TOWARDS MICROROBOT SWARM PATH PLANNING

A Dissertation

Presented to

the Faculty of the Department of Electrical and Computer Engineering

University of Houston

in Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

in Electrical Engineering

by

Li Huang

August 2019

TOWARDS MICROROBOT SWARM PATH PLANNING

Li Huang

Approved:

Chair of the Committee
Aaron T. Becker, Assistant Professor
Department of Electrical and Computer Engineering

Committee Members:

Frank J. 'Fritz' Claydon, Professor
Department of Electrical and Computer Engineering

David Mayerich, Assistant Professor
Department of Electrical and Computer Engineering

Hien Nguyen Van, Assistant Professor
Department of Electrical and Computer Engineering

Peggy Lindner, Research Assistant Professor
Department of Mechanical Engineering

Suresh K. Khator, Associate Dean,
Cullen College of Engineering

Badrinath Roysam, Professor and Chair,
Electrical and Computer Engineering

Acknowledgements

I am profoundly grateful and fortunate to have Dr. Aaron T. Becker as my advisor. During my doctoral study, Dr. Becker has shared with me his great passion for robotics and life, and has offered me valuable insights and inspirations. All of the work presented in this thesis could not have been possible without his support, advice, and encouragement.

I would like to show great appreciation to Dr. Frank J. 'Fritz' Claydon, Dr. David Mayerich, Dr. Hien Nguyen Van, and Dr. Peggy Lindner for being my defense committee members and offering insightful suggestions for my research. I would like to thank our major collaborators Dr. Min Jun Kim, Luis Rogowski, Xiao Zhang, Jung Soo Lee, Sam Sheckman, and Hoyeon Kim, who have closely worked with me in microrobot hardware experiments, and have provided exciting demonstrations in microrobot fabrication and manipulation. Thanks to other co-authors of my work, Dr. Sandor P. Fekete, Dr. Jason M. O'Kane, Dr. Dylan A. Shell, Dr. Nikolaos V. Tsekos, Dr. Robert Stewart, Dr. Julien Leclerc, Arne Schmidt, Sheryl Manzoor, Phillip Keldenich, Dominik Krupke, Li Chang, Srikanth Sudarshan, and Xin Liu. I am thankful to the current colleagues and alumni of the Robotic Swarm Control Lab, Nick Anderson, Daniel Bao, Daniel Biediger, Shriya Bhatnagar, Javier Garcia, Rhema Ike, Benedict Isichei, Parth Joshi, Julien Leclerc, Lillian Lin, Jarrett Lonsford, Arun Mahadev, Sheryl Manzoor, Victor Montano, An Nguyen, Adi Pasic, Shiva Shahrokhi, Steban Soto, Mohammad Sultan, Mable Wan, Mike Yannuzzi, and Haoran Zhao. I am thankful to Daniel J. Block for introducing me to Dr. Becker.

Finally, this thesis is dedicated to my parents Yan Li and Guangwen Huang, and my fiancee Xuan Zhu, for having so much love, care, support for me, and always being there when I needed you the most.

TOWARDS MICROROBOT SWARM PATH PLANNING

An Abstract

of a

Dissertation

Presented to

the Faculty of the Department of Electrical and Computer Engineering

University of Houston

in Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

in Electrical Engineering

by

Li Huang

August 2019

Abstract

Tiny robots have many promising applications in medical treatment including targeted drug delivery, non-invasive diagnosis, and minimally invasive surgery; and in micro-assembly/-fabrication for Micro-Electro-Mechanical-Systems (MEMS). Microrobots are often deployed in large populations, and typically steered by uniform driving signals, including magnetic, electromagnetic, electrostatic, optical, gravitational, thermal, and chemical. The homogeneity of the microrobots and the uniformity of the control input make microrobot swarm manipulation difficult in constrained workspaces such as human vascular networks. The control laws and path-planning algorithms designed for macro-size robotics do not scale well to a microrobot swarm, so new methodology must be developed to address more efficient planning with constraints for a multi-agent problem in microscale.

This thesis addresses the path-planning problem of a swarm of microrobots using a global control input. It begins with an introduction to state-of-the-art research and applications in microrobots. Chapter 2 gives an analysis of 2D and 3D position control of heterogeneous microrobots in the free space, together with demonstrations in simulations and hardware experiments. Motivated by the need for higher computational efficiency and capability of swarm manipulation with spatial constraints, chapter 3 discusses strategies of planning in 2D vascular networks for a swarm of homogeneous microrobots given a shared, global control input. Multiple path-planning methods and control algorithms are proposed, and their performance is compared in multiple vascular networks with different scale and complexity. The algorithms are validated with simulations and hardware experiments. Chapter 4 investigates reinforcement learning strategies to further improve path-planning efficiency, and to overcome local minima dilemmas in online algorithms. Chapter 5 reports automatic steering methods in multi-bifurcation vessels with flow, and reinforcement learning algorithms are implemented for improvement in microrobot delivery rate.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Microrobots and Potential Applications

Micro- and nanorobots have great potential impacts in the fields of biology, medical diagnosis and treatment, and industrial manufacturing. Especially for biological and medical tasks, micro- and nanorobots are promising for various applications such as cell manipulation, bio-sensing, targeted drug delivery, blood clot removal, minimally invasive or non-invasive surgical interventions, etc., because of their small size, flexibility, large population, and the capability of carrying drug loads. For example, magnetic resonance imaging (MRI) guided *in vivo* navigation of a microrobot has been reported by Martel et al. [1] in 2007, where microrobot tracking, control, and actuation have been achieved in the carotid artery within a living swine. Such techniques are envisioned to deploy micro- or nanorobots within the human body to reach remote regions and perform interventions that conventional treatment cannot. Another example is drug delivery.

Current chemotherapy procedures inject boluses of drug from a catheter—the medical device employed to treat disease or perform surgery—and the drug indiscriminatingly circulates the human body without control, which is why it kills healthy and tumor cells alike. To reduce toxic drug exposure to healthy cells, targeted drug delivery seeks to steer chemotherapy directly to diseased tissue. Magnetic micro- and nano-carriers with drugs can play a key role in such work, navigating through blood vessels by external magnetic fields and directly accessing the target tumor area. This dissertation is focused on the path-planning and control algorithms for targeted drug delivery with microrobot swarms.

Unlike macro-scale robots, due to space constraints microrobots usually have limited

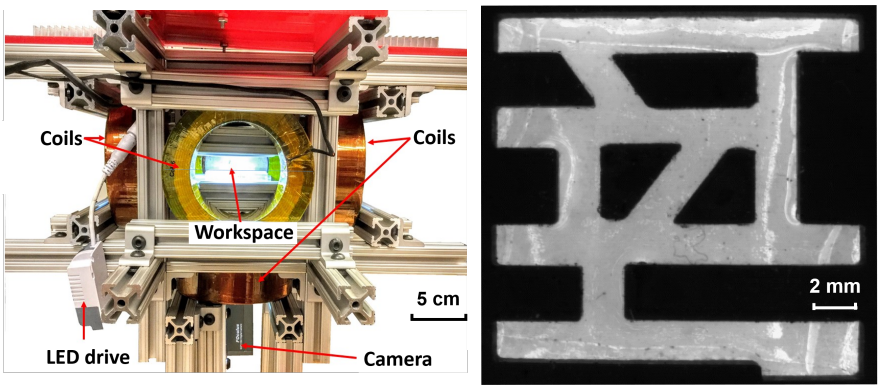
capabilities for onboard computation, communication, sensing, and actuation. Typically an artificial microrobot might just be a magnetic bead, a chain-like structure of magnetic particles, or a bulk of micro materials, such as the Mag-Mite magnetic microrobots [2], the therapeutic magnetic microcarriers (TMMC) [3], the magnetic thin-film microrobots [4], the paramagnetic Janus particles [5], the Mag-μBot [6, 7], the paramagnetic microparticles [8], the magnetic microtransporters [9], flagellated nanoparticles [10], achiral microswimmers [11], etc. Other microrobot forms include self-propelled cells or bacteria, for example, Magnetotactic Bacteria (MTB) [12–14] and *Tetrahymena pyriformis* cells [15, 16]. Due to the limitation of the sensing and actuating, external sensors (e.g., MRI, cameras) and actuators (e.g., electric/magnetic fields or thermal/chemical/optimal/acoustic source) must be employed to localize and power the microrobots.

1.2 Motivations and Objectives

In many aforementioned possible applications, microrobots are manipulated using a global control input in highly constrained environments such as the circulatory system, the central nervous system, the eye, the ear, etc. [17].

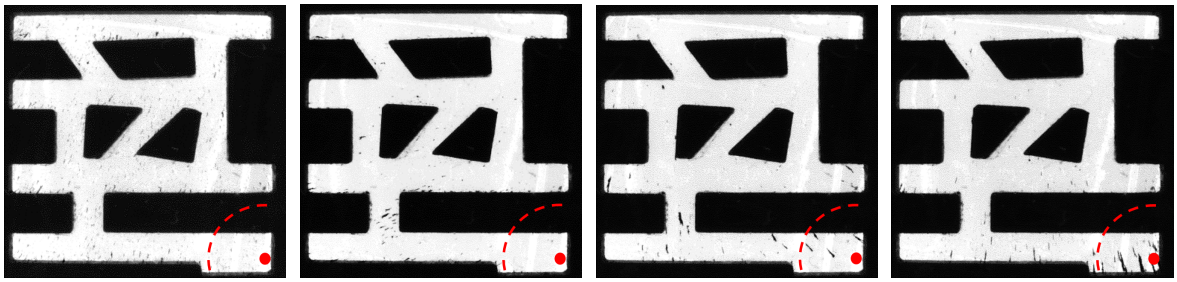
Many strategies and algorithms have been developed for navigation and motion control of microrobots. Path planning and closed-loop control were proposed in free space [18–20]. Khalil et al. demonstrated the control of a single microrobot in a micro-fabricated maze [21]. Scheggi et al. implemented and compared six path-planning algorithms using magnetic microrobots [22]. However, it is not clear if the single-robot control strategy can be scaled to microrobot swarm scenarios, or whether the feedback control solution in free space can be well adapted to vascular networks.

Pierson et al. proposed a control strategy that by introducing herders to drive a swarm of herding animals to the desired location with repelling potential fields [23]. Fine et al. reported how to actively design environments to assist the process of controlling multiple agents using



(a) (b)

Figure 1.1: (a) The six-coil electromagnetic system with a bottom-view camera. (b) A vascular network tested in experiments.



(a) (b) (c) (d)

Figure 1.2: (a)-(d) Captured frames from a experiment. The goal location is marked with a red point. (a) $t = 0$ min, (b) $t = 4$ min, (c) $t = 19$ min, (d) $t = 36$ min.

shape grammars [24]. This method addresses the automatic generation of environments given specific swarm objective and a control model of agents. Becker et al. showed particle computation methods to perform permutations between different swarm formations by designing unit-size obstacles in a grid workspace, where they used mobile particles with maximal motion (particles moved until they hit an obstacle or an obstructed particle) and a global input [25]. Bobadilla et al. gave another example of exploiting environment, where a state space is partitioned into discrete transition systems, and gates are configured to guide a swarm of simple robots to achieve state transition, and thereby to accomplish high-level tasks [26]. Mahadev et al. explored microrobot swarm aggregation in a planar grid environment, where microrobots are of different sizes, capable of overlapping, moving in discrete steps and directed by a share,

(a) (b) (c) (d)

Figure 1.3: (a)—(d) Vascular network examples in simulations.

global, control input [27].

The objectives of this thesis include the following three aspects: (i) exploring closed-loop control laws for manipulating a microrobot swarm in free space; (ii) proposing efficient strategies for real-time path-planning and control to steer the swarm in highly constrained environments, and to validate the results in both simulations and experiments; and (iii) utilizing learning-based algorithms to optimize the efficiency of path-planning and control policies, and thus indicating the direction to improve the real-time planning process.

1.3 Dissertation Organization

The dissertation is arranged as follows. Chapter 2 gives an analysis of 2D and 3D position control of a swarm of heterogeneous microrobots in the free space. Chapter 3 discusses strategies of online planning in 2D vascular networks for a swarm of homogeneous microrobots given a shared, global control input. Chapter 4 investigates reinforcement learning strategies to further improve path-planning efficiency, and to overcome local minima dilemmas in online algorithms. Chapter 5 reports automatic steering methods in multi-bifurcation vessels with flow, and reinforcement learning algorithms are implemented for improvement in microrobot delivery rate.

Chapter 2

Steering a Heterogeneous Microrobot Swarm by a Shared, Global Control Input

2.1 Introduction

Interest in swarm robotics in the control and robotics communities has increased. Compared to highly intelligent and advanced robots, each agent in a swarm robot system is inexpensive, easy to manufacture, and suitable to deploy in large populations [28]. At macro scales, swarm robots such as micro aerial vehicles and 2D autonomous ground vehicles have great potential to be applied to sensing, mapping, localization, surveillance, rescue, etc. At micro scales, agents such as ferromagnetic microrobots, magnetotactic bacteria, and catalytic Janus particles are researched for target drug delivery, non-invasive surgery, micro assembly, etc.

This chapter considers a swarm of simple robots with limited communication capability such that agents are commanded by a central system, and agent-to-agent information exchange is not applicable. The swarm system might consist of hundreds or thousands of agents, but each agent receives a copy of the same control input. For example, one potential application is steering catalytic Janus particles with uniform magnetic fields. These particles are self-propelled by a reaction between platinum on the particle and the liquid the particle swims in. The particles are also magnetic. The external magnetic field applies a torque that aligns the magnetic dipole of the particle with the external field. The results in [5] demonstrate steering an individual particle to a desired location and demonstrate that multiple particles could be made to move in different directions when under the control of a single magnetic field.

Consider a swarm with n agents in free space, with their headings randomly initialized. The agents are modeled as self-propelled points that move at a constant velocity along a thrust

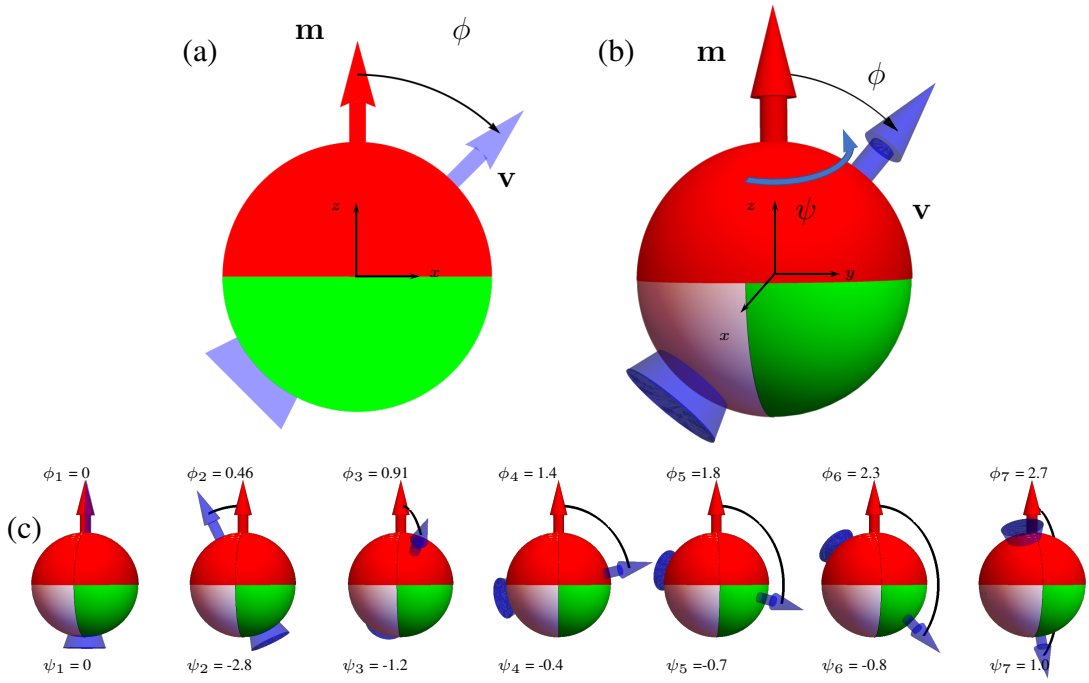


Figure 2.1: Schematic of self-propelled agent in 2D (a) and in 3D (b). 2D thrust vector (blue) is defined by the offset angle ϕ from the local coordinate frame, while 3D motion is offset by ϕ and ψ. (c) Seven agents with different thrust vector orientations.

vector that is fixed in their local coordinate frame. A central system controls agents via a broadcast mechanism. The only control inputs allowed are rotation commands that rotate each agent's local coordinate frame by the same rotation matrix.

This model is similar to ensemble control systems [29–39]. In these problems an ensemble of nearly identical agents that differ only in a set of one or more parameters, are each steered by the same control input. However, the major challenge in this chapter is the constrained control input in Equation (2.30), where $R(t)$ needs to be a rotation matrix.

This chapter addresses the 3D position control problem of a multi-agent system using a shared rotation control input, including (i) with no state perturbations, simultaneously steering up to three agents to arbitrary locations; (ii) with independent rotation perturbations, enabling simultaneously steering many agents ($n > 3$) to arbitrary locations. While boundary interactions could be another method to steer large populations of agents [40], this chapter focuses on simple agents in free-space that are affected only by their thrust, local coordinate frames, and

the shared control command.

2.2 Related Work

Previous works have addressed control strategies for 3D multi-agent navigation and planning, for example, Roussos et al. discusses a distributed control scheme based on Navigation Functions to drive aircraft-like vehicles towards their targets while avoiding colliding into each other or obstacles [41]; Xu et al. proposes an acyclic minimally structural persistent graph based formation control to steer a group of autonomous agents in 3D space [42]; and Nikou et al. developed a decentralized 3D formation control algorithm for a multi-agent system with unknown dynamics [43]. However, these control schemes rely on independent control of each agent, which is not applicable within the scope of this chapter.

The key results of the this chapter include techniques to make n agents follow orbits (section 2.4.2), steering a swarm of agents to arbitrary x, y, z positions, and reducing the variance of the swarm (sections 2.4.4 to 2.4.6). Simulation code is available at GitHub [44] and Wolfram Demonstration Projects [45], and see the video at See video at https://youtu.be/sSSQgnmjmJw [46].

2.3 2D Control of Self-Propelled Agents

This section analyzes a 2D version of the broadcast control problem. Figure 2.1 shows a schematic of a self-propelled point robot in 2D (a disk) and 3D (a sphere).

Bretl [47] and Das et al. [48] have discussed the shared control problem in a 2D plane. Bretl proposes a control law that directs two agents to meet at the same location simultaneously, and with input perturbations. Das demonstrates the possibility of achieving position consensus for large number of agents.

Considering n agents in a 2D xy plane, the origin of each local coordinate frame coin-

cides with the individual center of mass, and let the local coordinate frame be initially aligned with the global coordinate frame. The kinematics of the i^{th} agent is given by

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + R_\theta \mathbf{v}_i(t), \tag{2.1}$$

with $\mathbf{x}_i(t) \in \mathbb{R}^{2\times 1}$ the position at time t, $\mathbf{v}_i(t) \in \mathbb{R}^{2\times 1}$ the thrust vector, and R_θ the shared control command that rotates the agent along its local z-axis,

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{2.2}$$

Given two self-propelled agents with any initial positions $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^{2\times 1}$, and thrust vectors $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^{2\times 1}$ ($\mathbf{v}_1 \nparallel \mathbf{v}_2$), they can meet at only one unique location

$$\hat{\mathbf{x}}_{1,2} = \mathbf{x}_2 + \frac{1}{2}\begin{bmatrix} 1 & \tan\frac{2}{\phi_2-\phi_1} \\ \tan\frac{-2}{\phi_2-\phi_1} & 1 \end{bmatrix}(\mathbf{x}_1 - \mathbf{x}_2), \tag{2.3}$$

where $\phi_i = \arctan(\mathbf{v}_i)$. For $n > 2$ such agents in a 2D workspace, there are $n(n-1)/2$ potential collision locations, as shown in Figure 2.2. The potential collision locations are drawn as gray disks determined by the initial agent location and orientation. The radius of the collision disk is a function of agent radius and difference in orientation (online demonstration available at [45]).

In general, a sequence of commands for shared control cannot make large number of agents ($n > 2$) to meet simultaneously except for some special initial conditions. If thrust vector perturbations are permitted, Das shows the possibility of point convergence for all agents in [48].

2.4 3D Control of Self-Propelled Agents

The effect of any command sequence for 2D position control (without perturbations) of self-propelled agents can be replicated by three commands: an initial rotation along the local

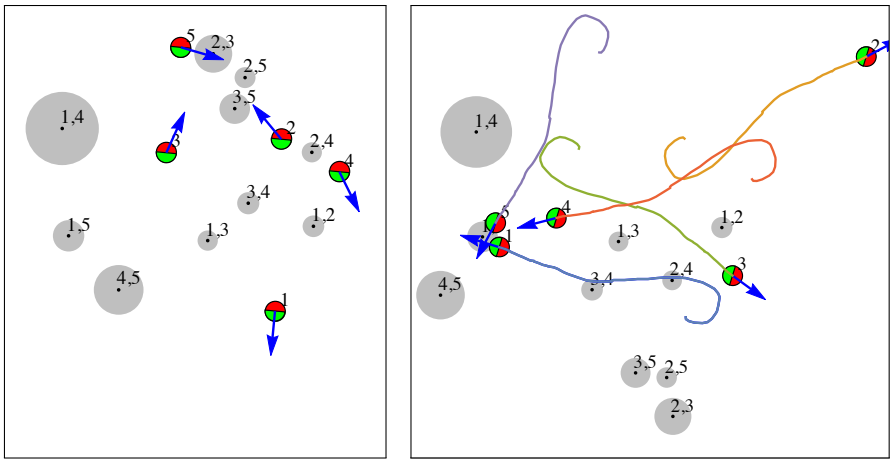


Figure 2.2: The configuration space for self-propelled agents in 2D is in $\mathbb{R}^2$. Right panel shows a set of control inputs that brings agents 1 and 5 into collision.

z-axis, a translation, and a final rotation along the local z-axis. Therefore, the configuration space for these agents' position is two-dimensional, no matter how many agents are used. This is because the ending position of each agent is the result of the same rigid body transformation modulo an initial rotation. Interestingly, in 3D-space multiple rotation and translation operations can be concatenated to control more than three degrees of freedom.

This work reports on simultaneous 3D position control for multiple self-propelled agents. Without loss of any generality, each agent is initialized with different thrust vectors. The origin of each agent's coordinate frame is at the agent's position, and the x, y, and z axes are aligned with the global coordinate frame. The agents implement shared rotation commands based on their local coordinate frames. For simplification, the actuation time for rotations is negligible compared to the time of translation.

2.4.1 Steer One Self-Propelled Agent

To deliver a single self-propelled agent from its initial position $\mathbf{x} \in \mathbb{R}^{3\times1}$ to a goal location $\hat{\mathbf{x}}$, the agent must be rotated so that the thrust vector points toward $\hat{\mathbf{x}}$, Let $\mathbf{v}$ be the initial thrust vector defined in the global coordinate frame. The desired thrust vector can be

described as

$$\hat{\mathbf{v}} = \frac{\hat{\mathbf{x}} - \mathbf{x}}{\|\hat{\mathbf{x}} - \mathbf{x}\|_2}. \tag{2.4}$$

First, identify the normal vector $\mathbf{k}$ of a plane containing both $\mathbf{v}$ and $\hat{\mathbf{v}}$,

$$\mathbf{k} = \mathbf{v} \times \hat{\mathbf{v}}. \tag{2.5}$$

Next, rotate θ about $\mathbf{k}$ to align the thrust vector with $\hat{\mathbf{v}}$, where

$$\theta = \arccos(\mathbf{v}, \hat{\mathbf{v}}). \tag{2.6}$$

2.4.2 Station-Keeping (Orbits) with Multiple Agents

The following sections provide control laws for steering 3D agents to goal positions. A preliminary challenge is to *keep* multiple agents at given locations. The solution in 2D is to revolve about the local z-axis at a constant rate, where the z-axis is perpendicular to the motion plane. The orbital radius is the thrust velocity divided by the angular frequency $r = |\mathbf{v}|/\omega$. The faster it revolves, the tighter the orbit. In three dimensions this technique no longer works.

The position change of the ith agent under a shared command that rotates $\theta(t)$ radians about an axis k is given by $\int_0^t R_{k,\theta(\tau)}\mathbf{v}_i \, d\tau$. It is easy to show that a rotation about the x-axis does not, in general, return all agents to the initial location. As shown in Figure 2.3, rotating θ radians at 1 radian per second about the local x-axis moves the agents to

$$[\theta v_x, v_z(\cos\theta - 1) + v_y \sin\theta, v_y(1 - \cos\theta) + v_z \sin\theta]^\top.$$

After one revolution, each agent has been displaced by $[2\pi v_x, 0, 0]$, and any agent with a non-zero v_x has followed a helical trajectory. Self-propelled agents with positive v_x will have moved in the positive x direction, and the others in the negative x.

A solution that returns all self-propelled agents to their initial positions is given by eight revolutions that toggle between revolving about the local x and y-axes. All revolutions proceed at a constant angular velocity and, as in 2D, the maximum deviation scales linearly with the

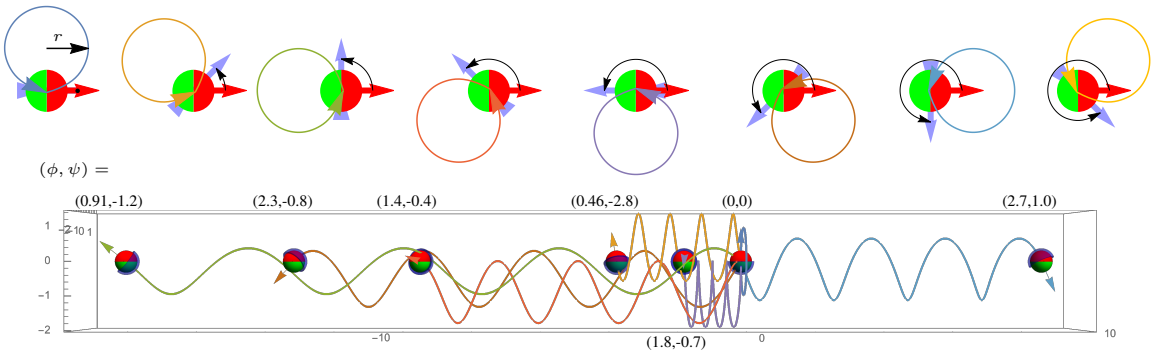


Figure 2.3: (Top) In 2D, revolving about the local z-axis results in circular orbits for self-propelled agents. This is not generally true in 3D (Bottom), and revolving four times around the local x-axis results in deviation from their initial locations.

inverse of the angular velocity,

$$R_{x,\pi}R_{y,-\pi}R_{x,-\pi}R_{y,\pi}R_{x,-\pi}R_{y,\pi}R_{x,\pi}R_{y,-\pi} = I_3. \tag{2.7}$$

All x, y, z subscripts for any rotation R refer to the current local x, y, z-axes in the following sections. The positions for an agent initially at the origin after each rotation are

$$\dot{\theta}^{-1}\left(\begin{bmatrix} \pi v_x \\ -2v_z \\ 2v_y \end{bmatrix}, \begin{bmatrix} \pi v_x + 2v_z \\ -\pi v_y - 2v_z \\ 2v_x + 2v_y \end{bmatrix}, \begin{bmatrix} 2v_z \\ -\pi v_y \\ 2v_x + 4v_y \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 4v_x + 4v_y \end{bmatrix}, \right.$$
$$\left. \begin{bmatrix} \pi v_x \\ 2v_z \\ 4v_x + 2v_y \end{bmatrix}, \begin{bmatrix} \pi v_x - 2v_z \\ -\pi v_y + 2v_z \\ 2v_x + 2v_y \end{bmatrix}, \begin{bmatrix} -2v_z \\ -\pi v_y \\ 2v_x \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right). \tag{2.8}$$

Trajectories of this input sequence are shown in Figure 2.4.

2.4.3 Simultaneous Position Control

For a system of $n \geq 2$ arbitrary self-propelled agents in a 3D free-space, there are $6n$ DOF: the position vectors $\mathbf{x}_i \in \mathbb{R}^{3\times 1}$ and the thrust vectors $\mathbf{v}_i \in \mathbb{R}^{3\times 1}$. This work provides both open-loop and closed-loop algorithms to control up to nine DOF in positions with no state perturbations.

For ease of exposition, the shared controller only uses R_{x,θ_x} and R_{y,θ_y} as rotation primitives to control the thrust vector heading, because any 3D rotations $R_{k,\theta}$ about a local axis k

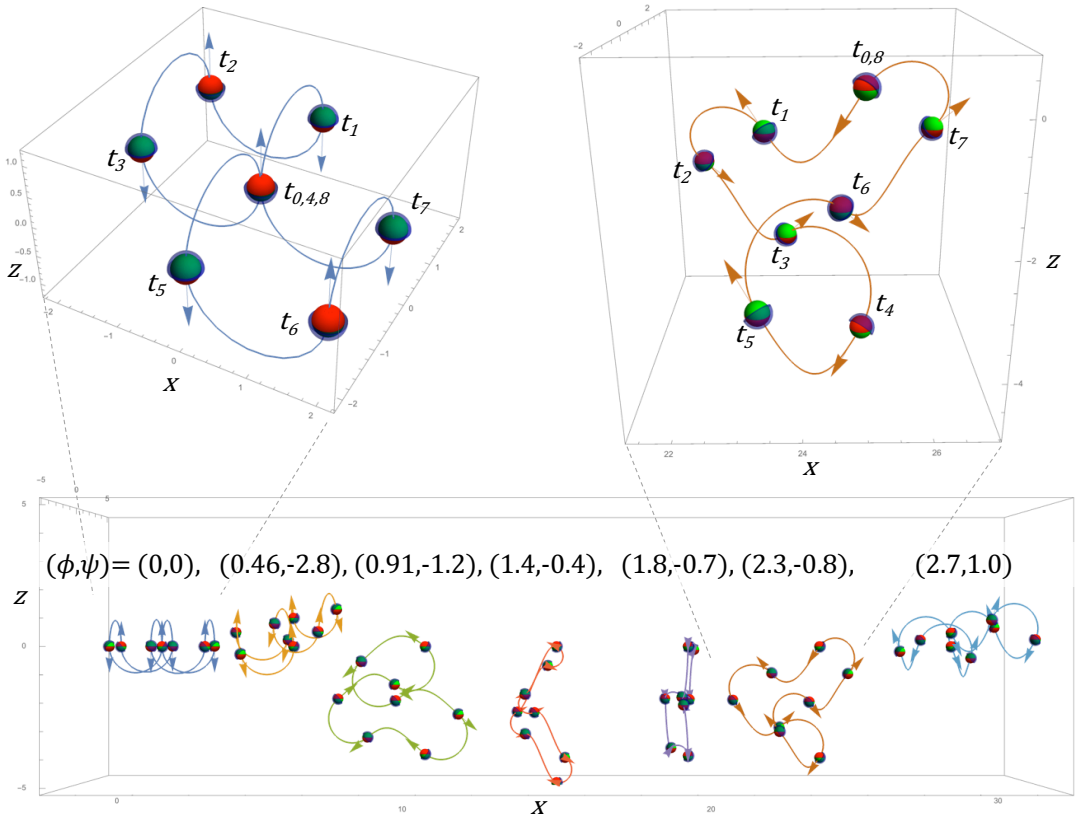


Figure 2.4: Periodic orbits of the seven agents shown in Figure 2.1c, under the open-loop input (2.7). The agents and current thrust arrows are redrawn at $t_i = k\pi$.

can be represented by a series of rotations about the current local x, y, and x axis,

$$R_{k,\theta} = R_{x,\theta_1} R_{y,\theta_2} R_{x,\theta_3}. \tag{2.9}$$

If the shared command steers agents to rotate θ_x about the current local x-axis, the corresponding rotation matrix and resultant thrust orientations in the global coordinate frame are

$$R_{x,\theta_x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & \sin\theta_x \\ 0 & -\sin\theta_x & \cos\theta_x \end{bmatrix} \text{ and} \tag{2.10}$$

$$\hat{\mathbf{v}}_i = R_{x,\theta_x} \mathbf{v}_i, \tag{2.11}$$

where the subscripts x, θ_x in R_{x,θ_x} denote the local x-axis and rotation angle. If the next command is to rotates θ_y about the current local y-axis, the corresponding rotation matrix and

thrust orientations in the global coordinate frame are

$$R_{y,\theta_y} = \begin{bmatrix} \cos\theta_y & 0 & -\sin\theta_y \\ 0 & 1 & 0 \\ \sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \text{ and} \tag{2.12}$$

$$\hat{\mathbf{v}}_i = R_{x,\theta_x} R_{y,\theta_y} \mathbf{v}_i. \tag{2.13}$$

These rotations are performed about the current local x or y-axis instead of the fixed global coordinate frame, and both Equation (2.11) and (2.13) give a thrust vector $\hat{\mathbf{v}}_i$ defined in the global frame.

Each rotation is followed by a translation motion. To simplify the derivation, the time required for any rotation is assumed to be negligible compared to the translation actuation time.

In 3D space multiple self-propelled agents cannot be driven to arbitrary goal locations $\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_n \in \mathbb{R}^{3\times 1}$ with one rotation and translation. However, concatenating the rotation operations (2.10) and (2.12) followed by translations enables controlling additional DOF.

This can be written in matrix form as

$$\begin{bmatrix} \Delta\mathbf{x_1} \\ \vdots \\ \Delta\mathbf{x_n} \end{bmatrix} = \begin{bmatrix} R_1\mathbf{v}_1 & R_2\mathbf{v}_1 & \cdots & R_N\mathbf{v}_1 \\ R_1\mathbf{v}_2 & R_2\mathbf{v}_2 & \cdots & R_N\mathbf{v}_2 \\ \vdots & \vdots & \vdots & \vdots \\ R_1\mathbf{v}_n & R_2\mathbf{v}_n & \cdots & R_N\mathbf{v}_n \end{bmatrix} \mathbf{t} = R_v\mathbf{t}, \tag{2.14}$$

with $\Delta\mathbf{x}_i = \hat{\mathbf{x}}_i - \mathbf{x}_i$, $R_j = R_{x,\theta_1} R_{y,\theta_2} \cdots R_{x,\theta_j}$, $R_{j+1} = R_j R_{y,\theta_{j+1}}$, and $\mathbf{t} = [t_1, t_2, \cdots, t_N]^\top$. If all R_j and $\mathbf{t}$ are unknown, solving Equation (2.14) directly is computationally intensive. Instead, randomly generating the N angles is computationally cheap and works well in simulation, though methods shown later can outperform this.

Given the N rotation angles, $\Delta\mathbf{x}_i$, and $\mathbf{v}_i$, where $i \in \{1, 2, \cdots, n\}$, the goal is to choose $\mathbf{t}$ for Equation (2.14) that minimizes $\|\mathbf{t}\|_1$ with $t_j \geq 0,\ \forall j \in [1, N]$.

Any three independent vectors in $\mathbb{R}^{3\times 1}$ forms a basis for the 3D space. Without loss of generality, assume that $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ in R_v (Equation (2.14)) are independent and all $\mathbf{v}_i$ are unique.

So any $\mathbf{v}_i$ of R_v can be expressed as a linear combination of $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$,

$$\mathbf{v}_i = \lambda_{i,1}\mathbf{v}_1 + \lambda_{i,2}\mathbf{v}_2 + \lambda_{i,3}\mathbf{v}_3, \tag{2.15}$$

where $\lambda_{i,1}, \lambda_{i,2}, \lambda_{i,3} \in \mathbb{R}$. Recall that row $3i-2$ to row $3i$ in R_v has the form

$$R_v(3i-2:3i,*) = [R_1\mathbf{v}_i \quad R_2\mathbf{v}_i \quad \cdots \quad R_N\mathbf{v}_i], \tag{2.16}$$

where $\mathbf{v}_i \in \mathbb{R}^{3\times 1}$. Substitute $\mathbf{v}_i$ with Equation (2.15),

$$\begin{array}{rcl} R_v(3i-2:3i,*) & = & \lambda_{i,1}[R_1\mathbf{v}_1 \quad R_2\mathbf{v}_1 \quad \cdots \quad R_N\mathbf{v}_1] \\ & + & \lambda_{i,2}[R_1\mathbf{v}_2 \quad R_2\mathbf{v}_2 \quad \cdots \quad R_N\mathbf{v}_2] \\ & + & \lambda_{i,3}[R_1\mathbf{v}_3 \quad R_2\mathbf{v}_3 \quad \cdots \quad R_N\mathbf{v}_3], \end{array}$$

which indicates that any row of R_v is a linear combination of its first nine rows. So the row rank of R_v is at most nine.

Recall that column j in R_v has the form

$$R_v(*,j) = \begin{bmatrix} R_j\mathbf{v}_1 \\ R_j\mathbf{v}_2 \\ \vdots \\ R_j\mathbf{v}_n \end{bmatrix}. \tag{2.17}$$

Assuming there are at most k independent columns of R_v, without loss of generality, let $R_v(*,1:k)$ be these columns. Hence, $R_v(*,j)$ can be expressed as

$$R_v(*,j) = l_{j,1}R_v(*,1) + l_{j,2}R_v(*,2) + \cdots + l_{j,k}R_v(*,k) \text{ and} \tag{2.18}$$

$$\begin{bmatrix} R_j\mathbf{v}_1 \\ R_j\mathbf{v}_2 \\ \vdots \\ R_j\mathbf{v}_n \end{bmatrix} = \begin{bmatrix} (l_{j,1}R_1 + l_{j,2}R_2 + \cdots + l_{j,k}R_k)\mathbf{v}_1 \\ (l_{j,1}R_1 + l_{j,2}R_2 + \cdots + l_{j,k}R_k)\mathbf{v}_2 \\ \vdots \\ (l_{j,1}R_1 + l_{j,2}R_2 + \cdots + l_{j,k}R_k)\mathbf{v}_n \end{bmatrix}. \tag{2.19}$$

That is, $\forall i \in \{1, 2, \ldots, n\}$,

$$(l_{j,1}R_1 + l_{j,2}R_2 + \cdots + l_{j,k}R_k - R_j)\mathbf{v}_i = 0_{3\times 1}. \tag{2.20}$$

Therefore

$$l_{j,1}R_1 + l_{j,2}R_2 + \cdots + l_{j,k}R_k = R_j. \tag{2.21}$$

Write R_k as

$$R_k = \begin{bmatrix} r_{k,11} & r_{k,12} & r_{k,13} \\ r_{k,21} & r_{k,22} & r_{k,23} \\ r_{k,31} & r_{k,32} & r_{k,33} \end{bmatrix}, \tag{2.22}$$

and let $\mathbf{r}_k = [r_{k,11}, r_{k,12}, r_{k,13}, \cdots, r_{k,31}, r_{k,32}, r_{k,33}]^\top$, $\mathbf{l}_j = [l_{j,1}, l_{j,2}, \cdots, l_{j,k}]^\top$. Then Equation (2.21) can be written as

$$[\ \mathbf{r}_1 \quad \mathbf{r}_2 \quad \cdots \quad \mathbf{r}_k\ ]\mathbf{l}_j = \mathbf{r}_j. \tag{2.23}$$

To have a unique solution to $\mathbf{l}_j$, $[\mathbf{r}_1, \mathbf{r}_2, \cdots, \mathbf{r}_k]$ needs to be invertible, that is, $k = 9$. It can be concluded that the column rank of R_v is at most nine. Because the maximum rank of R_v is nine, $n \leq 3$ self-propelled agents can be steered to arbitrary goal locations, and only if these agents have linearly independent initial thrust vectors.

Only nine of $3n$ DOF can be manipulated if $n > 3$. A special case is to make four agents to meet simultaneously. There is a unique position $\hat{\mathbf{x}}$ where four such agents can meet,

$$\begin{aligned} \hat{\mathbf{x}} &= \mathbf{x}_1 + \sum_{i=1}^{N} R_i \mathbf{v}_1 t_i = \mathbf{x}_2 + \sum_{i=1}^{N} R_i \mathbf{v}_2 t_i \\ &= \mathbf{x}_3 + \sum_{i=1}^{N} R_i \mathbf{v}_3 t_i = \mathbf{x}_4 + \sum_{i=1}^{N} R_i \mathbf{v}_4 t_i. \end{aligned} \tag{2.24}$$

Let $R_t = \sum_{i=1}^{N} R_i t_i$, so

$$\begin{aligned} &\mathbf{x}_1 - \mathbf{x}_2 = R_t(\mathbf{v}_2 - \mathbf{v}_1), \\ &\mathbf{x}_2 - \mathbf{x}_3 = R_t(\mathbf{v}_3 - \mathbf{v}_2), \text{ and} \\ &\mathbf{x}_3 - \mathbf{x}_4 = R_t(\mathbf{v}_4 - \mathbf{v}_3). \end{aligned} \tag{2.25}$$

Flatten R_t as a vector $\mathbf{r}_t$, and rewrite the right side of Equation (2.25) as

$$\begin{bmatrix} \mathbf{x}_1 - \mathbf{x}_2 \\ \mathbf{x}_2 - \mathbf{x}_3 \\ \mathbf{x}_3 - \mathbf{x}_4 \end{bmatrix} = \begin{bmatrix} R_\Lambda(\mathbf{v}_2, \mathbf{v}_1) \\ R_\Lambda(\mathbf{v}_3, \mathbf{v}_2) \\ R_\Lambda(\mathbf{v}_4, \mathbf{v}_3) \end{bmatrix} \mathbf{r}_t = R_{\Lambda\mathbf{v}}\mathbf{r}_t, \tag{2.26}$$

where

$$R_\Lambda(\mathbf{v}_i, \mathbf{v}_j) = \begin{bmatrix} \mathbf{v}_i^\top - \mathbf{v}_j^\top & 0_{1\times 3} & 0_{1\times 3} \\ 0_{1\times 3} & \mathbf{v}_i^\top - \mathbf{v}_j^\top & 0_{1\times 3} \\ 0_{1\times 3} & 0_{1\times 3} & \mathbf{v}_i^\top - \mathbf{v}_j^\top \end{bmatrix}. \tag{2.27}$$

So $\mathbf{r}_t$ (i.e., R_t) has a unique solution if and only if $R_{\Lambda\mathbf{v}}$ is invertible. Hence the meeting point can be derived,

$$\hat{\mathbf{x}} = \mathbf{x}_1 + R_t\mathbf{v}_1. \tag{2.28}$$

2.4.4 Open-Loop Control Using Linear Programming

One way to solve Equation (2.14) is via linear programming. The objective is to find a vector $\mathbf{t}$ such that the total control time $\|\mathbf{t}\|_1$ is minimized subject to (2.14) with $n = 3$,

$$\begin{aligned} &\min \sum_{j=1}^{N} t_j, \text{ such that } t_j \geq 0, j = 1, ..., N, \text{ and} \\ &\begin{bmatrix} R_1\mathbf{v}_1 & R_2\mathbf{v}_1 & \dots & R_N\mathbf{v}_1 \\ R_1\mathbf{v}_2 & R_2\mathbf{v}_2 & \dots & R_N\mathbf{v}_2 \\ R_1\mathbf{v}_3 & R_2\mathbf{v}_3 & \dots & R_N\mathbf{v}_3 \end{bmatrix} \mathbf{t} = \begin{bmatrix} \Delta\mathbf{x}_1 \\ \Delta\mathbf{x}_2 \\ \Delta\mathbf{x}_3 \end{bmatrix}, \end{aligned} \tag{2.29}$$

where the initial thrust vectors are linearly independent. Because the t_j must be nonnegative, the number of rotation matrices (N) should be much greater than nine, as shown in Figure 2.7a.

2.4.5 Feedback Control

Let the state space representation of n such agents be

$$\begin{bmatrix} \dot{\mathbf{x}}_1 & \cdots & \dot{\mathbf{x}}_n \end{bmatrix} = R(t) \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_n \end{bmatrix}, \tag{2.30}$$

where $\mathbf{x}_i$ denotes position of the i^{th} agents, $\mathbf{v}_i$ describes the initial thrust vector, and $R(t)$ is the shared control input. Construct an objective function based on the sum of squared Euclidean distance error

$$V(t) = \frac{1}{2}\sum_{i=1}^{n} \left(\hat{\mathbf{x}}_i - \mathbf{x}_i(t)\right)^\top \left(\hat{\mathbf{x}}_i - \mathbf{x}_i(t)\right), \tag{2.31}$$

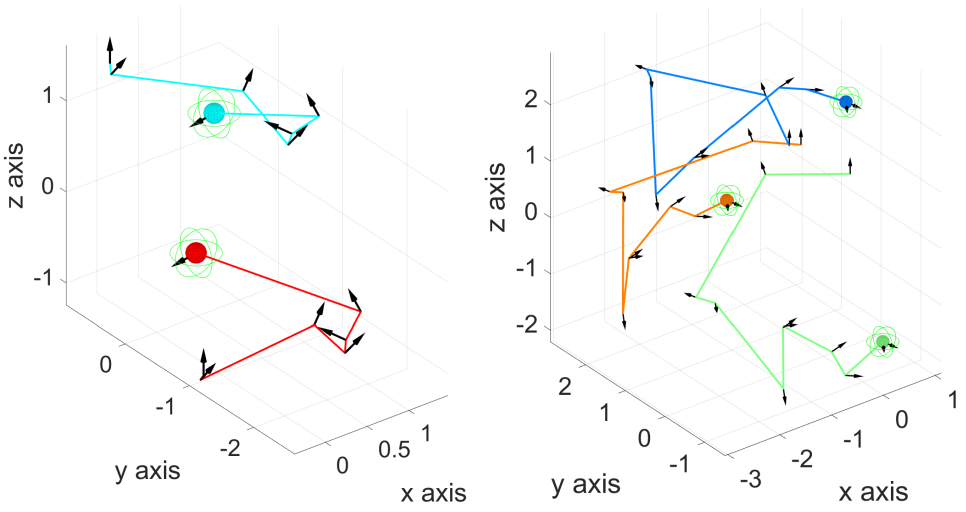


Figure 2.5: Open-loop control simulations using linear programming. The goal locations are indicated by green orbits. In each figure, all spheres move the same total distance and reach the goal location at the same time.

where $\hat{\mathbf{x}}_i$ is the i^{th} goal location. Hold $R(t)$ constant for time interval $[t_k, t_k + \tau_k)$, such that τ_k minimizes the convex objective function. Consider the first-order necessary condition,

$$\begin{aligned} \dot{V}(t_k + \tau_k) = \tfrac{d}{d\tau_k} V(t_k + \tau_k) = 0 \text{ and} \\ -\sum_{i=1}^{n} (\hat{\mathbf{x}}_i - \mathbf{x}_i(t_k) - \tau_k R(t_k)\mathbf{v}_i))^\top R(t_k)\mathbf{v}_i = 0, \text{ and thus} \end{aligned} \tag{2.32}$$

$$\tau_k = \frac{\sum_{i=1}^{n} (\hat{\mathbf{x}}_i - \mathbf{x}_i(t_k))^\top R(t_k)\mathbf{v}_i}{\sum_{i=1}^{n} \|R(t_k)\mathbf{v}_i\|_2^2}. \tag{2.33}$$

Note that the time interval τ_k must be non-negative, so if there exists an $R(t_k)$ such that $\tau_k > 0$, then $\int_{t_k}^{t_k+\tau_k} \dot{V}(t)dt < 0$, and the total distance error decreases monotonically with time; otherwise, the objective function has reached the minimum. Thus the system kinematics can also be written as

$$\mathbf{x}_i(t_{k+1}) = \mathbf{x}_i(t_k) + R(t_k)\mathbf{v}_i\tau_k. \tag{2.34}$$

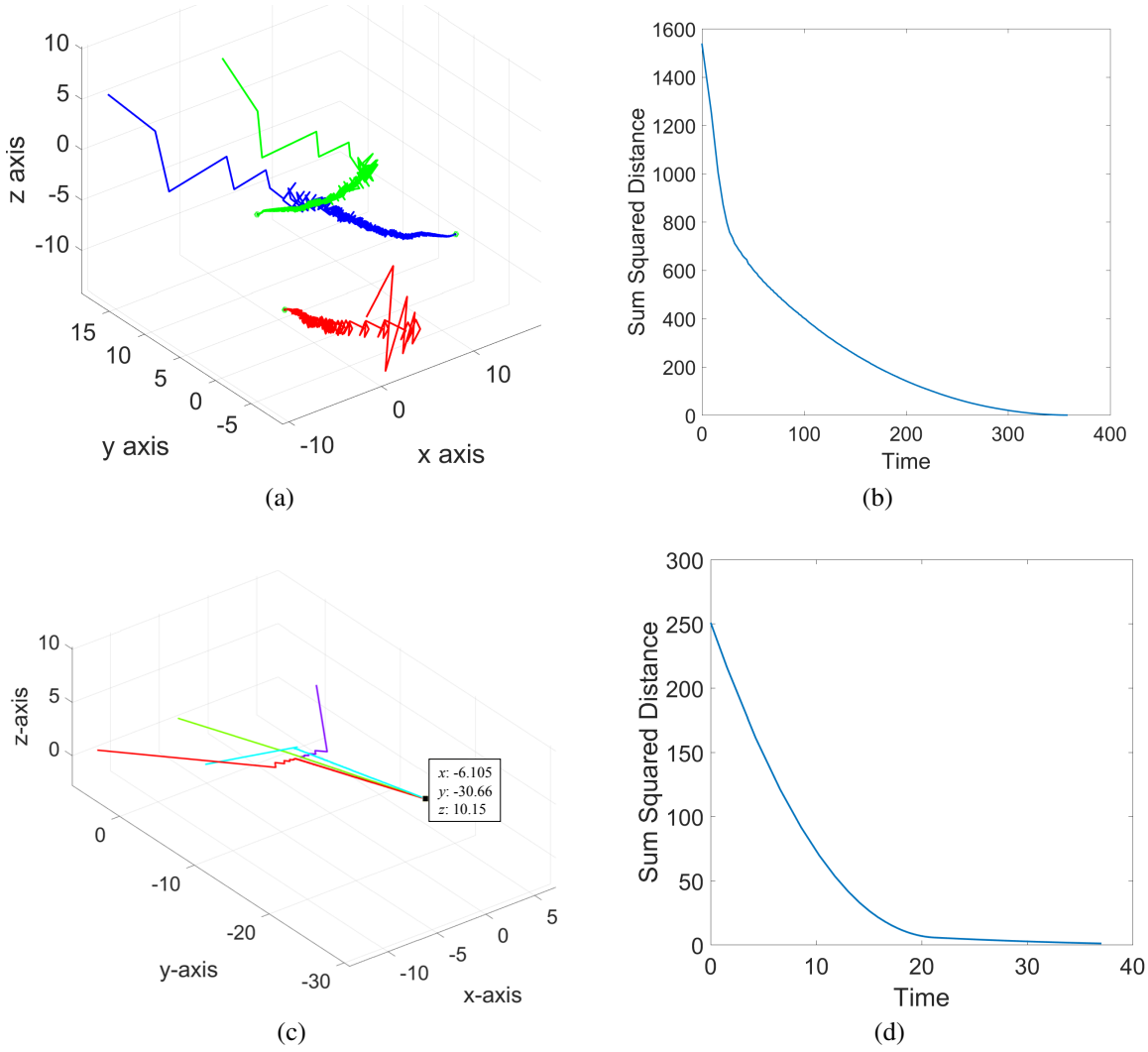


Figure 2.6: (a) & (c) Simulations of self-propelled agents using greedy optimal control. All agents reach the goal location at the same time. (b) & (d) The corresponding objective function plot with simulation time.

2.4.6 Greedy Optimal Control

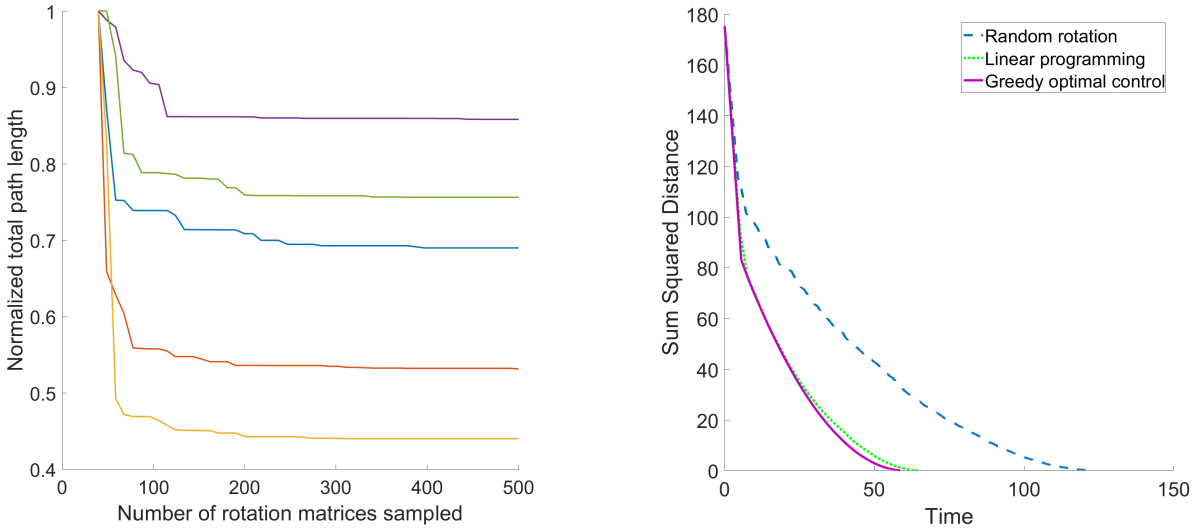
It has been discussed that the optimal actuation time to minimize the objective function $V(t)$ with a constant rotation matrix $R(t)$ within time interval $[t, t+\tau_k)$. By carefully choosing this $R(t)$, the objective function follows the steepest gradient during each time interval,

$$R(t_k) = \underset{\alpha_k,\beta_k,\gamma_k}{\operatorname{argmin}} \dot{V}(t_k), \tag{2.35}$$

with $R(t_k)$ a function of rotation angles α_k, β_k, and γ_k,

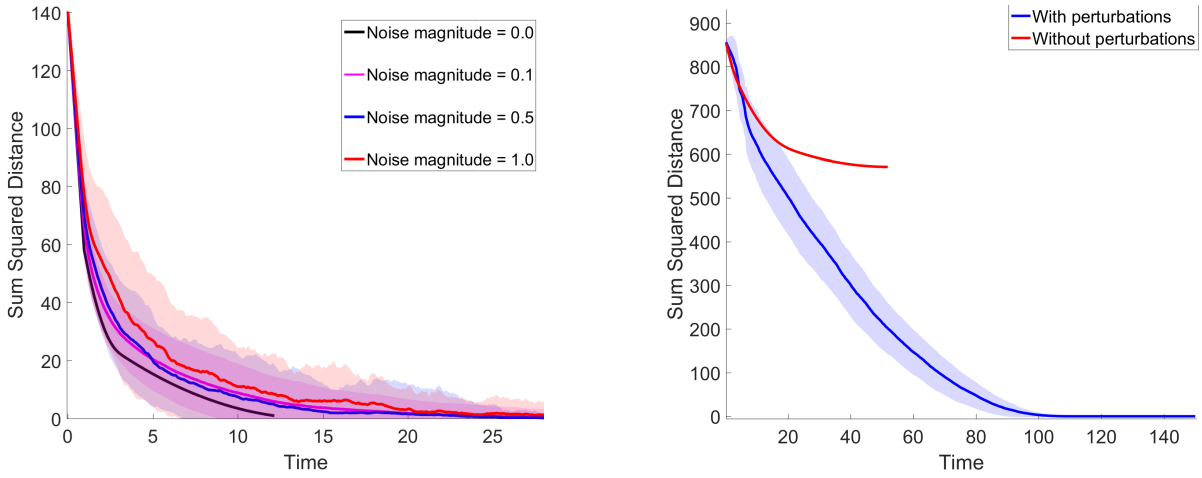
$$R(t_k) = R(t_{k-1}) R_{x,\alpha_k} R_{y,\beta_k} R_{x,\gamma_k}. \tag{2.36}$$

This indicates that at t_k, each agent implements the shared rotation control $R(t_k)$ which is equivalent to rotating α_k about the current local x-axis, then β_k about the current y-axis, and finally rotating γ_k about the current x-axis.



(a) (b)

Figure 2.7: Representative parameter optimization for controlling three self-propelled agents in 3D. (a) In open-loop control, path lengths decrease monotonically with the number of rotation matrices N. (b) Performance comparison of feedback control laws.



(a) Control of three agents with position noises.

(b) Control of ten agents with thrust perturbations.

Figure 2.8: Controlling multiple self-propelled agents in 3D. The solid lines are the average results of 50 simulations, and the shaded areas represent the corresponding standard deviation.

2.4.7 Control with State Perturbations

Under ideal conditions, agent motions are assumed to be perfectly implemented without errors. Up to four agents can be steered to meet simultaneously with a shared control input. In general, for $n > 4$ agents, the objective function reaches a local minimum, and the shared control cannot bring all agents to their target locations simultaneously.

In practice, agent rotations and translations may not be precise due to process noise and measurement noise. Inspired by the 2D results in [48], it is possible to escape local minima in 3D, if we relax previous assumptions and assume agents are subjected to independent random disturbances on their thrust vectors after each translation. In the following we show that there always exist disturbances that enable steering the agents closer to their goal. In simulation, it shows that randomly perturbing each agent's thrust vector enables convergence.

Section 2.4.5 shows that $\tau_k \leq 0$ if the objective function reaches a local minimum, and according to Equation (2.33),

$$\sum_{i=1}^{n} (\hat{\mathbf{x}}_i - \mathbf{x}_i(t_k))^\top \mathbf{v}_i(t_k) \leq 0, \tag{2.37}$$

where $R(t_k)\mathbf{v}_i$ is replaced by $\mathbf{v}_i(t_k)$. Let $\boldsymbol{\delta}_i(t_k) \in \mathbb{R}^3$ be a perturbation of the i^{th} thrust vector, such that the perturbed thrust vector $\mathbf{v}'_i(t_k) \parallel (\hat{\mathbf{x}}_i - \mathbf{x}_i(t_k))^\top$, where

$$\mathbf{v}'_i(t_k) = \frac{\mathbf{v}_i(t_k) + \boldsymbol{\delta}_i(t_k)}{\|\mathbf{v}_i(t_k) + \boldsymbol{\delta}_i(t_k)\|_2}\|\mathbf{v}_i(t_k)\|_2. \tag{2.38}$$

Therefore

$$\sum_{i=1}^{n}(\hat{\mathbf{x}}_i - \mathbf{x}_i(t_k))^\top \mathbf{v}'_i(t_k) > 0, \tag{2.39}$$

except for the case that the total position error is 0.

This example proves the existence of a perturbation that makes the objective function monotonically decreasing, even if the system is in a local minimum. In this chapter's simulations, the thrust vector perturbations are sampled from a zero-mean normal distribution.

2.5 Simulation

This section investigates the controllability for self-propelled agents in 3D free-space, including steering up to three spheres to arbitrary goals with a shared open-loop control (linear programming) and a shared closed-loop control, and control of four agents to meet. With perturbations on thrust vectors, it shows the capability of driving ten agents to the origin.

In Figure 2.5, open-loop control with linear programming is implemented with up to three self-propelled agents and drives them to arbitrary goals. A colored line describes the trajectory of each sphere. Black arrows indicate the local coordinate frame z-axis for the subsequent move. According to section 2.4.4, N angles are randomly generated to provide a large number of rotation matrix candidates, but linear programming selects N_k among N ($N_k \ll N$) matrices to apply actuation and steer the spheres to goal locations. In the simulation, a rotation matrix could be generated by $R(t) = R_{x,\alpha}R_{y,\beta}R_{x,\gamma}$, where $\alpha, \beta, \gamma \in [0, 2\pi]$. N is set to 200, and usually $N_k = 9$. Figure 2.7a shows that N can affect the performance of linear programming: if N is small, the provided paths to goals might be much longer than the optimal

solution. The total path length decreases monotonically with N. The total path length has little change when $N \geq 200$.

Section 2.4.5 introduces a closed-loop control law such that a rotation matrix $R(t)$ is held constant for each actuation time interval τ_k. Aptly choosing each τ_k, the cost function decreases monotonically along the trajectory. In simulation, $R(t)$ is generated with the following methods: (i) random angle generation, (ii) using the N_k rotation matrices selected by linear programming, and (iii) minimizing $\dot{V}(t)$ in Equation (2.32) with respect to $\alpha_k, \beta_k, \gamma_k$. Figure 2.6 shows the trajectories of greedy optimal control with 3 and 4 agents using method (iii). Not all $\alpha_k, \beta_k, \gamma_k$ must be non-zero values. For example, let $\alpha_k, \beta_k = 0$, or $\alpha_k, \gamma_k = 0$, or $\gamma_k = 0$ for greedy optimal control. For more simulation results, please refer to the repository on GitHub [44].

A performance comparison of the above three methods is shown in Figure 2.7b, which indicates that methods (ii) and (iii) have competitive performance, while method (i) takes about twice as long to converge.

Section 2.4.3 shows controlling up to nine DOF with no state perturbations. Figure 2.6 gives two examples of controlling up to four self-propelled agents: steering three agents to predefined goal locations, and moving four spheres to their mean positions, as shown in Equation (2.28).

When thrust vector perturbations are considered, the shared control is capable of moving out of local minima and bringing many agents ($n > 4$) to arbitrary locations, as shown in Figure 2.8b. In addition, the influence of position noises are compared in Figure 2.8a.

2.6 Conclusion

This chapter proves limitations on control for self-propelled agents that all receive the same rotation commands, but extends the existing literature which focuses on two dimensional results to show that nine degrees-of-freedom of position can be controlled. In 2D only one

agent can be steered to an arbitrary position, and two agents have only one possible meeting point. In 3D up to three agents may be steered to arbitrary positions, and four agents have only one possible meeting point.

There are many avenues for future work. These include optimal control results and analytical solutions to the optimal rotations for the controllers in section 2.4. In particular, calculate the meeting location for two spheres that requires the shortest control sequence. This problem is trivial in 2D, but potentially hard in 3D.

These controllers have potential insights for real-world systems that are self-propelled and can be steered by the orientation of a global field. The size of micro-scale robots makes it difficult to include onboard computation, so they are often steered by external fields. Examples include steering magnetized single-celled organisms [49–52], magnetotactic bacteria [12, 53–55] and catalytic Janus particles with magnetic cores [5, 56–58].

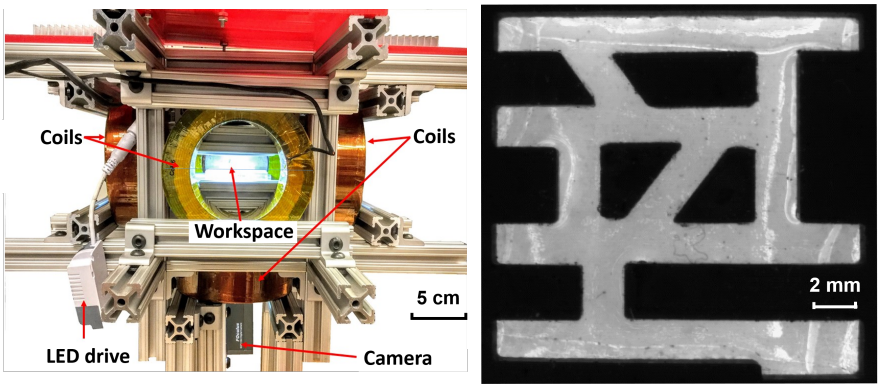
Chapter 3

Path Planning and Aggregation for a Microrobot Swarm in Vascular Networks Using a Global Input

3.1 Introduction

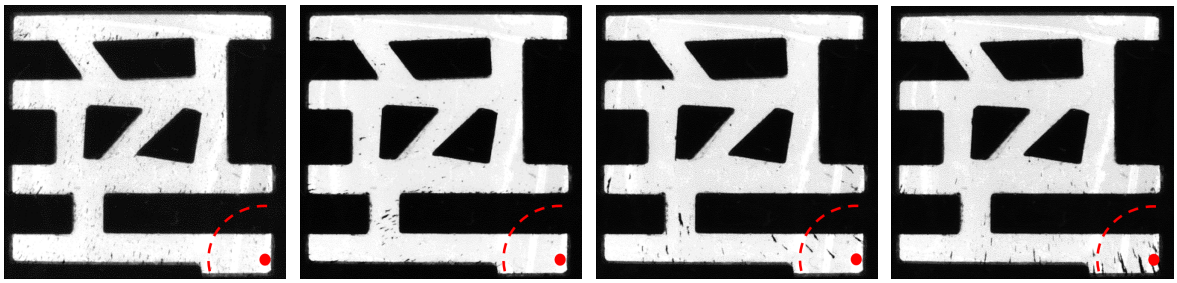
Microrobots have great potential to be used in non-invasive surgery for drug delivery. Traditional drug delivery circulates the human body indiscriminatingly, which is why chemotherapy kills healthy and tumor cells alike. To reduce toxic drug exposure to healthy cells, *targeted drug delivery* seeks to steer chemotherapy directly to diseased tissue. Many methods for drug delivery have been explored, including beaded delivery formulations, liposomal delivery systems, encapsulated chemotherapy in nanoparticles, and magnetic microcarriers navigated by magnetic fields [59].

Recent works have investigated many strategies to manipulate a swarm of simple robots with limited computation and communication [23–26].



(a) (b)

Figure 3.1: (a) The six-coil electromagnetic system with a bottom-view camera. (b) A vascular network tested in experiments.



(a) (b) (c) (d)

Figure 3.2: Captured frames from a experiment. The goal location is marked with a red point. (a) $t = 0$ min, (b) $t = 4$ min, (c) $t = 19$ min, (d) $t = 36$ min.

Pierson et al. proposed a control strategy that by introducing herders to drive a swarm of herding animals to the desired location with repelling potential fields [23]. Fine et al. reported how to actively design environments to assist the process of controlling multiple agents using *shape grammars* [24]. This method addresses the automatic generation of environments given specific swarm objective and a control model of agents. Becker et al. showed particle computation methods to perform permutations between different swarm formations by designing unit-size obstacles in a grid workspace, where they used mobile particles with maximal motion (particles moved until they hit an obstacle or an obstructed particle) and a global input [25]. Bobadilla et al. gave another example of exploiting environment, where a state space is partitioned into discrete transition systems, and gates are configured to guide a swarm of simple robots to achieve state transition, and thereby to accomplish high-level tasks [26].

However, many microrobots have limited capabilities for sensing and actuating, so external sensors (e.g. magnetic resonance imaging (MRI), cameras) and actuators (e.g. external electric or magnetic fields) must be employed. Experimentally, microrobot swarms such as paramagnetic microparticles [8], *Tetrahymena pyriformis* [16], and magnetotactic bacteria [13, 14] have attracted growing attention in many applications of micro-assembly and targeted therapies. These microrobots usually are physically simple agents, and are steered by global fields where every robot receives the same control signal. Many strategies and algorithms have been developed for navigation and motion control of microrobots in free space [18–20]. Khalil

et al. demonstrated control of a single microrobot in a micro-fabricated maze [21]. Scheggi et al. implemented and compared six path planning algorithms using magnetic microrobots [22]. Mahadev et al. explored microrobot swarm aggregation in a planar grid environment, where microrobots are of different sizes, capable of overlapping, moving in discrete steps and directed by a share, global, control input [27].

This chapter addresses aggregating a microrobot swarm in vascular networks using only a global input. This is divided into three challenges: (i) generating swarm trajectories, (ii) realizing robust swarm transitions, and (iii) constructing swarm-level strategies to reduce task time complexity. To address (i) and (ii), this chapter uses an augmented rapidly-exploring random tree (RRT) for path planning. A divide-and-conquer strategy is employed to address (iii) for swarm aggregation. The problem formulation and modeling are elaborated in 3.2.1. Section 3.2.2 and 3.2.3 introduce trajectory generation and algorithms for aggregation. Section 3.3 compares performance with different maps, aggregation methods, and swarm populations. A hardware implementation is described in Section 3.4.

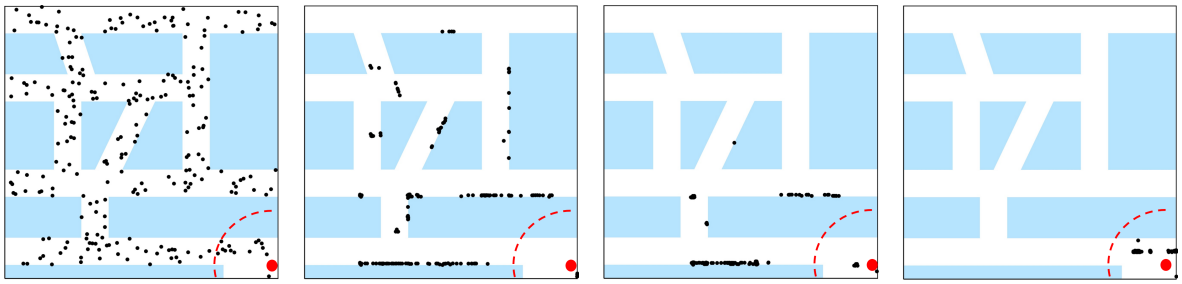
3.2 Methodology

3.2.1 Problem Formulation

Given a bounded 2D space $G \subset \mathbb{R}^2$, the plane is partitioned by obstacles into free space (G_{free}), obstacle space (G_{obs}) and contact space (∂G_{obs}), where ∂G_{obs} is the set of boundaries of all obstacles. G_{free} is continuous, and connected by paths of width at least w. A population of n simple microrobots, identified by $r_i,\ i \in \{1, \cdots, n\}$, are randomly distributed in free space at time t_0, with their positions represented by $\mathbf{p}_{t_0}^{r_i} \in G_{free}$. These microrobots are physically simple and have no on-board computation or communication. Their bodies are small compared to w, and interaction between agents is ignored. They are under the control of a global signal, that is, all units move in the same direction at the same speed until they hit an obstacle. We assume that microrobots in contact with the wall have zero velocity if the control input has a

component directed into the wall.

The goal is to quickly collect this swarm at a gathering point $\mathbf{q}^g$ (Figure 3.3). It can be inferred from [27] that there exists a solution to the problem. First, this chapter focuses on trajectory generation for each microrobot, and then presents control strategies for swarm aggregation.



(a) (b) (c) (d)

Figure 3.3: In simulation, blue polygons represent obstacles, white channels are free space, and a red dot for the goal location. (a) Simulated aggregation process after 1 step, (b) 200 steps, (c) 500 steps, and (d) 800 steps.

Define a distance metric $d : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}_{\geq 0}$ which denotes the cost to reach $\mathbf{p}^j$ from $\mathbf{p}^i$ via an accessible path in G_{free}. A microrobot is modeled by a discrete-time dynamic system, with a control input vector $\mathbf{u}_t$,

$$\mathbf{p}_{t+1}^{r_i} = \mathbf{p}_t^{r_i} + \delta \cdot \mathbf{u}_t, \tag{3.1}$$

where $\mathbf{p}_t^{r_i} \in \mathbb{R}^2$ represents the position of robot r_i at time t, $\|\mathbf{u}_t\| \ll w$, and δ is a scale factor that depends on $\mathbf{u}_t$ and $\mathbf{p}_t^{r_i}$,

$$\delta = \begin{cases} 1, & \text{if the path is collision-free,} \\ \|\mathbf{p}_t^{r_i} - \mathbf{p}^x\|_2, & \text{otherwise.} \end{cases} \tag{3.2}$$

If $\mathbf{u}_t$ steers the robot from $\mathbf{p}_t^{r_i}$ into any obstacle, r_i stops at the obstacle boundary $\mathbf{p}^x$.

An n microrobots swarm has $2n$ degrees-of-freedom, but a global input has only x and y components, so the system is under-actuated.

Let a cost function $F(n,t)$ at time t be the average distance of all robots to the gathering point $\mathbf{q}^g$,

$$F(n,t) = \frac{1}{n}\sum_{i=1}^{n} d\langle \mathbf{p}_t^{r_i}, \mathbf{q}^g \rangle. \tag{3.3}$$

The trajectory generation problem is interpreted as determining the single-source shortest paths in a weighted graph using distance metric d. The aggregation problem finds a deterministic policy to decrease the cost function, such that as $t \to \infty, F(n,t) < \sigma$, for some small $\sigma \in \mathbb{R}^+$.

3.2.2 Swarm Path Planning

This chapter introduces an obstacle-weighted rapidly-exploring random tree (RRT) planner to explore the environment, and discovers collision-free routes to the goal location $\mathbf{q}^g$.

Sampling-based motion planning algorithms have shown great success in exploring collision-free paths for many scenarios. Probabilistic roadmaps (PRMs) [60] and rapidly-exploring random trees (RRTs) [61] are two popular planners. These planners generate random configurations in free space, connect them to create a graph of feasible paths, and link start and goal locations. This chapter focuses on multi-shot 2D path planning with RRT and its extensions. Many attempts have shown success in improving the performance of RRTs near obstacles, such as narrow passage and tight region problems using a retraction strategy [62, 63].

Applying an RRT-based path to a microrobot swarm using a global input can lead to problems: moving one particular agent may cause all the others to drift away from their initial locations since all agents receive the same control signal. These locations may not be near any existing configurations on the tree ($\mathcal{T}$). Hence the RRT planner should have the following feature: for any robot r_i in G_{free},

$$\|\mathbf{p}^{r_i} - \mathbf{q}_v\|_2 \leq \epsilon, \tag{3.4}$$

for some $\epsilon > 0$, where $\mathbf{q}_v$ is the nearest vertex in $\mathcal{T}$. Grow a tree in an unbiased manner, such that sampling configurations are distributed uniformly. Also sufficient configurations are

Table 3.1: Variables, and functions used in Algorithm 1 and 2.

V — the set of vertices (configurations) of $\mathcal{T}$.
V_{obs} — the set of sampling nodes on the boundary of obstacles.
V^* — the set of configurations near medial axes of G_{free}.
$\pi(\mathbf{q}_v)$ — the predecessor of the configuration $\mathbf{q}_v$ in $\mathcal{T}$.
$Adj(\mathbf{q}_v)$ — the set of adjacent vertices.

required to guarantee the constraint Equation (3.4).

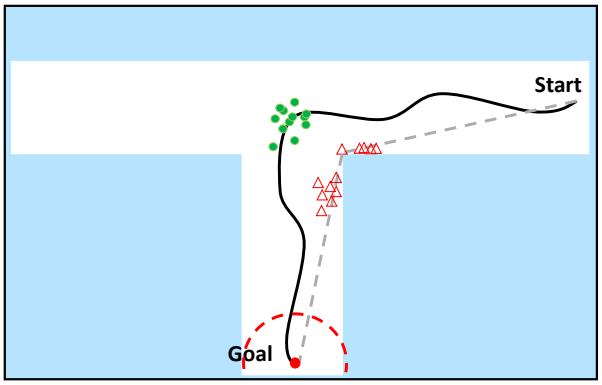


Figure 3.4: One swarm (green circles) follows a black solid trajectory near the medial axes. Another swarm (red triangles) follows the shortest path to the goal, which traps some microrobots at the corner and slows the aggregation process.

Another issue with microrobot swarms is the environment interference. For example, in Figure 3.4 a gray dashed line (trajectory 1) is the shortest path to the goal location, and a black solid line (trajectory 2) represents a near-medial-axis path. Although trajectory 1 is shorter than 2, such a path significantly slows the aggregation process near obstacles. An approach is proposed to reroute existing paths towards medial axes of free space. Compared to retraction-based planners, this approach replans a new route with existing configurations in $\mathcal{T}$ instead of generating biased sampling of tree nodes.

The basic RRT planner builds a connected tree rooted at the goal location, and samples tree nodes randomly in free space of G to explore the graph. This process (Algorithm 1) proceeds as follows: to grow the tree, generate a random point $\mathbf{q}_{rand}$ in G_{free}, and perform the

Algorithm 1 *RRT*.
Input: configuration space G, goal location $\mathbf{q}^g$, total number of configurations $NumNode$.
Output: an RRT $\mathcal{T}$

1: $V = \{\mathbf{q}^g\}, V_{obs} = \emptyset, \mathcal{T} = \{V, V_{obs}\}$
2: **while** $|V| \leq NumNode$ **do**
3: $\quad \mathbf{q}_{rand} \leftarrow$ a randomly generated point in G
4: $\quad \mathbf{q}_{near} \leftarrow$ the nearest neighbor of $\mathbf{q}_{rand}$ in V
5: $\quad \mathbf{q}_{new} \leftarrow$ extend $\mathbf{q}_{near}$ towards $\mathbf{q}_{rand}$ for unit length
6: $\quad$ **if** $(\mathbf{q}_{near}, \mathbf{q}_{new}) \cap G_{obs} = \emptyset$ **then**
7: $\quad\quad V = V \cup \{\mathbf{q}_{new}\}$
8: $\quad\quad \pi(\mathbf{q}_{new}) = \mathbf{q}_{near}$
9: $\quad\quad d\langle \mathbf{q}_{new}, \mathbf{q}^g \rangle = d\langle \mathbf{q}_{new}, \mathbf{q}_{near} \rangle + d\langle \mathbf{q}_{near}, \mathbf{q}^g \rangle$
10: $\quad$ **else**
11: $\quad\quad \mathbf{q}_{obs} \leftarrow (\mathbf{q}_{near}, \mathbf{q}_{new}) \cap \partial G_{obs}$
12: $\quad\quad V_{obs} = V_{obs} \cup \{\mathbf{q}_{obs}\}$
13: $\quad$ **end if**
14: **end while**
15: return $\mathcal{T}$

nearest neighbor query (lines 3-4); next, $\mathbf{q}_{near}$ is extended towards $\mathbf{q}_{rand}$ with unit length, and ends with $\mathbf{q}_{new}$; if the edge $(\mathbf{q}_{near}, \mathbf{q}_{new})$ is collision-free, which corresponds to a control input steering robots from $\mathbf{q}_{near}$ to $\mathbf{q}_{new}$, add the new node to the tree and update the distance metric (lines 6-9). Since all sampling points are connected to the tree, a robot can reach the goal location from any tree nodes simply by following their predecessors iteratively.

Note that in Algorithm 1, lines 10-12 are different from the original RRT [61]: if there is a collision along the path, we retract $\mathbf{q}_{new}$ to the boundary of the obstacle $\mathbf{q}_{obs}$. $\mathbf{q}_{obs}$ is not considered as a valid configuration in $\mathcal{T}$, instead, add it to V_{obs} to grow an obstacle-weighted RRT. These obstacle nodes assist in steering paths away from obstacles.

This process is illustrated in Algorithm 2, and compare it with the original RRT in Figure 3.5. The weight of a node $\mathbf{q}_v$ in $\mathcal{T}$ is calculated as

$$w(\mathbf{q}_v) = e^{-a\|\mathbf{q}_v - \mathbf{q}_{obs}^{near}\|_2 + b}, \tag{3.5}$$

where $a, b \in \mathbb{R}^+$. Therefore, weight decreases with distance from nearby obstacles. Hence, the new path tends to proceed near medial axes of free space if it identifies a gradient descent

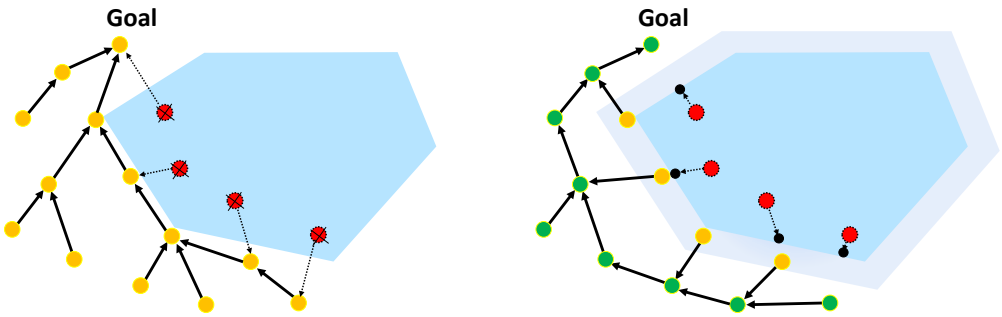
Algorithm 2 *Obstacle-weighted RRT.*
Input: the RRT $\mathcal{T}$ of Algorithm 1.
Output: an obstacle-weighted RRT: $\mathcal{T}_{obs}$

1: $V^* = \{\mathbf{q}^g\}, \mathcal{T}_{obs} = \{V, V_{obs}, V^*\}$
2: **for** all $\mathbf{q}_v \in V$ **do**
3: $\quad \mathbf{q}_{obs}^{near} \leftarrow$ the nearest neighbor of $\mathbf{q}_v$ in V_{obs}
4: $\quad w(\mathbf{q}_v) = e^{-a\|\mathbf{q}_v - \mathbf{q}_{obs}^{near}\|_2 + b}$
5: $\quad$ **if** $w(\mathbf{q}_v) < \zeta$ **then**
6: $\quad\quad V^* = V^* \cup \{\mathbf{q}_v\}$
7: $\quad$ **end if**
8: **end for**
9: **for** all $\mathbf{q}_v^* \in V^*$ **do**
10: $\quad d\langle \mathbf{q}_v^*, \mathbf{q}^g \rangle = \infty$
11: $\quad$ **for** all $\mathbf{q}_u^* \in Adj(\mathbf{q}_v^*) \cap V^*$ **do**
12: $\quad\quad$ **if** $d\langle \mathbf{q}_v^*, \mathbf{q}^g \rangle > d\langle \mathbf{q}_v^*, \mathbf{q}_u^* \rangle + w(\mathbf{q}_u^*) + d\langle \mathbf{q}_u^*, \mathbf{q}^g \rangle$ **then**
13: $\quad\quad\quad \pi(\mathbf{q}_v^*) = \mathbf{q}_u^*$
14: $\quad\quad\quad d\langle \mathbf{q}_v^*, \mathbf{q}^g \rangle = d\langle \mathbf{q}_v^*, \mathbf{q}_u^* \rangle + w(\mathbf{q}_u^*) + d\langle \mathbf{q}_u^*, \mathbf{q}^g \rangle$
15: $\quad\quad$ **end if**
16: $\quad$ **end for**
17: **end for**
18: **for** all $\mathbf{q}_v \in V$ **do**
19: $\quad$ **if** $\pi(\mathbf{q}_v) \notin V^*$ **then**
20: $\quad\quad \mathbf{q}_{near}^* \leftarrow$ the nearest neighbor of $\mathbf{q}_v$ in V^*
21: $\quad\quad \pi(\mathbf{q}_v) = \mathbf{q}_{near}^*$
22: $\quad$ **end if**
23: **end for**
24: return $\mathcal{T}_{obs}$

path to the goal with minimum-weight nodes. A near-medial-axis set of configurations is constructed as

$$V^* = \{\mathbf{q}_v \in V | w(\mathbf{q}_v) < \zeta\}, \tag{3.6}$$

for some $\zeta \in \mathbb{R}^+$. The trajectory generation (lines 7-16) is shown in Figure 3.5(b). We trim the tree to remove edges not connecting to vertices in V^*, and perform adjacent neighbors query to regrow the tree towards near-medial-axis regions. Section 3.2.3 shows that obstacle-weighted RRT decreases aggregation time.



(a) (b)

Figure 3.5: (a) RRT: yellow dots are configurations of $\mathcal{T}$, and red dots are abandoned extensions within the obstacle. (b) Obstacle-weighted RRT: green dots are near-medial-axis configurations $\in V^*$, yellow dots are elements in V affected by the obstacle.

3.2.3 Swarm Aggregation

This section presents a divide-and-conquer aggregation method with heuristic strategies to improve performance. The motivation behind microrobot swarm aggregation is efficient control strategies for drug delivery in vascular networks. However, a global input with a highly under-actuated swarm system makes it difficult constructing an optimal controller. Pioneering research has proposed different strategies for the aggregation/gathering problem, but most of them are element-wise algorithms, that is, performing the task in terms of individuals. The goal of this chapter is to propose a swarm-level strategy to carry out swarm aggregation, and reduce time complexity compared to element-wise methods.

Benchmark Aggregation

A benchmark heuristic for aggregation is to move one microrobot to the goal, and then move the next agent. Repeat this till all robots gather near the goal location. In this chapter, the benchmark heuristic moves the farthest microrobot to the goal.

The benchmark aggregation is presented in Algorithm 3. with the following assump-

tions: (i) the graph G is connected and bounded, and (ii) the goal location is inside a closed region (Definition 1) at an end point (Definition 2). The second assumption is inspired by the concept of discrete system transitions in [26], where gates are constructed to guide transitions from one region to another. In practice, a closed region indicates that once microrobots reach this area, it is hard for them to escape, given global inputs driving robots to $\mathbf{q}^g$. This is reasonable and essential, because chemotherapy molecules are designed to release from carriers once they reach the region of tumor cells [59].

Algorithm 3 *Heuristic Aggregation*.
Input: $\mathcal{T}_{obs} = \{V, V_{obs}, V^*\}$, initial positions $\{\mathbf{p}_{t_0}^{r_i}\}$ of all robots r_i.
Output: $\mathbf{u}_t$

```
while F(n,t) > σ do
    r_i ← the farthest robot
    q_v^{r_i} ← the nearest vertex ∈ V to r_i
    u_t ← move r_i towards q^g via q_v^{r_i}
end while
```

Definition 1. (closed region.) *Considering a global input $\mathbf{u}_t$ that drives all robots to the goal $\mathbf{q}^g$, a closed region is a positive invariant set $\mathcal{M} \subset G_{free}$, $\mathbf{q}^g \in \mathcal{M}$. Given $\mathbf{u}_t$ and a robot r_i, if $\mathbf{p}_{t_0}^{r_i} \in \mathcal{M}$ at t_0, then $\mathbf{p}_t^{r_i} \in \mathcal{M}$ for all $t > t_0$. $\mathcal{M}$ is bounded, so $\forall\ \mathbf{p}^{r_i} \in \mathcal{M}, \exists\ c > 0$, such that*

$$d\langle \mathbf{p}^{r_i}, \mathbf{q}^g \rangle < c\sigma. \tag{3.7}$$

Definition 2. (endpoint.) *An endpoint is a set $\mathcal{D} \subset \mathcal{M}, \mathbf{q}^g \in \mathcal{D}$, with the following properties: (1) $\forall \mathbf{p}^{r_i} \in \mathcal{D}$, $d\langle \mathbf{p}^{r_i}, \mathbf{q}^g \rangle < \sigma$; (2) given that all robots $\{r_i\} \in G_{free}$ aggregate inside $\mathcal{M}$ and a global input $\mathbf{u}_t$ moves robots towards $\mathcal{D}$, if $\mathbf{p}^{r_i} \in \mathcal{D}$ at t_0, then $\mathbf{p}^{r_i} \in \mathcal{D}$ for all $t > t_0$.*

Divide-and-Conquer Aggregation

This method recursively aggregates microrobots into a smaller region that contains the goal. A proper definition of 'region' reduces aggregation time. If each microrobot is manipulated all the way to the goal location, the algorithm is transformed into the heuristic aggregation.

The divide-and-conquer technique has two stages. it begins by splitting the aggregation problem into subproblems in smaller regions. Then it recursively performs discrete region transitions of microrobot swarms.

The first stage 'divide' performs map segmentation of vascular systems like Figure 3.9. In these maps, vessels are connected by junctions, and most of them are T-junctions.

Definition 3. (Region R_i.) *Define a partition of a map G_{free} as non-overlapping regions* $\{R_i\}_{i=1,2,\cdots,N_R}$, *such that* $\{R_i \in G_{free} | \bigcup_{i=1}^{N_R} R_i = G_{free}, R_i \cap R_j = \emptyset, \forall i \neq j\}$.

As shown in Figure 3.6(a), junction nodes (green dots) can be separated from other nodes in straight vessels (orange dots) by their spatial distributions. Considering the set $\{\mathbf{q}_v \cup adj(\mathbf{q}_v)\}$, i.e. a configuration and its adjacent neighbors in Cartesian coordinates, the shape ratio is defined as the eigenvalue associated with the principal component of the set divided by the other eigenvalue. If the shape ratio is less than a threshold, such configuration is regarded as a junction node; otherwise, a vessel node. These junction nodes can determine boundaries between regions. Here the maximal width of local channels are taken as the range of adjacent neighbors. With these definitions, perform map segmentation using results from obstacle-weighted RRT. This process is presented in Algorithm 4, and illustrated in Figure 3.6:

1. use near-medial-axis configurations $\mathbf{q}_v^* \in V^*$ to identify junction nodes (lines 1-4);
2. partition the set of junction nodes (V_J) using Euclidean distance (line 5), and yield N_R junction clusters;
3. split free space into N_R regions corresponding to the N_R junction clusters (line 6);
4. partition each region into branches $\{B_{j,k}\}_{k=1,2,\cdots}$ by their orientations (lines 7-10), where $B_{j,k}$ is the k-th branch in R_j.

In step 3, region segmentation proceeds in three phases. First select a cluster of V_J and set the junction nodes as seeds. Then grow the region with these seeds by adding their

Table 3.2: Variables and functions used in Algorithm 4.

Clustering(V_J, 'distance') — partition elements of V_J into junction clusters with the Euclidean distance metric, and returns the centroid of each junction $\{\mathbf{q}_j^d\}$.
RegionSeg($V^*, \{\mathbf{q}_j^d\}$) — partition elements of V^* into clusters by junctions, and returns the region ID: $\{R_j\}$. First, assign a unique region ID R_j to the centroid of a junction ($\mathbf{q}_j^d$) and its adjacent neighbors *Adj*($\mathbf{q}_j^d$). Then, assign all descendants of *Adj*($\mathbf{q}_j^d$) the same region ID.
Clustering($S, \mathbf{q}_j^d$ 'orientation') — partition elements $s_i \in S$ into clusters by the orientation of a directed edge $(s_i, \mathbf{q}_j^d)$, and returns $\{\mathbf{q}_{j,k}^o\}$ the mean orientation..
BranchSeg($S, \{\mathbf{q}_{j,k}^o\}$)— assign a branch ID to each element of S by orientation, where $\mathbf{q}_{j,k}^o$ is the orientation of branch $B_{j,k}$.

Algorithm 4 *Map Segmentation*.
Input: The obstacle-weighted RRT $\mathcal{T}_{obs} = \{V, V_{obs}, V^*\}$.
Output: Map segmentation: $M = \{V_J, \psi(V^*), \phi(V^*)\}$

1: $V_J = \{\mathbf{q}^g\}$
2: **for** all $\mathbf{q}_v^* \subset V^*$ **do**
3: **if** $\mathbf{q}_v^*$ is a junction node **then**
4: $V_J = V_J \cup \{\mathbf{q}_v^*\}$
5: **end if**
6: **end for**
7: $\{\mathbf{q}_j^d\}_{j=1,2,\cdots,N_d} \leftarrow$ *Clustering*(V_J,'distance')
8: $\{R_j\}_{j=1,2,\cdots,N_R} \leftarrow$ *RegionSeg*($V^*, \{\mathbf{q}_j^d\}$)
9: **for** $(j = 1; j = j + 1; j \leq N_R)$ **do**
10: $S \leftarrow \{\mathbf{q}_v^* \in V^* \cap R_j\}$
11: $\{\mathbf{q}_{j,k}^o\}_{k=1,2,\cdots} \leftarrow$ *Clustering*($S, \mathbf{q}_j^d$, 'orientation')
12: $\{B_{j,k}\}_{k=1,2,\cdots} \leftarrow$ *BranchSeg*($S, \{\mathbf{q}_{j,k}^o\}$)
13: **end for**

descendants generation by generation in $\mathcal{T}_{obs}$. The region expansion stops at the next junction. In step 4, branch segmentation is a result of clustering. Considering all $\mathbf{q}_v^*$ in the j-th region, connect $\mathbf{q}_j^d$ to $\mathbf{q}_v^*$, where $\mathbf{q}_j^d$ is the centroid of the j-th junction cluster. Branches can be obtained by clustering all these directed edges $(\mathbf{q}_j^d, \mathbf{q}_v^*)$.

The second stage 'conquer' is presented in Algorithm 5 and illustrated in Figure 3.7. A global planner moves a swarm of microrobots from region R_j to $R_{j.next}$, where region R_j and $R_{j.next}$ share an edge, and region $R_{j.next}$ is closer to the goal: $d\langle \mathbf{q}_{j.next}^d, \mathbf{q}^g \rangle < d\langle \mathbf{q}_j^d, \mathbf{q}^g \rangle$. A local planner assigns priorities to microrobots at different branches of region R_j, and leads

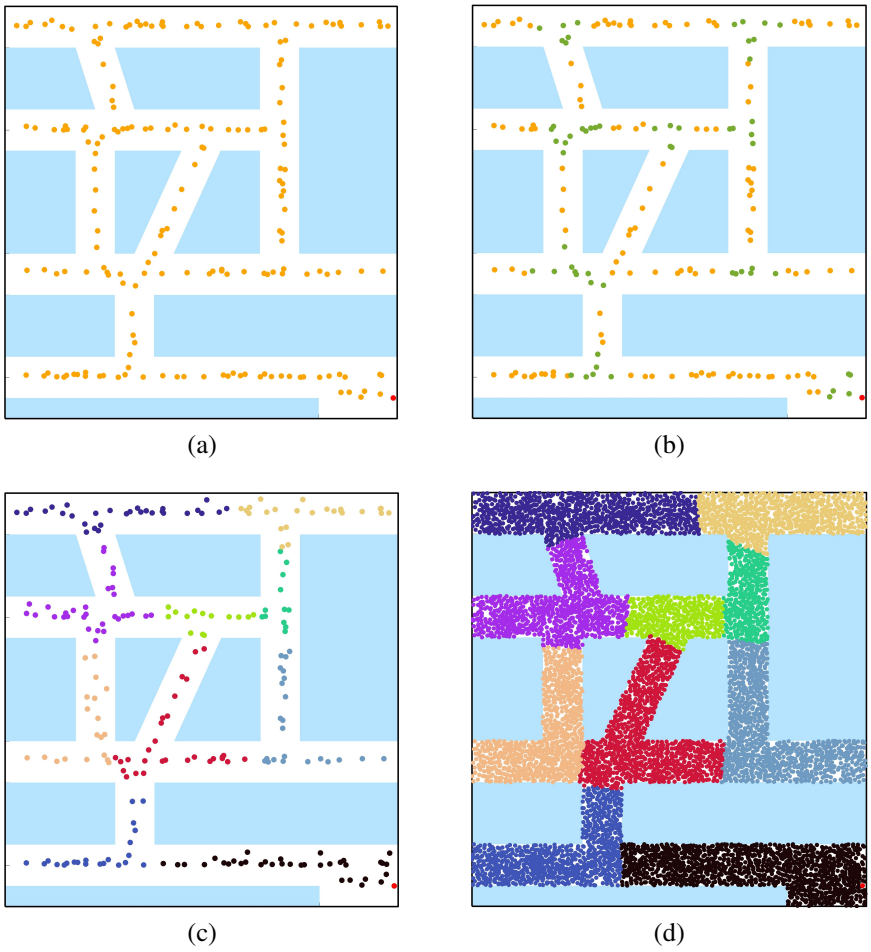


Figure 3.6: Algorithm 4 illustration. The goal (red dot) is located at (99,5). (a) represents step 1 and 2. (b) and (c) show step 3, where regions are marked as different colors. (d) illustrates step 4 in a region, where branches are marked as different colors.

them to the closer region $R_{j.next}$.

This divide-and-conquer algorithm consists of three *while* loops: (i) identify the farthest region R_j where microrobots exist (Figure 3.7(a)), and $d\langle \mathbf{q}_j^d, \mathbf{q}^g \rangle$ describes the cost from the j-th junction centroid to goal; (ii) pick a branch with the highest priority, i.e., with more robots closer to the junction centroid $\mathbf{q}_j^d$ (Figure 3.7(b)); (iii) identify the closest robot r_i to $\mathbf{q}_j^d$, and drive it to the nearest vertex $\mathbf{q}_v^{r_i} \in V$, and then move it towards some vertex in $R_{j.next}$ (Figure 3.7(b) and (c)). After moving all robots in R_j to $R_{j.next}$, a discrete region transition is completed.

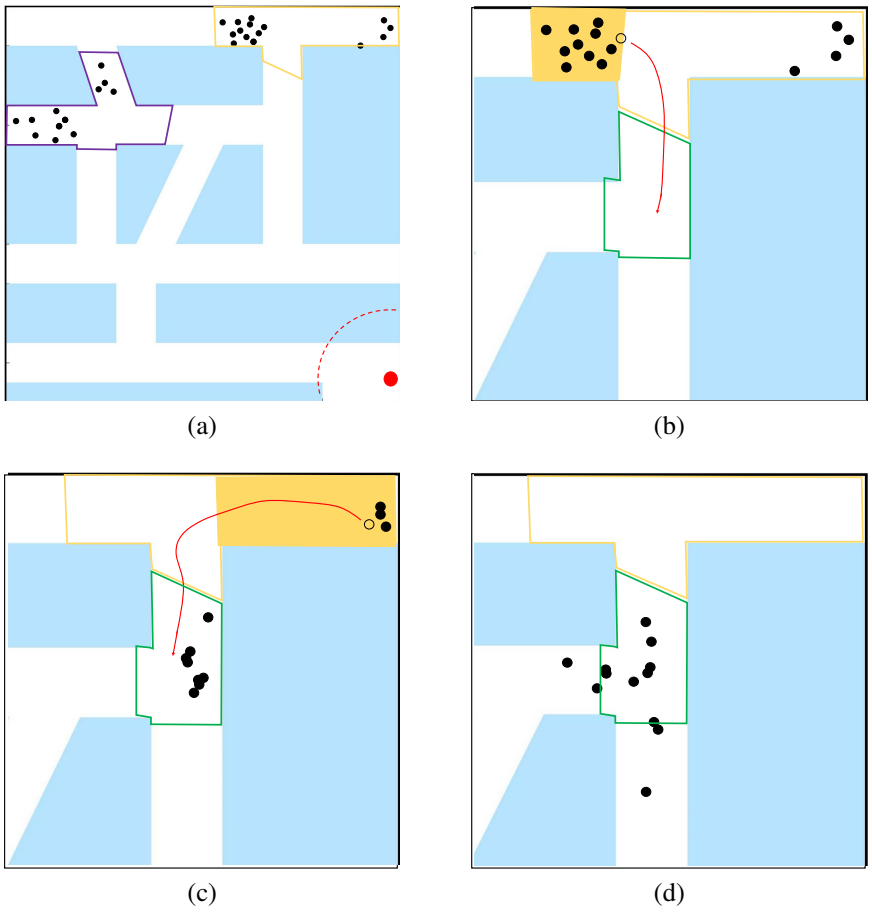


Figure 3.7: (a) Microrobots (black dots) exist in two regions (purple R_i and orange R_j). R_j is farther from the goal (red dot). (b) & (c) A control input drives a robot (hollow circle) to $R_{j.next}$ (green). Repeat this until transport the swarm from R_j to $R_{j.next}$.

To analyze time complexity of the divide-and-conquer recurrence, the following assumptions are needed: (i) the map is connected and bounded, (ii) 'closed region' and 'endpoint' definitions, and (iii) aggregation time is proportional to map area and population. Let G denote a map with $\textit{Area}(G) = m$, $\textit{Population}(G) = n$, $\textit{Density}(G) = \rho = n/m$. If $T(mn)$ is the running time for map G, level 0 of recurrence is

$$T(mn) = T(\xi mn) + f((1-\xi)\rho m \cdot (1-\xi)m), \tag{3.8}$$

where $f((1-\xi)^2\rho m^2)$ denotes the aggregation time in the farthest region (R_j) where microrobots exist, with $(1-\xi)\rho m$ the population, $(1-\xi)m$ the area, and ξ a discount factor ($0 < \xi < 1$). After we move out all robots of R_j, the aggregation map shrinks.

Algorithm 5 *Divide-and-conquer Aggregation*.
Input: $\mathcal{T}_{obs} = \{V, V_{obs}, V^*\}$, $M = \{V_J, \{R_j\}, \{B_{j,k}\}\}$, positions $\{\mathbf{p}_{t_0}^{r_i}\}$ of all robots $\{r_i\}$.
Output: $\mathbf{u}_t$

while $F(n,t) > \sigma$ **do**
  $R_j \leftarrow$ the region with the highest priority
  while there exists any robot in R_j **do**
    $B_{j,k} \leftarrow$ the branch with the highest priority
    while there exist any robot in $B_{j,k}$ **do**
      $r_i \leftarrow$ the closest robot to $\mathbf{q}_j^d$
      $\mathbf{q}_v^{r_i} \leftarrow$ the nearest vertex $\in V$ to r_i
      $\mathbf{u}_t \leftarrow$ move r_i towards $R_{j.next}$ via $\mathbf{q}_v^{r_i}$
    end while
  end while
end while

It is easy to derive the recursive form for level i using two models. In the first recurrence model, it is assumed that the aggregation map shrinks with a constant discount factor ξ each time, then $\mathit{Area}(G(i)) = \xi \cdot \mathit{Area}(G(i-1)) = \xi^{i+1}m$, and

$$T(\xi^i mn) = T(\xi^{i+1}mn) + f(\xi^i(1-\xi)\rho m \cdot \xi^i(1-\xi)m), \tag{3.9}$$

where the density is assumed to be a constant in $f(\cdot)$. In fact, the density decreases with aggregation as microrobots overlap. So this assumption does not reduce the difficulty of the subproblem. The base case is $T(n) = f(n)$ with $m = 1$. If $f(x)$ is simplified with a linear model $f(x) = kx$, then

$$\begin{aligned} T(mn) &= \sum_{i=0}^{\log_{1/\xi} m} f(\xi^{2i}(1-\xi)^2\rho m^2) \\ &= k\rho m^2(1-\xi)^2 \sum_{i=0}^{\log_{1/\xi} m} \xi^{2i}. \end{aligned} \tag{3.10}$$

Assuming $\log_{1/\xi} m$ is an integer, and $m \gg \xi$, then Equation (3.10) can be simplified to

$$T(mn) = k\rho m^2 \left(\frac{2}{1+\xi} - 1\right). \tag{3.11}$$

In the second model, the map is reduced by a constant area $(1-\xi)m$ each time, then

level i has the form

$$T((1-i(1-\xi))mn) = T((1-(i+1)(1-\xi))mn) + f((1-\xi)^2\rho m^2). \tag{3.12}$$

Hence,

$$T(mn) = k\rho m^2 \sum_{i=0}^{1/(1-\xi)} (1-\xi)^2. \tag{3.13}$$

Assuming $\frac{1}{1-\xi}$ is an integer, Equation (3.13) can be written as

$$T(mn) = k\rho m^2(1-\xi)(2-\xi). \tag{3.14}$$

The performances of different discount factors ξ are shown in Figure 3.8. As ξ increases, the scaled running time decreases fast, despite some fluctuations in the second model. This means that the more map size is reduced each time, the less efficient divide-and-conquer aggregation becomes. As $\xi \to 0$, it becomes the benchmark aggregation instead. This is equivalent to decreasing the map size from m to 1 with one recurrence ($\xi = \frac{1}{m}$). For both models (Equation (3.11) and (3.14)), as $\xi \to \frac{1}{m}$, $T(mn) \to O(m^2)$; as $\xi \to 1 - \frac{k^*}{m}$, for some $k^* \in \mathbb{R}^+$, $k^* \ll m$, $T(mn) \to T(m)$. Hence, the divide-and-conquer strategy makes it possible to reduce time complexity from $T(m^2)$ to $T(m)$. Note that k^* is dependent on junctions in a map: the finer a map can be split, the smaller k^* is.

3.3 Simulation

This section reports the simulation results to evaluate path planning approaches, RRT and obstacle-weighted RRT (OWRRT), compare the divide-and-conquer aggregation (DCA) with the heuristic aggregation (HRA), and study the impact of map and swarm population. Three sets of simulations are investigated, and each set presents two algorithms for aggregation and two methods for path planning.

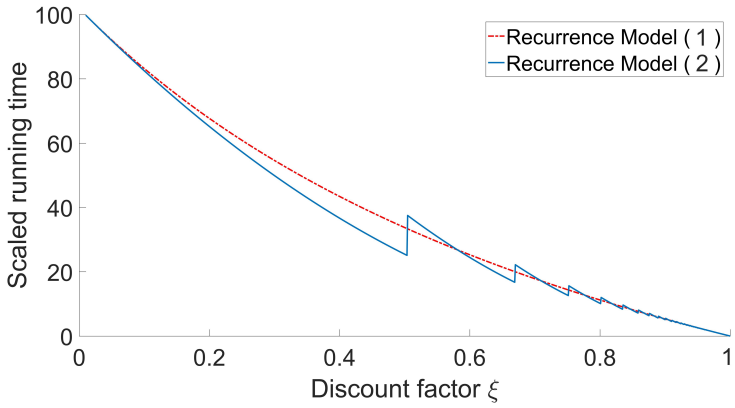


Figure 3.8: Running time estimation of the first recurrence model in Equation (3.10) and the second recurrence model in Equation (3.13).

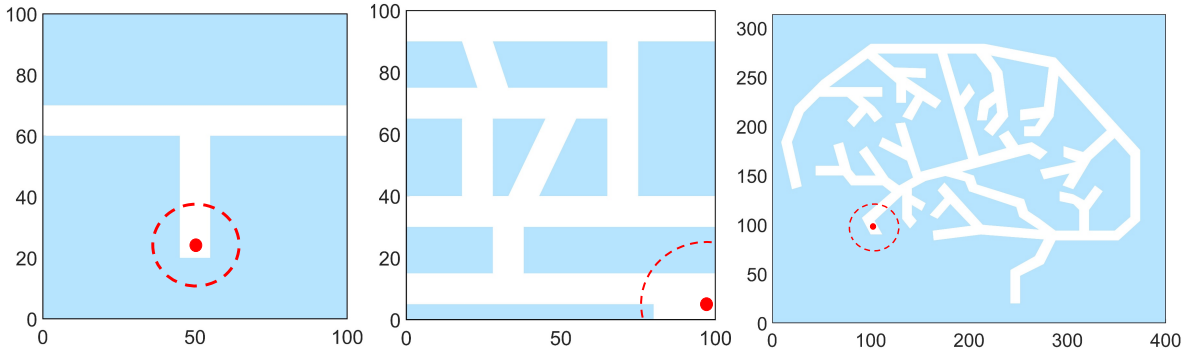


Figure 3.9: Blue polygons represent obstacles, and white channels are free space. We place a red dot at each goal location. These maps increase in size and complexity: (a) T-junction map, (b) a vascular network I and (c) vascular network II.

Path planning and aggregation are carried out in three simulated maps (Figure 3.9), including a T-junction map, and two vascular networks. The obstacles are marked as blue polygons, and free space is white. To initialize, n microrobots are randomly initialized in the free space, where $n \in \{2^1, 2^2, 2^3, \cdots, 2^{10}\}$, and each microrobot is represented by a point with no area. Given a global input $\mathbf{u}_t$ at time t, all microrobots will move towards the assigned direction for one discrete step of unit length (Equation (3.1) and (3.2)). The goal is to gather microrobots to the goal location. In practice, the task is accomplished if the average position of the swarm is near the goal, or $F(n,t) < \sigma$ (dashed circle in Figure 3.9). Count the total number of steps to approximate the running time for swarm aggregation in a map. In

each map, ten different microrobot populations are used. For each population, 30 simulations are performed for three combinations of aggregation algorithms and path planning methods respectively: DCA+OWRRT, HRA+OWRRT, and HRA+RRT. The results of simulations are compared using violin plots in Figure 3.10.

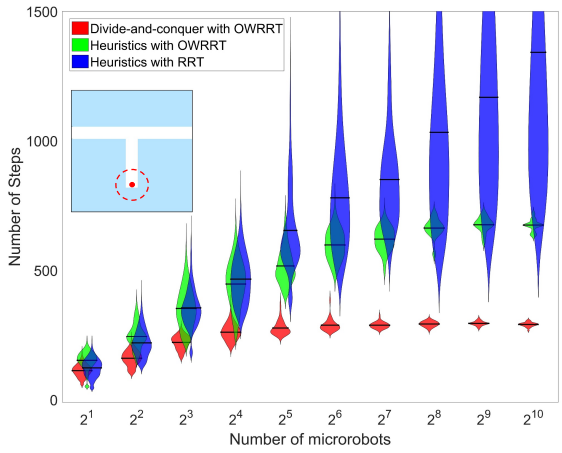
The performance of these algorithms are evaluated by their average running time (number of steps) and data distributions. DCA+OWRRT outperforms any other combinations in all these simulation when the swarm population is large enough ($n \geq 2^3$). The average aggregation time of DCA+OWRRT does not grow as fast as others, and it tends to approach an upper bound asymptotically in each vascular network. Also, this combination shows reliability and efficiency with different environments and swarm populations. For each independent trial, the aggregation time has small standard deviation. Neither HRA+OWRRT nor HRA+RRT can compete with DCA+OWRRT in average aggregation time when the swarm size is greater than 2^3. The average running time of HRA+OWRRT and HRA+RRT increases with $\log n$ in most cases with large standard deviation, and the worst case can be extremely inefficient.

3.4 Experiment

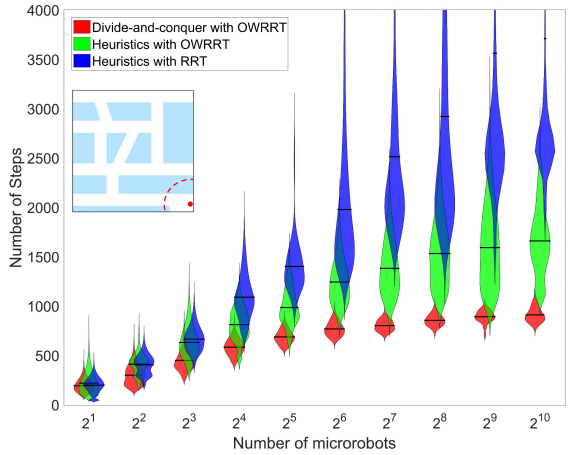
The experiments were conducted using ferromagnetic micro-particles steered by a global magnetic field generated by six electromagnets.

3.4.1 Electromagnetic Platform

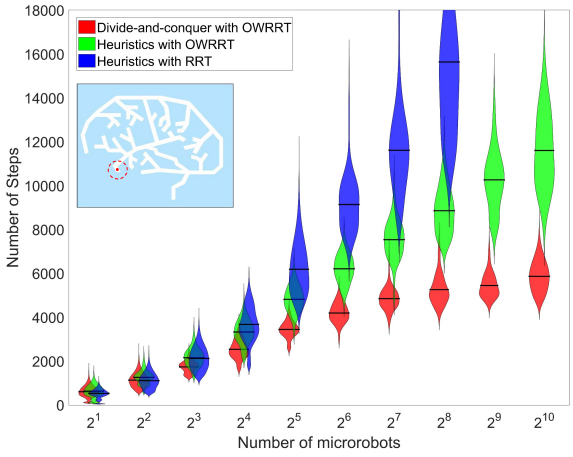
The experiment uses a custom-made electromagnetic platform which consists of three orthogonal pairs of coils with separation distance equivalent to the outer diameter of a coil (Figure 3.1a). The coils (18 AWG, Custom Coils, Inc) are powered by six SyRen10-25 motor drivers with Tekpower HY3020E DC power supply. An Arduino Mega 2560 provides six PWM signals to control the motor drives, and images are acquired using an IEEE 1394 camera (50 fps) with the region of interest approximate 20 mm^2. Each image has 379×366 pixels, and



(a)



(b)



(c)

Figure 3.10: Particle aggregation in maps (a,b,c). The violin plot shows the probability density of the simulation data and the black line indicates the mean value. We performed 30 simulations for each combination of methods and swarm populations.

each pixel represents an area of 40 μm^2 of the workspace. Microrobot detection and tracking are processed in MATLAB using blob analysis and Kalman filters, and the control input $\mathbf{u}_t$ (i.e., the orientation of the magnetic field) is sent to the Arduino Mega via USB serial port communication. In experiments, the electromagnetic platform (with iron cores) can provide over 300 Gauss magnetic fields along any direction in the 20 mm^3 workspace center.

3.4.2 Experiment Setup

The vascular network for experiments is shown in Figure 3.1 (b) and Figure 3.9 (b). This maze is made of two layers of acrylic cut using a Universal Laser Cutter, one layer as the base, and the other as the polygonal obstacles. The frame is a 20$\times$20 mm^2 square, and the channel width is 2 mm. In each experiment, the maze is filled with a mixture of microrobots and vegetable oil (0.45 mL) at the same concentration, and placed in the workspace center. The microrobots are composed of ferromagnetic particles (30 microns Fe_3O_4, Alpha Chemicals). These microparticles aggregate into microrobots that vary in sizes and shapes, with an initial population of over 300 microrobots. Microrobots align with magnetic fields when the magnitude is larger than 100 Gauss. Because the density of ferromagnetic particles is over seven times larger than water, gradient fields provided by our electromagnetic platform are not able to drag microrobots around due to friction. Hence rotational fields are deployed to make the microrobots roll along the base. Rolling a uniform field in the vertical plane at 5 Hz causes microrobots to move at an average velocity of 80 μm/s, and their maximum velocity is over 350 μm/s.

3.4.3 Validation of Aggregation Algorithms

The results of the divide-and-conquer aggregation are compared with those of the benchmark heuristic aggregation as shown in Figure 3.11. The running time is approximated by number of processed image frames for each experiment ($\approx$ 45 fps). The swarm population

is estimated by averaging the number of pixels classified as robots in last 13500 frames ($\approx$ 5 min). With similar swarm populations, the average running time for the benchmark is 93,063 frames ($\approx$ 34.5 min), and 60,628 frames ($\approx$ 22.5 min) for the divide-and-conquer algorithm, which is a reduction by 34.9%. Hence the divide-and-conquer aggregation outperforms the benchmark.

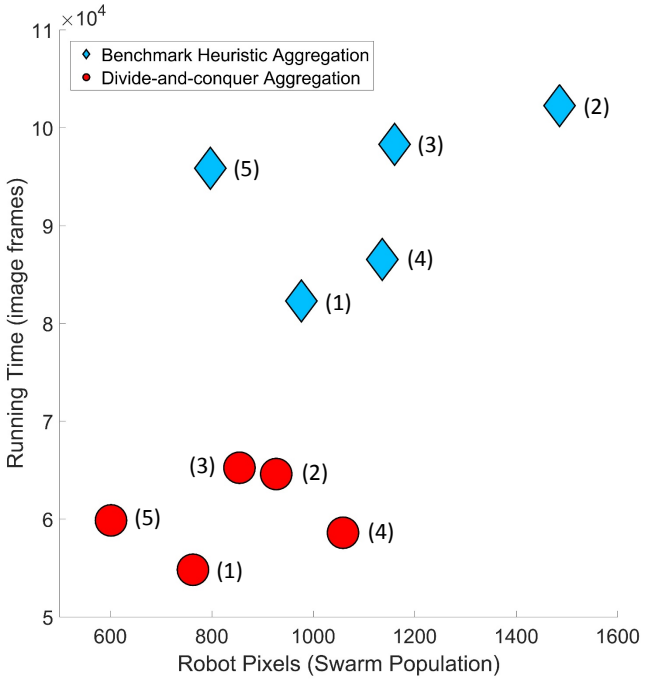


Figure 3.11: Blue diamonds are benchmark data, and red circles are results for divide-and-conquer aggregation, with an experiment number next to each marker.

3.5 Conclusion

This chapter compares two path-planning methods and two control strategies applied to the problem of aggregating microrobot swarms in vascular networks using a global input. An obstacle-weighted RRT is proposed and it steers microrobots towards near-medial-axis regions to reduce environment interference. A divide-and-conquer strategy is deployed to perform swarm-level aggregation via discrete region transitions. Compared to the benchmark strategy, the divide-and-conquer aggregation reduces the task time complexity.

Chapter 4

Path Planning Optimization Using Reinforcement Learning

4.1 Introduction

In Chapter chapter 3, online path planning and delivery methods are proposed for a homogeneous microrobot swarm, which takes as input the binary image of a vascular network for path planning, and the microrobot positions for real-time control and delivery. Compared to the benchmark algorithm, divide-and-conquer (D&C) shows considerable improvement in delivery efficiency (40%–50%). However, there are more difficulties in realism that planners need to overcome. First, D&C still is a local planning strategy and thus lack of the big picture—it only focuses on one region at a time while regardless of all others. Second, it is time-consuming to provide an analytical solution for optimal control given a highly constrained vascular network, not to mention implementation in different environments. Furthermore, the delivery targets are set near blocked outlets (endpoints), preventing microrobots from escaping for simplicity. When it comes to open outlets or branch points, both D&C and the benchmark algorithm have difficulty bringing microrobots to a target—they get trapped in local minima in such scenarios.

These challenges consist of three aspects: identical agents (homogeneous microrobots), a shared and uniform control input, and highly constrained environments. In optimal planning and control, the first two aspects belong to the area of under-actuated control problem or constraints in control input; the third aspect is categorized to unknown system dynamics as the environmental constraints cannot be easily specified in mathematical forms.

Reinforcement learning (RL) learns optimal or at least near-optimal strategies for sequential decision-making problems. The learned strategies are iterated via errors and trials

when interacting with the environment. RL agents get feedback (rewards) from the environment and try to maximize the cumulative rewards for an optimal solution. RL is a promising approach to study the optimal control problem with constraints and unknown system models, and to overcome the local minima dilemma in planning.

4.1.1 Reinforcement Learning

Reinforcement learning (RL) is an aspect of machine learning that studies the optimization of sequential decision-making problems [64]. An RL agent learns to map sensory input to action space to maximize a reward signal from the environment. Unlike supervised or unsupervised learning, the training data in RL are collected via interacting with the environment, there are no instructions (labels) but delayed rewards to tell agents which action to take, and the objective cannot be set as minimizing the difference between predictions and labels.

RL has a broader research scope and the capability of generalization to non-traditional control problems. For example, the optimal control theory generally assumes that a system model is well-defined and agent behavior can be predicted without interactions, each agent has an individual controller for planning, and the whole system is controllable under the proposed control law. These prerequisites are not satisfied in the scenario of homogeneous microrobot swarm path planning.

Well-established theories of path planning, such as breadth-first search, A* and rapidly random exploring tree (RRT), can provide both efficient and optimal solutions to single-robot navigation in a complex environment. However, homogeneous microrobot swarm path planning with a shared and uniform control input can cause problems, as fore-mentioned algorithms fail to see the whole swarm—only one robot is handled at a time. Hence the efficiency decreases significantly compared to that of a single-robot case.

RL offers a general abstraction for control problems as a Markov decision process, and formulates the problem as agents optimize cumulative rewards by making decision based on

the observation in an environment. Generalizable, globally optimal, and model-free are three significant advantages of RL in the case of homogeneous microrobot swarm navigation. RL agents use the trial-and-error experience to modify their learned skills and thus do not require lots of pre-existing knowledge to provide suitable solutions. The model-free property makes RL adapt to different scenarios easily without changing the algorithm.

4.1.2 Deep Learning

Deep learning is another aspect of machine learning that employs deep neural networks (number of layers$\geq$3) to learn high-level features from raw data (e.g., image, sound, text, etc.) and to make prediction on classification and regression problems. Recent development in deep learning has led to significant development in areas of speech recognition, image classification, object detection, semantic segmentation, text generation, etc. [65]. Contrast to conventional machine learning techniques, which heavily rely on hand-crafted features and domain knowledge to process raw data, deep learning approaches allow end-to-end learning from raw data directly to training objectives, automatically extracting abstractive representations from high-dimensional data.

4.1.3 Deep Reinforcement Learning

Deep reinforcement learning (DRL) makes use of deep learning techniques for abstractive state representations and nonlinear function approximations in RL. This enables RL to learn from high-dimensional sensory input without artificial feature engineering, and automatically generate complex control laws to direct agents. The recent successes of DRL include mastering of the game of Go [66], playing Atari 2600 games from pixel input [67, 68], and learning humanoid parkour and flexible behaviors [69–71].

4.2 Background

4.2.1 Markov Decision Process

A Markov decision process (MDP) provides the mathematical framework to describe agent-environment interactions and to model the decision-making process in RL. An *agent* is defined as a decision maker which learns to generate actions according to observations from an environment. The *environment* is a set of objects outside the agent. An MDP defines a stochastic control problem composing of five-tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where

- $\mathcal{S}$: a finite set of states (observations);
- $\mathcal{A}$: a finite set of actions (continuous or discrete);
- $\mathcal{P}$: $P^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$, is a state transition probability matrix, where $s', s \in \mathcal{S}, a \in A$;
- $\mathcal{R}$: $\mathcal{R}^a_s = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$, is a scaler reward function provided by the environment;
- γ: $\gamma \in [0, 1]$ is a scaling factor that discounts the future rewards.

At each time step, the agent take an action $a \in \mathcal{A}$ with respect to its current state $s \in \mathcal{S}$, the environment dynamics transition the agent to the next state $s' \in \mathcal{S}$ and provide a reward r for reaching the state. A sequential decision-making process is Markov if its state at time step $t+1$ only depends on its state and action at time step t,

$$\mathbb{P}[S_{t+1} = s_{t+1} | S_t = s_t, A_t = a_t] = \mathbb{P}[S_{t+1} | s_0, a_0, s_1, a_1, \ldots, s_t, a_t]. \tag{4.1}$$

4.2.2 Returns, Policy and Value Functions

RL agents try to maximize the cumulative rewards from the environment, where the rewards of a sequence of states are $R_{t+1}, R_{t+2}, R_{t+3}, \ldots, R_T$, the *return* G_t is defined as the

weighted sum of rewards from time step t,

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_T = \sum_{k=0}^{T} \gamma^k R_{t+k+1}, \tag{4.2}$$

where T denotes a final time step. If $T = \infty$, the task never ends; if T is finite, the task ends after T steps, followed by a reset to a starting point. A finite T defines the total time steps of an episodic task.

The *policy* is a mapping from a state to a probability distribution of actions,

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]. \tag{4.3}$$

The policy describes the behavior model of agents, and MDP policies only depend on the current state instead of the historical data.

The *value function* $v_\pi(s)$ denotes an estimation of the return from a state to evaluate how goodness the state is

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]. \end{aligned} \tag{4.4}$$

The value function can be decomposed into immediate reward R_{t+1} and the discounted return of successor state.

The *action-value function* $q_\pi(s, a)$, also known as Q value function, defined as the estimated return by taking action a at state s, to evaluate how goodness an action-state pair is

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a]. \end{aligned} \tag{4.5}$$

Again, the action-value function can be decomposed into the current reward by taking action a, and the discounted successor action value $q_\pi(S_{t+1}, A_{t+1})$.

Solving an MDP problem is equivalent to finding an optimal policy that maximizes the value function/action-value function. The optimal value function $v_*(s)$ and the optimal action-value function $q_*(s,a)$ are defined as

$$v_*(s) = \max_{\pi} v_\pi(s) \text{ and} \tag{4.6}$$

$$q_*(s,a) = \max_{\pi} q_\pi(s,a). \tag{4.7}$$

Define a partial ordering over policies, such that $\pi(s) \geq \pi'(s)$ if and only if $v(s) \geq v'(s), \forall s \in \mathcal{S}$. For any MDP, there exists at least one optimal policy π_* that is better or equal than all others, and such policy achieves the optimal value function/action-value function.

4.2.3 Problem Formulation

Given a swarm of simple homogeneous (identical) point robots distributed in a 2D vascular network, they receive a shared, global control input and thus head the same direction. The goal is to deliver a certain amount of microrobots to a target area with as few steps as possible. Specifically, consider the standard episodic task, RL agents use discrete actions to steer microrobots interacting with an environment for a finite number of steps. The environment is reset once RL agents have reach the maximum number of steps. The goal is to find a mapping from states to actions, such that cumulative rewards from the environment are maximized. Here, the set of actions are defined as eight discrete directions: $N(\uparrow), NE(\nearrow), E(\rightarrow), SE(\searrow), S(\downarrow), SW(\swarrow), W(\leftarrow), NW(\nwarrow)$. The states are defined as sensory inputs such as microrobot positions or an image sequence.

4.3 Related Work

Model-free control is aimed at solving RL problems that are either with unknown system model, but the data can be sampled from interaction, or with known dynamics, but too complicated to employ except by samples. Both *on-policy* and *off-policy learning* methods are widely

investigated in model-free RL research. The on-policy learning optimizes a policy based on state-action samples drawn from the most recent policy, i.e., a near-optimal but not optimal policy. While the off-policy learning periodically updates an outdated target policy based on experience sampled from a behavior policy where the samples are 'off' the target policy [64].

4.3.1 Q-Learning

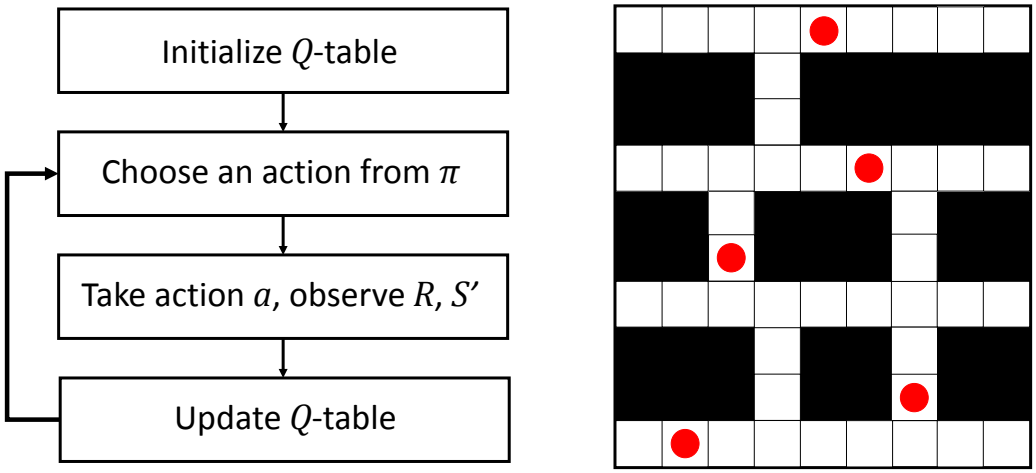
Q-learning [72] is an off-policy learning method which approximates the optimal action-value function $q_*(s, a)$ by a learned function Q, and the policy is indirectly derived from the Q function. The Q function is defined as

$$Q(s, a) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right], \tag{4.8}$$

where α is the learning rate, γ is the discount factor, and the action a is sampled from $\mathcal{A}$.

Tabular Q-Learning

Tabular Q-learning is deployed in cases of small, finite state spaces, where Q function is approximated using vectors or tables. The algorithm is briefly described in Figure 4.1a. For example, consider a 9×10 maze with 46 obstacle-free grids shown in Figure 4.1b, and a total of N homogeneous point robots are randomly distributed in the free space. To steer all these robots to a goal location only with a global control input, a Q-table and a value function table are constructed as a $m \times n$ matrix and a $m \times 1$ vector respectively, where m is the number of states, and $n = 4$ is the action space $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$. The required memory is shown in Table 4.1 given all variables are saved in *float64* format (8 bytes)—simulating hundreds of robots is beyond the capability of tabular Q learning.



(a) Tabular Q-learning.

(b) Q-learning example.

Figure 4.1: The Q-learning example is a 9×10 maze with white grids as pathways and black grids as obstacles. Point robots (red) move in pathways one grid per step, subjecting to a global control input.

Table 4.1: Memory required for tabular Q learning.

Number of robots	Required memory (GB)
7	5.76
8	31.52
9	148.86
10	10615.06

Deep Q-Network

To solve MDP with a large state space, function approximation is employed to estimate Q function, and such approximation makes it possible to generalize to unseen states. However, training RL agents with a nonlinear function approximator such as the neural network causes instability and divergence, as they are not guaranteed to converge.

A recent breakthrough deep Q-network (DQN) proposed by Mnih et al. achieves human-level or above control in classic Atari video games, such as Pong, Breakout, and Boxing [67, 68]. DQN learns policies directly from pixels and scores and output actions—such end-to-end reinforcement learning only requires minimal domain knowledge and thus the network architecture can be applied to a wide range of scenarios.

The success of DQN in RL relies on the following key elements: convolutional neural network (CNN), experience replay, and the target network. CNN captures features in the high-dimensional space (images) and maps them to a latent vector space, which reduces data dimension and learns an abstractive representation for similar features. Experience replay provides a replay buffer that collects a million (or similar) MDP transitions from agent-environment interactions. At the training stage, a mini batch of samples (e.g., 32 transitions) are randomly sampled from the replay buffer to update the network. Such randomness in samples removes correlation in observation sequences and thus brings the data distribution closer to identically independently distributed (i.i.d.) assumption. DQN keeps two action-value functions, Q and $\hat{Q}$ for the network, parameterized by θ and θ^- respectively. The agent behaviors are generated by the first network Q, and training targets are provided by the second network $\hat{Q}$. At the training, only θ gets updated, and θ^- synchronize its weight with θ periodically. Such fixed Q-value targets reduce data correlations and improve training stability and convergence.

Afterward, there are many succeeding innovations based on deep Q-network, such as double DQN [73], DQN with prioritized experience replay [74], dueling network structure in DQN [75], and rainbow DQN [76].

4.3.2 Policy Gradient

Policy gradient is an on-policy RL algorithm which directly maps a state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}$. Unlike value-based RL, where policy is indirectly generated, for example, via ϵ-greedy action selection, policy-based RL approximates the policy directly, and it has better convergence and is effective in a high-dimensional or continuous action space. Let $\pi_\theta(a|s)$ be a policy function parameterized by θ, and the objective is to find the best policy that maximizes

the cumulative returns,

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right] \\ &= \sum_{t=0}^{\infty}\sum_{a}\pi_\theta(a|s_t)q_{\pi_\theta}(s_t,a), \end{aligned} \tag{4.9}$$

with $r(\tau)$ the return at time τ. Hence, the policy gradient is

$$\begin{aligned} \nabla J(\theta) &= \sum_{t=0}^{\infty}\sum_{a}\nabla_\theta\pi_\theta(a|s_t)q_{\pi_\theta}(s_t,a) \\ &= \sum_{t=0}^{\infty}\sum_{a}\pi_\theta(a|s_t)\nabla_\theta \log \pi_\theta(a|s_t)q_{\pi_\theta}(s_t,a) \\ &= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty}\nabla_\theta \log \pi_\theta(a|s_t)q_{\pi_\theta}(s_t,a)\right] \\ &= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty}\nabla_\theta \log \pi_\theta(a|s_t)G_t\right], \end{aligned} \tag{4.10}$$

where G_t is defined in Equation (4.2), and this gradient is used to update θ,

$$\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta), \tag{4.11}$$

with α the learning rate.

Baseline Function

Despite the better convergence property, policy gradient may suffer from high variance which results in slow learning [64]. To reduce the variance, a state-dependent baseline function $b(s)$ is subtracted from the objective. The baseline function should not be related to the policy

parameters θ, and thus the gradient becomes

$$\begin{aligned}
\nabla J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty}\nabla_\theta \log \pi_\theta(a|s_t)(G_t - b(s_t)) + \sum_{t=0}^{\infty}\nabla_\theta \log \pi_\theta(a|s_t) b(s_t)\right] \\
&= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty}\nabla_\theta \log \pi_\theta(a|s_t)(G_t - b(s_t))\right] + \sum_{t=0}^{\infty}\sum_{a}\nabla_\theta \pi_\theta(a|s_t) b(s_t) \\
&= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty}\nabla_\theta \log \pi_\theta(a|s_t)(G_t - b(s_t))\right] + \sum_{t=0}^{\infty} b(s_t)\nabla_\theta \sum_{a}\pi_\theta(a|s_t) \\
&= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty}\nabla_\theta \log \pi_\theta(a|s_t)(G_t - b(s_t))\right],
\end{aligned} \tag{4.12}$$

because $\sum_a \pi_\theta(a|s_t) = 1$ and $\nabla_\theta 1 = 0$. Without loss of generality, the value function $v(s)$ can be chosen as the baseline function, and the advantage function $A(s,a)$ is defined as

$$A_\pi(s,a) = q_\pi(s,a) - v_\pi(s), \tag{4.13}$$

so the gradient is rewritten as

$$\nabla J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty}\nabla_\theta \log \pi_\theta(a|s_t) A_\pi(s_t, a_t))\right]. \tag{4.14}$$

Actor-Critic Methods

Actor-critic methods take advantage of both value function and policy gradient methods, where the *critic* and *actor* are deployed to describe the performance of them. The actor-critic RL further reduces the variance and makes learning faster via bootstrapping [64]. The loss function of actor-critic methods consists of the policy loss $\mathcal{L}_\pi$ and the value function loss $\mathcal{L}_v$,

$$\mathcal{L} = \mathcal{L}_\pi + \mathcal{L}_v, \tag{4.15}$$

where

$$\mathcal{L}_\pi = -\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)(R_t + \gamma v_w(s_{t+1}) - v_w(s_t)), \tag{4.16}$$

and

$$\mathcal{L}_v = \frac{1}{2}\sum_{t=0}^{T}\|R_t + \gamma v_w(s_{t+1}) - v_w(s_t)\|^2, \tag{4.17}$$

with T the maximum number of steps taken for each update. Instead of taking all returns into account, actor-critic methods bootstrap the value with T time steps. Here the critic is used to update parameters w of the value function $v_w(s)$, and the actor is used to update parameters θ of the policy $\pi_\theta(a|s)$ in the direction indicated by the critic.

4.4 Methodology

In Chapter 3, a few examples of vascular networks are given to show how to steer homogeneous microrobots using online path planning and control algorithms, where the delivery strategies are planned locally and the controllers only track one robot at each step. Here with deep reinforcement learning (deep RL), the entire image is taken into account, that is, the microrobots and the environment. This large sensory space brings the first challenge: the curse of dimensionality. For example, given $N = 100$ microrobots randomly distributed in a 50×50 obstacle-free pixel space within a 100×100 image, with each microrobot occupies a pixel, the number of the possible states is $\sum_{n=1}^{100}\binom{2500}{n}$, approximately 10^{181} states. Random exploration in such a vast state space is unlikely, or least efficient to bring all microrobots to a target.

Another challenge that needs addressing is the sparse reward problem with the delivery task. Unlike supervised learning where all data are labeled, RL agents have to deal with long-horizon tasks when exploring various states and only get rewarded for important achievements, but not for every action. This problem occurs because the positive rewards for meaningful tasks are provided at a much larger time-scale than that of an RL agent operate at, e.g., every step the agent choose a direction to steer the microrobot swarm, while the outcome of each action cannot be justified until it finishes the delivery task. In Breakout, Pong, and similar Atari games, the environment produces positive rewards every a few actions. Contrast to Atari

games, an RL agent takes every 20 actions or more to get a positive reward for a simple delivery task within a vascular network, even if it follows an optimal policy.

To overcome the curse of dimensionality and sparse rewards, curiosity-driven mechanism and reward shaping are introduced in this section. First, advantage actor-critic (A2C) RL framework with proximal policy optimization (PPO) provides the benchmark results for microrobot swarm delivery; then RL auxiliary tasks for motivating exploration are presented and compared with the benchmark, including intrinsic curiosity mechanism (ICM) and random network distillation (RND); third, a general reward function is proposed for the delivery task to accelerate learning.

4.4.1 Reinforcement Learning Framework

Advantage Actor-critic (A2C)

Synchronous Advantage Actor-Critic (A2C) method and proximal policy optimization (PPO) are deployed as the benchmark RL algorithm for microrobot swarm delivery task. A2C is a synchronous variant of Asynchronous Advantage Actor-Critic (A3C) [77], and both A2C and A3C employ multiple agents to collect experience. While A3C agents keep their copies of environment and network weights running in parallel, A2C agents synchronize their experience and weights once all of them finish their playing for each update. In this chapter, an OpenAI implementation of A2C is used (code available on GitHub [78]). According to OpenAI, A2C is more cost-effective than A3C on machines with a single GPU [79].

As a variant of actor-critic method, there is a small difference in the loss function between A2C and Equation (4.15), (4.16), and (4.17),

$$\mathcal{L}_\pi = -\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)(R_t + \gamma v_w(s_{t+1}) - v_w(s_t)) - \eta \mathcal{H}(\pi_\theta(a_t|s_t)), \tag{4.18}$$

where $\eta = 0.001$ is the entropy coefficient, and the entropy term $\mathcal{H}$ is added to encourage agents for exploration and to bootstrap from local minima.

Algorithm 6 *Synchronous Advantage Actor-Critic (A2C).*

Initialize N parallel environments
Initialize network parameters θ and w
Number of rollouts $n = 0$, max number of rollouts $n_{\text{max}} = 1e9$
Number of rollouts per update $n_{seg} = 2048$
while $n \leq n_{\text{max}}$ **do**
    Collect n_{seg} rollouts for each agent
    Calculate n_{seg}-step returns and advantages
    Calculate the loss $\mathcal{L}_{A2C}$ and the gradient with respect to θ and w
    Update network weights with $n_{seg}N$ rollouts
    $n = n + 1$
end while

Many optimization methods can be used to update the actor part, such as natural policy gradient, trust-region policy optimization (TRPO) [80], and proximal policy optimization (PPO) [81]. The objective is to maximize the advantage function defined as

$$\begin{aligned} &\max_{\theta} \ \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right], \text{ such that} \\ &\hat{\mathbb{E}}_t \left[\text{KL} \left[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t) \right] \leq \delta \right], \end{aligned} \tag{4.19}$$

where π_θ is the current policy being updated, π_{old} is used to interact with the environment and to collect rollouts, and the KL-divergence measures the differences in these policies.

In general, natural policy gradient is sensitive to step size, as too large size would be challenging to train and too small size leads to slow progress. Furthermore, the natural policy gradient does poor in sampling efficiency, which requires too many interactions to get updates. The idea of TRPO is that by constraining each update of the policy within a trust region, not too far away from previous policy, it brings more robust in training, less dependent on hyperparameter tuning, and better efficiency in sampling. TRPO takes KL-divergence into consideration and solves the optimization problem using conjugate gradients. Still, the implementation of TRPO with conjugate gradients is more complicated and less cost-effective compared to stochastic gradient descent (SGD). Practically, TRPO does not work well on deep convolutional network training.

PPO offers similar performance as TRPO does, but is compatible with first-order optimizers such as SGD. The clipped objective of PPO is defined as

$$\mathcal{L}_{\pi_\theta} = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t^{\pi_{\text{old}}}, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t^{\pi_{\text{old}}}) \right], \tag{4.20}$$

with the ratio $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{\text{old}}}(a_t|s_t)$ measuring the changes in the policy. Typically, policy π_{old} synchronizes with policy π_θ every four updates to prevent the difference from getting too large, and the clip portion is set to $\epsilon = 0.1$ in this thesis. The clipped objective stabilizes the training by preventing the policy changing too much of its gradient direction, which achieves similar performance as using KL-divergence metric, while largely simplifying the computation complexity because the gradients now can be approximated with first-order terms only.

Intrinsic Curiosity Mechanism (ICM)

RL agents using A2C and PPO can deliver microrobot swarm to a target in vascular networks, but limited to simple cases. As the complexity increases, it takes A2C long time to converge or even fails to complete the delivery because the state space grows to a scale that A2C can barely handle given the sparse reward function. So it is possible to produce intrinsic reward in addition to the extrinsic reward from the environment, such that RL agents are motivated to travel towards unvisited states? Pathak et al, [82], Burda et al [83] introduce intrinsic curiosity mechanism (ICM) as an auxiliary task to solve the problem of insufficient exploration.

The idea of ICM is illustrated in Figure 4.2. Within the ICM module, the RL agent predicts the next state in a feature space, given the current state and action; the intrinsic reward R_t^i is defined as the Euclidean distance between the predicted state and the actual state in the feature space,

$$R_t^i = \frac{1}{2}\|\hat{\phi}(s_{t+1})) - \phi(s_{t+1})\|^2. \tag{4.21}$$

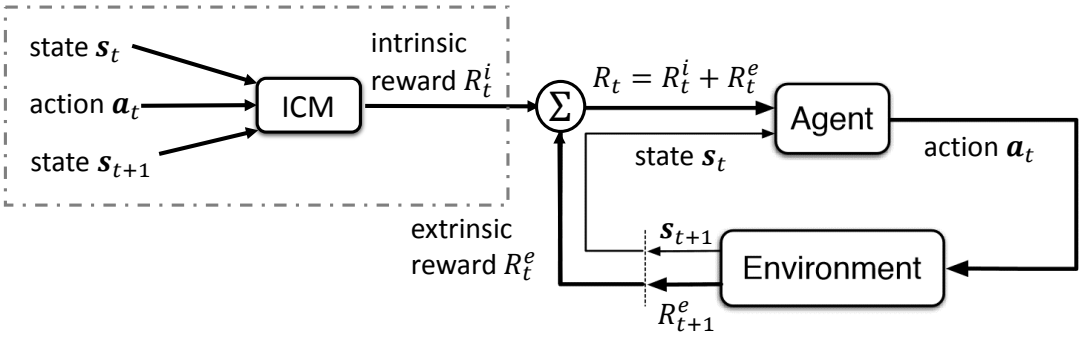


Figure 4.2: Illustration of intrinsic curiosity mechanism (ICM).

Hence the total rewards are the summation of intrinsic and extrinsic rewards,

$$R_t = R_t^i + R_t^e. \tag{4.22}$$

In other words, when the RL agent visits some unfamiliar states, the prediction error is large and thus the intrinsic reward encourages the RL agent to visit this state over-and-over again, till it gets more familiar with the transition and produces fewer errors.

Specifically, ICM includes a forward dynamic model and an inverse dynamic model. The forward model provides the prediction loss to the RL agent as intrinsic rewards. First, the state s_t is mapped from pixel space to a feature space $\phi(s_t)$ using a deep neural network. Then the RL agent learns an encoding that takes inputs $\phi(s_t)$ and the action a_t and projects them to the feature space,

$$\hat{\phi}(s_{t+1}) = f_{\theta_f}(\phi(s_t), a_t), \tag{4.23}$$

where the neural network is parameterized by θ_f. The corresponding auxiliary loss function is denoted as

$$\mathcal{L}_{\text{fwd}}^{\theta_f} = \frac{1}{2}\|\hat{\phi}(s_{t+1})) - \phi(s_{t+1})\|^2. \tag{4.24}$$

The inverse model learns a function $g_{\theta_i}(\cdot)$ that predicts the action which transitions state s_t to s_{t+1},

$$\hat{a}_t = g_{\theta_i}(\phi(s_t), \phi(s_{t+1})), \tag{4.25}$$

where $\hat{a}_t$ is the estimation of the action a_t, and the network is parameterized by θ_i. The corresponding auxiliary loss function is defined as the cross-entropy between the prediction $\hat{a}_t$ and the label a_t,

$$\mathcal{L}_{inv}^{\theta_i} = -\sum_{i=1}^{\dim(\mathcal{A})} a_t \log(\hat{a}_t), \tag{4.26}$$

where a_t is one-hot encoded defined in a discrete action space $\mathcal{A}$, and $\hat{a}_t$ is a vector of the same dimension after soft-max operation. Therefore the loss function for ICM is denoted as

$$\begin{aligned}
\mathcal{L} &= \mathcal{L}_\pi + \mathcal{L}_v + \beta_1 \mathcal{L}_{fwd} + \beta_2 \mathcal{L}_{inv} \\
&= \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t^{\pi_{\text{old}}} \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t^{\pi_{\text{old}}}) \right] \\
&- \eta \mathcal{H}(\pi_\theta(a_t|s_t)) + \frac{1}{2}\sum_{t=0}^{T} \|R_t + \gamma v_w(s_{t+1}) - v_w(s_t)\|^2 \\
&+ \beta_1 \frac{1}{2} \|\hat{\phi}(s_{t+1})) - \phi(s_{t+1})\|^2 - \beta_2 \sum_{i=1}^{\dim(\mathcal{A})} a_t \log(\hat{a}_t).
\end{aligned} \tag{4.27}$$

Random Network Distillation (RND)

Random network distillation (RND) shares a similar idea with ICM, relying on self-supervised learning to motivate the exploration towards infrequently visited states [84]. Burda et al. pointed that ICM might potentially suffer from incomplete feature representation, and the learned features are unstable as the data distribution changes along with the evolution of the target network [83]. RND selects a constant feature embedding by fixing the target network once initialized, and thus the target features are stable to learn. Still, RND cannot guarantee the completeness of the feature space.

Specifically, the target network maps the state into a feature space $\psi_{\theta^*}(s_t)$, with a neural network parameterized by θ^* with fixed, randomized weights. Hence the mapping of any state is constant and unique throughout the whole learning. The prediction network learns mapping

from a state to a feature space $\hat{\psi}_\theta(s_t)$ of the same dimension as the ψ_{θ^*}, and the objective is to minimize the Euclidean distance between the predicted feature and the target,

$$\mathcal{L}_{\text{aux}} = \|\psi_\theta(s_t) - \psi_{\theta^*}(s_t)\|^2. \tag{4.28}$$

Similar to ICM, the agent learns to predict more accurately given a frequently visited state, and once it reaches a new state, the resultant prediction error gives a large reward to encourage the agent to reach this state again. Unlike ICM, RND tries to predict a state in the feature space $\psi_\theta(s_{t+1})$ based on the state s_{t+1} itself, rather than including information of the previous state s_t and the action a_t. The RND loss function resembles Equation (4.27), except for the auxiliary losses $\mathcal{L}_{fwd}$ and $\mathcal{L}_{inv}$ replaced by $\mathcal{L}_{\text{aux}}$ in Equation (4.28).

Implementation Details

The data preprocessing and hyperparameter settings in this dissertation are slightly different from the original code on GitHub [85]. The environment wrapper is similar to those of Atari games: first, apply sticky actions and max pooling, and then resize the gray-scale image into a 84×84 format, and feed the network with a stack of four successive frames which are normalized with states mean and standard deviation from 10,000 random samples.

The network architecture remains unmodified as shown in Figure 4.3. The feature extraction uses four convolutional layers with 32 ($8 \times 8, s = 4$), 64 ($4 \times 4, s = 2$), and 64 ($2 \times 2, s = 1$) filters, respectively. The output of each layer is activated by a leaky rectified linear unit (Leaky ReLU). After flattening, the output of the last convolutional layer is mapped to the policy (dimension$= 4$ or 8, depending on available actions) via a fully connected layer (512 units). The value function is also mapped from the last convolutional layer, and the output dimension is 1.

A2C employs 128 parallel agents with different microrobot distributions to collect experience. The learning rate is set to 0.0001. Each agent collects 2048 rollouts (steps) before the four-epoch update in network weights. During each update, the mini batch size is set to 32.

The clip range of PPO is [-0.9, 1.1]. Other changes in hyperparameters are listed in Table 4.2.

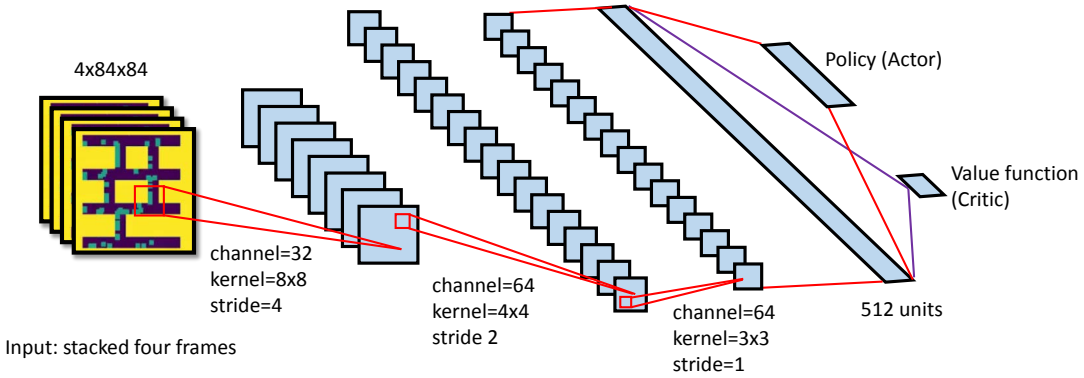


Figure 4.3: Illustration of A2C neural network architecture.

Table 4.2: Hyperparameter table for A2C framework and ICM/RND.

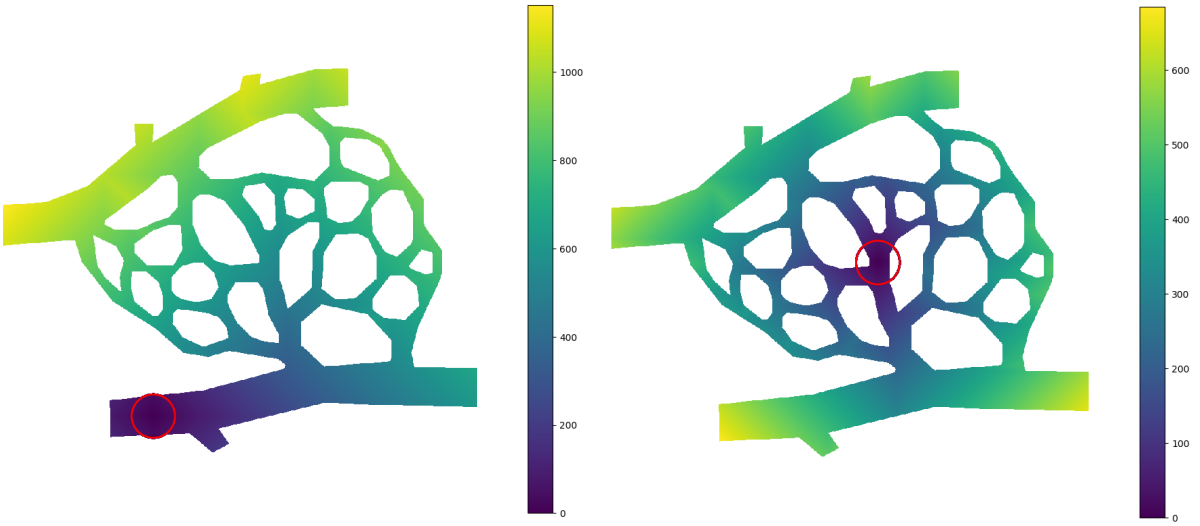
Hyperparamter/Operation	Default value
Extrinsic reward clipping	False
Extrinsic reward normalization	False
Intrinsic reward clipping	False
Intrinsic reward normalization	True
Max frames per episode	environment dependent
Stacked frame normalization	$x \rightarrow (x - \mu)/\sigma$
Rollout length	2048
Total number of rollouts	∞
Number of mini batches	16
Number of optimization epochs	4
Coefficient of extrinsic reward	1.0
Coefficient of intrinsic reward	0.5
Number of environments	128
Learning rate	0.0001
Entropy coefficient	0.001
Clip range	[0.9, 1.1]

4.4.2 Reward Shaping

The reward function design plays a key role in successful RL training [64]. Although the RL agent does not require the instructional information during learning, a well-designed reward signal helps divide a task into sub-goals and produces intermediate rewards for meaningful accomplishment at each stage. The reward signal leads the agent to the ultimate target and

reduces the time spent on less relevant states. The choice of subgoals can be tricky, as too many intermediate rewards may not necessarily benefit the learning process, but mislead the agent to sit on these rewards and to deviate from the target.

In general, the environment produces a reward at some milestone state/observation directly leading to the target, $f_{rew} : \mathcal{S} \rightarrow \mathcal{R}$. This works well for many single-robot environments, as one state can be easily distinguished from the other using coordination, orientation, or similar sensory information. However, the state of a swarm of N microrobot, which should include all microrobot coordinate information, does not explicitly show the 'milestone' property to the human as the state is in a high dimensional space and beyond our common sense. In other words, the sub-goals can be hard to describe if only the sensory information is given.



(a) (b)

Figure 4.4: Cost-to-go map with the target region around (a) [130, 61] and (b) [75, 100], shown as red circles.

This dissertation provides guidance for reward signal design, especially for environments with high-dimensional states. Starting from a single-robot case, the state space $\mathcal{S}$ is mapped to scalar function, such as a cost-to-go function $j : \mathcal{S} \rightarrow \mathcal{J}$, where $\mathcal{J} \subset \mathbb{R}$. Next, collecting cost-to-go functions from the swarm, a mapping to the reward can be constructed based on the mean and maximum of the collection: $f_{rew} : \mathcal{J} \rightarrow \mathcal{R}$.

Considering a vascular network shown in Figure 4.4a and 4.4b, where the 'viridis' color regions are obstacle-free pathways. Two targets are selected, and the breadth-first search is used to assign a cost to each pixel coordinate in pathways based on the Manhattan distance. The corresponding cost-to-go maps are visualized in Figure 4.4a and 4.4b, respectively. Let u, v be the average cost and the maximum cost to the target of all microrobots. If the task is to deliver the swarm to a target region such that $v < 10$, a discrete reward function can be designed as Table 4.3. Note that each reward can only be granted once until the task completed or environment reset when exceeding the time limit.

Table 4.3: Reward function design example.

(a)

Average cost u	Reward	Completed
< 10	8	False
< 20	8	False
< 40	4	False
< 80	4	False
< 120	2	False
< 160	2	False
$\vdots$	$\vdots$	$\vdots$

(b)

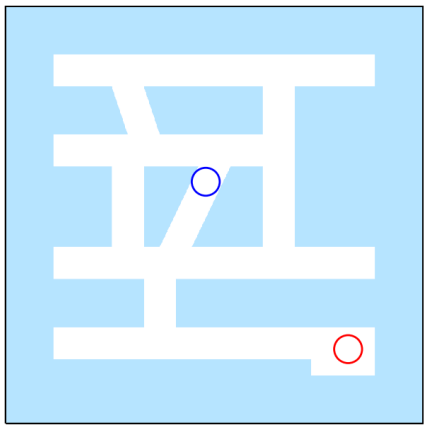
Maximum cost v	Reward	Completed
< 10	100	True
< 20	8	False
< 40	8	False
< 80	4	False
< 120	4	False
< 160	2	False
$\vdots$	$\vdots$	$\vdots$

4.5 Simulation

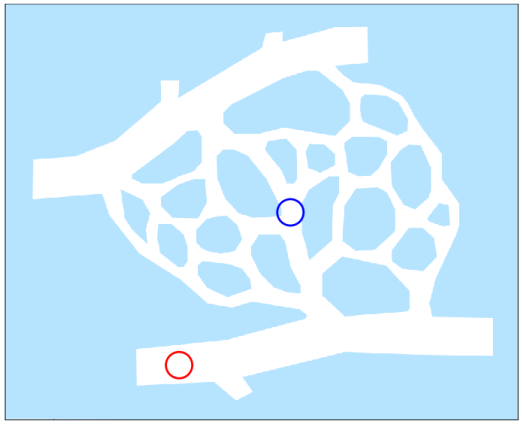
This section includes the training results of different vascular networks (mazes), with increasing complexity in targets and maze structures as shown in Figure 4.5. The performance of the heuristic (online) planner and RL (offline) planners are compared to evaluate improvement in delivery efficiency.

4.5.1 Comparison of Online and Offline Planners

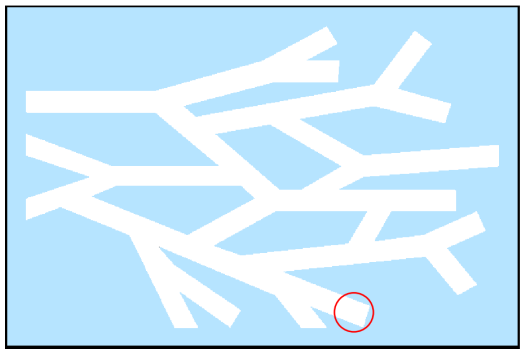
RL planning (ICM) shows a significant improvement in delivery efficiency compared to heuristic planning (D&C, a.k.a divide-and-conquer). Figure 4.7a and 4.7b show screenshots of



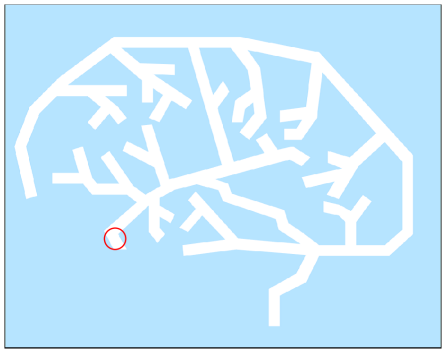
(a) Vascular network I, 100×100.



(b) Vascular network II, 180×150.



(c) Vascular network III, 180×120.



(d) Vascular network IV, 380×300.

Figure 4.5: (a) Target 1 (red): [82, 82], target 2 (blue): [42, 48]. (b) Target 1 (red): [130, 61], target 2 (blue): [75, 100]. (c) Target (red): [107, 122]. (d) Target (red): [204, 96].

delivery for ICM and D&C within the vascular network I, the easy target assigned, as it is at an endpoint that prevents microrobots escaping. Both of them complete their tasks, where ICM takes 1070 steps to finish the task, and 2886 steps for D&C. Note that ICM results are scaled for proper comparison with D&C. Figure 4.9a and 4.9b illustrate their strategies of delivery in terms of the cost-to-go function, where D&C (Figure 4.9b) has a more jittering curve compared to ICM (Figure 4.9a).

Figure 4.8a and 4.8b show screenshots of delivery for ICM and D&C within the vascular network I, the hard target assigned, as it is at a branch point that can barely keep microrobot staying. Only ICM completes its task, taking 1070 steps. D&C fails to finish the delivery within 6000 steps, and as revealed in Figure 4.10a and 4.10b, D&C has been trapped in a local

minimum after 2000 steps, while ICM is capable of decreasing both the average cost and the maximum cost to the target range.

In the vascular network II, two targets are assigned, with target 1 (red) an easy one, and target 2 (blue) a hard one. The screenshots of delivery are compared in Figure 4.11a, 4.11b, 4.12a, and 4.12b. Figure 4.13a, 4.13b, 4.14a, and 4.14b illustrate their strategies of delivery in terms of the cost-to-go function. The goals of two tasks are to decrease both average and maximum cost-to-go to a small range around zero. ICM completes two tasks, while D&C fails on the second task.

The overall performances of two online algorithms (the benchmark and D&C) and the offline algorithm (ICM) are compared in vascular network I, II, and IV with the easy targets as shown in Figure 4.6. It can be concluded that ICM outperforms the online algorithms in terms of the delivery efficiency (time steps). Compared to D&C, ICM takes 62% less time (steps) in the first two networks, and 70% less time (steps) in the third network.

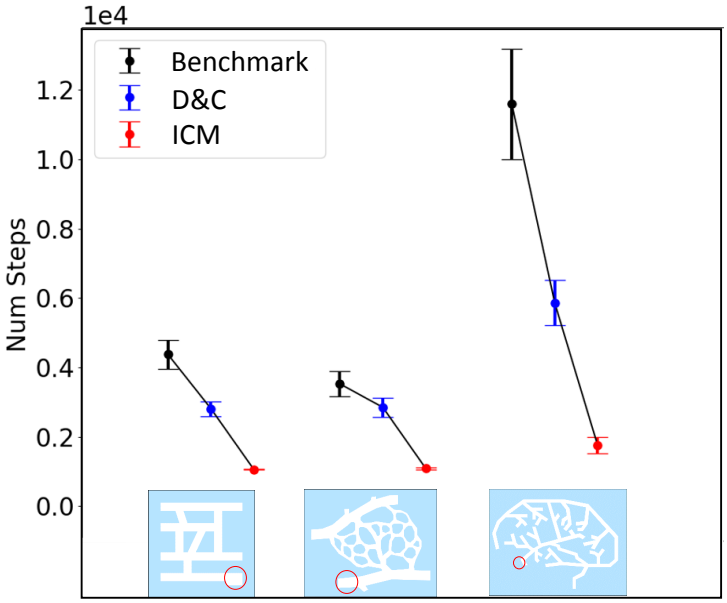
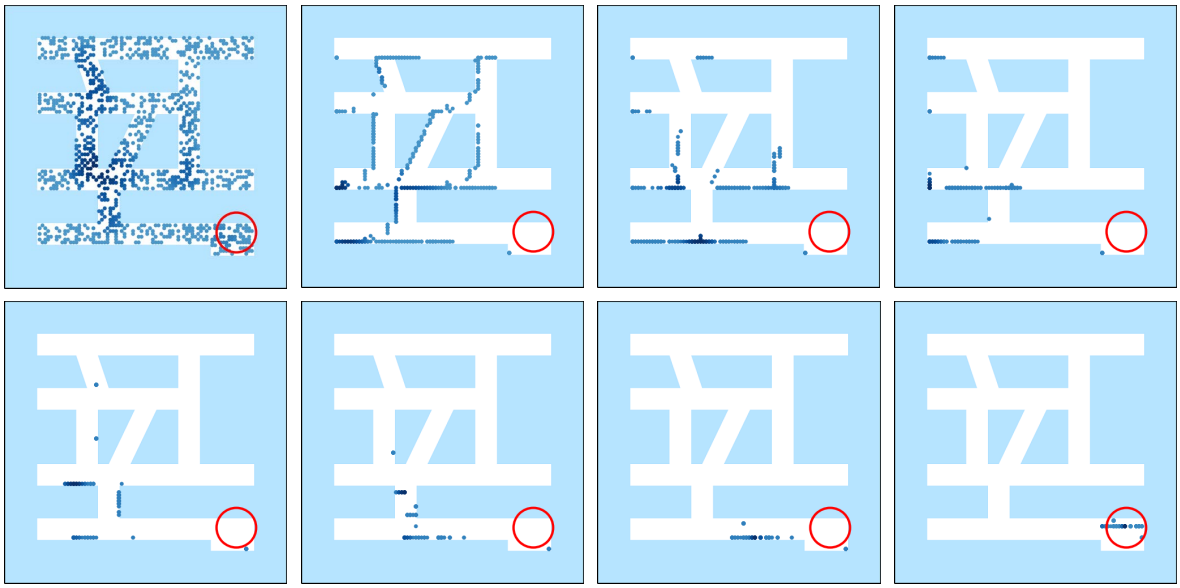
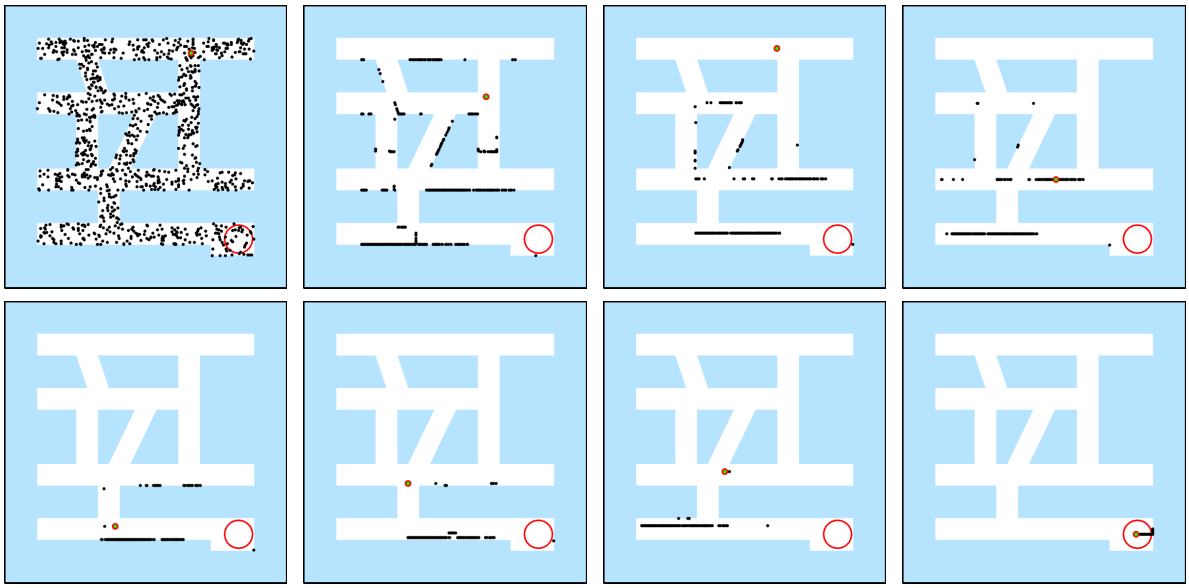


Figure 4.6: Simulation results of online and offline algorithms in vascular network I, II and IV, given the easy targets. Each data point represents the average steps of 128 trials initialized with 1000 microrobots.

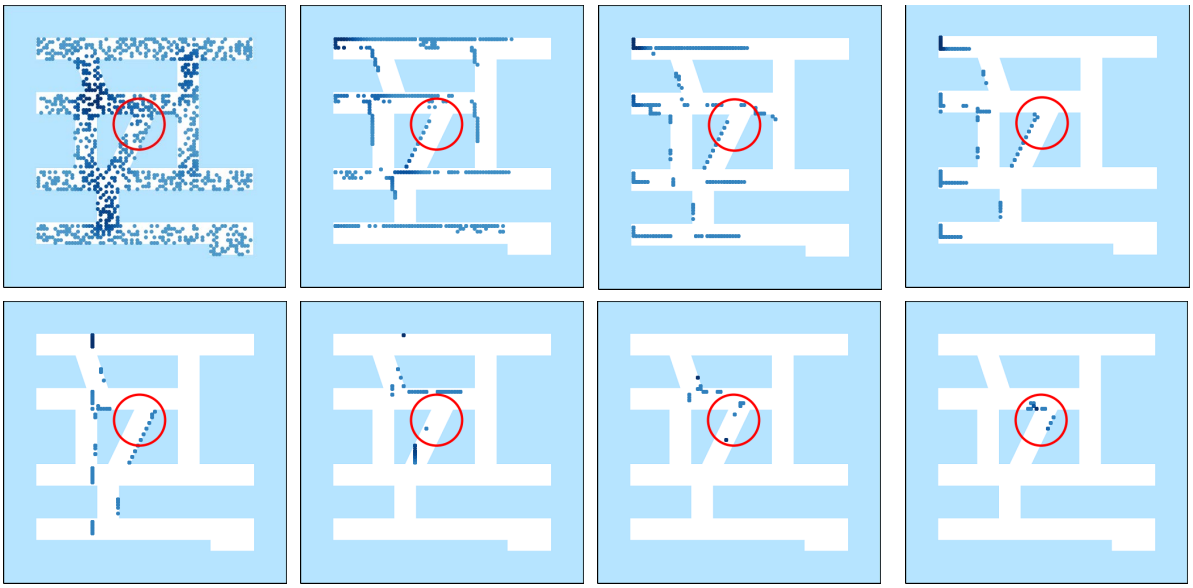


(a) ICM delivery (Python): 1070 steps, task completed. https://youtu.be/A8nyssHIVsI.

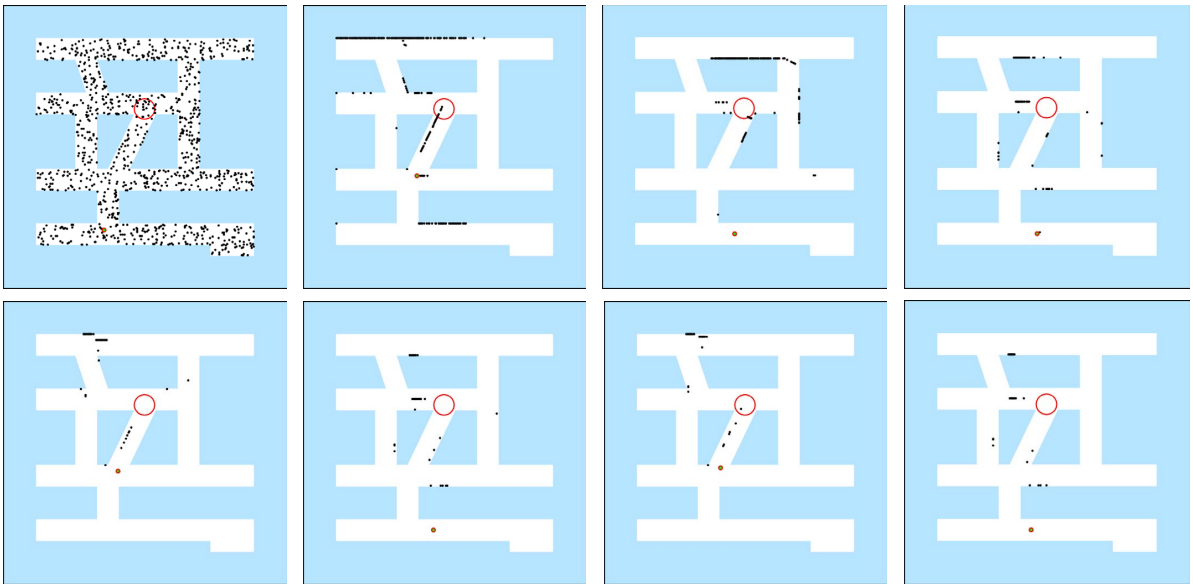


(b) D&C delivery (MATLAB): 2886 steps, task completed. https://youtu.be/MdyucazgDcU.

Figure 4.7: Screenshots of ICM and D&C demonstrations in vascular network I with the easy target, initialized with 1024 microrobots uniformly distributed. Each screenshot is taken every 1/7 total steps.

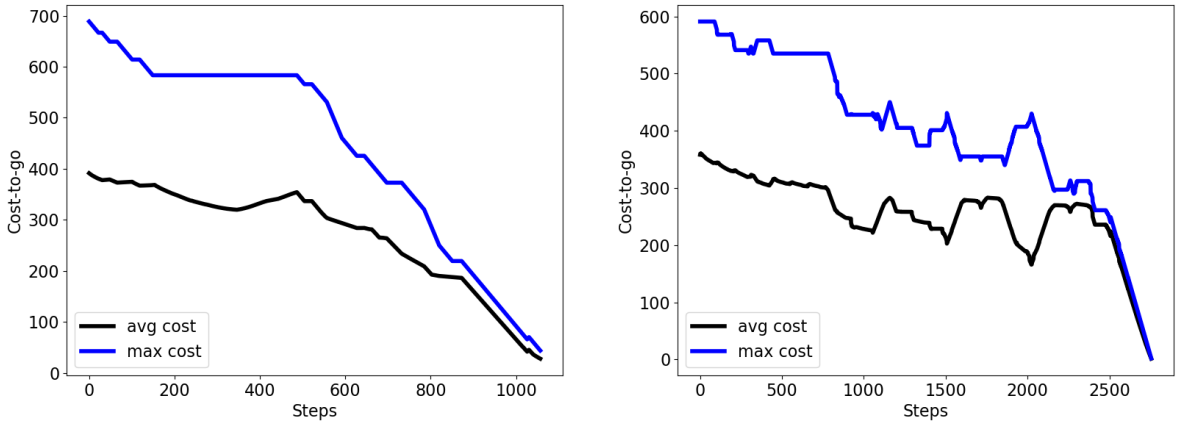


(a) ICM delivery (Python): 917 steps, task completed. https://youtu.be/JV7O3zIyFR8.



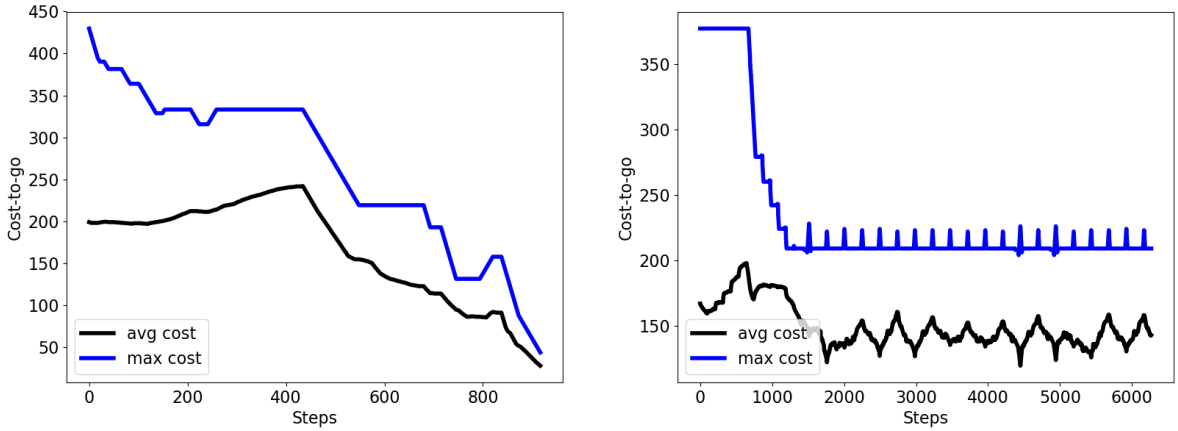
(b) D&C delivery (MATLAB): 6000 steps, task failed. https://youtu.be/oHItYu8vXZ0.

Figure 4.8: Screenshots of ICM and D&C demonstrations in vascular network I with the hard target, initialized with 1024 microrobots uniformly distributed. Each screenshot is taken every 1/7 total steps.



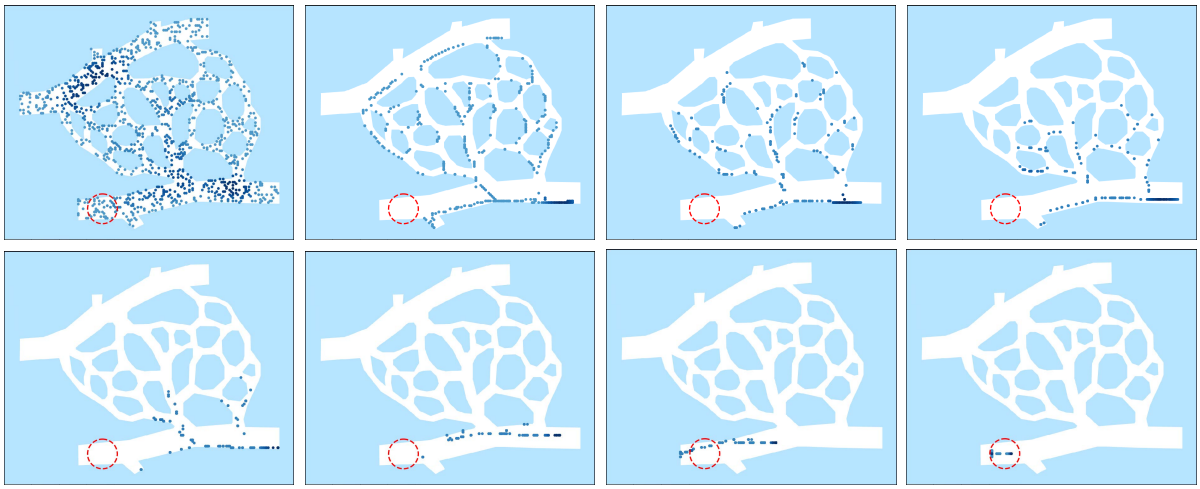
(a) (b)

Figure 4.9: Cost function plots of (a) ICM and (b) D&C in vascular network I with the easy target. The 'avg cost' and 'max cost' indicate the group mean and the maximum of cost-to-go of all microrobots.

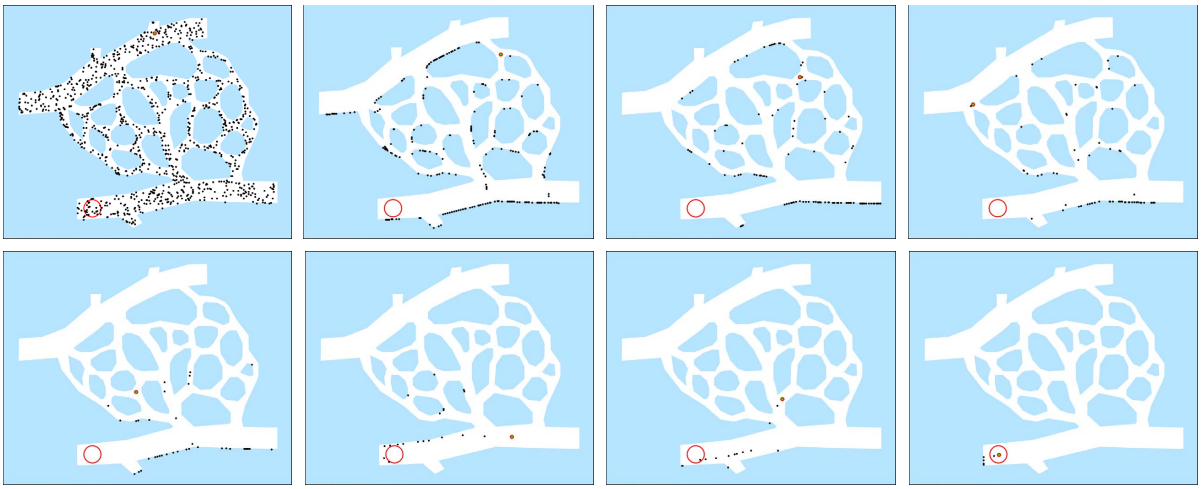


(a) (b)

Figure 4.10: Cost function plots of (a) ICM and (b) D&C in vascular network I with the hard target. The 'avg cost' and 'max cost' indicate the group mean and the maximum of cost-to-go of all microrobots.

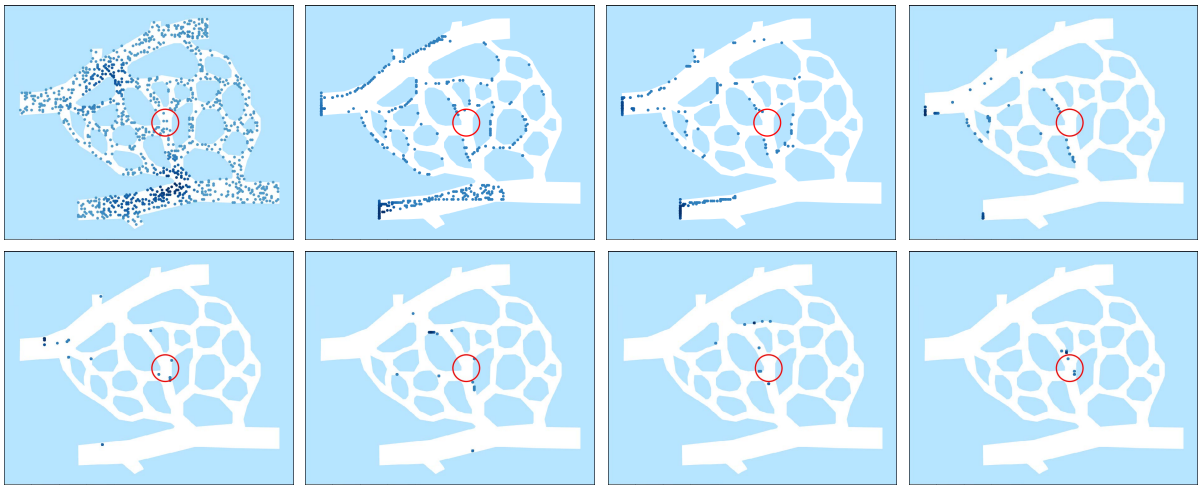


(a) ICM delivery (Python): 1079 steps, task completed. https://youtu.be/nZLgNM4SxMo.

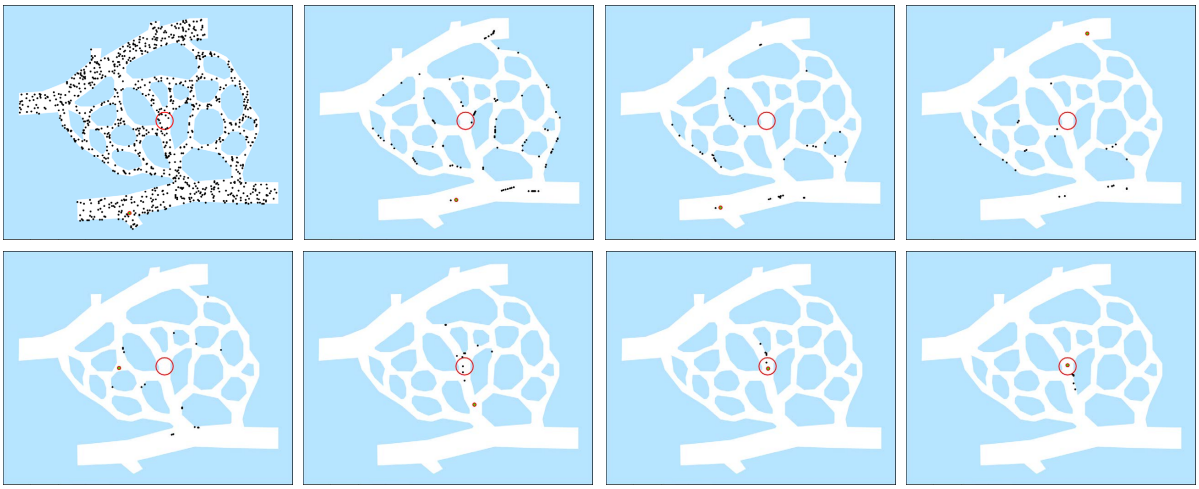


(b) D&C delivery (MATLAB): 2900 steps, task completed. https://youtu.be/uMvX2pByLcI.

Figure 4.11: Screenshots of ICM and D&C demonstrations in vascular network II with the easy target, initialized with 1024 microrobots uniformly distributed. Each screenshot is taken every 1/7 total steps.

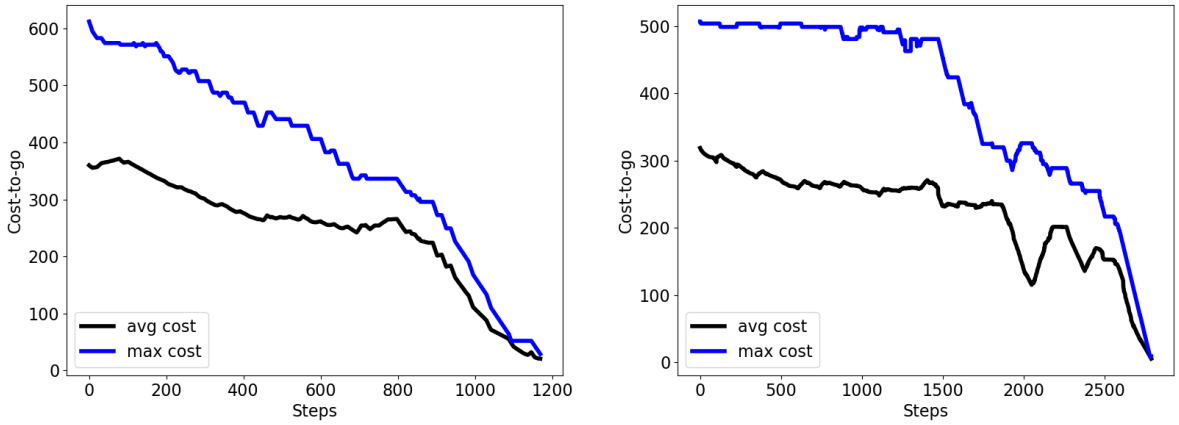


(a) ICM delivery (Python): 1300 steps, task completed. https://youtu.be/ERtfXlev1u4.



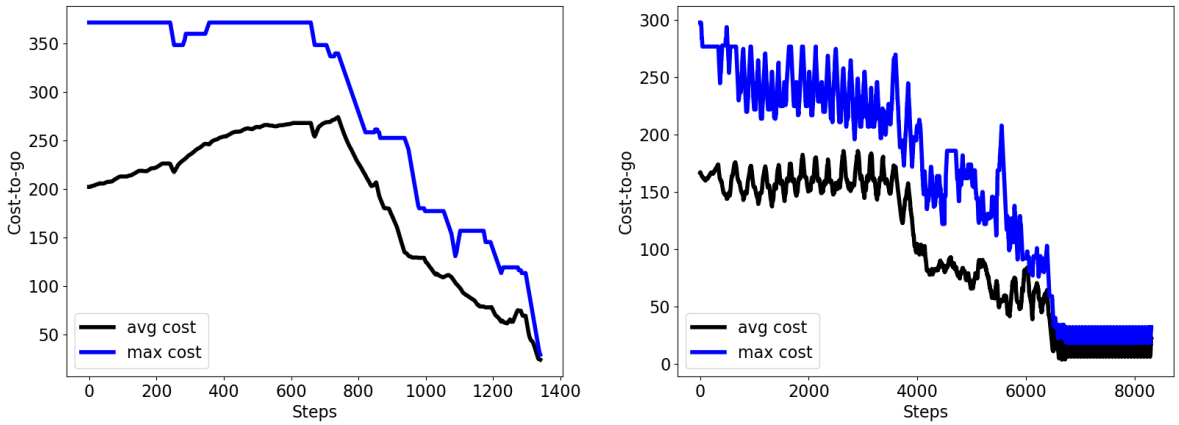
(b) D&C delivery (MATLAB): 8100 steps, task failed. https://youtu.be/zKMN23HDBHE.

Figure 4.12: Screenshots of ICM and D&C demonstrations in vascular network II with the hard target, initialized with 1024 microrobots uniformly distributed. Each screenshot is taken every 1/7 total steps.



(a) (b)

Figure 4.13: Cost function plots of (a) ICM and (b) D&C in vascular network II with the easy target. The 'avg cost' and 'max cost' indicate the group mean and the maximum of cost-to-go of all microrobots.



(a) (b)

Figure 4.14: Cost function plots of (a) ICM and (b) D&C in vascular network II with the hard target. The 'avg cost' and 'max cost' indicate the group mean and the maximum of cost-to-go of all microrobots.

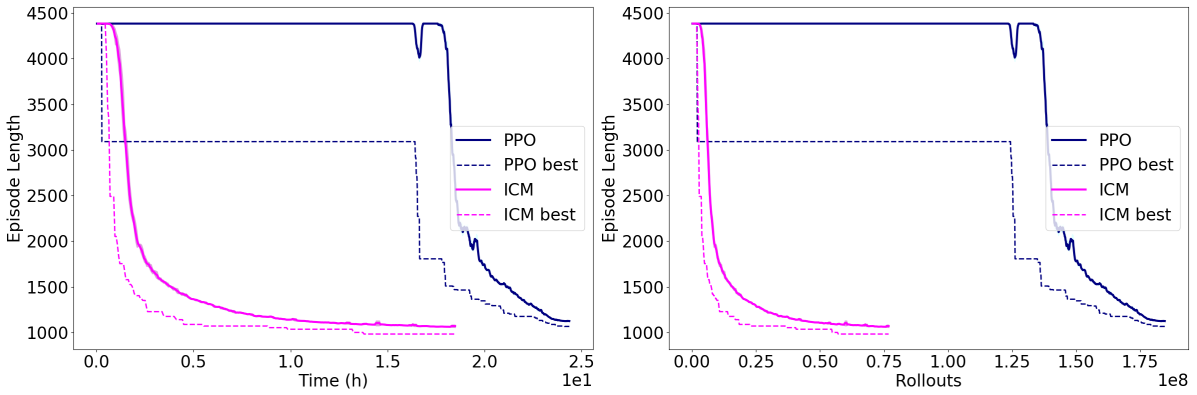
4.5.2 Comparison of Reinforcement Learning Algorithms

PPO, ICM, and RND algorithms are evaluated in vascular networks I, II, III, and IV, as shown in Figure 4.5. PPO agents are only rewarded by the environment (extrinsic), while both ICM and RND generate intrinsic rewards by self-learning in addition to extrinsic rewards.

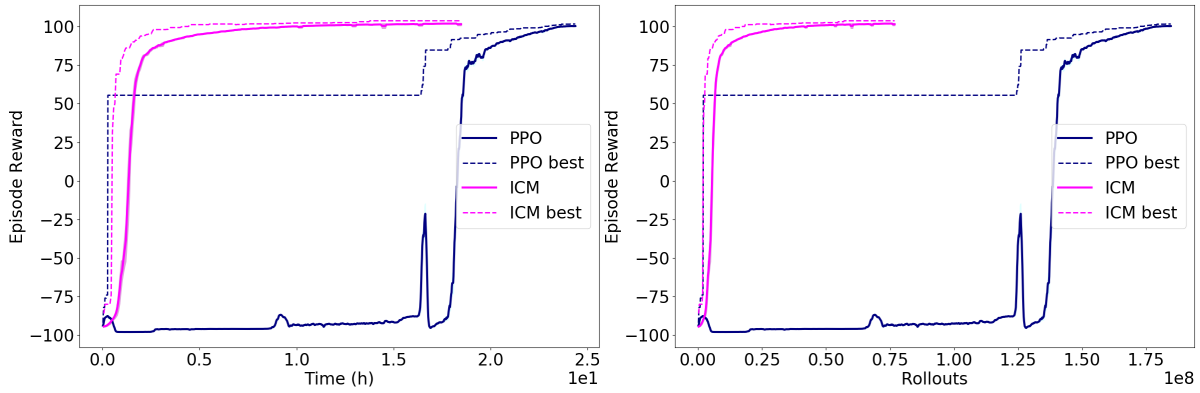
The performance of PPO and ICM are compared in vascular network I task 1 in Figure 4.15a and 4.15b. The 'PPO best' or 'ICM best' indicates the best results so far. Both PPO and ICM converge to the same level of the performance plateau, which indicates an optimal strategy for delivery. However, the difference is substantial in terms of cost-effective—PPO takes much longer and more samplings to achieve the similar performance as ICM does. These comparisons show that ICM agents get motivated by intrinsic rewards and thus explore more states than PPO agents who only receive extrinsic rewards.

Next, the results of target 2 in the same maze are compared in Figure 4.16a and 4.16b. This time, as it is considerably harder to deliver all microrobots to the target at a branch point, even the best of PPO agents fails to reach the goal. In vascular network II, ICM agents successfully deliver all microbots to the targets in two tasks, while PPO agents complete none of them, as shown in Figure 4.17a, 4.17b, 4.18a, and 4.18b.

ICM and RND algorithms are investigated in vascular networks II, III, and IV, with only one task for each. Both ICM and RND have similar learning process and converge to the optimal strategies (metric: number of steps for delivery) in the first two networks, as shown in Figure 4.19a, 4.19b, 4.20a, and 4.20b. However, in maze IV, ICM ends up with a suboptimal strategy, and takes longer to converge compared to RND, as shown in Figure 4.21a and 4.21b.

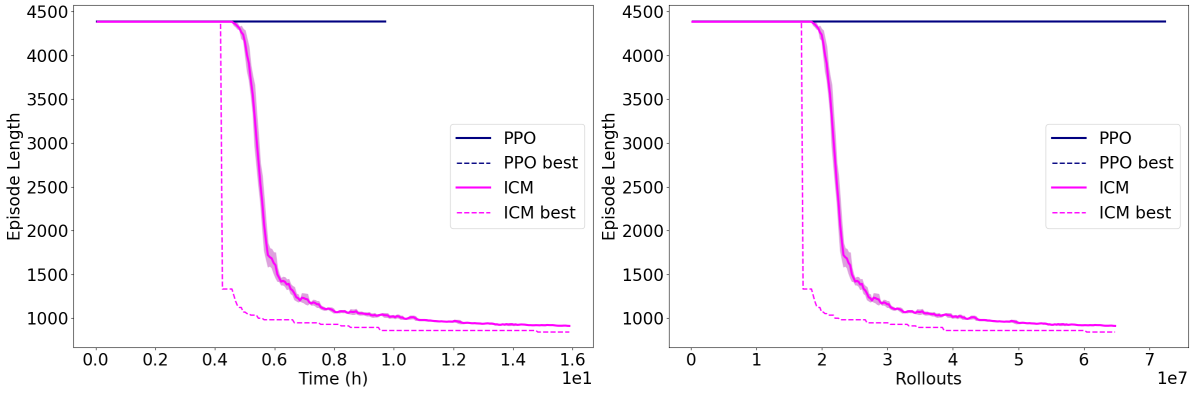


(a)

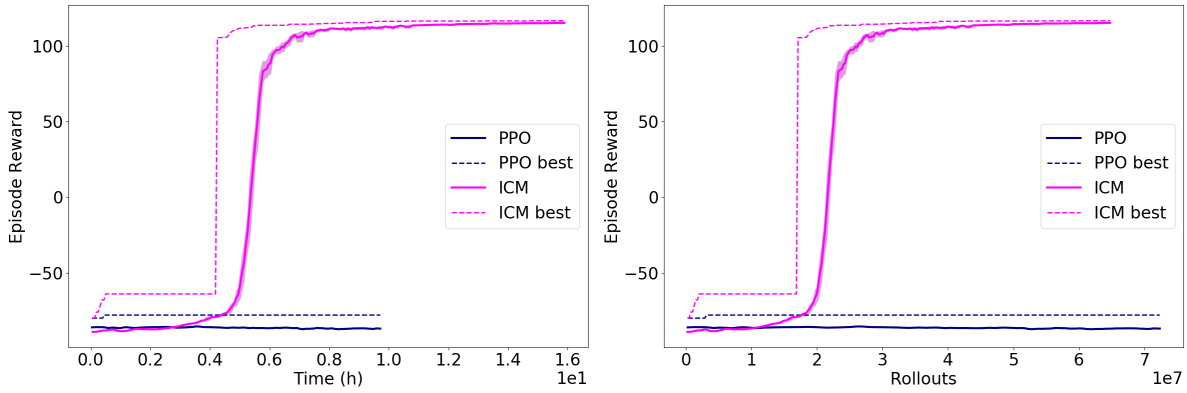


(b)

Figure 4.15: Vascular network I, target 1, comparisons of PPO and ICM. This evaluation shows (a) episode length and (b) episode reward versus the training time (left) and the number of rollouts (right). https://youtu.be/A8nyssHIVsI.

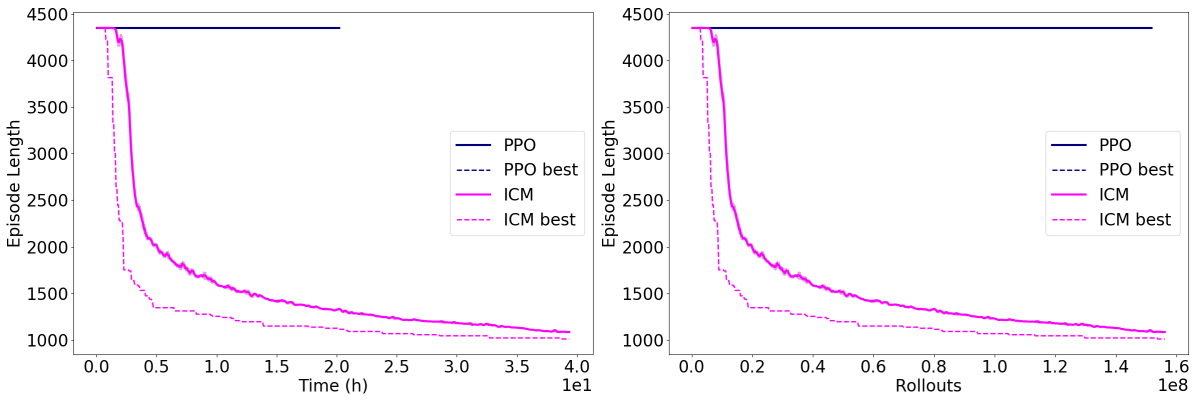


(a)

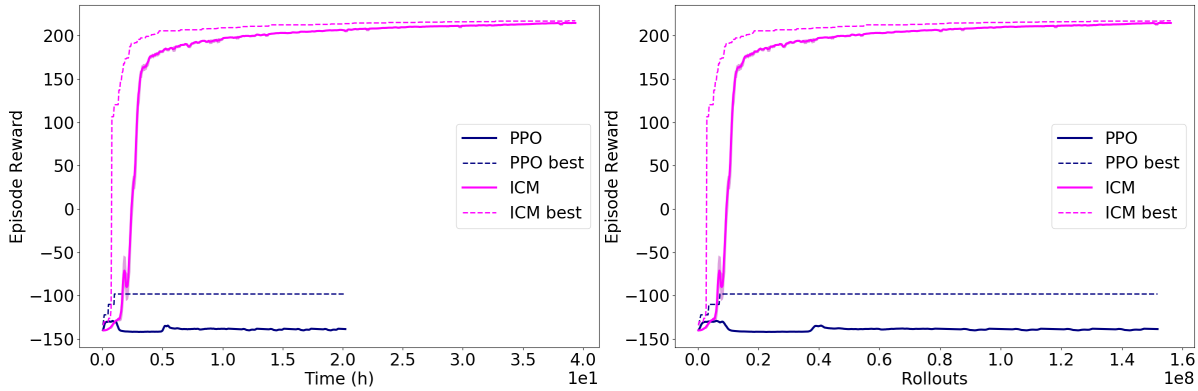


(b)

Figure 4.16: Vascular network I, target 2, comparisons of PPO and ICM. This evaluation shows (a) episode length and (b) episode reward versus the training time (left) and the number of rollouts (right). https://youtu.be/JV7O3zIyFR8.

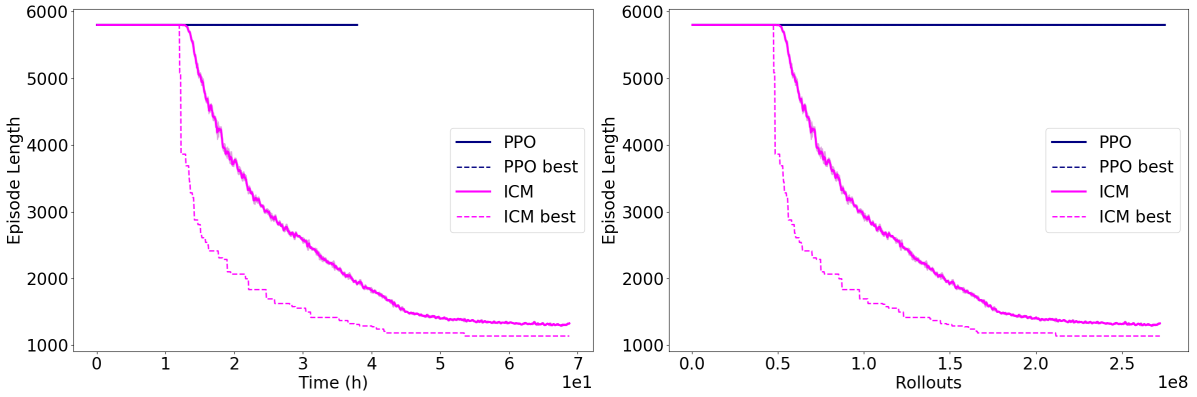


(a)

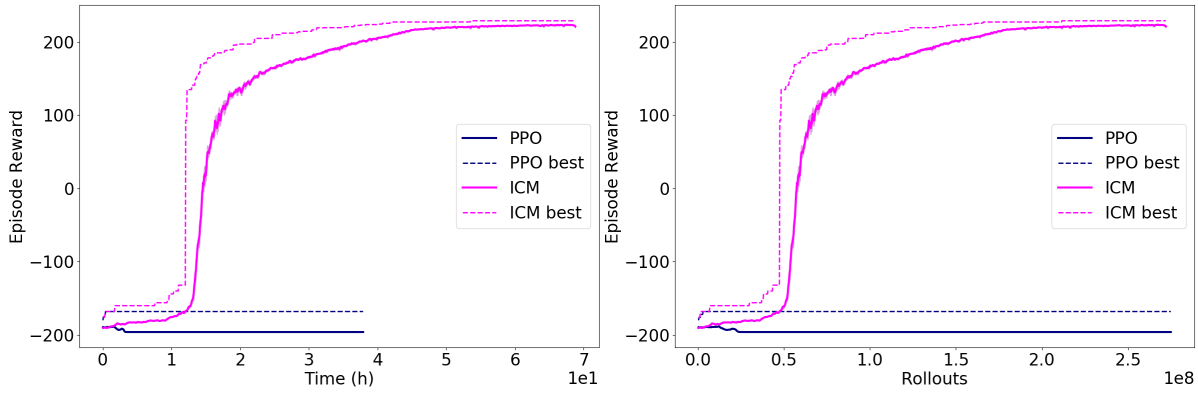


(b)

Figure 4.17: Vascular network II, target 1, comparisons of PPO and ICM. This evaluation shows (a) episode length and (b) episode reward versus the training time (left) and the number of rollouts (right). https://youtu.be/nZLgNM4SxMo.

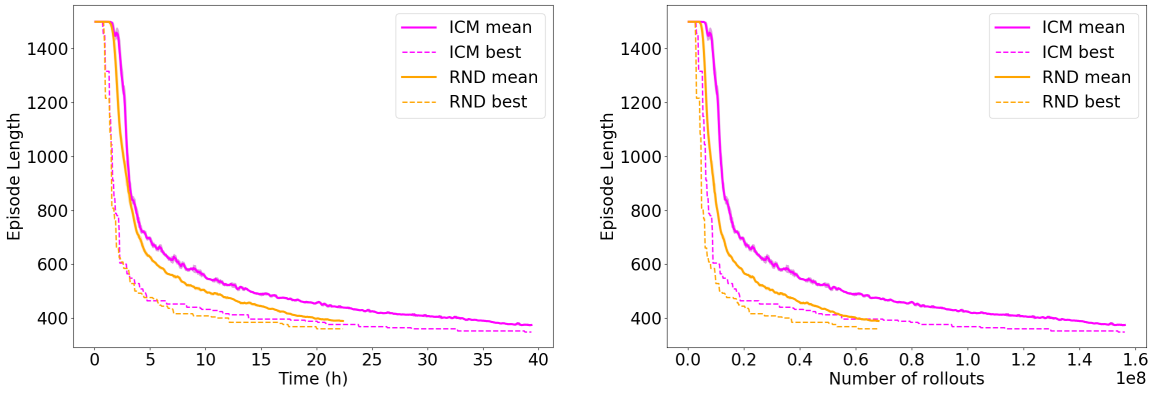


(a)

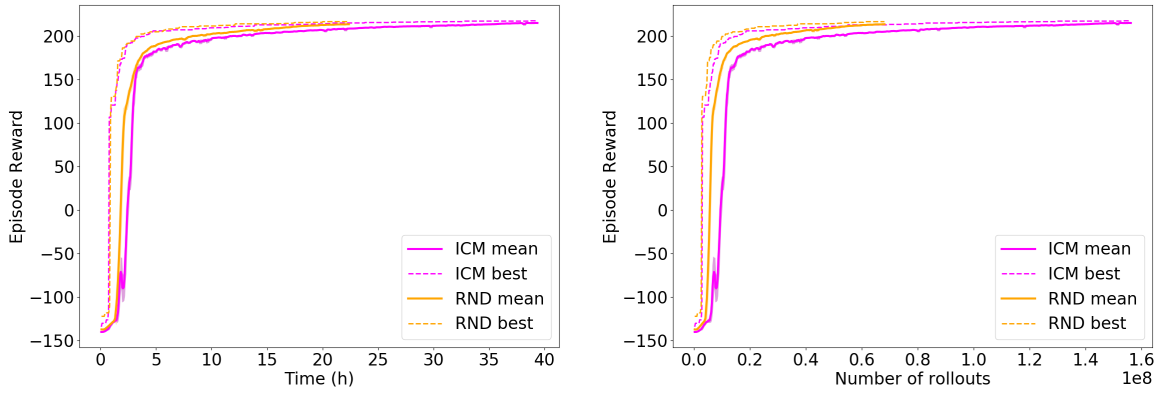


(b)

Figure 4.18: Vascular network II, target 2, comparisons of PPO and ICM. This evaluation shows (a) episode length and (b) episode reward versus the training time (left) and the number of rollouts (right). https://youtu.be/ERtfXlev1u4.

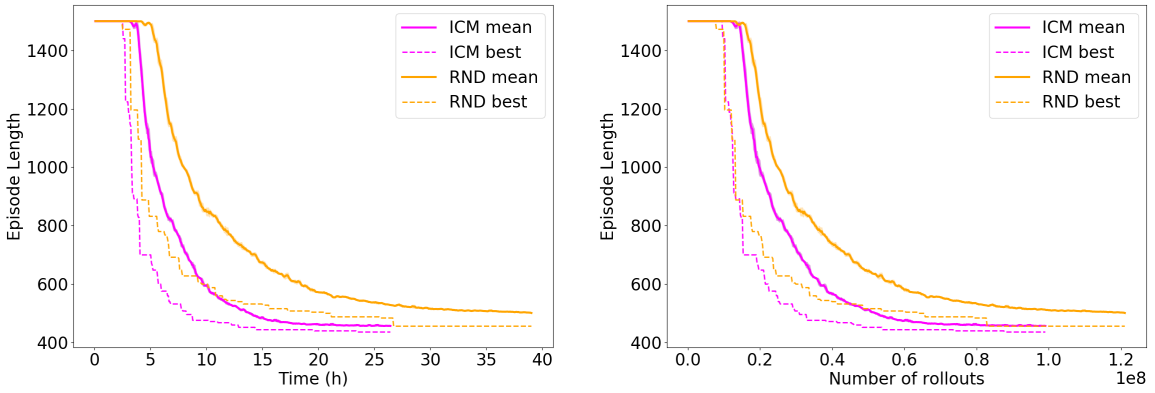


(a)

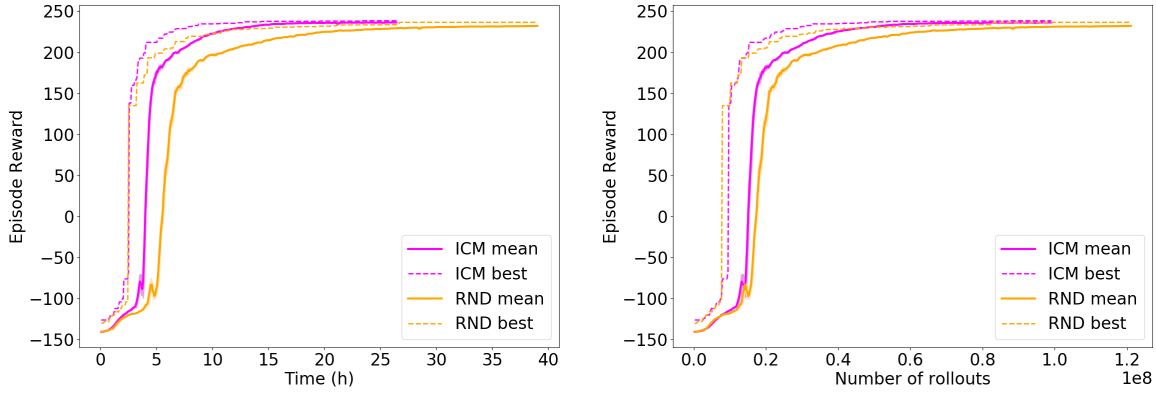


(b)

Figure 4.19: Vascular network II, comparisons of ICM and RND. This evaluation shows (a) episode length and (b) episode reward versus the training time (left) and the number of rollouts (right). https://youtu.be/nZLgNM4SxMo.

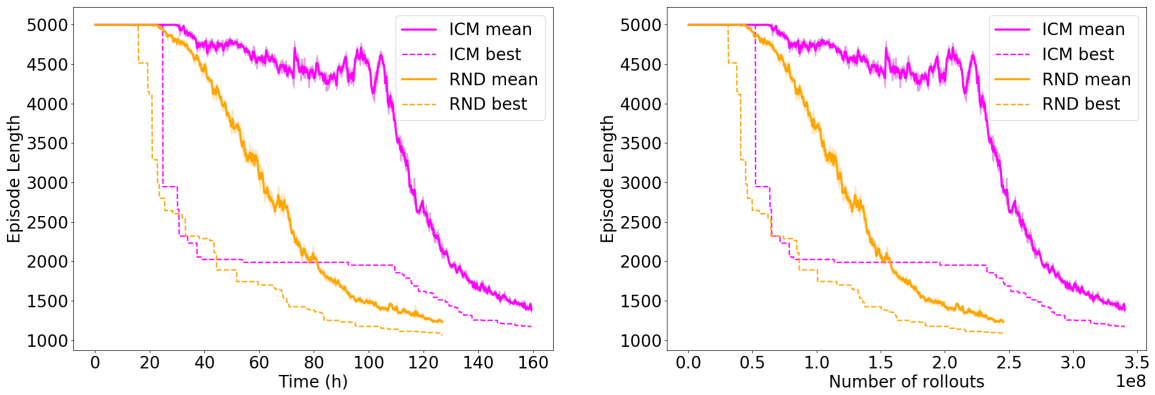


(a)

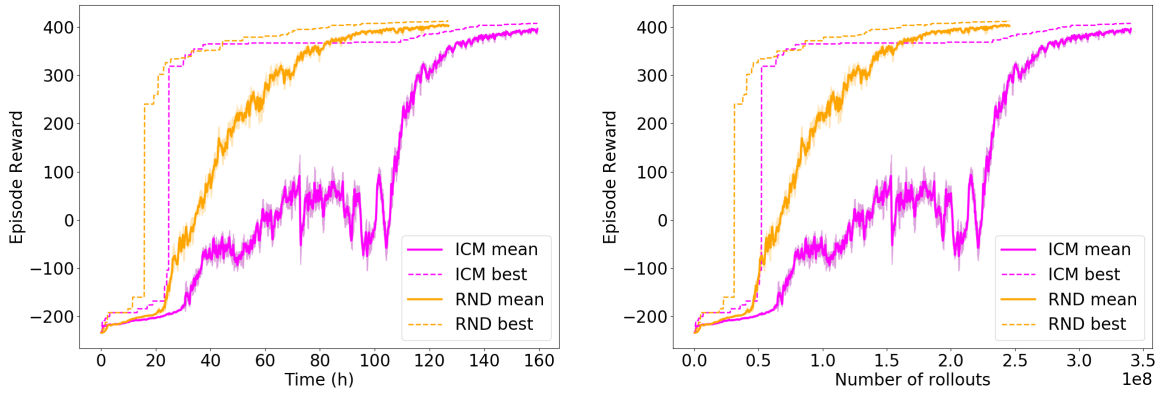


(b)

Figure 4.20: Vascular network III, comparisons of ICM and RND. This evaluation shows (a) episode length and (b) episode reward versus the training time (left) and the number of rollouts (right). https://youtu.be/gLIfxsYF1yY.



(a)



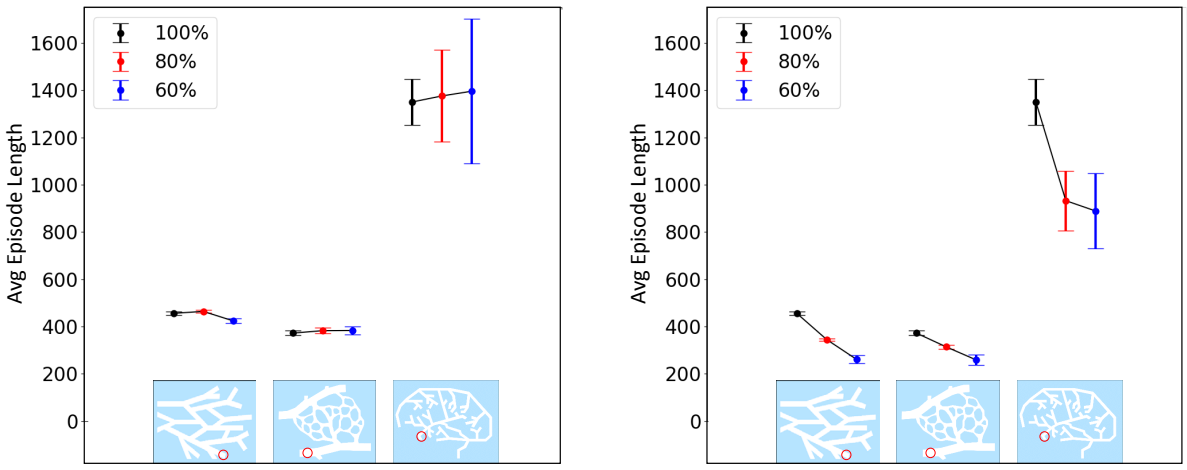
(b)

Figure 4.21: Vascular network IV, comparisons of ICM and RND. See videos at https://youtu.be/DAgtSBvDgpA and https://youtu.be/mSyXgN-ycsA.

4.5.3 Comparison of Targeted Delivery Rates

Delivering all microrobots to a target region might not be most cost-effective, as planning needs to take account corner cases which might involves a small population but takes plenty of extra time. The influence of targeted delivery rates are investigated in vascular network II, III and IV using ICM algorithm. Considering 80% as the targeted delivery rate, the RL agent can either learn particular strategies for different delivery rates, or learns a strategy aimed at 100% delivery and executes early stop when 80% of the microrobots reach the goal. This section investigates three delivery rates: 100%, 80%, and 60% and the two scenarios mentioned above.

In Figure 4.22a, RL agents are trained for 100% delivery rate, and applied to 80% and 60% cases. The results imply a minimal difference in the average episode length. While in Figure 4.22b, RL agents deploy particular strategies for each delivery rate. It indicates linear increase in the episode length as the targeted delivery rate increases for two easier environments, vascular network II and III. In the harder case, vascular network IV, the required episode length increases exponentially when the targeted delivery rate changes from 80% to 100%.



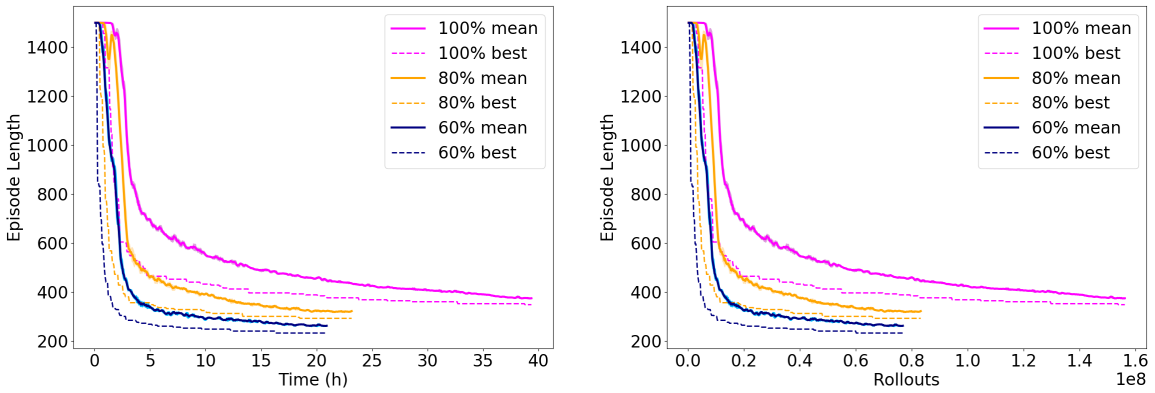
(a) One strategy for all delivery rates.

(b) One paticular strategy for each delivery rate.

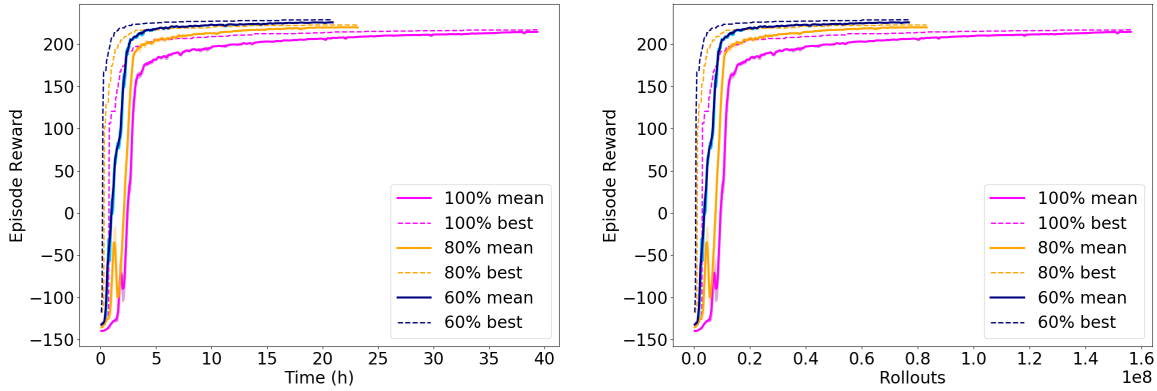
Figure 4.22: The influence of delivery rates to the episode length (the steps for delivery).

The learning processes in three vascular networks are illustrated in Figure 4.23, 4.24 and 4.25. The learning curves show that a lower targeted delivery rate leads to faster convergence,

and thus it requires less experience sampling and neural network updating. This is because RL agents learn to give up corner cases for a lower targeted delivery rate. It implies that to satisfy the dosage requirement for microrobots, a less demanding delivery goal with increasing swarm population have faster convergence for optimal strategies.

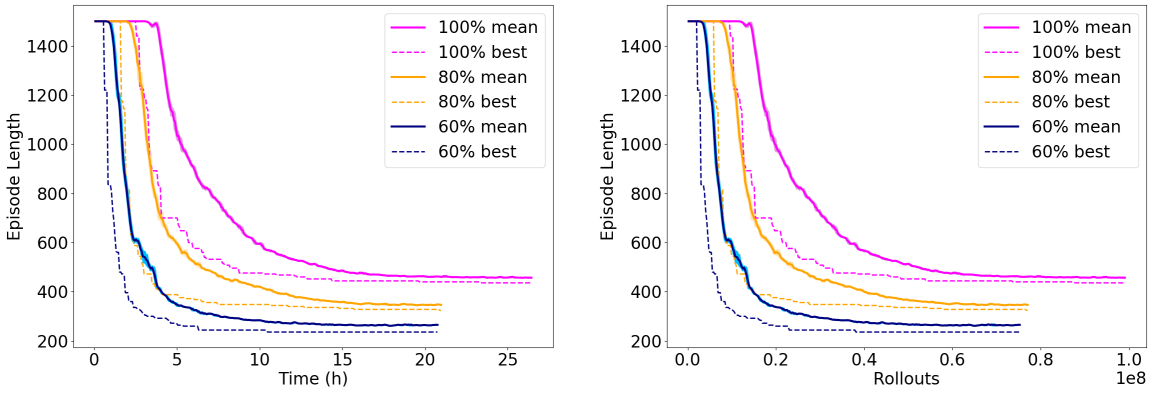


(a)

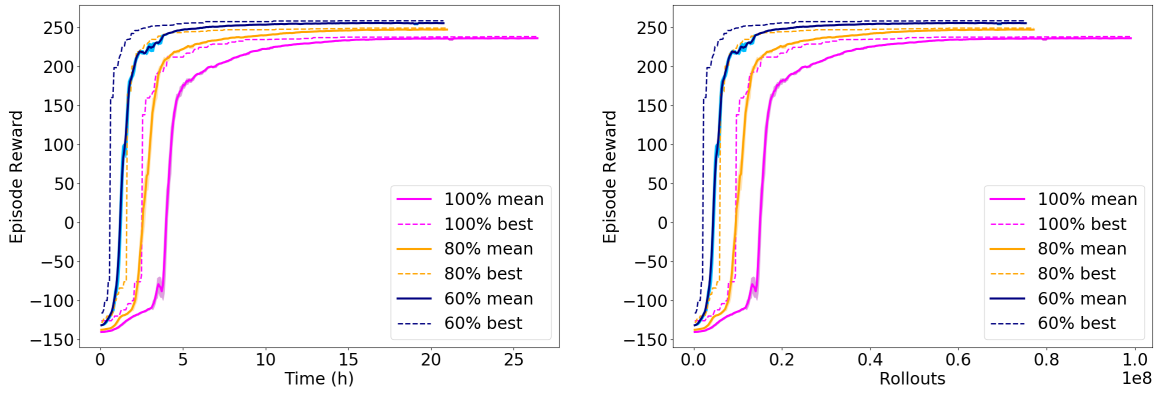


(b)

Figure 4.23: Vascular network II: a comparison of 100%, 80%, and 60% delivery rates. See videos at https://youtu.be/nZLgNM4SxMo, https://youtu.be/InqZljlfYOs, and https://youtu.be/odeTMIdBbdI.

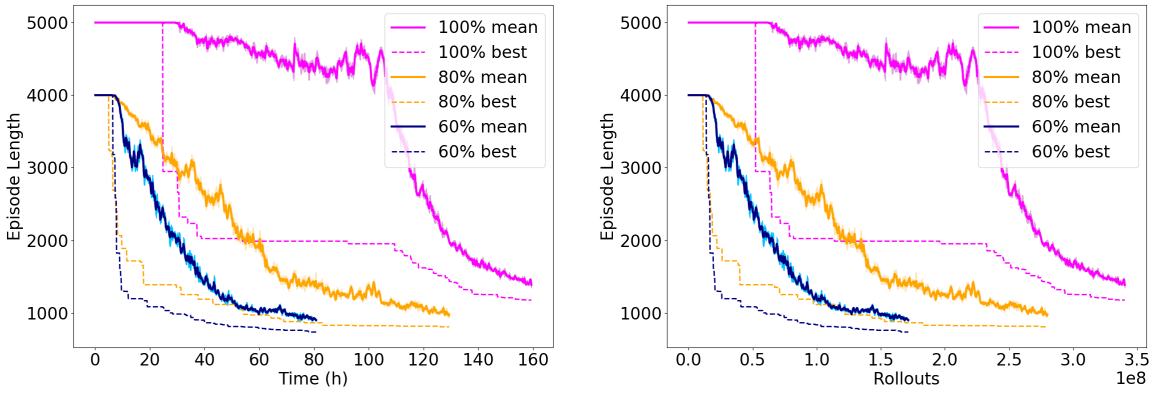


(a)

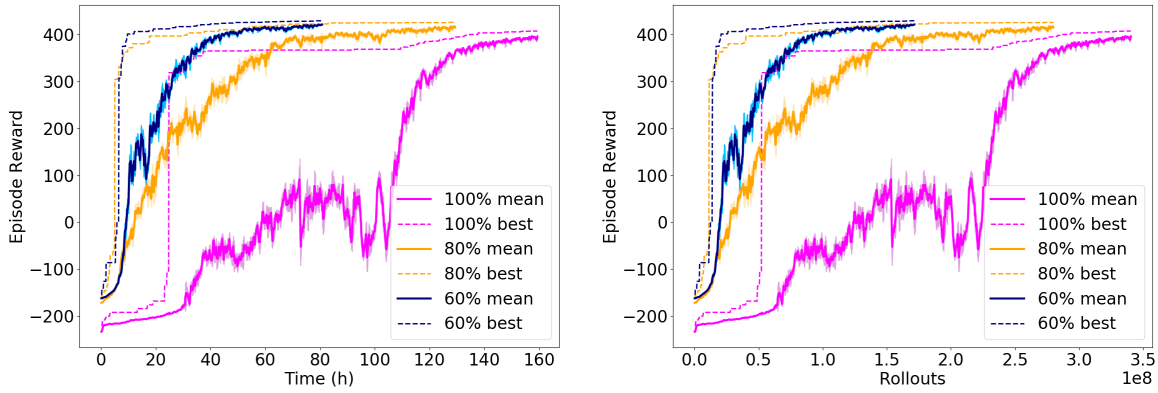


(b)

Figure 4.24: Vascular network III: a comparison of 100%, 80%, and 60% delivery rates. See videos at https://youtu.be/gLIfxsYF1yY, https://youtu.be/ePby3fsmeTo, and https://youtu.be/g6qBPQyZ7V8.



(a)



(b)

Figure 4.25: Vascular network IV: a comparison of 100%, 80%, and 60% delivery rates. See https://youtu.be/DAgtSBvDgpA, https://youtu.be/OmLhsxqyGsU, and https://youtu.be/AIun5uES8LI.

4.6 Conclusion

This chapter presents reinforcement learning (RL) algorithms aimed at optimizing the efficiency of delivering microrobots in vascular networks given a global control input. Curiosity-driven algorithms are implemented to overcome the delivery challenges such as the sparse rewards, the high-dimensional sensory space, and local minima dilemmas. RL (offline) strategies outperform the heuristic (online) planning methods in the simulations, capable of navigating microrobots to hard targets such as a branch point efficiently, and adapted to different environments without changing the neural network structures. This chapter also investigates the influence of different targeted delivery rates to the delivery efficiency and the RL learning process. It reveals that a lower targeted delivery rate requires fewer steps for a delivery task and brings faster convergence in training.

As for future work, more realistic scenarios would be considered to approach the ultimate goal of MRI-guided drug delivery, including adding physics-based microrobot dynamics to the simulation, introducing flow dynamics, and control limitations. RL strategies can provide instructions for future improvement in online planning algorithms. Besides, implementing these RL algorithms in hardware experiments would be an essential step for future research.

Chapter 5

Steering Microrobots in Multi-branch Vessels Using a Global Control Input

5.1 Introduction

In Chapter 3 and 4, heuristic and reinforcement learning path planners are proposed to steer a swarm of homogeneous microrobots in vascular networks with the following assumptions: (1) closed outlets (endpoints) prevent microrobots from escaping; (2) the environment is static without dynamic flows, so that microrobot motion along any directions is allowed; (3) microrobots follow deterministic motion without disturbances. This chapter addresses path-planning problems in multi-branch vessels with more realistic configurations. It simulates a magnetically guided drug delivery process similar to recent progress in hardware experiments, including dynamic flows, randomness in microrobot motion, and limited steering capabilities for a global controller.

In previous chapters, the global control input is capable of actuating microrobots and moving them in any directions regardless of the flow. However, this is not realistic in blood vessels. The average blood flow rates vary in the circulation system, for example, it is 450 mm/s in arteries (4 mm diameter), 50 mm/s in arterioles (0.05 mm diameter), 1 mm/s in capillaries (0.008 mm diameter), 3 mm/s in venules (0.02 mm diameter), and 5 mm/s in veins (5 mm diameter) [21, 86]. The microrobot velocity is at least an order of magnitude smaller compared to the blood flow except within capillaries. For example, Martel et al. reported several candidates of homogeneous microrobots, with the *Magnetospirillum gryphiswaldense* bacterium about 0.001-0.003 mm long and the speed in the range of 0.04-0.08 mm/s, MC-1 about 0.002 mm long, and the speed of 0.223 mm/s without wall effect, 0.18 mm/s within a 0.01 mm

microchannel, and 0.048 mm/s within a 0.004 mm microchannel [12]. Since the blood flow propels microrobots, such a scenario needs a steering input to direct microrobots towards the desired branch at a bifurcation, instead of a controller moving them without constraints [3]. This is illustrated in Figure 5.1, where blood flow propels microrobots towards downstream. A global control input, such as a magnetic gradient field applied at each bifurcation can direct microrobots to the desired branch.

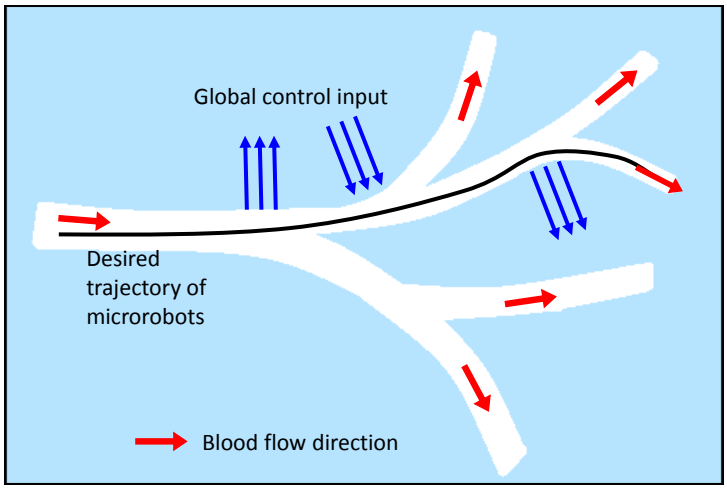


Figure 5.1: A global control input is used to direct microrobots in vessels with flow.

This chapter introduces two automatic steering (control) methods for microrobot navigation in multi-branch vessels with flow. For comparison, reinforcement learning (RL) is implemented to inspire the future development of the automatic steering methods.

5.2 Related Work

Mathieu et al. proved the concept of microparticle steering using a clinic MRI scanner for applications in the human blood vessels [87]. They have achieved 60% delivery rate in a Y-shape *in vitro* environment using ferromagnetic particles. Later, they increased the delivery rate up to 99% [86]. Bigot et al. successfully navigated a magnetic bead through multi-branch vasculature inside an MRI [88].

Martel et al. implemented the first feedback navigation control of a 1.5-mm ferromag-

netic bead in a live swine inside an MRI scanner [1]. They showed that the minimum time to acquire the coordinates of the bead was 16 ms. Pouponneu et al. demonstrated MRI-guided microparticle delivery in live rabbits. Both of these *in vivo* experiments were conducted in simple vessels [89].

Hoshiar et al. discussed a scheme of electromagnetic actuation to steer magnetic nanoparticles in a multi-channel vessel while reducing aggregation [90]. The scheme was demonstrated in a 3D vessel simulation, showing success in steering and disaggregation. Hoshiar et al. developed a simulation platform and an aggregation model to investigate steering magnetic microparticles in a Y-shape bifurcation [91]. The results help design magnetic actuation schemes with potential applications for improving drug delivery efficiency.

Hamdipoor et al. proposed a haptic guided scheme for human-in-the-loop targeted drug delivery, which is closely related to this chapter [92]. They developed a virtual environment to simulate multiple magnetic nanoparticles in multi-branch vessels with physics-based models. The haptic feedback was designed to assist the human with manipulating microparticles for efficient drug delivery. However, there are several limitations in their demonstration: (1) only one bolus of nano-/microparticles is released from the inlet, where the dose of drugs might not be sufficient; (2) the desired magnetic fields at all bifurcations are approximately along the same direction, which is a simple case; (3) the scheme needs a human operator, not fully automatic control.

Many works on microparticles steering are aimed at improving targeting/steering efficiency, including [3, 12, 21, 93, 94]. Larimi et al. reported CFD simulation of magnetic particles distribution within blood flow under the influence of the magnetic field in a bifurcation vessel [95]. Several related works investigate the behaviors of in blood flow under an external magnetic field, such as [96–101].

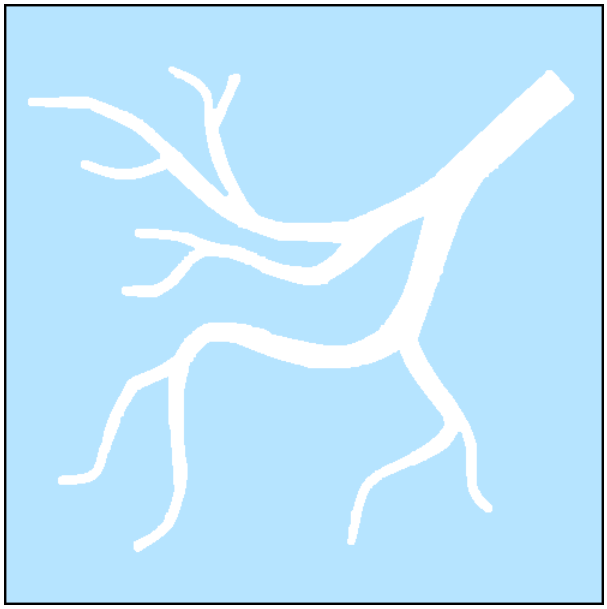
5.3 Methodology

This section proposes two automatic steering methods to deliver homogeneous microrobots to a targeted outlet in multi-channel vessels given the following challenges: (1) a virtual dynamic flow is introduced to propel microrobots; (2) the flow also causes randomness in microrobot motion; (3) the global control input has limited power and thus cannot pull microrobots backward; (4) the outlets (endpoints) are absorbing, so microrobots cannot escape. For comparison, this section implements deep reinforcement learning to investigate optimal steering strategies.

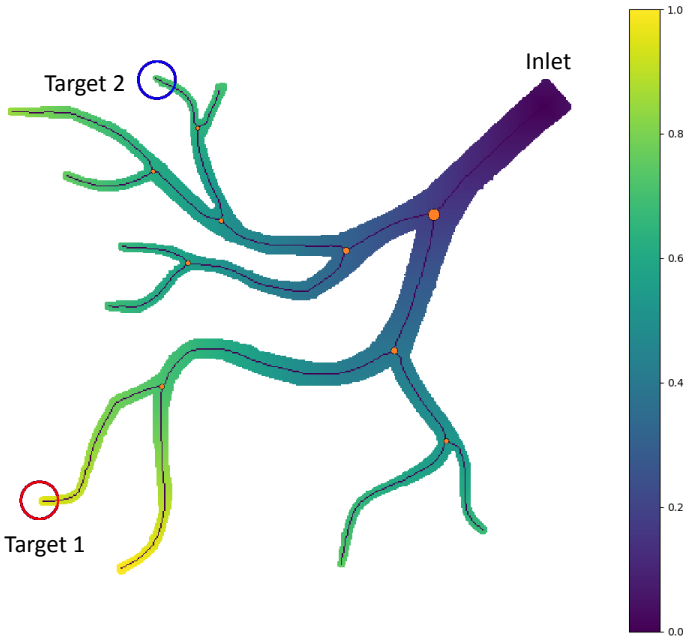
5.3.1 Data Preprocessing

Three multi-branch vessels are designed to demonstrate microrobot delivery, originated by [102–104], as shown in Figure 5.2a, 5.3a, and 5.4a, with an increasing number of branches and steering complexity. The map preprocessing (code available on GitHub [105]) is required before proceeding to automatic control, including target assignment, map skeletonizaion, cost-to-go map calculation, detection region construction, and steering direction identification.

The centerline of each vessel is extracted via skeletonization (*scikit-image*, Skeletonize, [106]), and each branch is marked as an orange dot. For simplicity, the flow leads to a statistically symmetric splitting of a swarm of microrobots at each bifurcation. The cost-to-go at each location is determined by the distance to the inlet. The centerline, branch points, and cost-to-go visualization are shown in Figure 5.2b, 5.3b, and 5.4b. Based on the cost-to-go map, each bifurcation is assigned with a detection region for microrobots as shown in Figure 5.5a. A detection region is identified by a branch point and its upstream coordinates within certain range of cost-to-go. The branch point together with the centerline are used to identify the flow direction at each bifurcation as shown in Figure 5.5b. According the local flow vector and the desired branch at a bifurcation, the steering direction is chosen to maximize the steering effect as shown in Figure 5.5b.

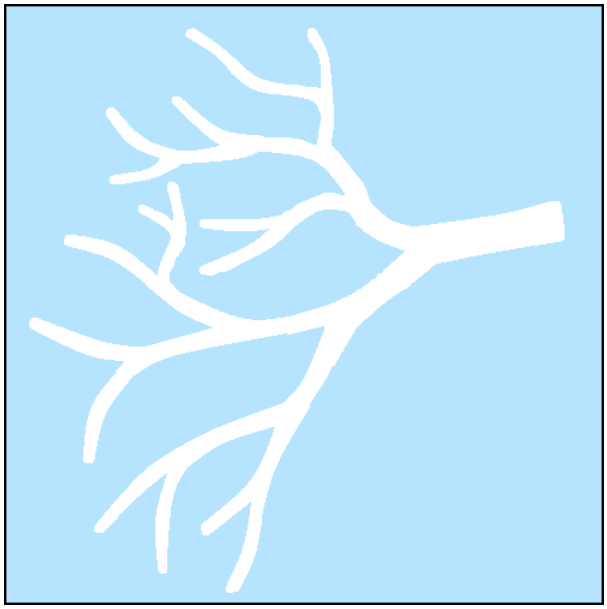


(a)

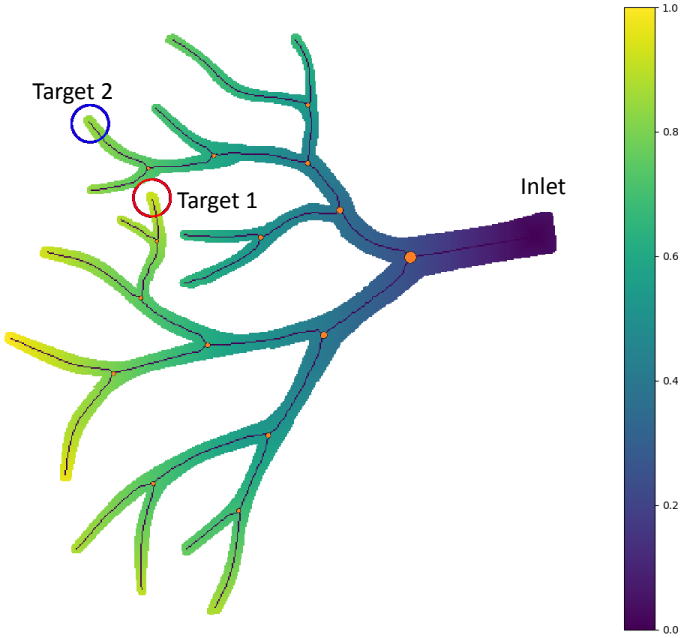


(b)

Figure 5.2: (a) Vessel I with 10 outlets and 9 bifurcations, size 200×200. (b) Cost-to-go map based on the distance to the inlet. Each branch is marked by an orange dot. Targeted outlets for two tasks are marked by a red and a blue circle.

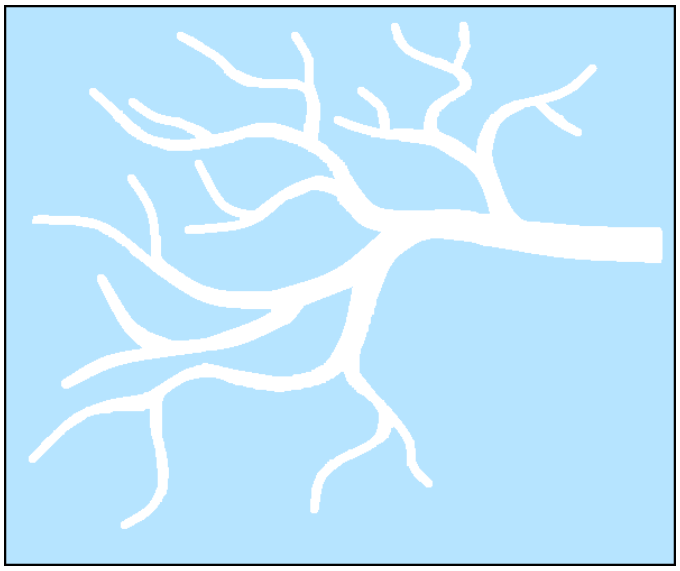


(a)

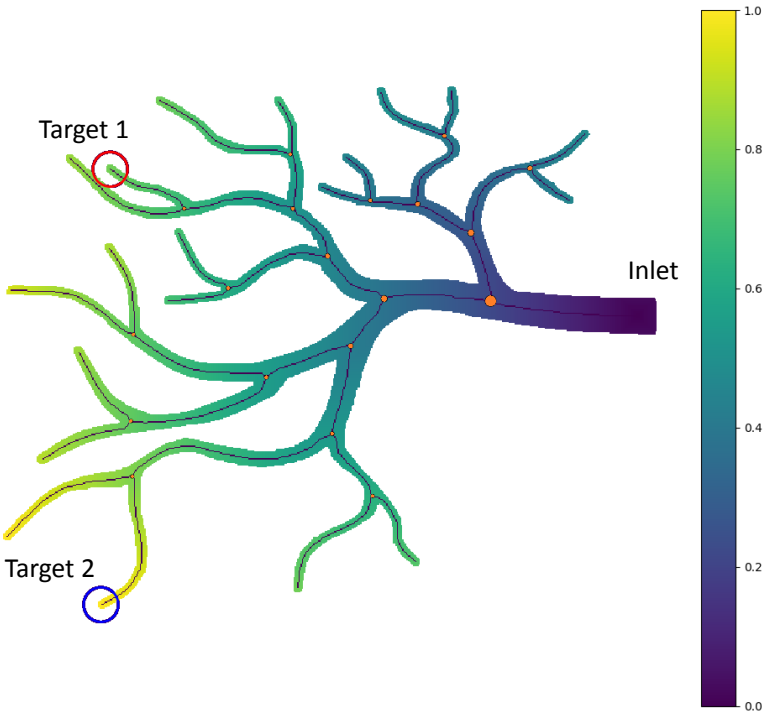


(b)

Figure 5.3: (a) Vessel II with 16 outlets and 15 bifurcations, size 200×200. (b) Cost-to-go map based on the distance to the inlet. Each branch is marked by an orange dot. Targeted outlets for two tasks are marked by a red and a blue circle.

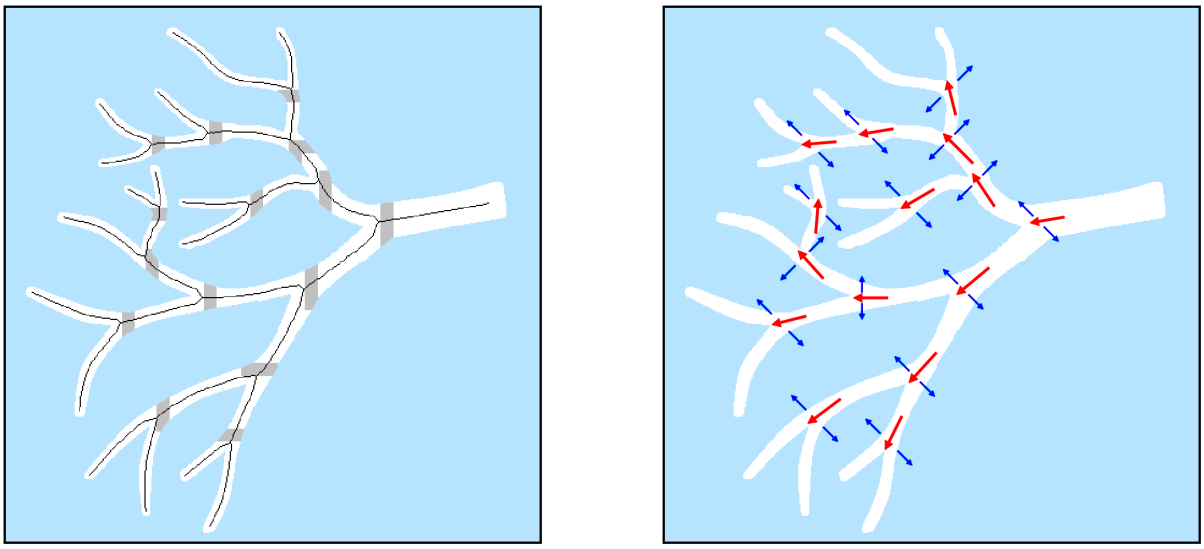


(a)



(b)

Figure 5.4: (a) Vessel III with 20 outlets and 19 bifurcations, size 300×360 (b) Cost-to-go map based on the distance to the inlet. Each branch is marked by an orange dot. Targeted outlets for two tasks are marked by a red and a blue circle.



(a) (b)

Figure 5.5: (a) Detection regions are denoted as gray bands. (b) The flow directions and steering directions are marked as red and blue arrows respectively at each bifurcation. The steering directions are chosen from $\{\leftarrow, \rightarrow, \uparrow, \downarrow, \nearrow, \searrow, \swarrow, \nwarrow\}$.

Section 5.2 introduces many works that have developed simulation platforms using physics-based model. The microrobot kinematics in blood flow are simplified here, as the primary concern is to provide automatic steering methods for microrobot navigation in multi-branch vessels. A bolus of microrobots released at the inlet follows the cost-to-go gradient ascent with probability p, where $0 < p \leq 1$, and might drift along the contour line, i.e. nearby locations with the same cost-to-go, with probability $1 - p$. Statistically, the bolus is symmetrically divided at each bifurcation, so that no bias is created due to the flow.

One of the key factors that affects steering effectiveness is the velocity ratio between the flow and the microrobot. Given a capillary of 0.005 mm wide, 1 mm long, with 0.5 mm/s flow, a microparticle (0.0021 mm) must accelerate to 0.003 mm/s at least, to swim into the desired branch before reaching the vessel bifurcation [87]. These examples are listed in Table 5.1, and the ratio κ, defined as $(l_1 v_2)/(d_1 v_1)$, falls in the range of [1.0, 2.0]. The properties of simulated vessels in this chapter are listed in Table 5.2, where the microrobot is assumed to move at 1 pixel/step. The flow velocity v_1 is derived from κ in Table 5.1, and it falls into the range [1.0,

3.1]. So the simulated flow is simplified to 3 pixel/step.

Table 5.1: Profiles of microchannels and microparticles.

Variable	**Microchannel**	1	2	3
l_1	Length (mm)	1.0e+0	1.0e+0	5.0e+2
d_1	Diameter (mm)	5.0e-3	1.0e-2	1.0e-1
v_1	Flow velocity (mm/s)	5.0e-1	5.0e-1	5.0e+1
	Microparticle			
d_2	Diameter (mm)	2.1e-3	4.2e-3	1.1e-2
v_2	Velocity (mm/s)	3.0e-3	1.0e-3	1.0e-1
	Ratio			
κ	$(l_1 v_2)/(d_1 v_1)$	1.2	2.0	1.0

Table 5.2: Profiles of microchannels and microparticles in this chapter. Note κ value inherits from Table 5.1, and the flow velocity is derived from κ.

Variable	**Microchannels**	1	2	3
l_1	Length (pixel)	101.0	68.9	78.7
d_1	Diameter (pixel)	32.6	33.1	31.0
v_1	Flow velocity (pixel/step)	[1.6, 3.1]	[1.0, 2.1]	[1.3, 2.5]
	Microrobot			
v_2	Velocity (pixel/step)	1.0	1.0	1.0
	Ratio			
κ	$(l_1 v_2)/(d_1 v_1)$	[1.0, 2.0]	[1.0, 2.0]	[1.0, 2.0]

5.3.2 Automatic Control Method

Figure 5.1 briefly illustrates the steering idea—the control input should be applied at the upstream of a bifurcation when there are microrobots approaching. This includes three steps: (1) microrobot detection near bifurcations; (2) steering direction identification based on the targeted outlet; (3) priority assignment if microrobots are detected at multiple bifurcations. These steps are illustrated in Figure 5.6. From the inlet to a targeted outlet, microrobots may travel through multiple bifurcations, and each bifurcation is assigned with a weight. There are five bifurcations along the desired trajectory in Figure 5.6. Their weights are denoted as w_i, $i \in \{1, 2, 3, 4, 5\}$, and the number of microrobots within each detection region is n_i. A bifurcation with the highest priority is determined by $\arg\max_{w_i}(w_i n_i)$, and this bifurcation is

in charge of the steering direction. In case $w_i a_i = 0 \ \ \forall i$, no control is applied.

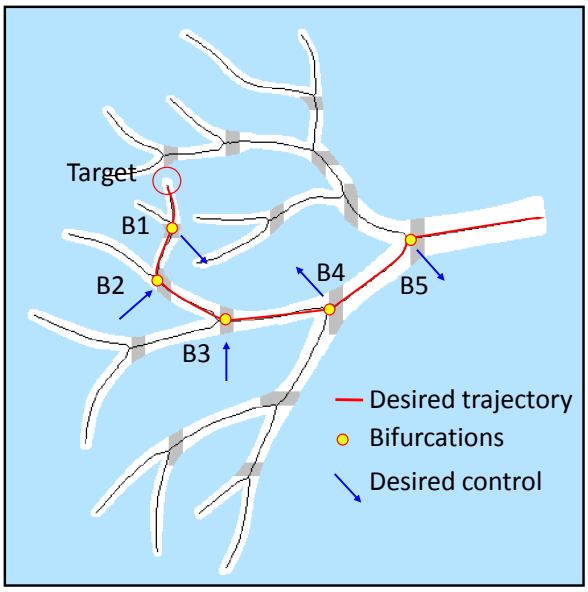


Figure 5.6: Illustration of map processing and automatic steering in Vessel II.

The *default* control assigns equal weight to each bifurcation, $w_i = 1$. Martel et al. suggested a learning-based scheme to navigate microparticles [107]. It relies on the injection of small amounts of magnetic microparticles to determine the optimal control sequences that maximize the steering effect. Then apply the control scheme to significant amounts of microparticles. Hence the *priority* control learns weights through trial and error, and selects the weights with the top delivery rate. This chapter chooses weights from a finite set $\{1, 2, 4, 8\}$.

5.3.3 Reinforcement Learning

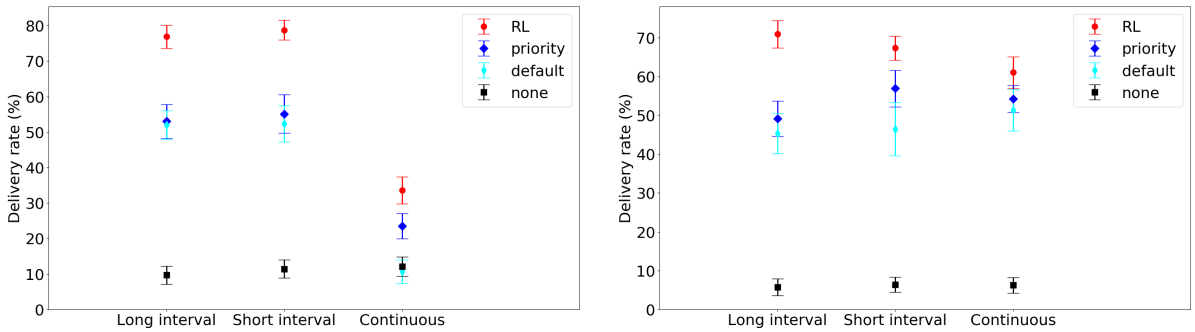
Reinforcement learning ICM algorithm (Chapter 4) is implemented to improve the delivery efficiency. The implementation details are reported in 4.4.1, except that only 32 parallel agents are used in this chapter, instead of 128. The RL agents are rewarded for any increase of the delivery rate. So maximizing the cumulative rewards is equivalent to maximizing the delivery rate.

5.4 Simulation

This section includes simulation results in different multi-channel vessels, microrobot distributions, and number of vessel bifurcations. The performance of automatic control methods and reinforcement learning are compared to evaluate their delivery efficiency.

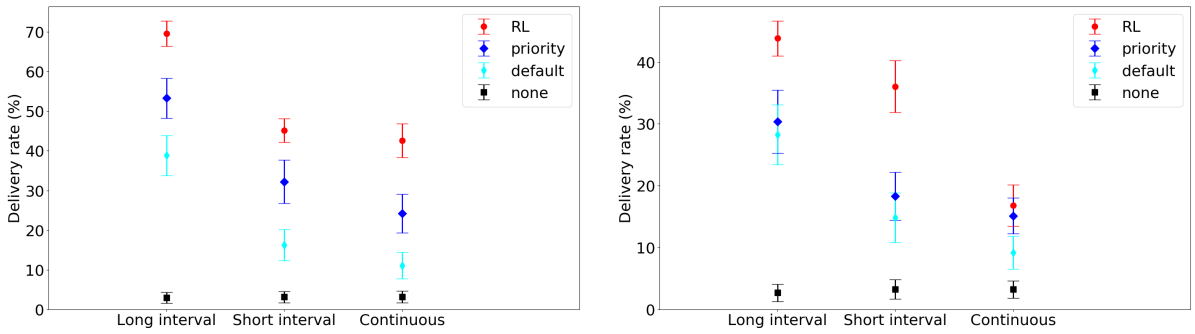
In each vessel, two targeted outlets are evaluated. Microrobots are released from the inlets at three intervals: long interval (64 time steps for Vessel I and II, 100 time steps for Vessel III), short interval (36 time steps for Vessel I and II, 60 time steps for Vessel III), and continuous. The total number of microrobots are 128 in Vessel I and II, or 256 in Vessel III, released in four times at intervals or continuously. The targets were selected such that the steering direction must change multiple times because the desired branch at each bifurcation alternates. A target reachable by a constant control input is just a corner case.

There are four steering strategies are evaluated in each vessel: RL, the priority control, the default control, and non-control. For each case, the priority control learns a set of weights defined in Section 5.3 through trial and error, and takes the weights with the maximum delivery rate. In the default control all weights are equal, and thus the bifurcation priority is decided by the number of microrobots detected. RL control refers to the strategies learned from training, and non-control receives no input.



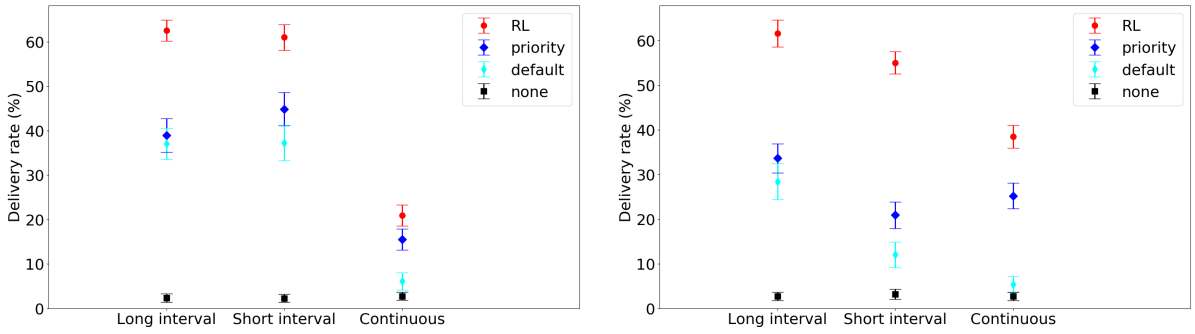
(a) (b)

Figure 5.7: Vessel I: comparison of delivery efficiency in two tasks (a) target 1 and (b) target 2, four steering methods, and three intervals of microrobot release.



(a) (b)

Figure 5.8: Vessel II: comparison of delivery efficiency in two tasks (a) target 1 and (b) target 2, four steering methods, and three intervals of microrobot release.



(a) (b)

Figure 5.9: Vessel III: comparison of delivery efficiency in two tasks (a) target 1 and (b) target 2, four steering methods, and three intervals of microrobot release.

The results are shown in Figure 5.7–5.9, with each data point based on 128 trials. It can be inferred that RL method achieves the maximum delivery rate in each scenario. Listed in Table 5.3, RL delivers 52.0% more microrobots on average compared to the average performance of the priority control, with the best performance of 163.4% more and the worst one 11.0% more. The default control delivers an average of 28.3% fewer microrobots compared to the average performance of the priority control, with the best performance 1.9% less and the worst 78.7% less. Table 5.4 indicates that a longer interval between microrobot releases significantly improves the delivery rate.

As shown in Figure 5.10–5.12, every RL training converges after 1e6 to 1e7 rollouts, despite the gaps between the average and the best performance. Such gaps may be caused by

Table 5.3: Comparison of the delivery efficiency.

ratio	mean	max	min
RL/priority	152.0%	263.4%	111.0%
default/priority	71.7%	98.1%	21.3%

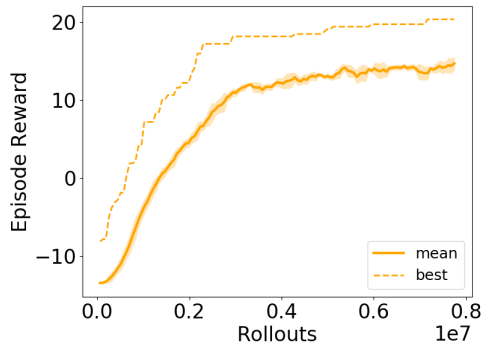
Table 5.4: Comparison of the delivery efficiency with respect to release intervals.

Control	long-interval	short-interval	continuous
RL	64.04%	57.22%	35.58%
priority	43.07%	38.05%	26.31%
default	38.32%	29.84%	15.62%
none	4.35%	4.94%	5.03%

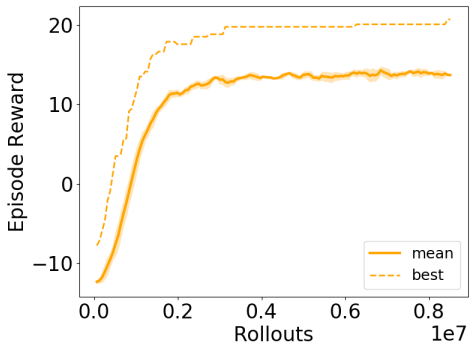
the stochastic motion in microrobots or random initializations.

5.5 Conclusion

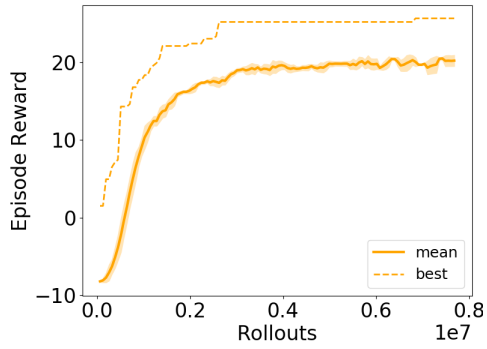
This chapter presents automatic control and reinforcement learning (RL) methods for microrobot swarm navigation in multi-branch vessels with flow. The automatic control methods, including the priority control and the default control, significantly improve the delivery rate to targets compared to non-control case. RL training results indicate there is space for future work aimed at increasing delivery efficiency. Both the priority control and RL control rely on offline training to achieve high delivery efficiency. The default control is provided for online planning, where the algorithm can be implemented in real-time operations. There are many aspects to explore for future work, such as extending the control input from the discrete space to the a continuous space, implementing the control algorithms on a more realistic simulation platform to demonstrate the effectiveness, where wall effects, Poiseuille flow, and microrobot interactions, and limited control power are considered, and demonstrating the control methods in hardware experiments.



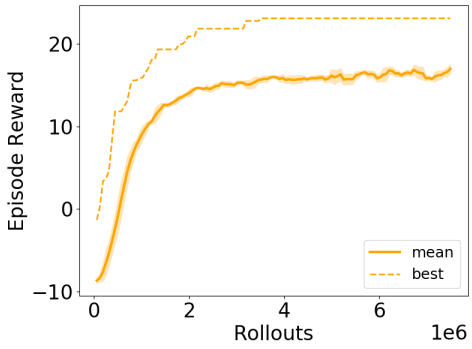
(a) Long intervals, target 1.
https://youtu.be/MVTredSna1M.



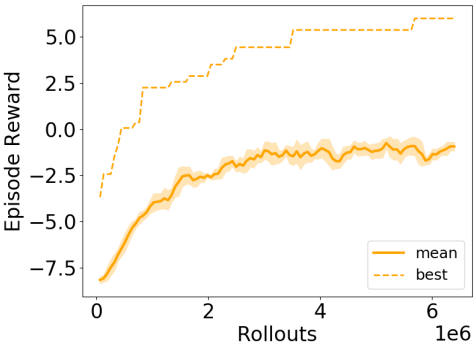
(b) Long intervals, target 2.
https://youtu.be/5UBUsMGSzH8.



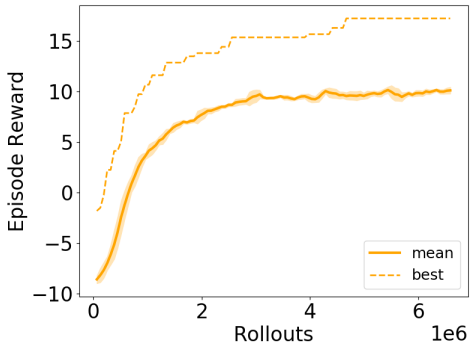
(c) Short intervals, target 1.
https://youtu.be/XFc9DFQji-0.



(d) Short intervals, target 2.
https://youtu.be/oz6sf6XSZ7A.

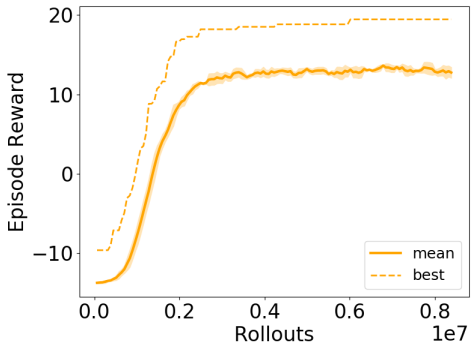


(e) Continuous, target 1.
https://youtu.be/PsoZMoTAHa4.

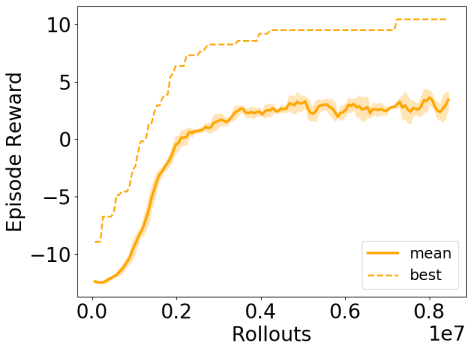


(f) Continuous, target 2.
https://youtu.be/zzWWJ4WnlbM.

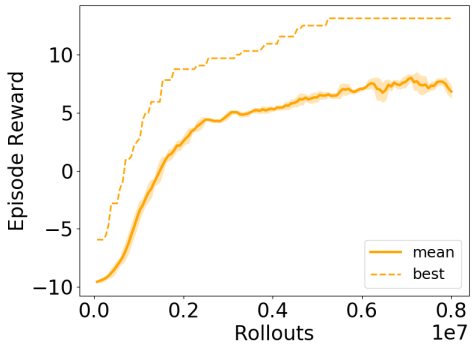
Figure 5.10: Vessel I: RL training results of different microrobot release intervals and targets. ‘mean’ and ‘max’ refer to the average and the best performance during updates.



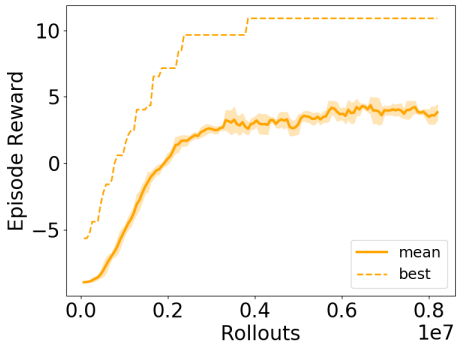
(a) Long intervals, target 1.
https://youtu.be/YFv5bHS9DWQ.



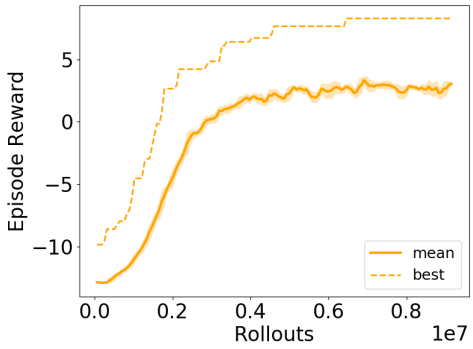
(b) Long intervals, target 2.
https://youtu.be/1J9F-1cdJDo.



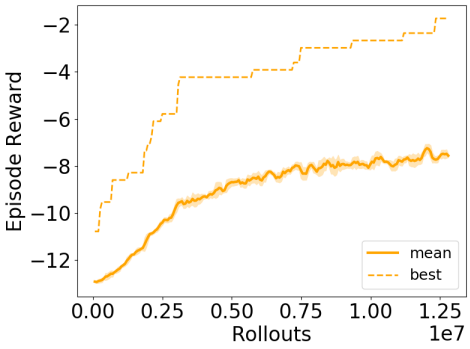
(c) Short intervals, target 1.
https://youtu.be/tp0nOTjSl1Y.



(d) Short intervals, target 2.
https://youtu.be/A2pG2hnZjh0.

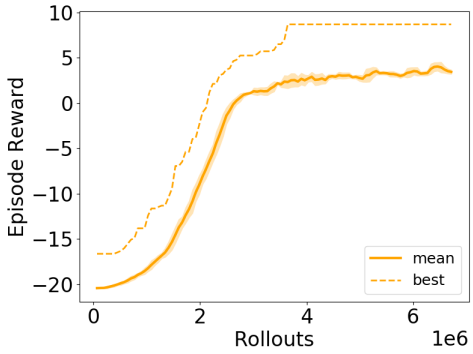


(e) Continuous, target 1.
https://youtu.be/Oe68t9SVP18.

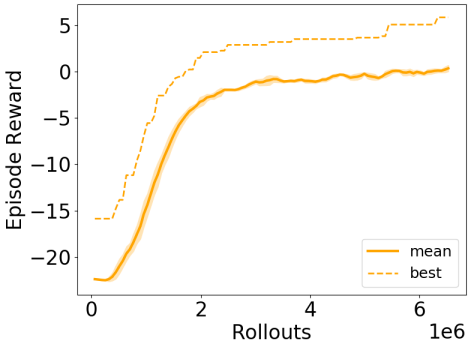


(f) Continuous, target 2.
https://youtu.be/4Ko-YkDPU0U.

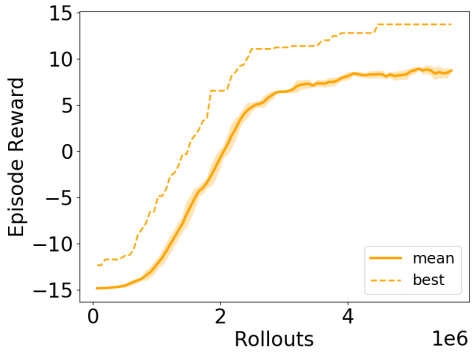
Figure 5.11: Vessel II: RL training results of different microrobot release intervals and targets. ‘mean’ and ‘max’ refer to the average and the best performance during updates.



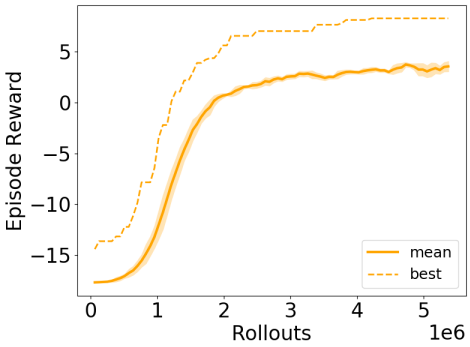
(a) Long intervals, target 1.
https://youtu.be/edF-DivriCo.



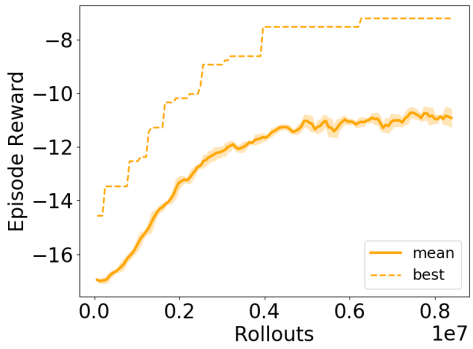
(b) Long intervals, target 2.
https://youtu.be/nH7XXKMXD-U.



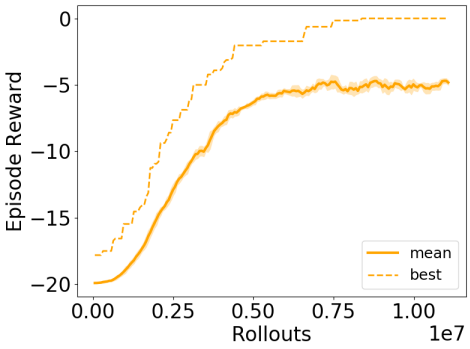
(c) Short intervals, target 1.
https://youtu.be/PPdxDHXkw3Q.



(d) Short intervals, target 2.
https://youtu.be/gqOcwhKg7SM.



(e) Continuous, target 1.
https://youtu.be/g0jpCXZdnJs.



(f) Continuous, target 2.
https://youtu.be/X4liqfdvIBE.

Figure 5.12: Vessel III: RL training results of different microrobot release intervals and targets. ‘mean’ and ‘max’ refer to the average and the best performance during updates.

Chapter 6

Conclusion

Microrobots are of great potential to be deployed in medical applications such as non-invasive surgeries and targeted drug delivery. These microrobots have the limited onboard source for actuation, communication, or computation, and thus an external, global control input is often required. Early-stage research, like path planning and control in the micro-scale, is essential for cost-effective medical applications. The goal of this dissertation is designing these strategies of navigating homogeneous microrobot swarm through vascular networks using a global control input. Chapter 2 proves limitations on the control for self-propelled agents (heterogeneous) that all receive the same rotation commands but extended the existing literature which focused on 2D results to show that nine degrees-of-freedom of position can be controlled in 3D. In 2D only one agent can be steered to an arbitrary position, and two agents have only one possible meeting point. In 3D up to three agents may be steered to arbitrary positions, and four agents have only one possible meeting point. Chapter 3 addresses path-planning and control problems in 2D vascular networks. Two path-planning methods and two control strategies are proposed to research the problem of homogeneous microrobot swarm delivery using a global input. The divide-and-conquer method reduces the delivery time and achieves considerable progress compared to the benchmark algorithm. Chapter 4 introduces deep reinforcement learning methods to improve further the delivery efficiency, where the reinforcement learning methods outperforms divide-and-conquer by 60% percent in terms of time steps for delivery. The reinforcement learning methods can quickly adapt to other environments without changing the algorithm, serving as a useful tool to direct optimization in the delivery strategy development. Chapter 5 proposes automatic steering methods to deliver microrobots in multi-channel vessels with flow. These methods help increase the delivery rate for microrobots in

a dynamic environment. Reinforcement learning methods are compared with the automatic control methods, and used to indicate directions for future work.

6.1 Future Work

This dissertation performs early-stage research for two primary scenarios in targeted drug delivery: static vascular networks and multi-channel vessels with flow. Multiple path-planning and control algorithms are designed to overcome challenges such as the global control input, identical microrobot swarm, and highly-constrained workspace. There are many directions to explore for future works. For automatic control, a more realistic simulation platform can be developed for accurate results, and thus making it more applicable to future clinic studies. Currently, the action space (control directions) only has up to eight discrete directions—this can be extended to continuous control space for flexible applications. Implementing these algorithms to hardware experiments is exciting, and can provide feedback for future algorithm design and optimization. There is still space for automatic control methods to catch up with the performance of reinforcement learning in delivery efficiency. For reinforcement learning, it is expected that hierarchical learning or transfer learning can be explored such that training in one vascular network can provide experience to learn strategies in other environments faster.

References

[1] S. Martel, J.-B. Mathieu, O. Felfoul, A. Chanu, E. Aboussouan, S. Tamaz, P. Pouponneau, L. Yahia, G. Beaudoin, and G. Soulez, "Automatic navigation of an untethered device in the artery of a living animal using a conventional clinical magnetic resonance imaging system," *Applied Physics Letters*, vol. 90, no. 11, p. 114105, 2007.

[2] D. R. Frutiger, B. E. Kratochvi, K. Vollmers, and B. J. Nelson, "Magmites-wireless resonant magnetic microrobots," in *IEEE International Conference on Robotics and Automation*. IEEE, 2008, pp. 1770–1771.

[3] S. Martel, "Magnetic navigation control of microagents in the vascular network: challenges and strategies for endovascular magnetic navigation control of microscale drug delivery carriers," *IEEE Control Systems Magazine*, vol. 33, no. 6, pp. 119–134, 2013.

[4] W. Jing, X. Chen, S. Lyttle, Z. Fu, Y. Shi, and D. J. Cappelleri, "A magnetic thin film microrobot with two operating modes," in *IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 96–101.

[5] S. Das, E. B. Steager, K. J. Stebe, and V. Kumar, "Simultaneous control of spherical microrobots using catalytic and magnetic actuation," in *Manipulation, Automation and Robotics at Small Scales (MARSS), 2017 International Conference on*. IEEE, 2017, pp. 1–6.

[6] S. Floyd, C. Pawashe, and M. Sitti, "Two-dimensional contact and noncontact micromanipulation in liquid using an untethered mobile magnetic microrobot," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1332–1342, 2009.

[7] E. Diller, S. Floyd, C. Pawashe, and M. Sitti, "Control of multiple heterogeneous magnetic microrobots in two dimensions on nonspecialized surfaces," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 172–182, 2012.

[8] I. S. Khalil, F. van den Brink, O. S. Sukas, and S. Misra, "Microassembly using a cluster of paramagnetic microparticles," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 5527–5532.

[9] M. S. Sakar, E. B. Steager, A. Cowley, V. Kumar, and G. J. Pappas, "Wireless manipulation of single cells using magnetic microtransporters," in *IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 2668–2673.

[10] J. Ali, U. K. Cheang, J. D. Martindale, M. Jabbarzadeh, H. C. Fu, and M. J. Kim, "Bacteria-inspired nanorobots with flagellar polymorphic transformations and bundling," *Scientific Reports*, vol. 7, no. 1, p. 14098, 2017.

[11] U. K. Cheang, H. Kim, D. Milutinović, J. Choi, and M. J. Kim, "Feedback control of an achiral robotic microswimmer," *Journal of Bionic Engineering*, vol. 14, no. 2, pp. 245–259, 2017.

[12] S. Martel, M. Mohammadi, O. Felfoul, Z. Lu, and P. Pouponneau, "Flagellated magnetotactic bacteria as controlled MRI-trackable propulsion and steering systems for medical nanorobots operating in the human microvasculature," *The International Journal of Robotics Research*, vol. 28, no. 4, pp. 571–582, 2009.

[13] S. Martel and M. Mohammadi, "Using a swarm of self-propelled natural microrobots in the form of flagellated bacteria to perform complex micro-assembly tasks," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 500–505.

[14] D. De Lanauze, O. Felfoul, J.-P. Turcot, M. Mohammadi, and S. Martel, "Three-dimensional remote aggregation and steering of magnetotactic bacteria microrobots for

drug delivery applications," *The International Journal of Robotics Research*, vol. 33, no. 3, pp. 359–374, 2014.

[15] A. Becker, Y. Ou, P. Kim, M. J. Kim, and A. Julius, "Feedback control of many magnetized: tetrahymena pyriformis cells by exploiting phase inhomogeneity," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 3317–3323.

[16] P. S. S. Kim, A. Becker, Y. Ou, A. A. Julius, and M. J. Kim, "Swarm control of cell-based microrobots using a single global magnetic field," in *Ubiquitous Robots and Ambient Intelligence (URAI), 2013 10th International Conference on*. IEEE, 2013, pp. 21–26.

[17] B. J. Nelson, I. K. Kaliakatsos, and J. J. Abbott, "Microrobots for minimally invasive medicine," *Annual Review of Biomedical Engineering*, vol. 12, pp. 55–85, 2010.

[18] U. Kei Cheang, K. Lee, A. A. Julius, and M. J. Kim, "Multiple-robot drug delivery strategy through coordinated teams of microswimmers," *Applied Physics Letters*, vol. 105, no. 8, p. 083705, 2014.

[19] D. Wong, E. B. Steager, and V. Kumar, "Independent control of identical magnetic robots in a plane," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 554–561, 2016.

[20] D. H. Kim, S. Brigandi, A. A. Julius, and M. J. Kim, "Real-time feedback control using artificial magnetotaxis with rapidly-exploring random tree (RRT) for tetrahymena pyriformis as a microbiorobot," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 3183–3188.

[21] I. S. Khalil, M. P. Pichel, B. A. Reefman, O. S. Sukas, L. Abelmann, and S. Misra, "Control of magnetotactic bacterium in a micro-fabricated maze," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 5508–5513.

[22] S. Scheggi and S. Misra, "An experimental comparison of path planning techniques applied to micro-sized magnetic agents," in *Manipulation, Automation and Robotics at Small Scales (MARSS), International Conference on*. IEEE, 2016, pp. 1–6.

[23] A. Pierson and M. Schwager, "Bio-inspired non-cooperative multi-robot herding," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1843–1849.

[24] B. T. Fine and D. A. Shell, "Eliciting collective behaviors through automatically generated environments," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 3303–3308.

[25] A. Becker, E. D. Demaine, S. P. Fekete, G. Habibi, and J. McLurkin, "Reconfiguring massive particle swarms with limited, global control," in *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*. Springer, 2013, pp. 51–66.

[26] L. Bobadilla, O. Sanchez, J. Czarnowski, K. Gossman, and S. M. LaValle, "Controlling wild bodies using discrete transition systems," *Advanced Robots*, 2011.

[27] A. V. Mahadev, D. Krupke, J.-M. Reinhardt, S. P. Fekete, and A. T. Becker, "Collecting a swarm in a grid environment using shared, global inputs," in *Automation Science and Engineering (CASE), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1231–1236.

[28] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt, "The robotarium: a remotely accessible swarm robotics research testbed," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1699–1706.

[29] J.-S. Li and N. Khaneja, "Ensemble control of bloch equations," *IEEE Transactions on Automatic Control*, vol. 54, no. 3, pp. 528–536, 2009.

[30] J.-S. Li, "Ensemble control of finite-dimensional time-varying linear systems," *IEEE Transactions on Automatic Control*, vol. 56, no. 2, pp. 345–357, 2011.

[31] J.-S. Li and N. Khaneja, "Control of inhomogeneous quantum ensembles," *Physical Review A*, vol. 73, no. 3, p. 030302, 2006.

[32] J. Ruths and S. Li, "Optimal ensemble control of open quantum systems with a pseudospectral method," in *Decision and Control (CDC), 2010 49th IEEE Conference on*. IEEE, 2010, pp. 3008–3013.

[33] A. Becker and T. Bretl, "Approximate steering of a plate-ball system under bounded model perturbation using ensemble control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5353–5359.

[34] J.-S. Li and N. Khaneja, "Ensemble control of linear systems," in *Decision and Control, 2007 46th IEEE Conference on*. IEEE, 2007, pp. 3768–3773.

[35] A. Becker and T. Bretl, "Approximate steering of a unicycle under bounded model perturbation using ensemble control," *IEEE Transactions on Robotics*, vol. 28, no. 3, pp. 580–591, 2012.

[36] J. Ruths and J.-S. Li, "Optimal control of inhomogeneous ensembles," *IEEE Transactions on Automatic Control*, vol. 57, no. 8, pp. 2021–2032, 2012.

[37] A. Becker and T. Bretl, "Motion planning under bounded uncertainty using ensemble control," in *Robotics: Science and Systems*, 2010.

[38] A. Becker, C. Onyuksel, and T. Bretl, "Feedback control of many differential-drive robots with uniform control inputs," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 2256–2262.

[39] A. Becker and J. McLurkin, "Exact range and bearing control of many differential-drive robots with uniform control inputs," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 3338–3343.

[40] S. Shahrokhi, A. Mahadev, and A. T. Becker, "Algorithms for shaping a particle swarm with a shared input by exploiting non-slip wall contacts," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 4304–4311.

[41] G. Roussos, D. V. Dimarogonas, and K. J. Kyriakopoulos, "3D navigation and collision avoidance for nonholonomic aircraft-like vehicles," *International Journal of Adaptive Control and Signal Processing*, vol. 24, no. 10, pp. 900–920, 2010.

[42] W. Xu, X. Lan, and Y. Wang, "Distance-based formation control of multi-agent systems moving in 3D space based on an amsp graph," in *37th Chinese Control Conference (CCC)*. IEEE, 2018, pp. 6676–6681.

[43] A. Nikou, C. K. Verginis, and D. V. Dimarogonas, "Robust distance-based formation control of multiple rigid bodies with orientation alignment," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 15 458–15 463, 2017.

[44] L. Huang and A. T. Becker, "Janus particle control simulation," https://github.com/RoboticSwarmControl/JanusParticleControl, 2018.

[45] A. T. Becker and J. Garcia, "Steering multiple radio control (RC) cars with one joystick," http://demonstrations.wolfram.com/SteeringMultipleRadioControlRCCarsWithOneJoystick/, 2018.

[46] A. T. Becker and L. Huang, "3D position control for a multi-agent system of self-propelled agents steered by a global control input," https://youtu.be/sSSQgnmjmJw, Feb 2019.

[47] T. Bretl, "Control of many agents using few instructions," in *Robotics: Science and Systems*, 2007.

[48] K. Das and D. Ghose, "Broadcast control mechanism for positional consensus in multiagent systems," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 5, pp. 1807–1826, 2015.

[49] D. H. Kim, U. K. Cheang, L. Kőhidai, D. Byun, and M. J. Kim, "Artificial magnetotactic motion control of tetrahymena pyriformis using ferromagnetic nanoparticles: a tool for fabrication of microbiorobots," *Applied Physics Letters*, vol. 97, no. 17, p. 173702, 2010.

[50] D. H. Kim, D. Casale, L. Kőhidai, and M. J. Kim, "Galvanotactic and phototactic control of tetrahymena pyriformis as a microfluidic workhorse," *Applied Physics Letters*, vol. 94, no. 16, p. 163901, 2009.

[51] Y. Ou, D. H. Kim, P. Kim, M. J. Kim, and A. A. Julius, "Motion control of magnetized tetrahymena pyriformis cells by a magnetic field with model predictive control," *The International Journal of Robotics Research*, vol. 32, no. 1, pp. 129–140, 2013.

[52] D. Hyung Kim, P. Seung Soo Kim, A. Agung Julius, and M. Jun Kim, "Three-dimensional control of tetrahymena pyriformis using artificial magnetotaxis," *Applied Physics Letters*, vol. 100, no. 5, p. 053702, 2012.

[53] D. Loghin, C. Tremblay, M. Mohammadi, and S. Martel, "Exploiting the responses of magnetotactic bacteria robotic agents to enhance displacement control and swarm formation for drug delivery platforms," *The International Journal of Robotics Research*, vol. 36, no. 11, pp. 1195–1210, 2017. [Online]. Available: https://doi.org/10.1177/0278364917728331

[54] O. Felfoul, M. Mohammadi, S. Taherkhani, D. De Lanauze, Y. Z. Xu, D. Loghin, S. Essa, S. Jancik, D. Houle, and M. Lafleur, "Magneto-aerotactic bacteria deliver drug-

containing nanoliposomes to tumour hypoxic regions," *Nature Nanotechnology*, vol. 11, no. 11, p. 941, 2016.

[55] O. Felfoul and S. Martel, "Assessment of navigation control strategy for magnetotactic bacteria in microchannel: toward targeting solid tumors," *Biomedical Microdevices*, vol. 15, no. 6, pp. 1015–1024, 2013.

[56] L. Baraban, D. Makarov, R. Streubel, I. Monch, D. Grimm, S. Sanchez, and O. G. Schmidt, "Catalytic janus motors on microfluidic chip: deterministic motion for targeted cargo delivery," *ACS Nano*, vol. 6, no. 4, pp. 3383–3389, 2012.

[57] L. Baraban, D. Makarov, O. G. Schmidt, G. Cuniberti, P. Leiderer, and A. Erbe, "Control over janus micromotors by the strength of a magnetic field," *Nanoscale*, vol. 5, no. 4, pp. 1332–1336, 2013.

[58] I. S. Khalil, V. Magdanz, S. Sanchez, O. G. Schmidt, and S. Misra, "Precise localization and control of catalytic janus micromotors using weak magnetic fields," *International Journal of Advanced Robotic Systems*, vol. 12, no. 1, p. 2, 2015.

[59] G. Tiwari, R. Tiwari, B. Sriwastawa, L. Bhati, S. Pandey, P. Pandey, and S. K. Bannerjee, "Drug delivery systems: an updated review," *International Journal of Pharmaceutical Investigation*, vol. 2, no. 1, p. 2, 2012.

[60] L. E. Kavraki and J.-C. Latombe, "Probabilistic roadmaps for robot path planning," 1998.

[61] S. M. LaValle, "Rapidly-exploring random trees: a new tool for path planning," 1998.

[62] L. Zhang and D. Manocha, "An efficient retraction-based RRT planner," in *Robotics and Automation (ICRA), 2008 IEEE International Conference on*. IEEE, 2008, pp. 3743–3750.

[63] X. Tang, J.-M. Lien, and N. Amato, "An obstacle-based rapidly-exploring random tree," in *Robotics and Automation (ICRA), 2006 Proceedings IEEE International Conference on*. IEEE, 2006, pp. 895–900.

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

[65] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[66] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, and A. Bolton, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[67] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[68] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[69] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, and M. Riedmiller, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017.

[70] J. Merel, Y. Tassa, S. Srinivasan, J. Lemmon, Z. Wang, G. Wayne, and N. Heess, "Learning human behaviors from motion capture by adversarial imitation," *arXiv preprint arXiv:1707.02201*, 2017.

[71] Z. Wang, J. S. Merel, S. E. Reed, N. de Freitas, G. Wayne, and N. Heess, "Robust imitation of diverse behaviors," in *Advances in Neural Information Processing Systems*, 2017, pp. 5320–5329.

[72] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[73] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[74] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[75] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[76] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[77] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[78] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," https://github.com/openai/baselines, 2017.

[79] Y. Wu, E. Mansimov, S. Liao, A. Radford, and J. Schulman, "Openai baselines," https://openai.com/blog/baselines-acktr-a2c/, 2017.

[80] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.

[81] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[82] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 16–17.

[83] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, "Large-scale study of curiosity-driven learning," *arXiv preprint arXiv:1808.04355*, 2018.

[84] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2018.

[85] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, "Large-scale curiosity," https://github.com/openai/large-scale-curiosity, 2018.

[86] J. Mathieu and S. Martel, "Steering of aggregating magnetic microparticles using propulsion gradients coils in an MRI scanner," *Magnetic Resonance in Medicine*, vol. 63, no. 5, pp. 1336–1345, 2010.

[87] J.-B. Mathieu and S. Martel, "Magnetic microparticle steering within the constraints of an MRI system: proof of concept of a novel targeting approach," *Biomedical Microdevices*, vol. 9, no. 6, pp. 801–808, 2007.

[88] A. Bigot, C. Tremblay, G. Soulez, and S. Martel, "Magnetic resonance navigation of a bead inside a three-bifurcation pmma phantom using an imaging gradient coil insert," *IEEE Transactions on Robotics*, vol. 30, no. 3, pp. 719–727, 2014.

[89] P. Pouponneau, J.-C. Leroux, G. Soulez, L. Gaboury, and S. Martel, "Co-encapsulation of magnetic nanoparticles and doxorubicin into biodegradable microcarriers for deep

tissue targeting by vascular MRI navigation," *Biomaterials*, vol. 32, no. 13, pp. 3481–3486, 2011.

[90] A. Hoshiar, T.-A. Le, F. Amin, M. Kim, and J. Yoon, "A novel magnetic actuation scheme to disaggregate nanoparticles and enhance passage across the blood-brain barrier," *Nanomaterials*, vol. 8, no. 1, p. 3, 2017.

[91] A. K. Hoshiar, T.-A. Le, F. U. Amin, M. O. Kim, and J. Yoon, "Studies of aggregated nanoparticles steering during magnetic-guided drug delivery in the blood vessels," *Journal of Magnetism and Magnetic Materials*, vol. 427, pp. 181–187, 2017.

[92] V. Hamdipoor, M. Afzal, T.-A. Le, and J. Yoon, "Haptic-based manipulation scheme of magnetic nanoparticles in a multi-branch blood vessel for targeted drug delivery," *Micromachines*, vol. 9, no. 1, p. 14, 2018.

[93] T. D. Do, Y. Noh, M. O. Kim, and J. Yoon, "An optimized field function scheme for nanoparticle guidance in magnetic drug targeting systems," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 4388–4393.

[94] C. Pacchierotti, V. Magdanz, M. Medina-Sánchez, O. G. Schmidt, D. Prattichizzo, and S. Misra, "Intuitive control of self-propelled microjets with haptic feedback," *Journal of Micro-bio Robotics*, vol. 10, no. 1-4, pp. 37–53, 2015.

[95] M. Larimi, A. Ramiar, and A. Ranjbar, "Numerical simulation of magnetic nanoparticles targeting in a bifurcation vessel," *Journal of Magnetism and Magnetic Materials*, vol. 362, pp. 58–71, 2014.

[96] A. Boghi, F. Russo, and F. Gori, "Numerical simulation of magnetic nano drug targeting in a patient-specific coeliac trunk," *Journal of Magnetism and Magnetic Materials*, vol. 437, pp. 86–97, 2017.

[97] S. Kenjereš and B. Righolt, "Simulations of magnetic capturing of drug carriers in the brain vascular system," *International Journal of Heat and Fluid Flow*, vol. 35, pp. 68–75, 2012.

[98] M. Larimi, A. Ramiar, and A. Ranjbar, "Numerical simulation of magnetic drug targeting with Eulerian-Lagrangian model and effect of viscosity modification due to diabetics," *Applied Mathematics and Mechanics*, vol. 37, no. 12, pp. 1631–1646, 2016.

[99] A. Nacev, C. Beni, O. Bruno, and B. Shapiro, "The behaviors of ferromagnetic nanoparticles in and around blood vessels under applied magnetic fields," *Journal of Magnetism and Magnetic Materials*, vol. 323, no. 6, pp. 651–668, 2011.

[100] S. Sharma, V. Katiyar, and U. Singh, "Mathematical modelling for trajectories of magnetic nanoparticles in a blood vessel under magnetic field," *Journal of Magnetism and Magnetic Materials*, vol. 379, pp. 102–107, 2015.

[101] M. Momeni Larimi, A. Ramiar, and A. A. Ranjbar, "Magnetic nanoparticles and blood flow behavior in non-newtonian pulsating flow within the carotid artery in drug delivery application," *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, vol. 230, no. 9, pp. 876–891, 2016.

[102] Adobe Stock: happyvector071, "File: 204976905, creative vector illustration of red veins isolated on background. human vessel, health arteries, art design. abstract concept graphic element capillaries. blood system," https://stock.adobe.com/images/creative-vector-illustration-of-red-veins-isolated-on-background-human-vessel-health-arteries-art-design-abstract-concept-graphic-element-capillaries-blood-system/204976905, [Online; accessed July 9, 2019].

[103] Mikhail Grachikov, "Vector id: 172739230, creative vector illustration of red veins isolated on background. human vessel, health arteries, art design. abstract concept graphic element capillaries. blood system," 2017, [Online; accessed July 9, 2019].

[104] Adobe Stock: happyvector071, "File: 204977507, creative vector illustration of red veins isolated on background. human vessel, health arteries, art design. abstract concept graphic element capillaries. blood system," https://stock.adobe.com/images/creative-vector-illustration-of-red-veins-isolated-on-background-human-vessel-health-arteries-art-design-abstract-concept-graphic-element-capillaries-blood-system/204977507, [Online; accessed July 9, 2019].

[105] L. Huang, "Vascular networks for microrobot swarm path planning," https://github.com/lihuang3/MazeEnv, 2018.

[106] S. van der Walt, J. L. Schnberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors, "Scikit-image: image processing in python," https://peerj.com/articles/453/, 2014.

[107] S. Martel, J.-B. Mathieu, O. Felfoul, A. Chanu, E. Aboussouan, S. Tamaz, P. Pouponneau, L. Yahia, G. Beaudoin, and G. Soulez, "A computer-assisted protocol for endovascular target interventions using a clinical MRI system for controlling untethered microdevices and future nanorobots," *Computer Aided Surgery*, vol. 13, no. 6, pp. 340–352, 2008.