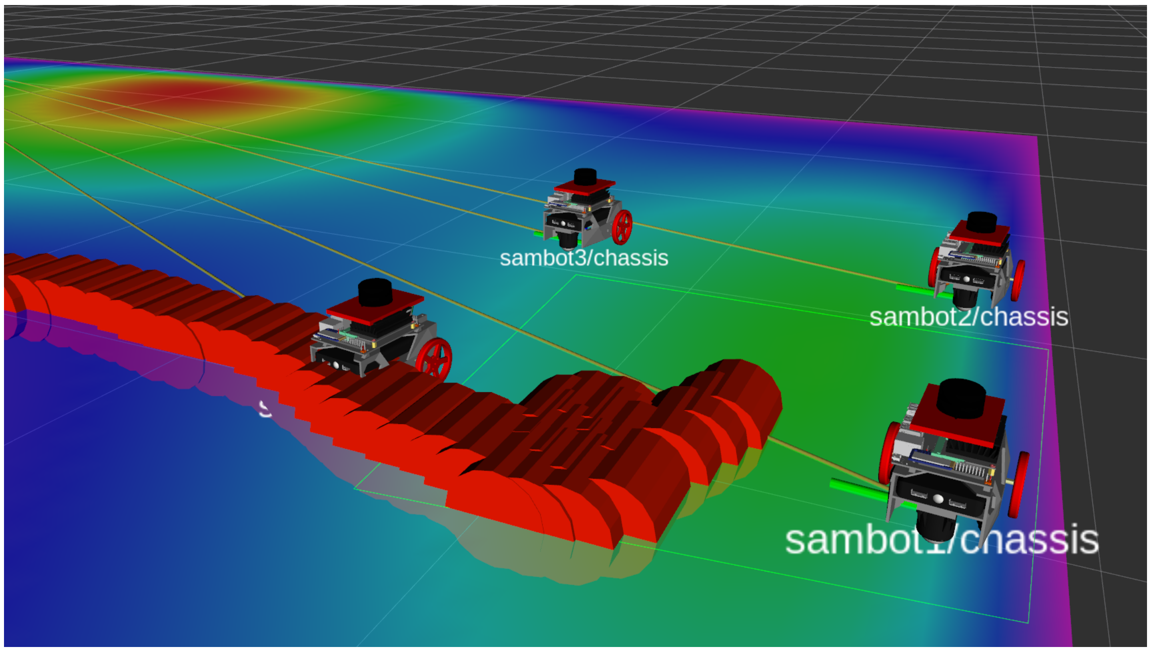


Fig. 60. Snapshot of the mobile robot formation moving in the simulated advection-diffusion field.

11 CONCLUSION

In this work, our primary goal was to build a path-planning algorithm that can be used for reconstruction of advection-diffusion fields using a number of mobile sensing robots traveling through the field domain of interest. We formulated the problem and proposed a deep reinforcement learning based algorithm that directs the mobile sensing robot formation to travel along information-rich trajectories. In addition, we added mechanisms to allow the robot formation to identify advection parameters required to successfully reconstruct the field. Moreover, our algorithm also encourages exploration in the field domain as stationary states are reached, which improves field reconstruction performance significantly as it allows the formation to reach multiple high-concentration zones that may exist in the field domain.

To train the deep reinforcement learning models, and to validate and test the performance of the proposed algorithm, we provide a Low-Fidelity Simulation Environment. We show sample episodes using the proposed algorithm on multiple training and test advection-diffusion fields that show satisfactory reconstruction results, and reduction in the mapping error for all episodes. Finally, we also provide a High-Fidelity Simulation Environment based on Robot Operating System (ROS) [11]. The High-Fidelity Simulation Environment simulates real robots and we show sample test episode using the proposed algorithm in this environment. We demonstrate that the proposed algorithm achieves expected results in both the Low-Fidelity Simulation Environment, as well as the High-Fidelity Simulation Environment.

12 FUTURE WORK

In this work, we propose a deep reinforcement learning based path-planning algorithm that is shown to perform advection-diffusion field reconstruction using multiple mobile sensing robots to a satisfactory degree in two simulation environments of differing fidelities. However, there are several practical improvements that be considered for the future to make the performance better and more generalizable. In this section, we will focus on future improvements for four components of this research: deep reinforcement learning, simulation, real-world applications, and extension to other types of mobile robots.

In Chapter 5.2, we showed how Deep Q-Networks can be trained to control the mobile robot formation to follow information-rich paths on a simulated advection-diffusion field. We also showed that a model trained on one advection-diffusion field is able to generalize on unknown fields as well. However, the performance of generalization can be improved by training the same model from different simulated advection-diffusion fields. Additionally, a larger and wider network architecture with more weights can learn good policies for many different kinds of advection-diffusion fields. Therefore, training on bigger networks with multiple fields must be tested in the future. In addition, we would also like to try other deep reinforcement learning algorithms and techniques to train the models. While DQN models provide good performance, methods such as policy gradient and actor-critic may result in better policies and/or faster training.

Currently, the High-Fidelity Simulation Environment introduced in Chapter 9 is based on the ROS version 1 [11]. A newer version of ROS, known as ROS2 [62], provides substantial benefits to message passing, robustness and scalability. In the future, it would be important to port the High-Fidelity Simulation Environment to ROS2 [62].

While the High-Fidelity Simulation Environment (Chapter 9) provides a fairly realistic environment, we still use a simulation of an advection-diffusion field in it. It is important to test the proposed algorithm in an actual environmental advection-diffusion field, and with mobile robots installed with sensors to directly measure the concentrations. While testing with real robots, in a real advection-diffusion field was originally planned, we were unable to perform these tests due to campus closures caused by the COVID-19 pandemic.

Finally, we would like to extend results produced in this work with other types of mobile robots, such as aerial drones and legged robots. Since both types of robots provide different kinds of operational field domains, the algorithm designs will need to be adapted for these domains. We plan to do this in high-fidelity simulation to begin with, and then extend them to real robots.

Literature Cited

[1] S. MartíNez and F. Bullo, "Optimal sensor placement and motion coordination for target tracking," *Automatica*, vol. 42, no. 4, pp. 661–668, 2006.

[2] R. Ghez, *Diffusion phenomena: cases and studies.* Springer Science & Business Media, 2013.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer networks*, vol. 38, no. 4, pp. 393–422, 2002.

[4] S. MartíNez and F. Bullo, "Optimal sensor placement and motion coordination for target tracking," *Automatica*, vol. 42, no. 4, pp. 661–668, 2006.

[5] M. A. Demetriou, "Guidance of mobile actuator-plus-sensor networks for improved control and estimation of distributed parameter systems," *IEEE Transactions on Automatic Control*, vol. 55, no. 7, pp. 1570–1584, 2010.

[6] M. A. Demetriou, N. A. Gatsonis, and J. R. Court, "Coupled controls-computational fluids approach for the estimation of the concentration from a moving gaseous source in a 2-D domain with a Lyapunov-guided sensing aerial vehicle," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 3, pp. 853–867, 2013.

[7] J. You and W. Wu, "Sensing-motion co-planning for reconstructing a spatially distributed field using a mobile sensor network," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 3113–3118, IEEE, 2017.

[8] You, Jie and Wu, Wencen, "Geometric Reinforcement Learning Based Path Planning for Mobile Sensor Networks in Advection-Diffusion Field Reconstruction," in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 1949–1954, IEEE, 2018.

[9] You, Jie and Zhang, Fumin and Wu, Wencen, "Cooperative filtering for parameter identification of diffusion processes," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 4327–4333, IEEE, 2016.

[10] W. Wu, J. You, Y. Zhang, M. Li, and K. Su, "Parameter Identification of Spatial–Temporal Varying Processes by a Multi-Robot System in Realistic Diffusion Fields," *Robotica*, p. 1–20, 2020.

[11] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[12] M. Krstic and A. Smyshlyaev, "Adaptive control of PDEs," *Annual Reviews in Control*, vol. 32, no. 2, pp. 149–160, 2008.

[13] L. A. Rossi, B. Krishnamachari, and C.-C. Kuo, "Distributed parameter estimation for monitoring diffusion phenomena using physical models," in *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004.*, pp. 460–469, IEEE, 2004.

[14] M. Krstic and A. Smyshlyaev, "Adaptive boundary control for unstable parabolic PDEs—Part I: Lyapunov design," *IEEE Transactions on Automatic Control*, vol. 53, no. 7, pp. 1575–1591, 2008.

[15] Z. Tang and U. Ozguner, "Motion planning for multitarget surveillance with mobile sensor agents," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 898–908, 2005.

[16] W. Burgard, M. Moors, C. Stachniss, and F. E. Schneider, "Coordinated multi-robot exploration," *IEEE Transactions on robotics*, vol. 21, no. 3, pp. 376–386, 2005.

[17] D. Ucinski, *Optimal measurement methods for distributed parameter system identification*. CRC press, 2004.

[18] D. Ucinski and Y. Chen, "Time-optimal path planning of moving sensors for parameter estimation of distributed systems," in *Proceedings of the 44th IEEE Conference on Decision and Control*, pp. 5257–5262, IEEE, 2005.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," 2013.

[20] B. Wang, Z. Liu, Q. Li, and A. Prorok, "Mobile Robot Path Planning in Dynamic Environments through Globally Guided Reinforcement Learning," 2020.

[21] X. Lei, Z. Zhang, and P. Dong, "Dynamic path planning of unknown environment based on deep reinforcement learning," *Journal of Robotics*, vol. 2018, 2018.

[22] A. S. Huang, E. Olson, and D. C. Moore, “LCM: Lightweight Communications and Marshalling,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4057–4062, 2010.

[23] “ZeroMQ.” https://zeromq.org/. (Accessed on 10/27/2020).

[24] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, E. Agafonov, T. H. Kim, E. Sterner, K. Ushiroda, M. Reyes, D. Zelenkovsky, and S. Kim, “LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving,” 2020.

[25] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “Carla: An open urban driving simulator,” 2017.

[26] “Deepdrive from Voyage - Push the state-of-the-art in self-driving.” https://deepdrive.voyage.auto/. (Accessed on 10/27/2020).

[27] “Webots: robot simulator.” https://cyberbotics.com/. (Accessed on 10/27/2020).

[28] N. Koenig and A. Howard, “Design and use paradigms for Gazebo, an open-source multi-robot simulator,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, pp. 2149–2154 vol.3, 2004.

[29] “Robot simulator CoppeliaSim: create, compose, simulate, any robot.” https://www.coppeliarobotics.com/index.html. (Accessed on 10/27/2020).

[30] “Autonomous Vehicle Simulation with Applied Intuition — From Sensor Simulation to Perception and Control Testing.” https://www.appliedintuition.com/. (Accessed on 10/27/2020).

[31] “Driving Simulation for autonomous driving, ADAS, vehicle dynamics and motorsport.” http://www.rfpro.com/. (Accessed on 10/27/2020).

[32] “Cognata — Autonomous and ADAS Vehicles Simulation Software.” https://www.cognata.com/. (Accessed on 10/27/2020).

[33] S. S. Srinivasa, P. Lancaster, J. Michalove, M. Schmittle, C. Summers, M. Rockett, J. R. Smith, S. Chouhury, C. Mavrogiannis, and F. Sadeghi, “MuSHR: A low-cost,

open-source robotic racecar for education and research," *CoRR*, vol. abs/1908.08031, 2019.

[34] J. M. Soares, I. Navarro, and A. Martinoli, "The Khepera IV Mobile Robot: Performance Evaluation, Sensory Data and Software Toolbox," in *Robot 2015: Second Iberian Robotics Conference* (L. P. Reis, A. P. Moreira, P. U. Lima, L. Montano, and V. Muñoz-Martinez, eds.), (Cham), pp. 767–781, Springer International Publishing, 2016.

[35] "TurtleBot." https://www.turtlebot.com/. (Accessed on 10/30/2020).

[36] "AI Robot Kits from NVIDIA JetBot Partners — NVIDIA." https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetbot-ai-robot-kit/. (Accessed on 09/24/2020).

[37] Zhang, Fumin and Leonard, Naomi Ehrich, "Cooperative filters and control for cooperative exploration," *IEEE Transactions on Automatic Control*, vol. 55, no. 3, pp. 650–663, 2010.

[38] W. Ren and R. W. Beard, *Distributed consensus in multi-vehicle cooperative control*, vol. 27. Springer, 2008.

[39] M. Lapan, *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing Ltd, 2018.

[40] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

[41] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," 2016.

[42] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[43] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[44] “NVIDIA Jetson Nano Developer Kit — NVIDIA Developer.” https://developer.nvidia.com/embedded/jetson-nano-developer-kit. (Accessed on 10/26/2020).

[45] “Ubuntu 18.04.5 LTS (Bionic Beaver).” https://releases.ubuntu.com/18.04.5/. (Accessed on 09/24/2020).

[46] “YDLIDAR-G4-Datasheet.” http://www.ydlidar.com/Public/upload/files/2020-04-13/YDLIDAR%20G4%20Datasheet.pdf. (Accessed on 10/26/2020).

[47] “sensor_msgs/LaserScan Documentation.” http://docs.ros.org/melodic/api/sensor_msgs/html/msg/LaserScan.html. (Accessed on 09/24/2020).

[48] “sensor_msgs/PointCloud2 Documentation.” http://docs.ros.org/melodic/api/sensor_msgs/html/msg/PointCloud2.html. (Accessed on 09/24/2020).

[49] “Wheel Encoder Kit - ROB-12629 - SparkFun Electronics.” https://www.sparkfun.com/products/12629. (Accessed on 10/26/2020).

[50] “Artificial Intelligence Computing Leadership from NVIDIA.” https://www.nvidia.com/en-us/. (Accessed on 10/30/2020).

[51] H. R. Kam, S.-H. Lee, T. Park, and C.-H. Kim, “RViz: A Toolkit for Real Domain Data Visualization,” *Telecommun. Syst.*, vol. 60, p. 337–345, Oct. 2015.

[52] P. Fankhauser and M. Hutter, “A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation,” in *Robot Operating System (ROS) – The Complete Reference (Volume 1)* (A. Koubaa, ed.), ch. 5, Springer, 2016.

[53] G. Guennebaud, B. Jacob, *et al.*, “Eigen v3.” http://eigen.tuxfamily.org, 2010.

[54] “tf2 - ROS Wiki.” http://wiki.ros.org/tf2. (Accessed on 09/25/2020).

[55] “rqt_tf_tree - ROS Wiki.” http://wiki.ros.org/rqt_tf_tree. (Accessed on 09/24/2020).

[56] P. J. Besl and N. D. McKay, “A method for registration of 3-D shapes,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[57] “Point Cloud Library (PCL): pcl::IterativeClosestPoint Class Template Reference.” https://pointclouds.org/documentation/classpcl_1_1_iterative_closest_point.html. (Accessed on 09/30/2020).

[58] R. B. Rusu and S. Cousins, “3D is here: Point Cloud Library (PCL),” in *IEEE International Conference on Robotics and Automation (ICRA)*, (Shanghai, China), May 9-13 2011.

[59] “geometry_msgs/Twist Documentation.” https://docs.ros.org/en/api/geometry_msgs/html/msg/Twist.html. (Accessed on 10/27/2020).

[60] “move_base - ROS Wiki.” http://wiki.ros.org/move_base. (Accessed on 10/27/2020).

[61] “geometry_msgs/PoseStamped Documentation.” http://docs.ros.org/en/api/geometry_msgs/html/msg/PoseStamped.html. (Accessed on 10/27/2020).

[62] “ROS 2 Overview.” https://index.ros.org/doc/ros2/. (Accessed on 10/26/2020).