NORTHWESTERN UNIVERSITY

Co-Design of Bodies and Strategies

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Mechanical Engineering

By

Ana Pervan

EVANSTON, ILLINOIS

December 2021

ABSTRACT

Co-Design of Bodies and Strategies

Ana Pervan

The field of robot design mostly focuses on careful construction of complex control and planning algorithms (e.g., tuning neural network weights) which bear sole responsibility for improving task performance, while the robot's body is often assumed to be part of the environment. In nature, however, biological organisms co-evolve both their neurological capabilities and their physical morphology to improve their chances for survival. This enables information and intelligence to be encoded not only in a centralized brain, but also in a distributed body. This thesis focuses on co-design of control algorithms and physical robot bodies.

This thesis begins by introducing elements of robot design on a minimal, micro-robotic system. I analyze tasks (like micro-manipulation and target localization) in terms of the fundamental capabilities and information required to achieve them, and use those to generate robot designs ideally consisting of only the most essential components necessary for the task. The resulting designs are compared in terms of task performance and design complexity. These principles are then applied to the design of a group of robots, in

which the collective system demonstrates emergent behaviors that the individuals are incapable of. Designing for ensemble behaviors requires predictions and analysis of inter-robot communication and collaboration. I compare the results of the collaborative robots with individuals attempting the same task to show that the emergent behaviors greatly benefit the system. I conclude this thesis by outlining a macroscopic extension of the contributions in which a robot co-designs a flexible tool by bending it into a shape that both informs and is informed by the control algorithm defining how the tool will be used to achieve a goal.

Acknowledgements

First and foremost, I'd like to thank my advisor Todd Murphey. For five years of unwavering support, actionable advice, trust in my instincts, intervention when necessary, and a genuine enthusiasm for interesting research ideas and the philosophy of math and engineering. You've inspired me to investigate what I find exciting and to value my own work. Your support and mentorship has made me a better researcher and scientist.

To my lab mates and collaborators, old and new: thank you for motivating me, teaching me, listening to me, and for your kindness, intelligence, and friendship. Thank you especially to Tommy Berrueta, for our philosophical ~~arguments~~ discussions (can a rock learn?) which are some of my fondest memories from my PhD. And thank you to Ola Kalinowska and Hannah Janssen, for being my friends and confidants — from classes and homework to research and conferences to weekends and brunches and life.

To my parents, Boris and Sherry Pervan, thank you for motivating me to pursue my PhD in the first place. For encouraging my scientific curiosity and teaching me to think for myself. You both truly inspire me and I'm lucky to have such amazing role models.

And lastly, thank you to Justin Wolfington. Thank you for valuing my work and pushing me to do my best. Thank you for being there for me during every failure and every success, and for keeping me (mostly) sane during these unprecedented times. This wouldn't have been possible without you, and I am endlessly grateful for you.

Table of Contents

List of Figures

CHAPTER 1

Introduction

One of the goals of the field of robotics is to generate autonomous systems that can perform tasks without external influence. To achieve this, roboticists must understand the information that a robot needs to solve a given task, and how the robot will receive that information and act on it.

Robots traditionally require three elements: sensors, computation, and actuators. Sensors record all inputs to the robot, ranging from video of a crowded street to the degree at which the robot's own knee is bent. Computation encompasses all of the planning algorithms and learning strategies that a robot might undertake in order to achieve its task. Actuation comprises any output from the robot, including picking up a block, driving a car, or saying hello.

In many instances, roboticists seek to develop autonomy by beginning with a task specification and a preexisting standard robot (e.g., a Sawyer arm) and developing often complex code, controllers, and algorithms to enable the robot to achieve the task. What if we were able to design not only the "brains" of the robot, but also the body? Instead of creating a control strategy for a robot based on the sensors and actuators it has, a potential control strategy could inform one's choice of sensors and actuators.

Simultaneous design of both the hardware and software of a robot could produce outcomes similar to evolution, where animals' brains and bodies develop concurrently to better accomplish their goals (survival and reproduction). This thesis investigates avenues and applications for **co-design** of both the physical body and planned strategy of robots.

1.1. Main Contributions

This thesis examines robot design with respect to a desired task. It explores the information required to accomplish a goal and how to algorithmically choose components from a library of possible sensors and actuators. Elements like design complexity, task performance, memory, and multi-agent systems are taken into account. Through theoretical analysis, simulations, and physical experiments, I show strategies for designing robots across scales for specific tasks and explore the impact of those choices using various metrics of task performance. *This thesis introduces methods and examples for autonomous co-design of physical robots and their control strategies, as well as validation of their task performance.*

1.1.1. Primitives and Logic for Robot Comparisons

In the first chapter, a task-centered formal analysis of the relative power of several robot designs is presented, inspired by the unique properties and constraints of micro-scale robotic systems. The task of interest is object manipulation. I present minimal conditions on the sensing, memory, and actuation requirements of periodic "bouncing" robot trajectories that move an object in a desired direction through the incidental forces arising from robot-object collisions. Several robot designs are compared using an information space framework and a hierarchical controller, emphasizing the information requirements of goal completion under different initial conditions, as well as what is required to recognize irreparable task failure.

The contributions of this chapter are as follows:

(1) I present a novel demonstration of the information-based approach in [**68**] for analyzing and comparing different robot designs with respect to an object manipulation task

(2) I describe a hierarchical method to combine high-level task completion guarantees with components of low-level controllers using information requirements of goal completion

(3) I use an information-based approach to determine what is required to recognize irreparable task failure.

I would like to acknowledge Alexandra Nilles, who introduced boundary interactions (which define a robot's motion strategy when it collides with objects) and analyzed the feasibility and dynamics of cyclic motion strategies, as well as Thomas Berrueta, who derived information space relationships for minimal robots and performed formal analysis of our system. This work was published in [**77**].

1.1.2. Algorithmic Design of Synthetic Cells

In robotics, complex control and planning algorithms often bear sole responsibility for improving task performance. This dependence on centralized control can be problematic for systems with computational limitations, such as mechanical systems and robots on the microscale. In these cases, we need to be able to offload complex computation onto the physical morphology of the system. In this chapter, I introduce an algorithm that arranges sensing and actuation components into a robot design, while enforcing a low level of control policy complexity. This enables the resulting robot designs to work offline – directly mapping sensory observations to actions – and on systems without computation available, while still maintaining a high level of task performance.

The contributions of this chapter are as follows:

(1) I present quantitative definitions for design complexity and task embodiment
(2) I develop an iterative algorithm to create control policies with low design complexity while increasing task information
(3) I introduce a mathematical projection operator which projects a low complexity control policy onto a physically realizable set of sensor-actuator interconnections
(4) I establish a methodology for algorithmically organizing components for robot design. The procedure begins either with a control policy based on a discrete set of actuators and interconnects them with different possible sets of sensors or begins with a control policy generated with a discrete set of sensors and combines them with a selection of discrete actuators.

This work was published in [**73**], [**74**], and [**76**].

1.1.3. Bayesian Particles

The work presented in this chapter is inspired by biological processes in which components (e.g., cells) are individually simple, but work together to create unimaginably complex structures and functions – like the mammalian immune response. The goal is to build simple agents that collectively exhibit emergent behaviors.

In this chapter, I design minimal agents which use only a few bits of memory to search for targets in a dynamic, stochastic environment. The exhibited behavior of the collective agents functions as a Bayesian update: the agents play the role of the individual particles in a particle filter. These results were validated in experiments using twelve driving robots with infrared sensors and radio communication.

The contributions of this chapter are as follows:

(1) I show emergence of global learning behavior from simple agents executing local algorithms and using only a few bits of memory
(2) I prove that this implementation of simple agents acts as a suboptimal Bayesian filter, and therefore that the system inherits formal properties in the form of guarantees on asymptotic performance and probabilistically predictable behavior
(3) I validate theoretical and simulated results in experiments with a macroscopic robotic swarm system.

Joshua Cohen designed (and sourced, manufactured, coded, and tested) the swarm robots that were used in the physical experiments. A custom-built system of collaborative robots enabled us to run experiments under the exact conditions needed. Karalyn Baird lead the physical experiments and data collection, which enabled analysis that validated our theoretical claims and simulated results. Jamison Weber and Professor Andrea Richa developed the definition of Uniformly Oriented (UO) graphs, and proved convergence rates and mixing times for various cases.

Some of this work has been published in [**75**] and another publication is in preparation [**78**].

1.1.4. Flexible Tool Design

Although robot design and capabilities are simplest to analyze in minimal, discrete systems, these methods and conclusions can also be applied to macroscopic, continuous examples. In this chapter, I introduce a methodology for designing a flexible tool. I use the extended example of a bendable wire being manipulated by a dual-armed robot. The robot grasps, bends, and re-grasps the rod-shaped tool while modelling the rod's configuration to create a desired shape. This type of geometric design (as opposed to a component-selection type of design) still requires the robot to co-design the morphology of the tool as well as its planned use of the tool.

The contributions of this chapter are as follows:

(1) I present results of a neural network model for reducing the dimensionality of the configuration of a continuous flexible tool
(2) I demonstrate simulated examples of a robot grasping a flexible tool to bend it into a desired shape
(3) I outline a framework and experimental plan for data-driven learning – bootstrapped by optimization-based simulations – for tool design and manipulation for task execution.

1.2. Thesis Outline

Chapter 2 will be an examination of robot primitives as the most fundamental behaviors and building blocks of a robot. These are used in an extended example as well as a discussion of the information requirements and capabilities of robots. Chapter 3 furthers these ideas and applies them to the design and selection of sensors and actuators for a robot attempting a task. Chapter 4 investigates the application of low complexity design to collectives of robots, and how the existence of multiple agents affects the design of the individuals. Lastly, Chapter 5 discusses future directions and work toward co-design on a macroscopic scale, in which a robot must simultaneously design both the shape of a flexible tool and its strategy for how to use the tool.

CHAPTER 2

Primitives and Logic for Robot Comparisons

In this chapter, a task-centered formal analysis of the relative power of several robot designs is presented, inspired by the unique properties and constraints of micro-scale robotic systems. The task of interest is object manipulation because it is a fundamental prerequisite for more complex applications such as micro-scale assembly or cell manipulation. Motivated by the difficulty in observing and controlling agents at the micro-scale, I, along with my collaborators Alexandra Q. Nilles and Thomas A. Berrueta, focus on the design of *boundary interactions*: the robot's motion strategy when it collides with objects or the environment boundary, otherwise known as a *bounce rule*. I present minimal conditions on the sensing, memory, and actuation requirements of periodic "bouncing" robot trajectories that move an object in a desired direction through the incidental forces arising from robot-object collisions. Several robot designs are compared using an information space framework and a hierarchical controller, emphasizing the information requirements of goal completion under different initial conditions, as well as what is required to recognize irreparable task failure. Finally, a physically-motivated model of boundary interactions is presented, and the robustness and dynamical properties of resulting trajectories are analyzed.

2.1. Introduction

Robots at the micro-scale have unique constraints on the amount of possible on-board information processing. Despite this limitation, future biomedical applications of micro-robots, such as drug delivery, tissue grafting, and minimally invasive surgery, demand sophisticated locomotion, planning, and manipulation [**93, 94**]. While certain robotic systems have succeeded at these tasks with assistance from external sensors and actuators [**104**], the minimal sensing and actuation requirements of these tasks are not well-understood.

Ideally, designs at this scale would not require fine-grained, individual motion control, due to the extreme difficulty in observing and communicating with individual agents. In fact, micro-scale locomotion is often a direct consequence of the fixed or low degree-of-freedom morphology of the robot, suggesting that direct co-design of robots and their motion strategies may be necessary [**19**]. The work presented in this chapter provides the beginning of a theory of task-centered robot design, used to devise micro-robot morphology and propulsion mechanisms. In order to inform the design of task-capable micro-robotic platforms, the information requirements of tasks will be analyzed [**31**].

The goal task will be *micromanipulation* with micro-robots, a fundamental task underlying more complex procedures such as drug delivery and cell transplantation [**54**]. In this chapter, I investigate micro-robot motion strategies that explicitly use *boundary interactions*: the robot's action when it encounters an environment boundary. Identifying minimal information requirements in this setting is essential for reasoning about robot performance, and is also a first step toward automating co-design of robots and their policies. In Section 2.3, physically motivated models are defined, aiming to roughly cover an interesting set of possible micro-robot realizations. Section 2.4 motivates the task of interest and the assumptions. In Section 2.5, several robot designs are compared and results on requirements for task completion are stated with respect to controller complexity, sensor power, and onboard memory.

2.2. Related Work

The scientific potential of precise manipulation at microscopic scales has been appreciated for close to a century [**21**]. As micro/nano-robots have become increasingly sophisticated, biomedical applications such as drug delivery [**54**] and minimally-invasive surgery [**94**] have emerged as grand challenges in the field [**107**].

To formally analyze the information requirements of planar micromanipulation, many physically-motivated actuators and sensors, such as odometry, range-sensing, differential-drive locomotion, and others are defined abstractly. Additionally, similarly to the work in [**31**], we avoid tool-specific manipulation by purposefully abstracting robot-object contacts, in an effort to be more general. However, the focus of this work is on collision-based manipulation instead of push-based. Recent publications have illustrated the value of robot-boundary collisions in generating reliable and robust robot behaviors [**48, 88**]. Thus, a model of collision-based micromanipulation that can serve as a test-bed for micro-robotic designs is developed and analyzed.

Despite substantial advances in micro-robotics, agents at micro/nano length-scales face fundamental difficulties and constraints. For one, as robots decrease in size and mass, common components such as motors, springs and latches experience trade-offs in force/velocity generation that limit their output and constrain the space of feasible mechanical designs [**42**]. Furthermore, battery capacities and efficiencies [**69**], as well as charging and power harvesting [**29**], experience diminishing performance at small scales, which restrict electrical designs of micro-machines. These limitations collectively amount to a rebuke of traditionally complex robotic design at the length-scales of interest, and suggest that a minimalist approach may be necessary. The minimal approach to robot design asks, "what is the most simple robot that can accomplish a given task?"

Minimalist approaches to microrobot design often exploit the close relationship between morphology and computation. For example, a DNA-based nano-robot is capable of capturing and releasing molecular payloads in response to a binary stimulus [**32**]. The inclusion of a given set of sensors or actuators in a design can enable robotic agents to sidestep complex computation and planning in favor of direct observation or action. Alternatively, in top-down approaches the formulation of high-level control policies can guide the physical design of robots [**73**]. While many approaches have seen success in their respective domains, the problem of optimizing the design of a robot subject to a given task has been shown to be NP-hard [**87**].

Due to information constraints, designing control policies for minimal robots remains a challenge. In order to achieve complex tasks, controllers are often hand-tuned to take advantage of the intrinsic dynamics of the system. For example, in [**1**] the authors show that one can develop controllers for minimal agents (solely equipped with a clock and a contact sensor) that can achieve spatial navigation, coverage, and localization by taking advantage of the details of the agents' dynamics. Hence, in order to develop control strategies amenable to the constraints of such robots, one requires substantial analytical understanding of the capabilities of minimal robots.

A unifying theory of robotic capabilities was established in [**68**]. The authors develop an information-based approach to analyzing and comparing different robot designs with respect to a given task. The key insight lies in distilling the minimal information requirements for a given task and expressing them in an appropriately chosen information space for the task. Then, as long as one is capable of mapping the individual information histories of different robot designs into the task information space, the performance of robots may be compared.

The main contribution of this work is a novel demonstration of the approach in [**68**] to an object manipulation task. Additionally, a hierarchical approach to combine the resulting high-level task completion guarantees with results on the robustness of low-level controllers is demonstrated.

2.3. Model and Definitions

Next, I introduce relevant abstractions for characterizing robots generally, as well as their capabilities for given tasks. This is largely followed from the work of [**51, 68**].

2.3.1. Primitives and Robots

In this work, a robot is modelled as a point in the plane; this model has obvious limitations, but captures enough to be useful for many applications, especially *in vitro* where the robot workspace is often a thin layer of fluid. Hence, its configuration space is $X \subseteq SE(2)$, and its configuration is represented as (x, y, θ). The robot's environment is $E \subseteq \mathbb{R}^2$, along with a collection of lines representing boundaries; these may be one-dimensional "walls" or bounded polygons. The environment may contain objects that will be static unless acted upon.

Following the convention of [**68**], I define a robot through sets of primitives. A *primitive*, P_i, defines a "mode of operation" of a robot, and is a 4-tuple $P_i = (U_i, Y_i, f_i, h_i)$ where U_i is the action set, Y_i is the observation set, $f_i : X \times U_i \to X$ is the state transition function, and $h_i : X \times U_i \to Y_i$ is the observation function. Primitives may correspond to use of either a sensor or an actuator, or both if their use is simultaneous (see [**51**] for examples). Time will be modeled as proceeding in discrete stages. At stage k the robot occupies a configuration $x_k \in X$, observes sensor reading $y_k^i \in Y_i$, and chooses its next action $u_k^i \in U_i$ for each i primitive in its set. A *robot* may then be defined as a 5-tuple $R = (X, U, Y, f, h)$ comprised of the robot's configuration space in conjunction with the elements of the primitives 4-tuples. With some abuse of notation, I occasionally write robot definitions as $R = \{P_1,\ ...,\ P_N\}$ when robots share the same configuration space to emphasize differences between robot capabilities.

2.3.2. Information Spaces

A useful abstraction to reason about robot behavior in the proposed framework is the information space (I-space). Information spaces are defined according to actuation and sensing capabilities of robots, and depend closely on the robot's history of actions and measurements. We denote the *history* of actions and sensor observations at stage k as $(u_1, y_1, \ldots, u_k, y_k)$. The history, combined with initial conditions $\eta_0 = (u_0, y_0)$, yields the *history information state*, $\eta_k = (\eta_0, u_1, y_1, \ldots, u_k, y_k)$. In this framework, initial conditions may either be the exact starting state of the system, or a set of possible starting states, or a prior belief distribution. The collection of all possible information histories is known as the *history information space*, $\mathcal{I}_{hist}$. It is important to note that a robot's history I-space is intrinsically defined by the robot primitives and its initial conditions. Hence, it is not generally fruitful to compare the information histories of different robots.

Derived information spaces should be constructed to reason about the capabilities of different robot designs. A derived I-space is defined by an *information map* $\kappa : \mathcal{I}_{hist} \to \mathcal{I}_{der}$ that maps histories in the history I-space to states in the derived I-space $\mathcal{I}_{der}$. Mapping different histories to the same derived I-space allows direct comparisons between different robots. The exact structure of the derived I-space depends on the task of interest; an abstraction must be chosen that allows for both meaningful comparison of the robots as well as determination of task success.

In order to be able to compare robot trajectories within the derived I-space, I introduce an *information preference relation* to distinguish between derived information states [**68**]. We discriminate these information states based on a distance metric to a given goal region $\mathcal{I}_G \subseteq \mathcal{I}_{der}$ which represents success for a task. Using a relation of this type "preference" over information states can be assessed, notated as $\kappa(\eta^{(1)}) \preceq \kappa(\eta^{(2)})$ if an arbitrary $\eta^{(2)}$ is preferred over $\eta^{(1)}$.

2.3.3. Robot Dominance

Here, a relation is defined that captures a robot's ability to "simulate" another given a policy. Policies are mappings π from an I-space to an action set: the current information state determines the robot's next action. Additionally, a function F is defined that iteratively applies a policy to update an information history. The updated history I-state is given by $\eta_{m+k} = F^m(\eta_k, \pi, x_k)$, where $x_k \in X$.

Definition 1. (Robot dominance from [**68**]**)** Consider two robots with a task specified by reaching a goal region $\mathcal{I}_G \subseteq \mathcal{I}_{der}$:

$$R_1 = (X^{(1)}, U^{(1)}, Y^{(1)}, f^{(1)}, h^{(1)})$$

$$R_2 = (X^{(2)}, U^{(2)}, Y^{(2)}, f^{(2)}, h^{(2)}).$$

Given I-maps $\kappa_1 : \mathcal{I}^{(1)}_{hist} \to \mathcal{I}_{der}$ and $\kappa_2 : \mathcal{I}^{(2)}_{hist} \to \mathcal{I}_{der}$, if for all: $\eta^{(1)} \in \mathcal{I}^{(1)}_{hist}$ and $\eta^{(2)} \in \mathcal{I}^{(2)}_{hist}$ for which $\kappa_1(\eta^{(1)}) \preceq \kappa_2(\eta^{(2)})$; and $u^{(1)} \in U^{(1)}$; there exists a policy, defined as $\pi_2 : \mathcal{I}^{(2)}_{hist} \to U^{(2)}$, generating actions for R_2 such that for all $x^{(1)} \in X^{(1)}$ consistent with $\eta^{(1)}$ and all $x^{(2)} \in X^{(2)}$ consistent with $\eta^{(2)}$, there exists a positive integer l such that

$$\kappa_1(\eta^{(1)}, u^{(1)}, h^{(1)}(x^{(1)}, u^{(1)})) \preceq \kappa_2(F^l(\eta^{(2)}, \pi_2, x^{(2)}))$$

then R_2 *dominates* R_1 under κ_1 and κ_2, denoted $R_1 \trianglelefteq R_2$. If both robots can simulate each other ($R_1 \trianglelefteq R_2$ and $R_2 \trianglelefteq R_1$), then R_1 and R_2 are *equivalent*, denoted by $R_1 \equiv R_2$.

Lemma 1. (from [**68**]) Consider three robots R_1, R_2, and R_3 and an I-map κ. If $R_1 \trianglelefteq R_2$ under κ, we have:

(1) $R_1 \trianglelefteq R_1 \cup R_3$ (adding primitives never hurts);

(2) $R_2 \equiv R_2 \cup R_1$ (redundancy does not help);

(3) $R_1 \cup R_3 \trianglelefteq R_2 \cup R_3$ (no unexpected interactions).

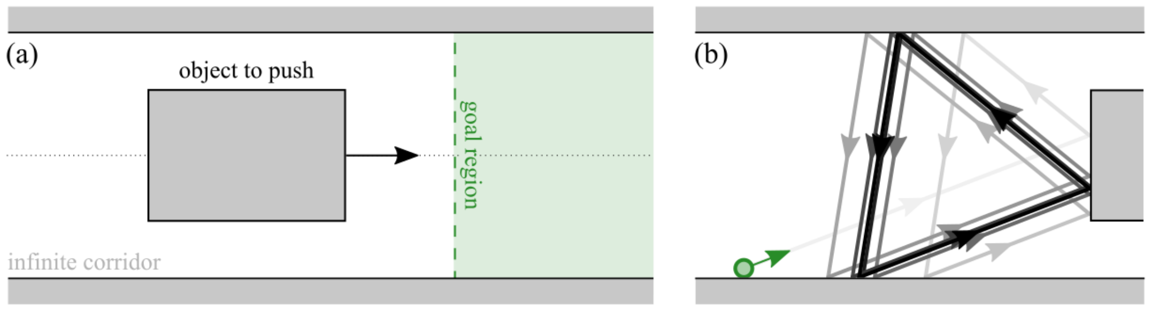


Figure 2.1. (a) A rectangular object in a long corridor that can only translate to the left or right, like a cart on a track. The robot's task is to move the object into the green goal region. (b) The robot, shown in green, executes a trajectory in which it rotates the same relative angle each time it collides.

2.4. Manipulating a Cart in a Long Corridor

The information requirements of micro-scale object manipulation will be analyzed by introducing a simple, yet rich, problem of interest. Consider a long corridor, containing a rectangular object, as shown in Fig. 2.1(a). The object may only translate left or right down the corridor, and cannot translate toward the corridor walls or rotate. This object can be abstracted as a cart on a track; physically, such one-dimensional motion may arise at the micro-scale due to electromagnetic forces from the walls of the corridor, from fluid effects, or from direct mechanical constraints. The task for the robot is then to manipulate the object into the goal region (green-shaded area in Fig. 2.1(a)). This simplified example will illustrate several interesting trade-offs in the sensing, memory, and control specification complexity necessary to solve the problem. On its own merits, this task solves the problem of *directed transport*, and can be employed as a useful component in larger, more complex microrobotic systems.

To break the symmetry of the problem, it is assumed that the object has at least two distinguishable sides (left and right). For example, the two ends of the object may emit chemical A from the left side and chemical B from the right side. The robot may be equipped with a chemical comparator that indicates whether the robot is closer to source A or source B (with reasonable assumptions on diffusion rates). The object may have a detectable, directional electromagnetic field. All these possible sensing modalities are admissible under our model. A second necessary assumption is that the dimensions of the corridor and the object are known and will be used to design the motion strategy of the robot—often a fair assumption in laboratory micro-robotics settings.

Here, I investigate the requirements of *minimal* strategies for object manipulation, given the constraints of micro-robotic control strategies. In the spirit of minimality, low-level control policies will be tailored to the natural dynamical behaviors of the system prior to the specification of increasingly abstract control policies. To this end, the system is composed of "bouncing robots," which have been shown to exhibit several behaviors, such as highly robust limit cycles, chaotic behavior, and large basins of attraction [**1, 67, 96**]. Once discovered, these behaviors can be chained together and leveraged towards solving robotic tasks such as coverage and localization without exceedingly complex control strategies [**6**]. The key insight in this paper is that by taking advantage of spontaneous limit cycles in the system dynamics, trajectories can be engineered that, purely as a result of the incidental collisions of the robot, manipulate objects in the robot's environment. Figure 2.1(b) shows an example of such a trajectory constructed from iterative executions of the natural cyclic behavior of the bouncing robots. A detailed analysis of this dynamical system and its limit cycles are presented in [**77**].

2.5. A Formal Comparison of Several Robot Designs

In order to achieve the goal of manipulating an object in a long corridor, several robot designs will be introduced and then policies will be constructed so that each robot might accomplish the task. Particularly, these robots were designed to achieve the limit cycle behavior of bouncing robots, and to use this cyclic motion pattern to push the object in a specified direction.

Prior to describing the robot designs, I will introduce the primitives that will be used to construct the robots. Figure 2.2 shows four robotic primitives taken directly from [**68**] (P_A, P_L, P_T, and P_R) and two additional primitives defined for the proposed task (P_B and P_Y). The primitive P_A describes a rotation relative to the local reference frame given an angle u_A. P_L corresponds to a forward translation over a chosen distance u_L, and P_T carries out forward translation in the direction of the robot's heading until it reaches an obstacle. In addition to these actuation primitives, P_R is defined as a range sensor where y_R is the distance to whatever is directly in front of the robot. For the 4-tuple specification of these primitives, the reader is referred to [**68**]. The chosen primitives were largely selected based on their feasibility of implementation at the micro-scale, as observed in many biological systems [**96, 46**].

To differentiate different facets of the object being manipulated, two primitives that are sensitive to the signature of each side of the target are employed. P_B is a blue sensor that measures $y_B = 1$ if the color blue is visible (in the geometric sense) given the robot's configuration (x, y, θ), and P_Y is a yellow sensor outputting $y_Y = 1$ if the color yellow is visible to the robot. More formally, we define $P_B = (0, \{0, 1\}, f_B, h_B)$ and $P_Y = (0, \{0, 1\}, f_Y, h_Y)$, where f_B and f_Y are trivial functions always returning 0, and

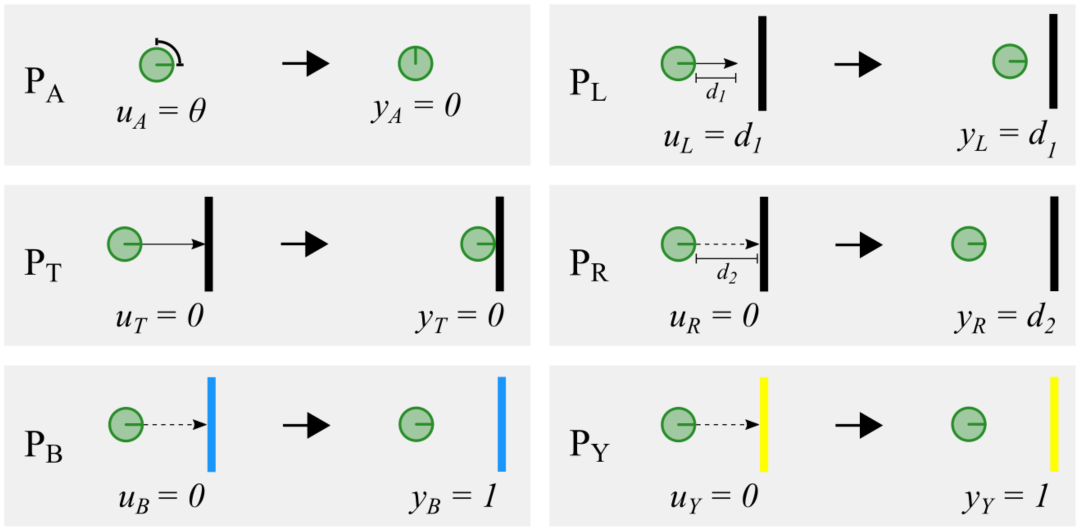


Figure 2.2. Simple robotic behaviors called primitives. P_A is a local rotation, P_T is a forward translation to an obstacle, P_L is a forward translation a set distance, P_R is a range sensor, P_B senses the color blue, and P_Y senses the color yellow.

the observation functions h_B and h_Y return 1 when the appropriate signature is in front of the robot. The "blue" and "yellow" sensors are deliberately abstract since they should be thought of as placeholders for any sensing capable of breaking the symmetry of the manipulation task, such as a chemical comparator, as discussed in Section 2.4.

These six simple primitives, shown in Fig. 2.2 can be combined in different ways to produce robots of differing capabilities. Notably, we develop modular subroutines and substrategies that allow us to develop hierarchical robot designs, enabling straightforward analysis of their capabilities. The corresponding task performance and design complexity trade-offs between the proposed robots and their constructed policies are explored in the following subsections.

2.5.1. Robot 0: Omniscient and Omnipotent

In many reported examples in micro-robot literature, the robots—as understood through the outlined framework—are *not* minimal. Often, instead of grappling with the constraints of minimal on-board computation, designers make use of external sensors and computers to observe micro-robot states, calculate optimal actions, and actuate the micro-robots using external magnetic fields, sound waves, or other methods [**54, 104, 105**]. Given the prevalence of such powerful robots in the literature, a "perfect" robot is introduced to demonstrate notation and compare to the minimal robots presented in the following sections.

The primitive for an all-capable robot can be specified as $P_O = (SE(2), SE(2) \times SE(2), f_O, h_O)$, where the action set U_O is the set of all possible positions and orientations in the plane, and the observation set Y_O is the set of all possible positions and orientations in the plane of both the robot and the object. The state transition function is $f_O(x, u) = (x + u_{O_x}\Delta t_k, y + u_{O_y}\Delta t_k, \theta + u_{O_\theta}\Delta t_k)$ where $u_{O_x}, u_{O_y}, u_{O_\theta} \in \mathbb{R}$, and $\Delta t_k \in \mathbb{R}^+$ is the time step corresponding to the discrete amount of time passing between each stage k. The observation function outputs the current configurations of the robot and object. A robot with access to such a primitive (*i.e.*, through external, non-minimal computing) would be able to simultaneously observe themselves and the object anywhere in the configuration space at all times and dexterously navigate to any location in the environment. Given that the configuration space of the robot is some bounded subset X of $SE(2)$, the robot definition for this omniscient robot is $R_0 = (X, SE(2), SE(2) \times SE(2), f_O, h_O)$. Since all robots employed in the task of manipulating the cart in the long corridor share the same bounded configuration space $X \subseteq SE(2)$, we also can define this robot using $R_0 =$

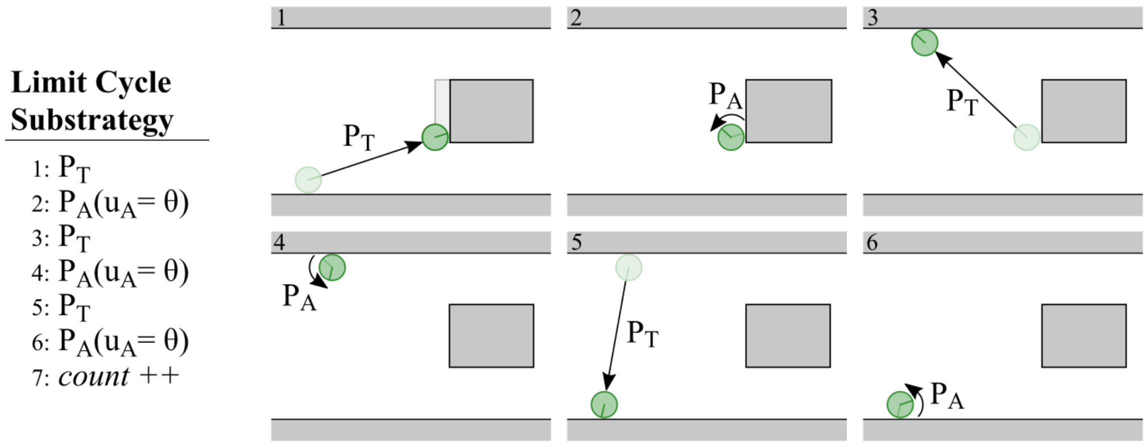


Figure 2.3. The *Limit Cycle* strategy uses the primitive P_T, which moves forward until coming into contact with an object, and P_A, which rotates relative to the robot heading. Once a limit cycle is completed, the *count* variable is incremented.

$\{P_O\}$. We use this alternative notation for robot definitions as it is less cumbersome and highlights differences in capabilities. While it is clear that such a robot should be able to solve the task through infinitely many policies, an example of such a policy π_0 that completes the task is provided in Algorithm 6 in Appendix A.

2.5.2. Robot 1: Complex

The underlying design principle behind each of the following minimal robotic designs is modularity. We use a hierarchical control approach: at the highest level, there are a few spatio-temporal states (see Fig. 2.4). In each state, the available primitives are used to develop *subroutines* corresponding to useful behaviors such as wall following, measuring distance to an object, and orienting a robot in the direction of the blue side of the cart. These subroutines are specified in Algorithm 5 in Appendix A. Through these subroutines, *substrategies* are constructed that transition the robots between states, such as moving from the right hand side of the object to the left hand side. The complete robot policy π_i is represented as a combination of these substrategies in the form of a finite state machine (FSM).

The key substrategy that enables the success of the minimal robot designs is the *Limit Cycle* substrategy (shown in Fig. 2.3), which requires two primitives, P_A and P_T, to perform. This substrategy is enabled by the fact that these bouncing robots converge to a limit cycle for a non-zero measure set of configurations (as shown in [**77**]). All other substrategies employed in the design of the robots serve the purpose of positioning the robot into a configuration where it is capable of carrying out the limit cycle substrategy.

Hence, the robot is defined as $R_1 = \{P_A, P_T, P_B, P_R, P_L, P_Y\}$, which makes use of all six of the primitives shown in Fig. 2.2. Its corresponding policy π_1, as represented by an FSM, is shown in Fig. 2.4. The details describing this policy are in Algorithm 7 in Appendix A. We highlight that through this policy the robot is capable of succeeding at the task from *any* initial condition. The substrategy structure of the FSM—containing *Initial*, *Left*, *Right*, *Middle* and *Limit Cycle* states—corresponds to different configuration domains that the robot may find itself in as represented by the shaded regions in Fig. 2.4. Due to the structure of the FSM and the task at hand, if the robot is in the *Limit Cycle* state it will eventually succeed at the task. Finally, although we allocate memory for the robot to track its success through a variable *count* (as seen in the substrategy in Fig. 2.3) the robot does not require memory to perform this substrategy and we have only included it for facilitating analysis.

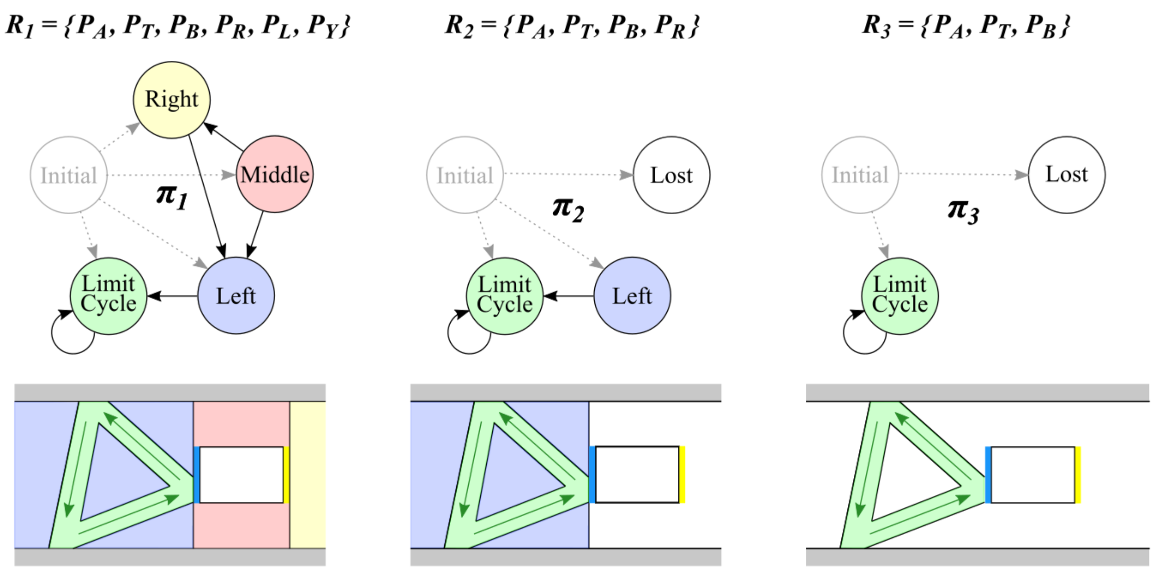


Figure 2.4. (Left) A complex robot (composed of 6 primitives) can successfully achieve its goal no matter its initial conditions. (Middle) A simple robot (composed of 4 primitives) can only be successful if its initial conditions are on the left side of the object. (Right) A minimal robot (composed of 3 primitives) can only be successful if its initial conditions are within the range of the limit cycle.

2.5.3. Robot 2: Simple

Robot 2, defined as $R_2 = \{P_A, P_T, P_B, P_R\}$, is comprised of a subset of the primitives from R_1. As a result, it is not capable of executing all of the same motion plans as R_1. As Fig. 2.4 shows, R_2 can enter its *Limit Cycle* substrategy if it starts on the left side of the object, but otherwise it will get lost. The *Initial* state uses sensor feedback to transition to the substrategy the robot should use next. The *Lost* state is distinct from the *Initial* state—once a robot is lost, it can never recover (in this case, the robot will move until it hits a wall and then stay there for all time). More details on policy π_2 and the specific substrategies that R_2 uses can be found in Algorithm 8 in Appendix A.

2.5.4. Robot 3: Minimal

Robot 3, $R_3 = \{P_A, P_T, P_B\}$, contains three robotic primitives, which are a subset of the primitives of R_1 and R_2. Under policy π_3 (detailed in Algorithm 9 in Appendix A), this robot can only be successful at the task if it initializes in the *Limit Cycle* state, facing the correct direction. Otherwise, the robot will never enter the limit cycle. Such a simple robot design could be useful in a scenario when there are very many "disposable" robots deployed in the system. Even if only a small fraction of these many simple robots start out with perfect initial conditions, the goal would still be achieved. Despite the apparent simplicity of such a robot, we note that R_3 (along with all other introduced designs) is capable of determining whether or not it is succeeding at the task or whether it is lost irreversibly. Such capabilities are not by any means trivial, but are included in the robot designs for the purposes of analysis and comparison.

2.5.5. Comparing Robots

We will compare the four robots introduced in this section, R_0, R_1, R_2, and R_3. In order to achieve this we must first specify the task and derived I-space in which we can compare the designs. The chosen derived I-space is $\mathcal{I}_{der} = \mathbb{Z}^+ \cup \{0\}$. Specifically, it consists of counts of the *Limit Cycle* state (the *count* variable is shown in Fig. 2.3 and in the algorithms in Appendix A). If we assume that after each collision the robot pushes the object a distance ϵ, task success is equivalently tracked in memory by *count* up to a scalar.

The goal for the task of manipulating the cart in a long corridor is $\mathcal{I}_G \subseteq \mathcal{I}_{der}$, where $\mathcal{I}_G$ is an open subset of the nonnegative integers. In this set up, as illustrated in Fig. 2.1(a), the robot must push the object some N times, corresponding to a net distance traveled, to succeed. More formally, the information preference relation can be expressed through the indicator $1_G(\eta)$ corresponding to whether a derived information history is within the goal region $\mathcal{I}_G$, thereby inducing a partial ordering over information states. Hence, the likelihood of success of any of the proposed robot designs (excluding R_0) is solely determined by their initialization, and the region of attraction of the limit cycle behavior for the bouncing robots, which will be explored in more detail.

2.5.5.1. Comparing R_1, R_2, and R_3. The comparison of robots R_1, R_2, and R_3 through the lens of robot dominance is straightforward given the modularity of the robot designs. Since R_1 and R_2 are comprised of a superset of the primitives of R_3, they are strictly as capable or more capable than R_3, as per Lemma 1(a). Therefore, it can be stated that R_1 and R_2 *dominate* R_3, denoted by $R_3 \trianglelefteq R_1$, and $R_3 \trianglelefteq R_2$. Likewise, using the same lemma, we can see that $R_2 \trianglelefteq R_1$. This is to say that for the task of manipulating the cart along the long corridor R_1 should outperform R_2 and R_3, and that R_2 should outperform R_3.

While the policies for each robot design are nontrivial, Fig. 2.4 offers intuition for the presented dominance hierarchies. Effectively, if either R_2 or R_3 are initialized into their *Lost* state they are incapable of executing the task for all time. Hence, it is the configuration space volume corresponding to the *Lost* state that determines the robot dominance hierarchy.

Let $\eta^{(1)} \in \mathcal{I}_{hist}^{(1)}, \eta^{(2)} \in \mathcal{I}_{hist}^{(2)}, \eta^{(3)} \in \mathcal{I}_{hist}^{(3)}$, and define I-maps that return the variable *count* stored in memory for each robot. The information preference relation then only discriminates whether the information histories correspond to a trajectory reaching $\mathcal{I}_G \subseteq \mathcal{I}_{der}$—in other words, whether a robot achieves the required N nudges to the object in the corridor. Note that since there are no time constraints to the task, this number is arbitrary and only relevant for tuning to the length-scales of the problem. Thus, the dominance relations outlined above follow from the fact that for non-zero volumes of the configuration space there exists no integer l for which $\kappa_1(\eta^{(1)}) \preceq \kappa_2(F^l(\eta^{(2)}, \pi_2, x))$. On the other hand for all $x \in X$, $\kappa_2(\eta^{(2)}) \preceq \kappa_1(F^l(\eta^{(1)}, \pi_1, x))$. Through this same procedure the rest of the hierarchies presented in this section can be deduced.

2.5.5.2. Comparing R_0 and R_1. To compare R_0 and R_1 a similar reachability analysis can be used. From any $x \in X$, R_1 and R_0 are capable of reaching the object and nudging it. This means that given that the domain X is bounded and information history states $\eta^{(0)} \in \mathcal{I}^{(0)}_{hist}, \eta^{(1)} \in \mathcal{I}^{(1)}_{hist}$ corresponding to each robot, there always exists a finite integer l such that $\kappa_0(\eta^{(0)}) \preceq \kappa_1(F^l(\eta^{(1)}, \pi_1, x))$, and $\kappa_1(\eta^{(1)}) \preceq \kappa_0(F^l(\eta^{(0)}, \pi_0, x))$. Therefore, $R_1 \trianglelefteq R_0$ and $R_0 \trianglelefteq R_1$, meaning that $R_1 \equiv R_0$. Thus, R_0 and R_1 are equivalently capable of performing the considered task.

It is important to note that despite the intuition that R_0 is more "powerful" than R_1 in some sense, for the purposes of the proposed task that extra power is redundant. However, there are many tasks where this is would not be the case (*e.g.*, moving the robot to a specific point in the plane).

2.5.5.3. Comparing R_0, R_2 and R_3. Lastly, while the relationship between R_0 and the other robot designs is intuitive, an additional lemma is necessary.

Lemma 2. (Transitive property) Given three robots R_0, R_1, R_2, if $R_2 \trianglelefteq R_1$ and $R_1 \equiv R_0$, then $R_2 \trianglelefteq R_0$.

Proof. The proof of the transitive property of robot dominance comes from the definition of equivalence. $R_1 \equiv R_0$ means that the following statements are simultaneously true:

$R_1 \trianglelefteq R_0$ and $R_0 \trianglelefteq R_1$. Thus, this means that $R_2 \trianglelefteq R_1 \trianglelefteq R_0$, which implies that $R_2 \trianglelefteq R_0$, concluding the proof.

Using this additional lemma, it can be shown that $R_2 \trianglelefteq R_0$, and $R_3 \trianglelefteq R_0$, as expected. Hence, we have demonstrated that minimal robots may be capable of executing complex strategies despite the constraints imposed by the micro-scale domain. Minimality in micromanipulation is in fact possible when robot designs take advantage of naturally occurring dynamic structures, such as limit cycles.

For more information on the necessary conditions for establishing such cycles, as well as the robustness properties of limit cycle behavior, which are important for extending this work to less idealized and deterministic settings, the reader is encouraged to consult [**77**]. The section on Feasibility and Dynamics of Cyclic Motion Strategies contains propositions and proofs of bouncing strategy conditions and robustness, largely developed by Alexandra Q. Nilles.

2.6. Discussion

In this chapter, robust motion strategies for minimal robots that have great promise for micromanipulation were designed. We have analyzed the information requirements for task success, compared the capabilities of four different robot designs, and found that minimal robot designs may still be capable of micromanipulation without the need for external computation. While the example of a rectangular obstacle in a corridor is simple, it can be interpreted as *robust directed transport*, a key building block for future work.

2.6.1. Future Directions

The setting of micro-robotics provides motivation for the approach laid out in this work. At the micro-scale, coarse high-level controllers that can be applied to a collection of many micro-robots are easier to implement than fine-grained individual controllers. This requires formal reasoning about all possible trajectories, in order to funnel the system into states that allow for task completion, as was illustrated in this chapter. In order to be more applicable in the micro-robotics domain, it will be important to extend the approach to multiple agents, as well as scenarios subject to noise. While these strategies passively provide some noise tolerance by virtue of the limit cycle region of attraction, there is much work to be done on more concrete applications to characterize and account for sensing and actuation noise.

Outside of this particular model and application, this work has implications for the future of robot behavior and design. Often, derived I-states are designed to infer information such as the set of possible current states of the robot, or the set of possible states that the robot could have previously occupied. In this work, the focus is on derived I-spaces that encode information about what *will happen* to the robot under a given strategy. With these forward-predictive derived I-spaces, a high density of task-relevant information is encoded into a few-state symbolic abstraction. By making use of such abstractions, minimal agents may be endowed with a passively predictive capacity leading to greater task-capability.

More broadly, this work provides an exciting glimpse toward more automated analysis, through a combination of system identification techniques, hybrid systems theory, and I-space analysis. Coarse-grained sensors provide an avenue for discretization useful for hierarchical control; such an approach is increasingly needed as our robotic systems become more data-driven. Such a unified approach may be able to simultaneously identify coarse-grained system dynamics, predict their task-capabilities, and design fine-tuned control strategies.

CHAPTER 3

Algorithmic Design of Synthetic Cells

In nature, biological organisms jointly evolve both their morphology and their neurological capabilities to improve their chances for survival. Consequently, task information is encoded in both their brains and their bodies. In robotics, the development of complex control and planning algorithms often bears sole responsibility for improving task performance. This dependence on centralized control can be problematic for systems with computational limitations, such as mechanical systems and robots on the microscale. In these cases we need to be able to offload complex computation onto the physical morphology of the system. To this end, we introduce a methodology for algorithmically arranging sensing and actuation components into a robot design while maintaining a low level of design complexity (quantified using a measure of graph entropy), and a high level of task embodiment (evaluated by analyzing the Kullback-Leibler divergence between physical executions of the robot and those of an idealized system). This approach computes an idealized, unconstrained control policy which is projected onto a limited selection of sensors and actuators in a given library, resulting in intelligence that is distributed away from a central processor and instead embodied in the physical body of a robot. The method is demonstrated by computationally optimizing a simulated synthetic cell.

3.1. Introduction

Embodied intelligence, the coupling of a system's controller and morphology, has been studied for quite some time [**14**], [**79**], [**80**], often in the context of biologically inspired systems [**28**], [**95**], [**102**]. Recently, some significant and comprehensive efforts have been made toward implementing components of embodiment in robotic applications [**5**], [**39**]. But still, most robot designers opt for approaches using centralized computations to manipulate existing robotic platforms, rather than offloading some of the computational effort onto a robot's morphology. In this work, we are motivated by a system that is computationally limited but flexible in terms of physical design, and is therefore an ideal candidate to take advantage of embodied intelligence.

The main challenge with embedding control information in material properties is the contradiction between continuous, often complex, classical control and the discrete, simpler capabilities that a materials-based system is likely to have. A conflict exists between equipping a robot with what is sufficient and what is necessary—what can enable a robot to succeed (and is likely complex) and what is minimally required for it to achieve its goal (and is necessarily simple).

This is especially evident when designing for robots without any on-board, CPU-based, traditional computational capabilities. Some robotic systems employ embodied intelligence (which we'll also refer to as embodied computation) to reduce weight and energy—for example fully mechanical devices, like passive dynamic walkers [**23**], [**97**] or those used in prosthetic limbs [**3**]—while others must *necessarily* resort to embodied computation because of scale, for example, robots on the micro- or nano-scales [**25**], [**55**]. We focus on the latter case later. So although robot design is traditionally identifying which sensory, computation, and actuation elements can be combined to best achieve a goal, here we will examine a framework for designing the sensory and actuation elements of robots so that no traditional, CPU-based computation is necessary to accomplish a goal.

We first examine the relationship between control policy design and physical robot body design. A control policy assigns an action for a system at each time or state. Symbolic control policies have been useful in robot control and motion planning [**7**], [**60**] for systems with limited computational power [**81**]. An example is shown in Fig. 3.1, where the system consists of three possible control modes: move right (blue), move up (red), and stay still (white) and its goal is to navigate to the upper right corner of the grid world. These simple control modes can also be thought of as primitives, as discussed in Chapter 2. A simple robot placed in this environment with one of the shown policies in its memory would be able to achieve its task without any traditional computation, replacing logical operators such as inequalities with physical comparators to relate sensor states to control actions.

Figure 3.1 illustrates the difference in complexity between these two policies. The policy in Fig. 3.1 (a) requires a combination of sensors that are capable of differentiating between three different regions of the state space, and the policy in Fig. 3.1 (b) requires sensors that are able to discern between fifteen different regions. It is simpler to physically implement the robot design implied by the control policy in Fig. 3.1 (a) than the policy in Fig. 3.1 (b).

We pose the physical design problem as the projection of a policy onto an admissible set of physical sensor-actuator interconnections, the complexity of which must be managed

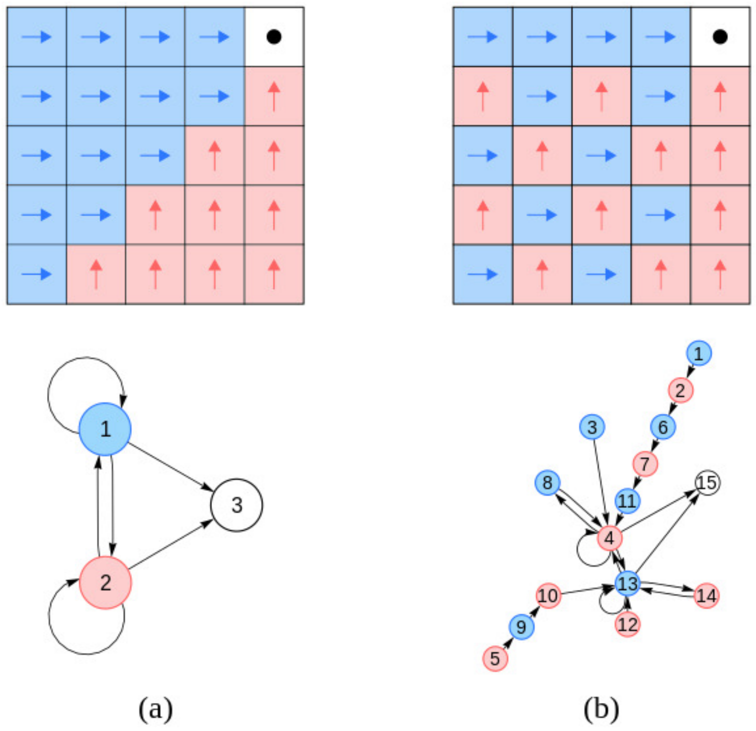


Figure 3.1. A simple example of a control policy. Here the state space is a two dimensional 5×5 grid and the desired state is in the upper right of the state space. At each state, the suitable control is indicated by its assigned color. (a) A *simple* control policy with only three different states in the finite state machine. (b) A *complex* control policy, with fifteen different states in the finite state machine. Both achieve the task with different implementations.

during policy iteration. This complexity is a measure of logical interconnections between sensor states and control modes (which correspond to arrows on the graphs in Fig. 3.1). The policy projection will be performed both by computing a control policy assuming discrete control modes (and continuous sensing) and then projecting onto a discrete sensor set, and by generating a control policy assuming discrete sensing (and unconstrained control authority) and then projecting onto a set of discrete control modes.

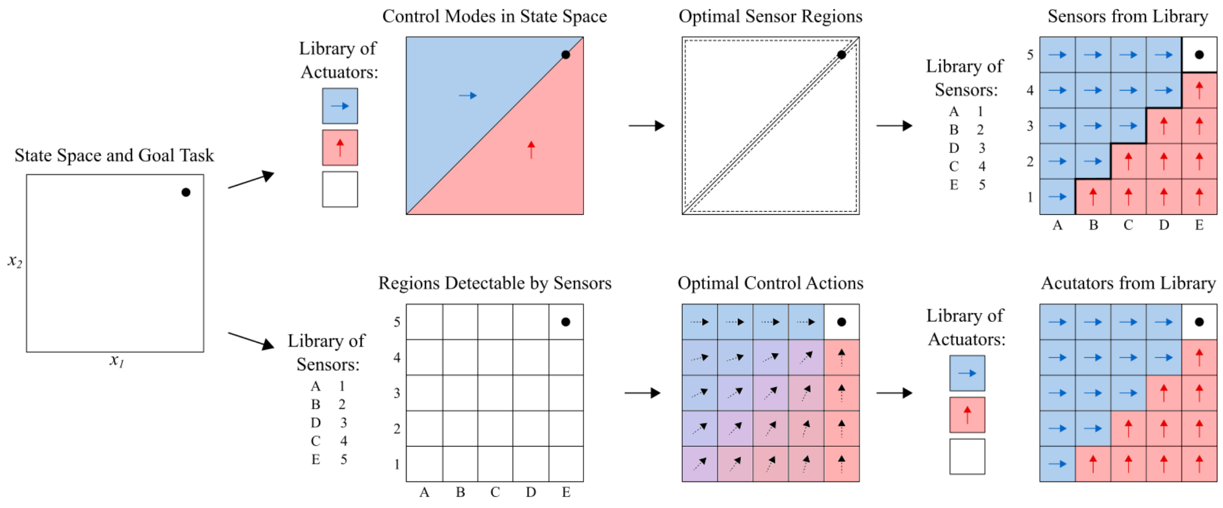


Figure 3.2. This chapter outlines two possible paths to robotic control policy design: first assigning actuation, then sensors, or vice versa. The top section of the flow diagram shown here illustrates beginning with a library of *actuators* and assigning control modes in space. These spatial assignments inform what the optimal sensor regions would be, which are then approximated using a library of available sensors. Conversely, the bottom section of the diagram illustrates the process beginning with a library of *sensors*, and using those to determine which regions in the state space are distinguishable. Then optimal control actions are calculated for each of the spatial regions, and those optimal control actions are approximated using a library of available actuators.

Methods from switched systems literature are used to organize these sensor-actuator connections so that they change minimally as a function of state. This is challenging because [**8**] shows that optimal solutions for discrete switched systems will chatter with probability 1, and chattering implies a complex policy. That is, when choosing among a finite number of sensor-actuator pairings, optimization of an objective function will necessarily lead to arbitrarily complex dependencies on state, and the physical implementation of such a policy would consequently be very complex (and often not physically realizable). Results from [**17**] show that solutions with slow mode switching are “almost” as good as chattering solutions. Properties from [**17**] are used here to design control policies with minimal mode switching and then project them onto sensor-actuator interconnections, as illustrated in Fig. 3.3, resulting in a methodology for designing robots with embodied intelligence.

After reviewing related work in Section 3.2, this methodology will be explored in terms of an extended example. The example system, called a synthetic cell, will be introduced in Section 3.3.1. The primary contributions of this work can be summarized as follows:

(1) Quantitative definitions are given for design complexity and task embodiment, described in Section 3.3.2.
(2) An iterative algorithm is developed in Section 3.4.1, and is used to create control policies with low design complexity while increasing task information.
(3) A projection operator is presented in Section 3.4.2, which projects a low complexity control policy onto a physically realizable set of sensor-actuator interconnections.
(4) A methodology for algorithmically organizing components for robot design is established. The procedure begins either with a control policy based on a discrete set of actuators (Sec. 3.4.1) and interconnects them with different possible sets of sensors (Sec. 3.4.2) or begins with a control policy generated with a discrete set of sensors (Sec. 3.4.3) and combines them with a selection of discrete actuators (Sec. 3.4.4).

These are supported by simulations of synthetic cells. Versions of this work were published in [**73**] and [**76**].

3.2. Related Work

Policy Optimization while Evolving Morphology (POEM) [**5**], evolves the physical body of a continuously controlled reinforcement learning agent and analyzes the relative importance of body changes using cooperative game theory. The POEM method was shown to produce stronger agents than optimizing the control policy alone. A common motivation of the work in [**5**] and this chapter is the theory that a physical body that is well-suited to a task is easier, and simpler, to control (and, in [**5**], easier to *learn* to control). In this work, we characterize design updates in terms of moving task information from centralized computations in control calculations to embodied computation in the physical body.

The work in [**18**], [**19**], and [**20**] defines design problems as relations between functionality, resources, and implementation and shows that despite being non-convex, non-differentiable, and noncontinuous, it is possible to create languages and optimization tools to define and automatically solve design problems. The optimal solution to a design problem is defined as the solution that is minimal in resources usage, but provides maximum functionality. We apply this definition by proposing a min-max problem in which the goal is to minimize design complexity (representative of the amount of sensors and actuators required, i.e., the resources), and maximize task embodiment (i.e., the functionality of the design).

A method for automatically designing action-based sensors was explored in [**35**]. This was done by generating a strategy for a robot task using a planner that assumes perfect sensing, and using that plan to specify sensors that tell the robot where to execute each action. The methodology in [**35**] is very similar to the work presented in this chapter, which also first develops a control policy assuming perfect sensing (or perfect actuation) and then specifies discrete sensors (or actuators) that approximate the original strategy. The work in this chapter differs from the contributions in [**35**] by taking complexity into account and by also designing actuators. In fact, Fig. 22 in [**35**], which shows the state space divided into regions discernible by sensors and which actions to use in each of them, closely resembles the ideas represented in Figs. 3.3 and 3.11 in this chapter.

Robotic primitives are introduced in Chapter 2 and [**70**] as independent components that may involve sensing or motion, or both. These are implemented in this chapter as actuator and sensor libraries from which we allow our algorithm to choose components. Task embodiment, which is defined in Sec. 3.3.2, parallels the dominance relation proposed in Chapter 2 and [**70**] that compares robot systems such that some robots are stronger than others based on a sensor-centered theory of information spaces.

Similarly, our definition of design complexity (Sec. 3.3.2) parallels an existing notion of conciseness, presented in [**71**]. The results in [**71**] are motivated by circumstances with severe computational limits, specifically addressing the question of how to produce filters and plans that are maximally concise subject to correctness for a given task. This is very related to our goal of finding the simplest way to physically organize sensors and actuators so that a (computationally limited) robot can achieve a given task.

The work presented in [**47**] produces asymptotically optimal sampling-based methods and proposes scaling laws to ensure low algorithmic complexity for computational efficiency. These algorithms were originally developed for path planning, but we apply similar ideas for generating simple control policies. The methods described in [**47**] start with an optimal, infinite complexity solution, and from that develop simpler plans. In

Sec. 3.4.1, we start with a zero complexity policy and move towards more complex, better performing solutions—while maintaining a level of computational complexity appropriate for physical implementations of embodied computation.

3.3. Model and Definitions

3.3.1. Motivating Example: The Synthetic Cell

How can we use control principles to organize sensor components, actuator components, and their interconnections to create desired autonomous behavior, without relying on traditional computation? To answer this question we will consider the extended example of a synthetic cell—a small robot that only has a finite number of possible sensor and actuator states and potential pairings between them [**59**]. The purpose of this example system is to show a concrete implementation of the methods in Sections 3.4.1-3.4.4, and to illustrate the relationship between control policy design and physical robot body design.

A synthetic cell is a mechanically designed microscopic device with limited sensing, control, and computational abilities [**59**]; it is essentially an engineered cell. A synthetic cell exists in a chemical bath and generates movement by interacting with its environment using chemical inhibitors, and it contains simple circuits that include minimal sensors and very limited nonvolatile memory [**58**]. Such a device is 100μm in size or less, rendering classical computation using a CPU impossible. But these simple movement, sensory, and memory elements can be combined with a series of physically realizable logical operators to enable a specific task. How can these discrete structures be algorithmically organized to combine sensing and control to accomplish an objective?

Synthetic cells can contain sensors like a photodiode to detect light, a chemical comparator to compare chemical concentrations, or limited amounts of nonvolatile memory—but they certainly cannot accurately estimate their location in a two (or three) dimensional space. This is why a complex control policy (like the one shown in Fig. 3.1(b)), which requires the agent to have knowledge of which relatively small region it is in, would be difficult to physically implement, and why a less complex design must be found.

For the example in this chapter, a synthetic cell operates in a two dimensional space, and its control authority is the ability to be attracted toward a specific chemical potential. So at any location (x, y) the robot may choose a control mode $\sigma \in \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6\}$, where σ_0 is zero control, and the other six modes are a potential that the synthetic cell can be attracted to (their locations are shown in Fig. 3.3), with dynamics

$$\mathbf{x} = \begin{pmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{pmatrix}, \quad f(\mathbf{x}, u) = \begin{pmatrix} \dot{x} \\ \frac{\text{sign}\,(x_{S_n} - x)}{r_n^2} \\ \dot{y} \\ \frac{\text{sign}\,(y_{S_n} - y)}{r_n^2} \end{pmatrix}, \tag{3.1}$$

where r_n is the distance from the synthetic cell to source n and (x_{S_n}, y_{S_n}) are coordinates of the source locations. Because of the inverse squared terms in the dynamics, chemical sources that are nearer to the synthetic cell will be able to accelerate the cell faster than those that are far away. Boundary conditions, like those discussed in [**15**], are necessary to avoid an infinite acceleration as $r_n \to 0$. We included a very small boundary ε around the chemical source [**15**], so that the cell cannot be co-located with the source. The maximum velocity was also bounded, to mimic terminal velocity in a fluid.

The control synthesis problem is to schedule σ in space (x, y), based on an objective (in this case, to approach a point $\mathcal{P}$) specified in a cost function J (3.2) made up of a running cost $\ell(x(t), u(t))$ (3.3) and a terminal cost $m(x(t_f))$ (3.4).

$$J(x(t), u(t)) = \int_0^{t_f} \ell(x(t), u(t))dt + m(x(t_f)). \tag{3.2}$$

$$\ell(x, u) = (x - x_d)^T Q(x - x_d) + u^T Ru \tag{3.3}$$

$$m(x) = (x - x_d)^T P_1(x - x_d). \tag{3.4}$$

For our simulations, we used the parameters: prediction time horizon $T = 0.1s$; time step $t_s = 0.02s$; final time $t_f = 5s$; desired state[1] $x_d = [2-\frac{\pi}{20}, 4-\frac{\pi}{15}, 0, 0]^T$; size of the source $\varepsilon = 0.001$; maximum velocity $v_{max} = 0.4$; cost weights $Q = P_1 = diag[10, 10, 0.001, 0.001]$ and $R = 0$; and source locations $(x_{S_1}, y_{S_1}) = (1, 5)$, $(x_{S_2}, y_{S_2}) = (3, 5)$, $(x_{S_3}, y_{S_3}) = (1, 3)$, $(x_{S_4}, y_{S_4}) = (3, 3)$, $(x_{S_5}, y_{S_5}) = (1, 1)$, and $(x_{S_6}, y_{S_6}) = (3, 1)$. The state space, desired point, and chemical sources are all shown in Fig. 3.3 (a)

[1]Point $\mathcal{P}$ is slightly off center, to avoid adverse effects of symmetry. This is reflected in the asymmetry of the resulting control policies (e.g., green and orange not being perfectly even).

3.3.2. Design Complexity and Task Embodiment

Graph entropy [**2**], [**27**] will be used as a measure of design complexity for comparing robot designs. The complexity of a control policy is equated with the measure of entropy of its resulting finite state machine.

A finite state machine consists of a finite set of states (nodes), a finite set of inputs (edges), and a transition function that defines which combinations of nodes and edges lead to which subsequent nodes [**84**]. The finite set of nodes that the system switches between are the *control modes*, and the edges—inputs to the system which cause the control modes to change—are the *state observations* (Fig. 3.1 (a)).

Finite state machines and their corresponding adjacency matrices are generated numerically, by simulating a synthetic cell forward for one time step, and recording control modes assigned at the first and second states. These control mode transitions are counted and normalized into probabilities, and the resulting data-driven adjacency matrix A is used in the entropy calculation,

$$h = -\sum_{i} A(i) \log\left(A(i)\right) \tag{3.5}$$

which results in a complexity measure h for each robot design. This measure of complexity is more informative than other metrics (e.g., simply counting states) because it is a function of the *interconnections* between states—which is what we want to minimize in the physical design.

We define *task embodiment* as the amount of information about a task encoded in a robot's motion (not to be confused with embodiment found in human-robot interaction [**41**], [**101**]). We focus on this motion-based task information so that the design update can be characterized in terms of moving task information from the centralized computations in the control calculations to embedded computation in the physical body. One measure that captures how much information one system encodes about another system is Kullback-Leibler divergence. Here we measure the K-L divergence between a distribution representing the task, P, and a distribution representing the robot design, Q [**10**],

$$D_{KL}\left(P \| Q\right) = -\sum_{\mathbf{x}} P(\mathbf{x}) \log\left(\frac{Q(\mathbf{x})}{P(\mathbf{x})}\right). \tag{3.6}$$

To define the goal task distribution P, a model predictive controller (MPC) is used to simulate the trajectories of a robot with an ideal (centralized, unlimited in sensing (Sec. 3.4.1) or actuation (Sec. 3.4.3)) controller. The same method is used to generate a distribution Q that represents the robot design—this time simulating trajectories using the generated control policy. Task embodiment is a measure of the difference in task executions between a robot with an ideal controller and a resource-limited robot with some embodied intelligence. We use Eq. (3.6) to compare the two distributions of trajectories: a low measure of K-L divergence indicates that the distributions are similar, and implies a high level of task embodiment, and therefore a better robot design.

In other words, if a task is well-embodied by a robot, only a simple control policy is necessary to execute it. Otherwise, more information, in the form of a more complex control

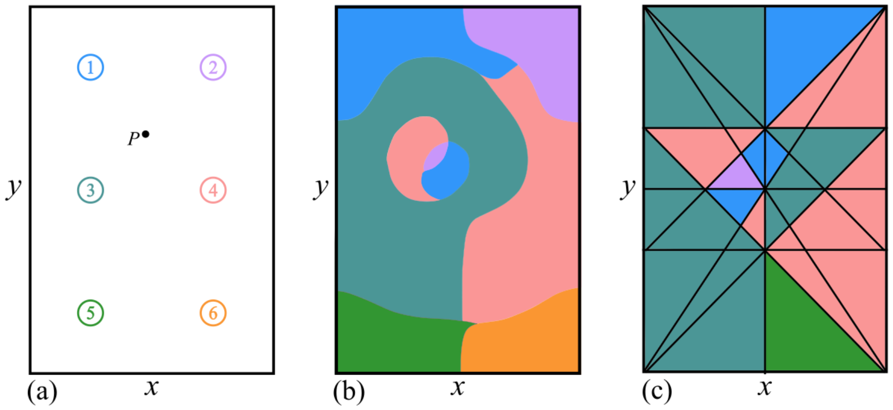


Figure 3.3. (a) The state space and controls for the synthetic cell example system introduced in Sec. 3.3.1. At any location in the state space, the robot is able to choose one of seven different control modes: attraction to chemical potentials at the six different sources or zero control. The goal is to reach $\mathcal{P}$. (b) A control policy for this system generated with discrete controls and unconstrained sensing. (c) The control policy projected onto a feasible set of sensor states. (These figures will be explained in detail in Sec. 3.4.1 and Sec. 3.4.2.)

policy, is required. To construct these control policies, we will explore two opposing procedures: optimizing with respect to actuation assuming unconstrained sensing (Sec. 3.4.1) and projecting onto discrete sensor sets (Sec. 3.4.2), or optimizing with respect to sensing assuming unconstrained actuation (Sec. 3.4.3) and projecting onto discrete control modes (Sec. 3.4.4).

3.4. Control Policy Generation

3.4.1. Generating Policies with Actuators First

The optimization problem of minimizing implementation complexity while maximizing task embodiment is challenging, with many reasonable approaches. We use techniques from hybrid optimal control because of properties described next.

It was proven in [**8**] that optimal control of switched systems will result in a chattering solution with probability 1. Chattering is equivalent to switching control modes very quickly in time. In the case of these control policies, this translates to switching between control modes very quickly in state. As a result, an implementation of the optimal control policy would be highly complex. Instead of an optimal solution, we are looking for a "good enough," near optimal solution that results in a minimal amount of mode switching. It was shown in [**17**] that the mode insertion gradient (MIG), which will be discussed in Section 3.4.1.2, has useful properties, including that when the MIG is negative at a point it is also negative for a region surrounding that point, and that a solution that switches modes slowly can be nearly as optimal as a chattering solution.

This section will first review the topics of switched systems [**8**], [**17**], [**33**] and the use of needle variations for optimization [**34**], [**90**], [**106**], then develop an algorithm for building low complexity control policies. The algorithm creates a simple control policy under the assumption that the system has perfect knowledge of its state. Mapping this policy to physically realizable sensors is the subject of Section 3.4.2.

3.4.1.1. Switched Systems. A switched-mode dynamical system is typically described by state equations of the form

$$\dot{x}(t) = \{f_\sigma(x(t))\}_{\sigma\in\Sigma} \tag{3.7}$$

with n states $x : \mathbb{R} \to X \subseteq \mathbb{R}^n$, m control modes[2] $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$, and continuously differentiable functions $\{f_\sigma : \mathbb{R}^{n+m} \to \mathbb{R}^n\}_{\sigma\in\Sigma}$ [**34**]. Such a system will switch

[2]Typically u is denoted as a control variable, but in this case the control value u could be anything in the greater context of the control mode σ. The control mode may, in fact, consist of many different values of u, which is why u does not appear in a significant way in the posing of this problem.

between modes a finite number of times N in the time interval $[0, t_f]$. The control policy for this type of switched system often consists of a mode schedule containing a sequence of the switching control modes $\mathcal{S} = \{\sigma(1), \ldots, \sigma(N)\}$ and a sequence of switching times $\mathcal{T} = \{\tau_1, \ldots, \tau_N\}$ [**33**], [**106**].

Here, we will consider a similar switched-mode system, but instead of implementing an algorithm to optimize transition *times* between modes (so that control modes are scheduled as a function of time $\sigma(t)$), we optimize transition *states* (so that control modes are a function of state $\sigma(x)$). This way a robot can directly map sensory measurements of state to one of a finite number of control outputs.

3.4.1.2. Hybrid Optimal Control. Let $\ell : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable cost function, and consider the total cost J, defined in Eq. (3.2). We use the Mode Insertion Gradient (MIG) [**34**], [**90**], [**106**] to optimize over the choice of control mode at every state. The MIG measures the first-order sensitivity of the cost function (3.2) to the application of a control mode σ_i for an infinitesimal duration $\lambda \to 0^+$. The MIG $d_i(x)$ is defined

$$d_i(x) = \frac{dJ}{d\lambda^+}\bigg|_t = \rho(t)^T (f_{\sigma_i}(x(t)) - f_{\sigma_0}(x(t))). \tag{3.8}$$

The adjoint variable ρ is the sensitivity of the cost function

$$\dot{\rho} = -(\frac{\partial f_{\sigma_0}}{\partial x}(x(t)))^T \rho - (\frac{\partial \ell}{\partial x}(x(t)))^T, \quad \rho(t_f) = 0. \tag{3.9}$$

The derivation of these equations is discussed in [**34**], [**90**], [**106**], but the key point is that $d_i(x)$ measures how much inserting a control mode σ_i locally impacts the cost J. When $d_i(x) < 0$, inserting control mode σ_i at state x will decrease the cost throughout a

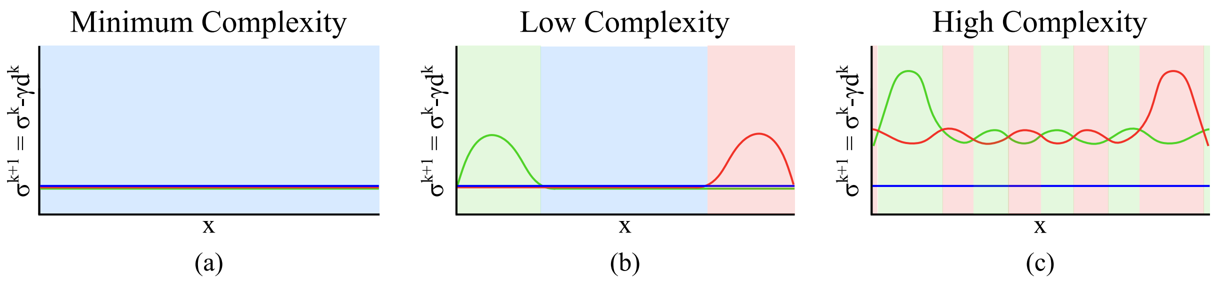


Figure 3.4. The curves show $\sigma^{k+1} = \sigma^k - \gamma d^k$ for three different control modes σ_1 (blue), σ_2 (green), and σ_3 (red), where γ is the line search parameter and the background colors indicate which mode is assigned to state x in the control policy. As step size γ increases from left to right, the magnitude of $\gamma d^k(x)$ surpasses that of the default control $\sigma^k(x)$ (here the default control is $\sigma_1 = 0$). (a) Minimum Complexity: Only one control mode is assigned throughout the entire state space. (b) Low Complexity: A few control modes are employed, indicating that the cost function can be reduced by including these extra control modes. (c) High Complexity: The control mode switches often but the values are similar, indicating that there is no significant difference in cost between these modes—despite the large increase in complexity (i.e., chattering).

volume around x, meaning a descent direction has been found for that state. The MIG can be calculated for each mode so that $d(x)$ is a vector of m mode insertion gradients: $d(x) = [d_1(x), ..., d_m(x)]^T$. Therefore the best actuation mode (i.e., the mode with the direction of maximum descent) for each state x has the lowest value in the vector $d(x)$.

As long as the dynamics $f(x(t))$ are real, bounded, differentiable with respect to state, and continuous in control and time and the incremental cost, $\ell(x(t))$, is real, bounded, and differentiable with respect to state, the MIG is continuous [**17**]. Sufficient descent of the mode insertion gradient is proven in [**17**], where the second derivative of the mode insertion gradient is shown to be Lipschitz continuous under assumptions guaranteeing the existence and uniqueness of both x, the solution to the state equation Eq. (3.7), and ρ, the solution of the adjoint equation Eq. (3.9). Combining this with the results of [**47**],

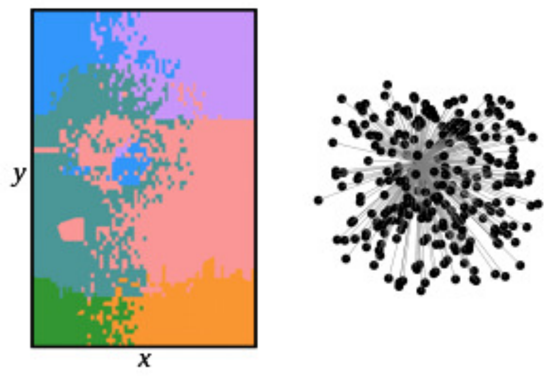


Figure 3.5. A chattering control policy. The corresponding graph has entropy $h = 7.6035$.

one can conclude that any sufficiently dense finite packing will also satisfy the descent direction throughout the volume of packing. As a result, although chattering policies may be the actual optimizers, finite coverings will generate descent throughout the state space, resulting in a non-optimal but "good enough" solution. This provides the required guarantee that we can locally control the complexity of the policy as a function of state. This will be discussed further in Sec. 3.4.1.3.

Figure 3.4 illustrates differences in complexity as a result of optimizing using the mode insertion gradient. The magnitude of the curves is the default control (the control we are comparing to) minus the step size (a scaling factor, and also the line search parameter) multiplied by the MIG. Therefore the magnitude of these plots correspond to the amount of reduction in cost that can be achieved by locally employing each control mode at the state x. The complex policy illustrated in Fig. 3.4 (c), occurs in simulation of the chattering policy of Fig. 3.5. This happens when there is similar utility in employing more than one mode in a region—there is only marginal benefit in choosing one control mode over another, which results in increased complexity.

3.4.1.3. Iterative Algorithm. An algorithm is introduced that can reduce the complexity of a control policy in as little as one iteration, based on the work in [**16**], [**17**].

Algorithm 1 Iterative Line Search Optimization

```
**Input Parameters:** $\epsilon_h$, $\epsilon_J$
**Initialize Variables:** $k = 0$, $\gamma = 0.001$
Choose default policy $\sigma^0(x)$
Calculate initial cost $J(\sigma^0(x))$
Calculate initial complexity $h^0$
Calculate initial descent direction $d^0(x)$
$h^{k-1} = \infty$
**while** $h^k < h^{k-1} + \epsilon_h$
  **while** $J(\sigma^k(x)) < J(\sigma^{k+1}(x)) + \epsilon_J$
    Re-simulate $\sigma^{k+1}(x) = \sigma^k(x) - \gamma d^k(x)$
    Compute new cost $J(\sigma^{k+1}(x))$
    Increment step size $\gamma$
  Calculate new complexity $h^{k+1}$
  Calculate $d^{k+1}(x)$
  $k = k + 1$
```

In this line search algorithm the default control may be chosen arbitrarily, but for simplicity we will show an example using a null default policy (in Fig. 3.6).

The cost $J(\sigma^k(x))$ of the entire policy is approximated by simulating random initial conditions forward in time and evaluating the total cost function for time t_f. We use cost J rather than task embodiment D_{KL} as the objective function because the line search in the algorithm is a function of $d(x) = \frac{dJ}{d\lambda}$. This way, the algorithm decreases the objective function (plus tolerance ϵ_J) each iteration. After choosing a default policy $\sigma^0(x)$, computing the initial cost $J(\sigma^0(x))$, and calculating the initial entropy h^0 (using Eq. (3.5)) the initial descent direction $d^0(x)$ is calculated for the set of points in S, as described in Sec. 3.4.1.2. A line search [**4**] is performed to find the maximum step size γ that generates a reduction in cost in the descent direction $d^k(x)$, and then the policy

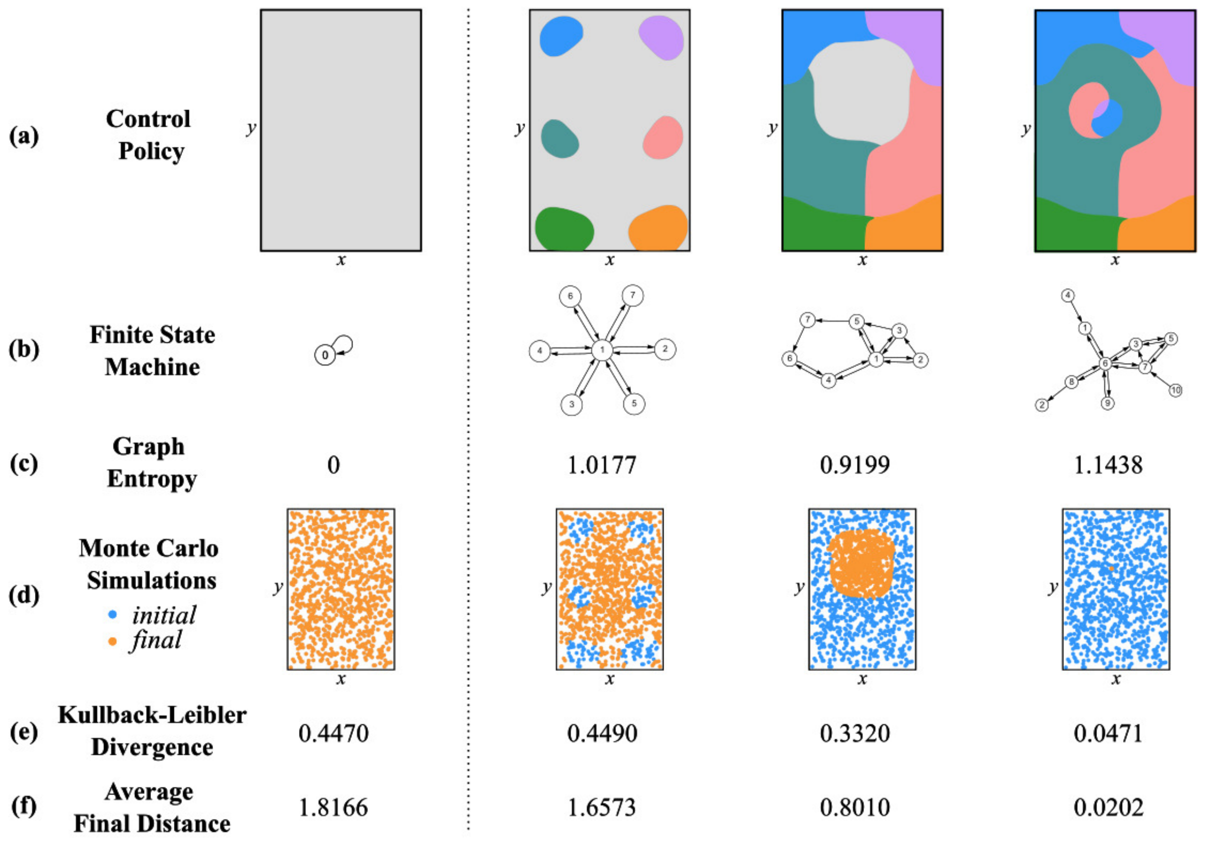


Figure 3.6. Control policies for the system in Fig. 3.3, starting from a null initial policy. (a) The control policies, mapping state to control for various iterations of the line search. (b) A finite state machine representation of each policy, representative of the complexity of the system. (c) The design complexity value calculated from Eq. (3.5). (d) 1000 Monte Carlo simulations illustrate the results of random initial conditions using the associated control policies. (e) The Kullback-Leibler divergence between the goal task distribution and the distribution generated by the control policy, from Eq. (3.6). (f) The average final distance of the final states of (d) from the desired point $\mathcal{P}$.

$\sigma^k(x)$ is updated to the policy $\sigma^{k+1}(x)$. The new design complexity h^{k+1} and descent directions $d^{k+1}(x)$ are calculated, and this is repeated until the cost can no longer be reduced without increasing the complexity beyond the threshold defined by ϵ_h.

The tolerances ϵ_h and ϵ_J are design choices based on how much one is willing to compromise between complexity and performance. In the example illustrated in the next section, the value for ϵ_h is significant because it represents the allowable increase in complexity—how much complexity the designer is willing to accept for improved task embodiment. For these figures, $\epsilon_h = 1.25$ and $\epsilon_J = 10$ were used.

This algorithm enforces low design complexity, meaning it will not result in chattering outputs. The work in [**17**] showed that if $d(x(\tau)) < 0$ then there exists an $\epsilon > 0$ such that $d(x(t)) < 0 \, \forall \, t \in [\tau-\epsilon, \tau+\epsilon]$. Since $d(x)$ is continuous in x (as discussed in Section 3.4.1.2), $d(x_0) < 0$ implies that there exists an $\epsilon > 0$ such that $d(x) < 0 \; \forall \; x \in B_\epsilon(x_0)$. Note that each point in $B_\epsilon(x_0)$ does not necessarily have the same mode of *maximum* descent, but they do each have a *common* mode of descent.

The MIG serves as a descent direction for a volume in the state space, rather than just at a point. This property allows us to assign one control mode throughout a neighborhood so that instead of choosing the optimal control mode (the direction of maximum descent) at each point and causing chattering, we select a good control mode (a direction of descent) throughout a volume and maintain relative simplicity in the policy. Figure 3.5 shows a control policy that is the result of assigning the *optimal* control mode at each point, which results in chattering.

Figure 3.6 begins with an initial control policy of zero control throughout the state space, and increases design complexity and task embodiment until the line search algorithm converges to a new control policy. Monte Carlo simulations were performed with 1000 random initial conditions, shown in row (d) and the average distance of the final points from the desired point is shown in row (f). Most interesting are the trends in rows (c) and (e). These correspond to the min-max problem posed earlier, in which we attempt to minimize design complexity, computed using graph entropy (c), and maximize task embodiment, calculated using K-L divergence (e). The graph entropy in row (c) increases as the K-L divergence in row (e) increases. This shows that the entropy must increase (from 0) to ensure some amount of task embodiment.

Synthetic cells can encode these simplified control policies by physically combining their movement, sensory, and memory elements with a series of logical operators, as discussed next, in Section 3.4.2.

3.4.2. Projecting Policies onto Discrete Sensors

Section 3.4.1 described synthetic cells with perfect state measurement. In this section, implementations using discrete sensors will be explored. In some design processes, it may be possible to *create* sensors that are able to detect exactly where a robot should switch between control modes (e.g., a sensor that can perfectly sense the boundary between the green and orange regions of the control policy). It is also possible that a designer may start with a *fixed* library of sensors, in which case the state space should first be divided into sensed regions and then control modes should be assigned, as described in Section 3.4.3. Another possible scenario, and the one we will examine in this section, is that a designer has many sensors to choose from, and will want to use some subset of them.

For the synthetic cell example, we will assume discrete sensing is provided by a chemical comparator—a device that compares the relative strength of two chemical concentrations. From a given library of sensors, how should the combination of sensors, actuators, and logical operators be chosen so that the task is best achieved?

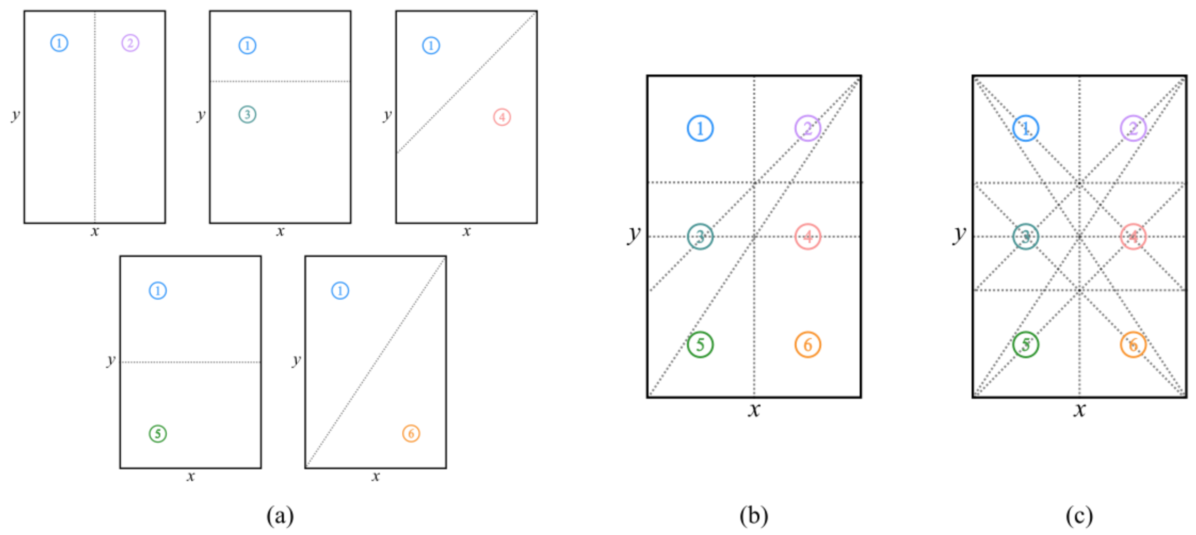


Figure 3.7. (a) Illustration of five individual chemical comparators, each comparing the chemical potential of Source 1 with one of the other sources. Each sensor can tell whether the robot is on one side of an equipotential—the dotted line—or the other. (b) Using a combination of the five sensors, a robot is able to sense which of these 9 regions it is in. (c) Sensor regions resulting from the combination of all possible chemical comparators in this environment.

Figure 3.7 (a) shows five different individual sensors: each comparing the strength of chemical source 1 to another of the chemical sources in the environment, and how each of these sensors is able to divide the state space into two distinct regions, while Fig. 3.7 (b) illustrates which regions of the state space are able to be discerned using these five sensors *combined*. Figure 3.7 (c) shows all possible combinations of comparators: all six chemical sources compared to each of their five counterparts, and therefore the maximum granularity of sensed regions in the state space using this sensor library[3].

[3]Note that some comparators divide the state space in the exact same way, e.g., comparing chemical sources 1 and 3 results in the same sensed regions as comparing sources 2 and 4 (this is true for three other sets of comparators: 1/2 = 3/4 = 5/6, 1/5 = 2/6, and 3/5 = 4/6).

The optimal scenario would be that these sensor regions correspond perfectly to the control regions found using the iterative algorithm in Fig. 3.6. Since this will almost never be the case, we must attempt to approximate our control policy using the library of sensors.

Figures 3.8 and 3.9 demonstrate how the control policy synthesized in the previous section combines with a library of sensors to create a physical design. Figure 3.8 (a) shows two comparators chosen from the sensor library and how they each divide the state space into sensed regions and Fig. 3.8 (b) is the policy that results from projecting the final control policy found in the algorithm onto the feasible sensor space.

This projection from continuous sensing to discrete sensing is done by simulating many rollouts, and finding the best discrete control mode for each sensor region. We pose the question: assuming the best possible control (the unconstrained-sensing control policy from Fig. 3.6) everywhere else in the state space, which control mode should be used inside each individual sensor region?

In each rollout, a trajectory begins from a random initial condition x within the state space (and therefore a random initial sensor region r in the set of sensor regions R). As the simulated synthetic cell executes its trajectory, it uses a single control mode s when it is inside its initial region r, and the continuous sensing control policy $C(x)$ (from Fig. 3.6) when it is outside that initial region r. This is executed for each of the S control modes, and the cost J of each trajectory is calculated based on how long it takes the particle to reach the desired point. In case a trajectory gets stuck in a loop, or for some other reason never reaches the desired point, the algorithm will break the loop after i_{max} increments, and record a large cost for that control mode and sensor region combination. After N

rollouts, we assign the lowest cost control mode s to each sensor region r to construct the projected, discrete control policy $D(r)$. As $N \to \infty$, repeated execution of the algorithm will not change the resulting policy $D(r)$. The projection algorithm also does not depend on the order of executions, and can be computed in parallel. An outline of this process can be found in Algorithm 2.

Algorithm 2 Projection

```
for each rollout $n \in N$
  $x$ = random initial condition
  $r$ = initial sensor region of $x$
  $i = 0$ (increment counter)
  for each control mode $s \in S$
    while $x \neq$ desired point $P$
      if current sensor region of $x = r$
        $u = s$
      else
        $u$ = control from continuous policy $C(x)$
      Update $x$ using dynamics $f(x, u)$
      Update cost $J$
      if $i > i_{max}$
        Break loop and record large cost
    Record cost $J$ for each region $r$ and control mode $s$
Generate discrete policy $D(r) = \underset{s}{\operatorname{argmin}} J(r) \quad \forall r \in R$
```

Logical operators can be combined with sensory observations to represent the state space with more fidelity than sensors alone (e.g., a single sensor in Fig. 3.7 (a))—so that actions can be associated with *combinations* of sensory observations (e.g., Fig. 3.7 (b)). Figure 3.8 (c) illustrates the logical diagram that would be physically encoded in circuitry onto a synthetic cell so that the policy in Fig. 3.8 (b) could be executed.

Figure 3.9 is similar to Fig. 3.8, but illustrates the physical design corresponding to the highest fidelity control policy from the library of comparator sensors. Figure 3.9 (a)

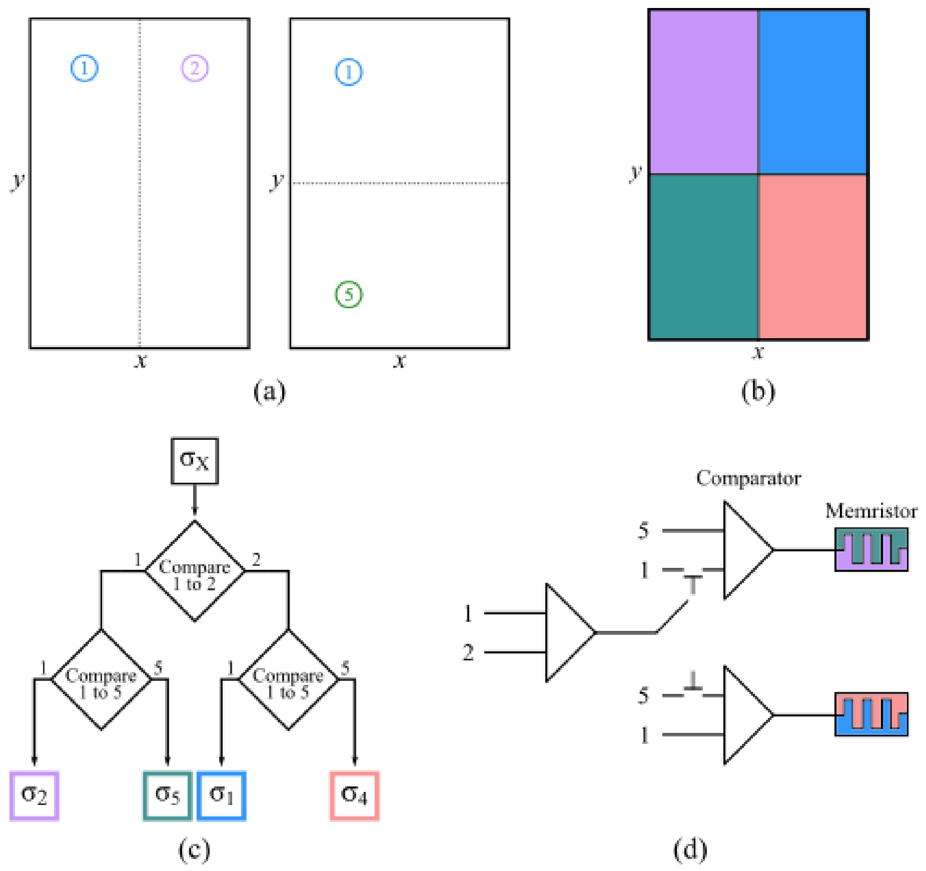


Figure 3.8. Low-fidelity Design. (a) Two sensors. Left: comparing chemical Sources 1 and 2 divides the state space into left and right. Right: comparing Sources 1 and 5 divides the space into top and bottom. (b) Control policy from Fig. 3.6 projected onto the sensed regions. (c) Logical decision diagram for this system. (d) Circuit diagram for physical synthetic cell design.

shows each of the sensors in the library, including the ones that repeat sensed regions due to the symmetry in this environment. The projected control policy shown in Fig. 3.9 (b) and Fig. 3.9 (c) illustrates the logic of the physical circuitry.

It is notable that the designs in Figures 3.8 and 3.9 are quite dissimilar. Figure 3.10 shows how each of the physically feasible designs compare to each other, to another physically feasible design, and to the control policy with perfect knowledge of state. The

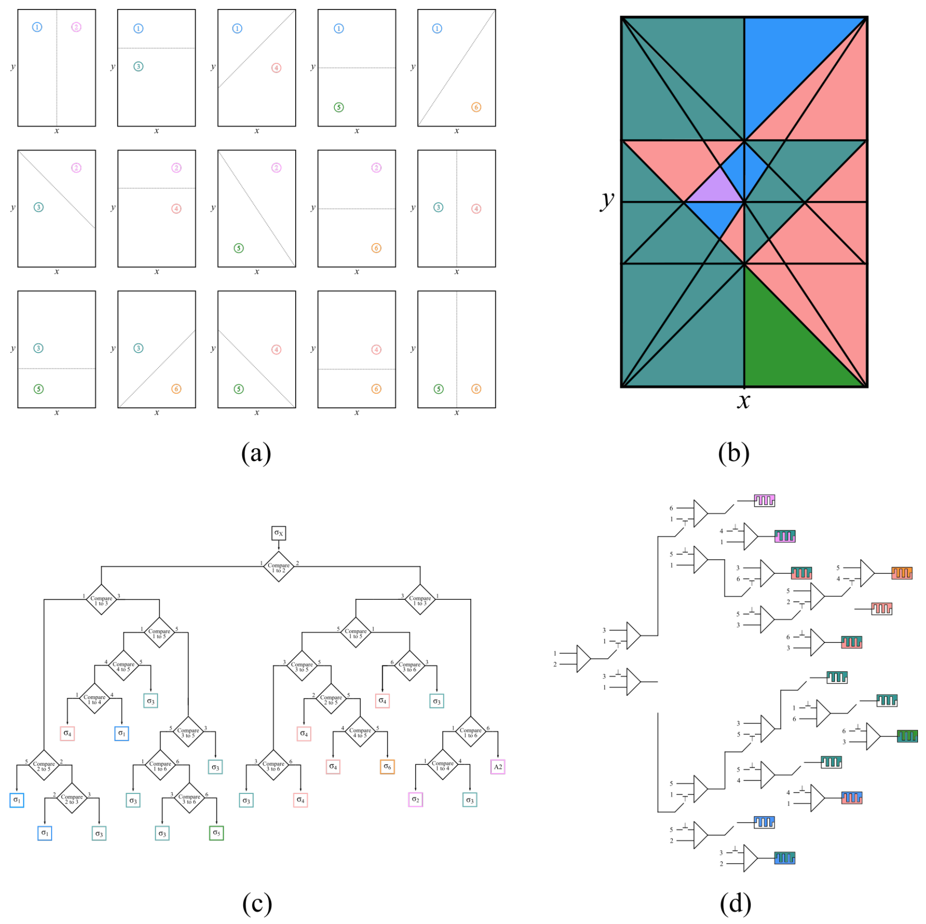


Figure 3.9. High-fidelity Design. (a) Ten distinct sensors. The equipotential lines demonstrate how the device can use chemical comparators to estimate its location in the environment. (b) Control policy from Fig. 3.6 projected onto the sensed regions. (c) Logical decision diagram for this system. (d) Circuit diagram for physical synthetic cell design.

high-fidelity design in the middle of Fig. 3.10 captures much of the structure of the sensor-agnostic policy, and the results are evident in the relatively low K-L divergence. The medium-fidelity design uses fewer sensors than the high-fidelity one and consequently does not embody the task quite as well. The low-fidelity design has the highest K-L divergence, corresponding to the worst task performance. But, depending on the goals of

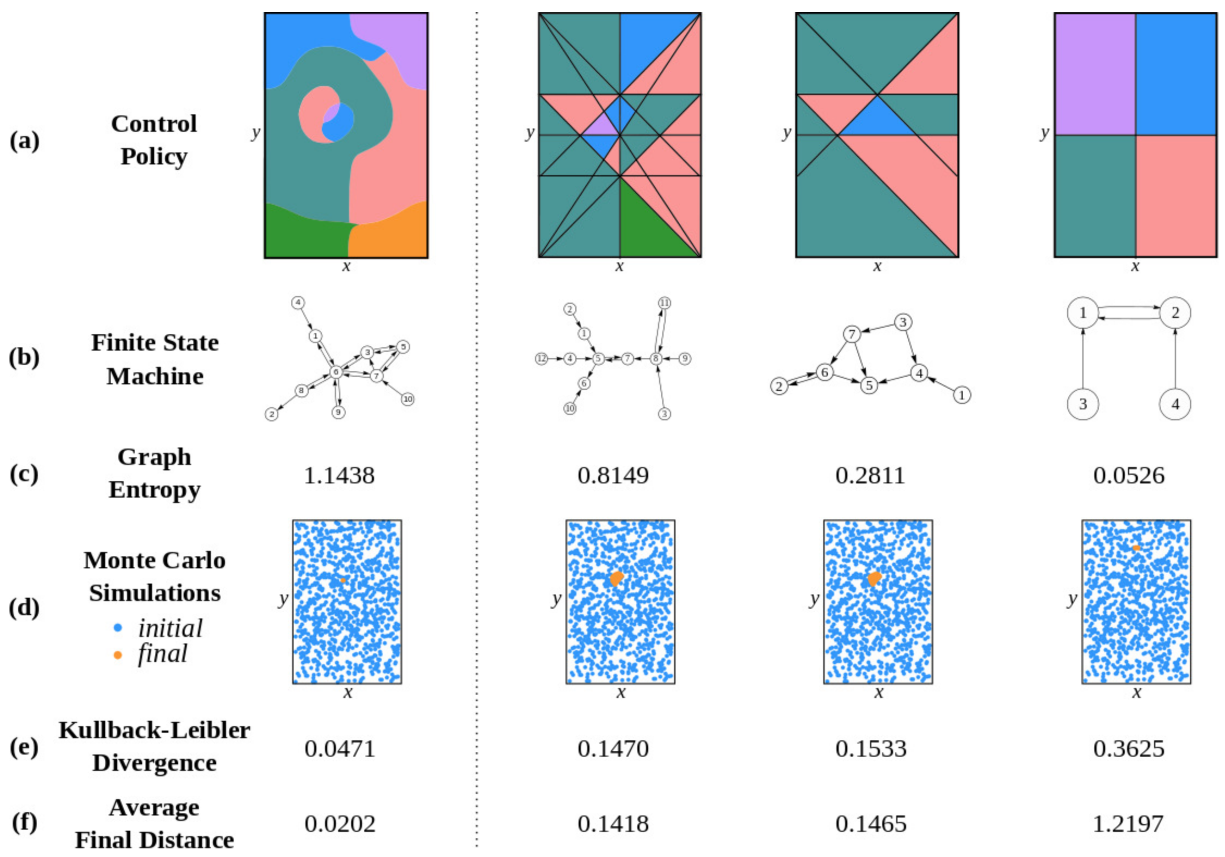


Figure 3.10. Left: The policy generated when perfect knowledge of state was assumed. Right: A high-fidelity design using all of the (10 distinct) sensors in the sensor library, as shown in Fig. 3.9. A medium-fidelity design, using five sensors. A low-fidelity design, using only two of the sensors from the sensor library, as shown in Fig. 3.8.

the designer, it's possible that even this task performance is good enough to sufficiently achieve the goal, and a synthetic cell would be designed in this simplest form.

3.4.3. Generating Policies with Sensors First

The main objective of this chapter has been to encode task information in material properties to produce a simple, physically feasible synthetic cell design that will best achieve a task. We have specified this problem statement to include a set library of sensors (chemical comparators) and actuators (attraction to chemical sources), and in Sections 3.4.1 and 3.4.2 we discussed taking a finite set of control modes, finding a control policy using those discrete control modes and assuming perfect sensing, and then projecting that policy onto discrete sensors. But there may be cases where a robot designer has good reason to solve this problem in the opposite order.

In this section, we will discuss generating a control policy with discrete sensors and assuming continuous control authority, and then projecting this policy onto discrete control modes.

Figure 3.11 (a) shows the (x, y) state space divided into 32 regions using chemical comparators. To compute a control policy assuming continuous control capabilities, we use the same type of control authority as the previous sections (the synthetic cell being attracted to a chemical potential) but the location of that chemical potential is no longer restricted to a few fixed positions.

Figure 3.12 shows an RGB (red, green, blue) color gradient that represents all possible locations of a chemical source in this system. The value of red is increased as the x location of a potential source increases (i.e., moves from left to right) and the value of blue is increased as the position of the chemical potential increases in the y direction (i.e., moves from the bottom to the top). The environment contains a constant amount of green, so where x and y are both small, the color green is most visible (e.g., in the bottom left corner, at $(x = 0, y = 0)$). This color-based representation of (x, y) locations of potential sources allows us to illustrate continuous control authority.

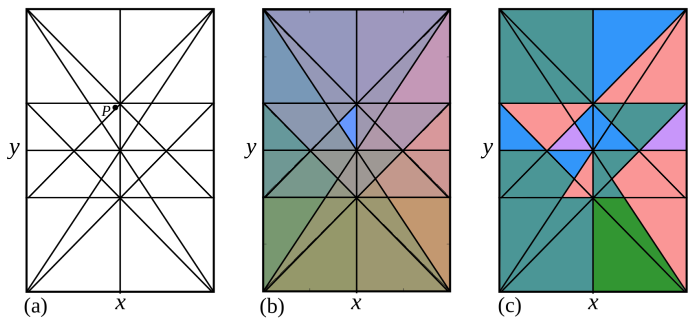


Figure 3.11. (a) Combinations of ten discrete sensors yield 32 distinct regions in the state space. (b) Control policy generated with discrete sensing and continuous control authority (chemical sources placed anywhere in the environment). (c) Control policy projected onto discrete control modes.

A control policy was generated using the discrete sensor regions shown in Fig. 3.11 (a) and the continuous control shown on the left of Fig. 3.12. The resulting policy is shown in Fig. 3.11 (b). This was computed using rollouts: for each sensor region, a control mode (i.e., a location for a chemical potential to be placed) was chosen that would minimize the cost of trajectories starting in that region, and its location is illustrated by the color of each region. The left of Fig. 3.13 shows the performance of this policy.

Note that we are using the same sensor regions (divided by equipotential lines from Fig. 3.7) as in the last section, even though we are choosing new locations for the chemical potentials. On synthetic cells, the ability to detect and compare specific chemicals (as a chemical comparator does) and the ability to be attracted to a certain chemical (resulting in locomotion) are distinct and unrelated. There might be many chemical stimuli in an environment that a synthetic cell can use for state estimation which have no affect at all on a cell's actuation. Here we continue with the assumption that the synthetic cell has chemical comparators on board that pertain to the 6 potentials shown in Fig. 3.3, but assert that there is some possibility in the design space that sources might be added at new positions, or that we otherwise would be interested to know where the best possible source locations are.

3.4.4. Projecting Policies Onto Discrete Actuators

Now that we have calculated our control policy with discrete sensors and unconstrained control authority, the optimal scenario would be to create actuators that perfectly align with the policy: if possible, we should place a chemical potential at each preferred location. Since this is unlikely to be easily achievable, we will approximate our control policy using a library of actuators.

For continuity, we assume that our library of actuators consists of the same control modes shown in Fig. 3.3. We project the computed policy onto 3 subsets of these actuators, shown on the right side of Fig. 3.12. The subsets are: all six control modes, a set of three of the control modes (1, 2, and 3) and a set of only two control modes (1 and 4). We generate synthetic cell designs with each of these sets of actuators by projecting the unconstrained controls computed in the policy to these discrete control modes, and then compare their design complexities and levels of task embodiment in Fig. 3.13.

The projection operator in this section is the same as the one described in Sec. 3.4.2. But in this case, the continuous control policy $C(x)$ (used outside each region being evaluated) is the discrete-sensing, continuous-actuation policy shown in Fig. 3.11, rather than the discrete-actuation, continuous sensing control policy shown in Fig. 3.3.

Figure 3.13 shows the projected control policies and the results of simulations performed with each different design. Monte Carlo simulations were performed with random initial conditions, shown in row (d), and the average distance of the final points from

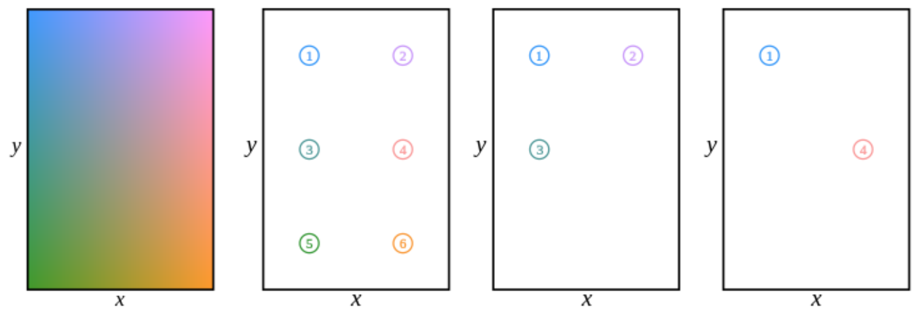


Figure 3.12. Left: All possible locations of chemical sources, represented by RGB (red, blue, green) colors. As the x location of a source increases, the color illustrating the source location becomes more red. As the y location of the source increases, more blue is added to the color. There is a constant level of the color green throughout the state space. Right: Three different subsets of discrete control modes are shown: 6 discrete source locations, 3 locations, and only 2 locations.

the desired point is shown in row (f). Note the trends in rows (c) and (e). Similarly to Fig. 3.10, we observe that as the graph entropy in row (c) decreases, the K-L divergence in row (e) decreases. This shows that as the designs become simpler (i.e., use fewer control modes), the task performance becomes worse. But, depending on the goals of the robot design process, it's possible that even the worst performance shown on the right would be a worthy trade off for the simplicity of the design and ease of fabrication.

3.5. Discussion

In this work we addressed the question of designing robots while minimizing complexity and maximizing task embodiment. We demonstrated our method of solving this min-max problem, which included both solving for the organization of actuators first and then projecting onto discrete sensors, and organizing sensors first and then projecting onto discrete actuators. To accomplish the former, an iterative algorithm was introduced that

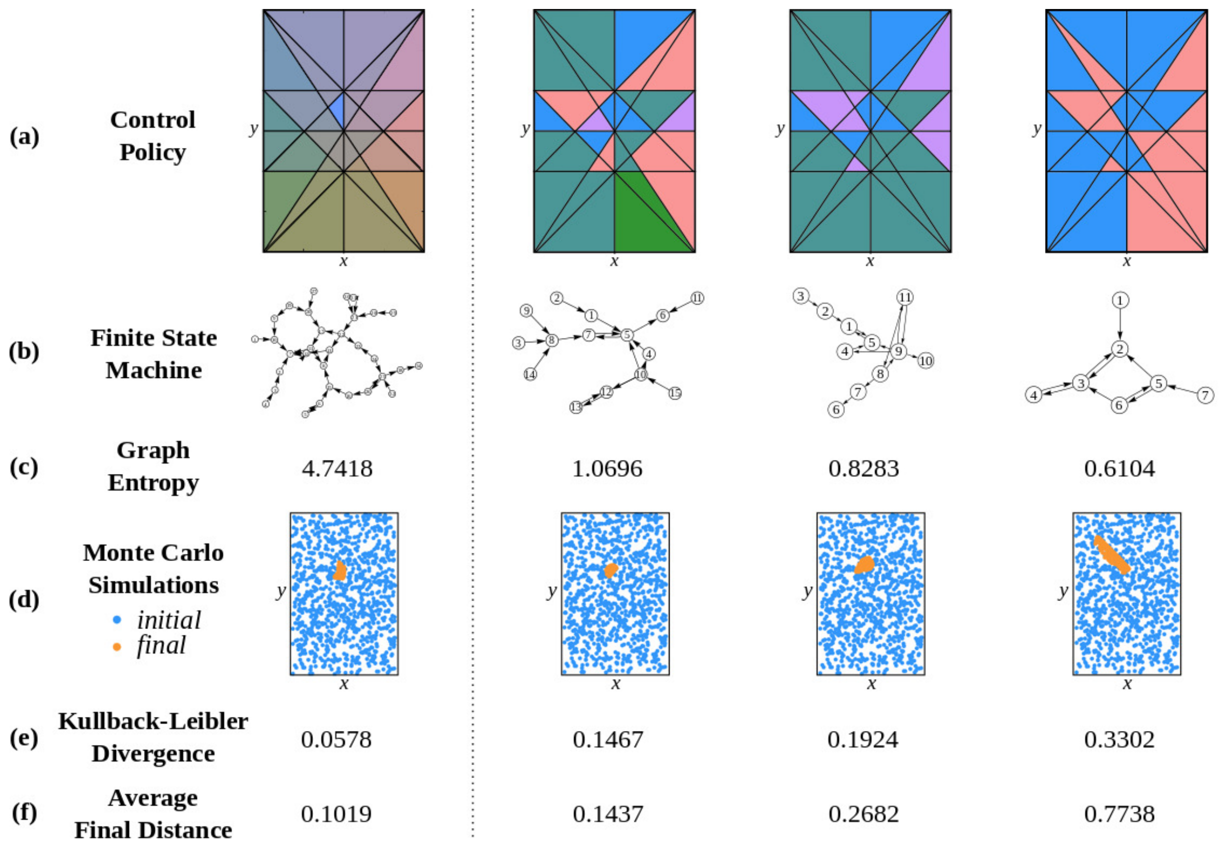


Figure 3.13. Left: The policy generated when continuous control was assumed. Right: A high-controllability design using all 6 of the discrete control modes in the actuator library, as shown in Fig. 3.12. A medium-controllability design using only three control modes (chemical sources 1, 2, and 3). A low-controllability design using only two of the actuators from the actuator library: chemical sources 1 and 4.

resulted in a simple control policy assuming discrete control modes and perfect sensing, and then projecting that policy onto a discrete space of sensed regions resulting from a library of sensors. This is not necessarily an optimal design pipeline for all robot design problems. There may be some instances where there is a fixed library of sensors, in which case one would solve the latter problem, by first dividing the state space into discrete sensor regions, then computing a control policy assuming discrete sensors and unconstrained control authority, and finally projecting the policy onto a discrete library of control modes from a library of actuators.

When these two approaches were applied to the same libraries of sensors and actuators, slightly different designs were generated (as seen in the second columns of Fig. 3.10 and Fig. 3.13). Also, although both methods resulted in similar levels of task embodiment across the different designs, the designs produced by the sensors-first method were more complex. This is because the complexity of a design is directly related to the number of states, which depends significantly on the quantity of sensor regions. In Fig. 3.3 each subsequent design had fewer and fewer sensors where as in Fig. 3.11 each design had the same number of sensor regions, keeping the complexity relatively high.

Overall, the algorithmic approach to robot design demonstrated in this chapter is capable of producing low complexity control policies that are applicable to simple robotic systems. Similar minimal robotic design analyses and comparisons were performed in the previous chapter. One limitation of these results is the fact that minimal and microscopic robots are constrained in their capabilities, and each agent can only have a small impact on its environment. In the next chapter we will investigate how robots can be designed for collaboration – so that they can work together to achieve things that are difficult or impossible individually.

CHAPTER 4

Bayesian Particles

Cells in the human body are individually simple, but create unimaginably complex structures and functions when they work together. We can mimic these emergent phenomena by building simple organisms that, when in a group, are capable of complex behaviors. In this chapter, we demonstrate a scalable framework for theoretical analysis of cyclic environments called *uniformly oriented graphs*. These UO graphs represent topologies like that of mammalian circulatory systems, ventilation systems in buildings, oil pipelines, and other naturally-occurring and man-made systems. We design simple, minimal agents (with capabilities up to current state of the art [**56**]) that search for targets in a dynamic, stochastic environment. Using only a few bits of memory, simulated agents navigate through a uniformly oriented graph, detect a moving target, and communicate their discovery to other agents. I show that the exhibited behavior of the ensemble functions as a physical implementation of a Bayesian update. Lastly, I validate the results by demonstrating our algorithms on macroscopic robots which work together to find a moving target in a model of the human circulatory system using infrared sensors and distance-based radio communication. These results pave the way for designing and producing microscopic robots for a diverse spectrum of applications.

4.1. Introduction

As robot size decreases to the order of a single cell, previously inconceivable applications and abilities emerge. These include monitoring of oil and gas conduits [**50**], electrophysiological recordings with neural dust motes [**89**], minimally invasive medical procedures [**93**], and much more. Because of recent advances in the manufacturing and development of colloidal robots [**56**], researchers can reasonably assume that small-scale untethered mobile robots can cooperate in collectives of hundreds or thousands of agents. Applications for such large groups of such tiny robots include exploration, coverage, and environmental monitoring. With these novel applications in mind, we pose the question: what is the simplest, physically realizable coordination strategy that micro-particles could use to guarantee physical coverage?

Many instances of microorganisms cooperating and self-organizing have been demonstrated [**109**]. Complex behaviors (e.g., learning) do not only exist in multi-cellular organisms. Collective behaviors emerge even in (possibly, especially in) simple systems [**88**]. Slime molds are single-celled organisms that have neither neurons nor brains, but work together as a single entity to achieve feats such as solving mazes and anticipating periodic events [**11**]. They have been shown to solve the U-shaped trap problem—a common test of autonomous navigational ability in robotics [**85**]. Our goal in this work is to emulate this type of ensemble behavior where each individual cell works to ensure the success of the group, using the extended example of mimicking the immune response.

The immune system protects the body by recognizing and responding to antigens, which are harmful agents like viruses, bacteria, and toxins [**22**]. When white blood cells find a target, they multiply and send signals to other cells to communicate their discovery

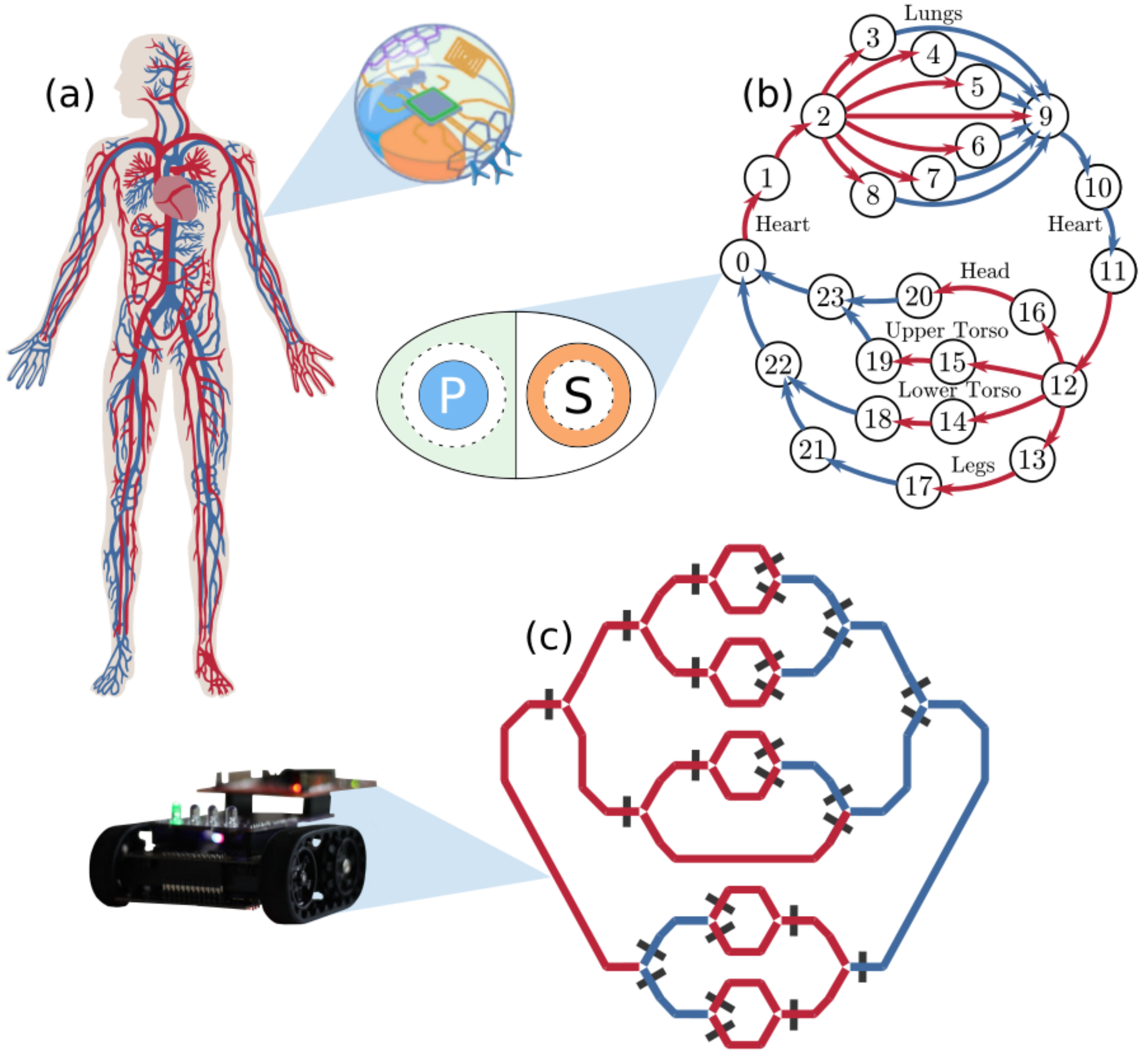


Figure 4.1. (a) An illustration of the human circulatory system, inset with an illustration of a synthetic cell [**57**]. (b) A graph representing a simplified human circulatory system, with an agent (Bayesian particle) containing a policy bit **P** and a success bit **S**. Similar to what's shown in [**57**] (c) The experimental set up of the circulatory maze and one of the Bayesian robots.

[**66**]. I show that a group of synthetic cells can imitate this discovery and communication behavior by collectively executing a simple algorithm that manifests itself as a Bayesian update over the control policy that brings cells to the location of an antigen.

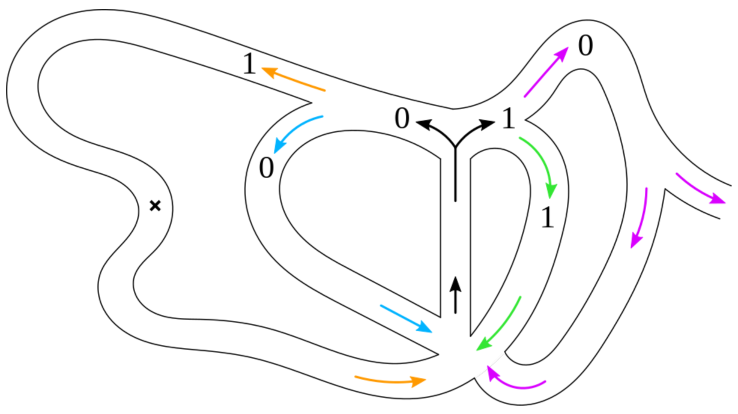


Figure 4.2. A cyclic maze that mimics aspects of the circulatory system. A group of synthetic cells would require at least three bits each to navigate through the four paths and record instances of target detection. One of the paths (10) leads to a juncture where cells might get lost and not return.

Synthetic cells are microscopic devices with limited sensing, control, and computational capabilities [**59**]. They can contain simple circuits that include minimal sensors and limited nonvolatile memory—barely a handful of bits [**58**]. These devices are around 100μm in size, rendering classical computation using a CPU likely impossible. But simple movement, sensory, and memory elements can potentially be combined with a series of physically realizable logical operators to enable a specific task [**73, 57**] and communication about how to achieve that task.

A group of synthetic cells can learn to find a target by beginning with different control policies—indicating *how*, and implicitly where, they will explore—and then communicating with each other that they have or have not been successful in detecting a target. After communicating their success, some synthetic cells will change their control policies to reflect the successes of others in the group. Thus the distribution of synthetic cell control policies reflects the expected location of the target.

This update is similar to a particle filter, where samples from a distribution are represented by a set of particles, and each particle has a likelihood weight assigned to it that corresponds to the probability of that particle being sampled from the distribution. Particle filters also include a resampling step, to mitigate weight disparity [**98**] before the weights become too uneven, which closely mirrors the communication step in this synthetic cell implementation, as discussed in Section 4.4.

In this chapter we show how minimal agents can use simple, local algorithms and only a few bits of memory to enable global learning behavior to refine their belief of a target location. We also show that this implementation of the ensemble of agents is a sub-optimal Bayesian filter, where the control policy of the agents is the independent variable. By constraining individuals to collectively behave as a Bayesian update, the group inherits formal properties in the form of guarantees on asymptotic performance and probabilistically predictable behavior. These properties will help us to reason about robustness and safety in task execution. That is, we are replacing the model of a distributed system with a single Bayesian filter. Lastly, we validate our results in experiments with a robot swarm, using line-following robots shown in Fig. 4.1.

4.2. Related Work

Literature surveys of previous work on nanotechnology and mobile microrobots can be found in [**58**] and [**93**], respectively. In the discussion of existing challenges associated with designing miniaturized robots for biomedical applications, [**93**] notes that most robots with dimensions less than $1mm$ use an "off-board" approach where the devices are externally actuated, sensed, controlled, or powered. In this work, we employ fully autonomous devices that process information and act independently of external drivers and centralized computers.

Micro- and nanorobots can also be classified as either synthetic or biohybrid, where the former are composed of fully synthetic materials (polymers, composites, metals) and the latter are made of both synthetic and biological materials (bacteria, algae, protozoa). Both have benefits, but biohybrid robots can only exist under specific biological conditions, requiring limitations on the environment, temperature, pH, salinity, and more [**50**]. This constrains the envisioned applications compared to purely synthetic devices, which is what we employ in the models in this work.

Research in nanofabrication and synthesis methods have yielded sophisticated synthetic devices, including particles that serve a particular function (e.g., light control for nanoactuation [**30**], performing clocked, multistage logic [**108**], actuation using external magnetic fields [**92**], [**110**]), but not particles possessing autonomous circuitry, logic manipulation, and information storage [**50**]. Besides the work published in [**50, 57**], existing micro- or nanoparticles do not autonomously process information when decoupled from their environment [**44**], [**45**]. The particles created in Koman and Liu et al. [**50, 57**] are the basis for the synthetic cells proposed in this chapter, and we will make the following assumptions based on this work:

(1) Synthetic cells can guide their own motion either mechanically [**36**], by means of elaborate swimming strategies like rotating helical flagella [**53**], or (more likely) chemically [**63, 62**], through use of Pt-Au bimetallic rods [**72**], self-electrophoresis [**38**], [**64**], self-diffusiophoresis [**40**], or self-thermophoresis [**43**].

(2) Synthetic cells can send and receive communications optically, using integrated LEDs [**103**] and solid-state or organic light emitting diodes [**103, 52**].

(3) Synthetic cells can detect a target by using a chemiresistor to recognize the target's specific chemical analyte [**50, 59, 58**].

Safety and robustness guarantees are the most important qualities for biomedical microrobots operating inside a body. If devices are to be employed in medical applications, they must not damage tissues or cause any negative reaction from the body. One of the primary contributions of the work in this chapter is constraining synthetic cells to behave as a physical implementation of a Bayesian update, creating a basis for formal properties and guarantees on their behavior. This ensures robustness in their performance, which can be translated to safety guarantees in specific environments.

Bayesian approaches have been applied to reinforcement learning [**82**], [**83**], but often in supervised scenarios where there are experts and learners. These methods also often employ passive observations by the learners, rather than explicit communication from experts to learners. In this chapter, we employ reinforcement learning of policy updates in rollouts of executions. We also enforce communication between agents, specifically from successful agents to unsuccessful ones when they encounter each other.

Many relevant papers on swarm robotics [**12**] and self-organized collective decision making [**26, 99, 86, 100**] exist, but they generally apply to robot swarms that have significantly more computation and capabilities than those presented in this chapter. Therefore, the formalisms, perspectives, and strategies presented in these works are informative, but not applicable at a scale where agents communicate only 6 bits of information.

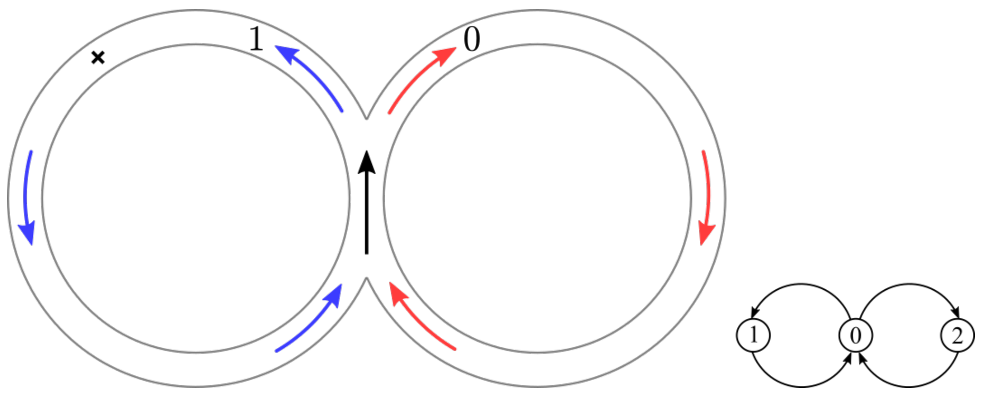


Figure 4.3. Left: An example cyclic maze with only two possible paths. Based on the control policy (one bit: a 0 or a 1) of each two-bit synthetic cell, the cell will follow either the right path or the left path in search of the target $\times$. The cell uses its second bit to encode whether or not it has detected the target. Right: A graphical representation of this synthetic cell environment with node 0 as the heart node that every agent passes through.

4.3. Algorithm and Problem Definition

4.3.0.1. Environment. Our goal is to mimic the immune response, so I simulate a model of the circulatory system [**37**] in Section 4.4. But this work also applies to cyclic systems of different types. With the help of my collaborator Jamison Weber, I have defined *uniformly oriented graphs* as the class of environments that will be considered. A uniformly oriented (UO) graph is a specific class of directed random graphs, in which there exists one vertex h that is an element of every directed cycle. More details about UO graphs can be found in Section 4.6.

In this section, I present a simplified, introductory model. This model consists of a maze, shown in Fig. 4.3, with only two possible paths: left or right. The central vertex, or “heart node,” is node 0. In this example, the agents will search for a target located at the black $\times$.

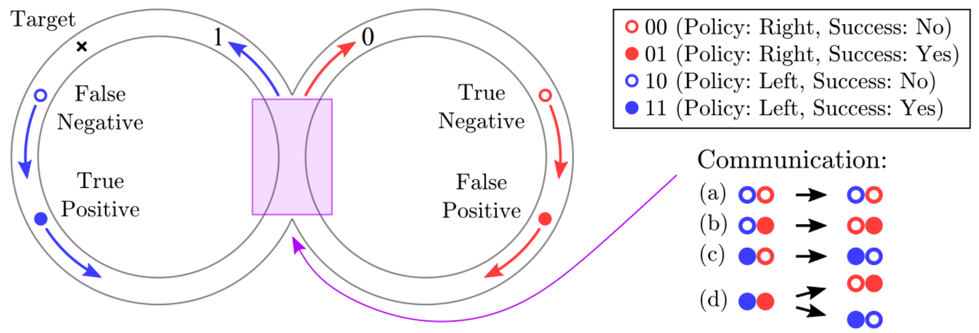

Target
1
0
False Negative
True Positive
True Negative
False Positive
○ 00 (Policy: Right, Success: No)
● 01 (Policy: Right, Success: Yes)
○ 10 (Policy: Left, Success: No)
● 11 (Policy: Left, Success: Yes)
Communication:
(a)
(b)
(c)
(d)

Figure 4.4. Each agent begins with an initial control policy of either a 1 or a 0, which causes it to turn either left or right in the maze. If the cell thinks it has found the target, it changes its second bit (its success bit) to a 1. Due to stochasticity in the environment and the cells, there is the possibility of a false positive or a false negative. Cells might communicate in the middle region, shown in purple. (a) If both cells are unsuccessful they will not communicate any information even if they are within range of each other. (b) and (c) If one cell is successful and one isn't, the successful cell will communicate its policy to the unsuccessful one. (d) If both cells are successful, one (selected randomly) will listen to the other.

4.3.0.2. Policy Execution. For this introductory example, each synthetic cell has only two bits: one for its control policy (1 for left or 0 for right) and one to indicate whether it has found the target (1) or not (0). Each cell begins with a random control policy and loops through the maze. They have a probability of a false positive p_{fp} (detecting the target when it is *not* there) and a probability of a false negative p_{fn} (*not* detecting the target when it is there). If a synthetic cell thinks it has detected the target, it changes its second bit, which we will call its *success bit*, to a 1. This policy execution is illustrated in Fig. 4.4.

4.3.0.3. Communication. Using methods of optical information transmission discussed in Section 4.2, synthetic cells are capable of local communication when they are within a certain distance of each other. Note that in the graphical representation of the environment, I model this as agents exchanging information when they are on the same node as each other.

When agents reconvene in the middle of the maze (defined by the purple box in Fig. 4.4, and the node 0 in Fig. 4.3), there is an opportunity for communication. I use a parameter ρ to characterize how many other synthetic cells, on average, each agent will interact with during one loop of the maze (no matter what control policy or success bit either cell has). This parameter ρ is related to the density of synthetic cells in the environment, and how likely they are to pass within communication range of each other.

If two agents come into contact, a successful agent (with a 1 for its success bit) will tell an unsuccessful agent (with a 0 for its success bit) its “correct” policy—even if it is successful because of a false positive. If both communicating agents are successful, one will listen to the other, but which one is the listener is randomly chosen. By “randomly” I mean that there is a probability of 0.5 that either agent will be chosen as the listener. And if both are unsuccessful they will not tell each other anything. In this way, the success bit also functions as a read/write bit. If it is a 0, the cell will listen to others (read) and if it is a 1, the cell will try to broadcast its policy to others (write). These different scenarios are depicted in Fig. 4.4.

4.3.0.4. Behavior Algorithm. These policy execution, communication, and target detection behaviors are described in detail in Algorithm 3. The agents begin at the heart node 0 with randomly assigned control policies and they step forward to successive nodes according to that policy (line 17). There is a parameter r indicating the amount of randomness that unsuccessful agents use to explore the environment (line 6). If the agents detect the target (lines 3 and 8), they record a success (lines 4 and 9), and will broadcast their policy (line 12) to any other agent in their proximity (i.e., at the same node as them). If an unsuccessful agent receives a message from a successful agent (line 14), it will switch its policy to that of the successful agent (line 15). If two successful agents hear each others' policies and they are different, one will randomly listen to the other's policy. An agent that has listened to a new policy will also change its success bit (line 16) to be *nearly* off – but not completely 0. This enables an agent to avoid randomizing its behavior immediately upon encountering the heart node (line 6), so that it has a chance to try out its new policy for a cycle or two.

Algorithm 3 Bayesian Particle Behavior

```
1: $s$ = success bit, $\tau$ = target, $u$ = current node, $\hbar$ = heart node, $p$ = policy bits,
   $r$ = random policy generation rate, $\delta$ = decay amount $\delta << 1$.

2: if $s = 0$ (agent unsuccessful)
3:     if $\tau = 1$ (target detected)
4:         $s \leftarrow 1$ (flip success bit)
5:     if $u = \hbar$ (at heart node)
6:         Generate new random policy $p$ with probability $r$
7: else if $s > 0$ (agent successful)
8:     if $\tau = 1$ (target detected)
9:         $s \leftarrow 1$ (recharge success bit)
10:    else if (target not detected)
11:        $s \leftarrow s - \delta$ (decay success bit)
12:    Broadcast policy $p$
13: Listen for policy $p_{new}$ from other agents
14: if $p_{new}$ received
15:    $p \leftarrow p_{new}$ (reset policy)
16:    $s \leftarrow clog(n)\delta$ (reset success bit)
17: Step forward according to policy $p$
```

When synthetic cells enact their simple algorithm of policy execution, possible target detection, and communication, the cells all end up with the policy that passes the target.

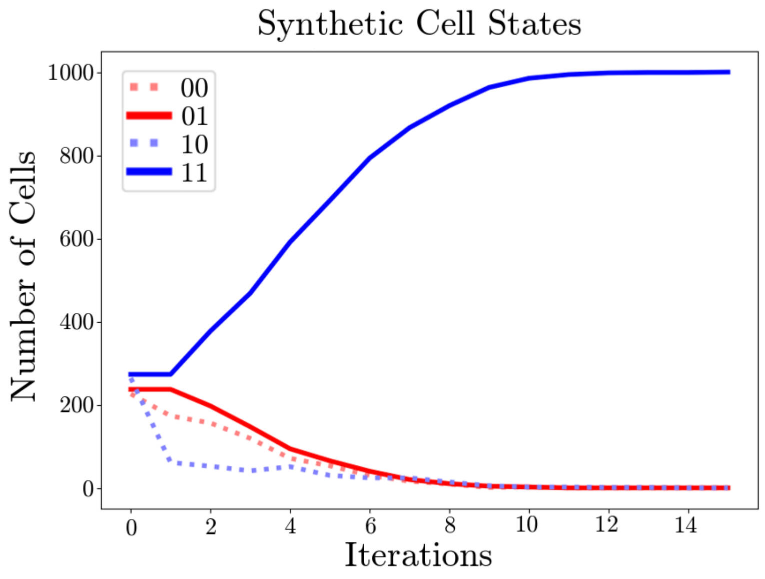


Figure 4.5. Top: Results of a simulation with 1000 two-bit synthetic cells for 15 iterations (15 loops around the maze shown in Figure 4.3), with parameters $p_{fp} = 0.2$, $p_{fn} = 0.2$, $\rho = 1.0$, and $r = 0.001$.

In Fig. 4.5, the number of agents with each state are shown as they loop through the maze multiple times and communicate with each other between loops. By the ninth iterate, every cell has a policy that takes it past the target. An animation of this simulation can be found at https://sites.google.com/u.northwestern.edu/bayesianparticles. This optimal final result always occurs in the case of this simple maze, as long as p_{fp} and p_{fn} are sufficiently small and $\rho > 0$. But with more complicated environments and possibilities (for example, the maze in Fig. 4.2) it becomes more difficult to ensure this result. To address this, we increase the number of bits on each cell.

4.4. Simulations

I now introduce a more complex example, where each synthetic cell has three bits and the maze has four possible paths. The environment is shown in Fig. 4.2, where the

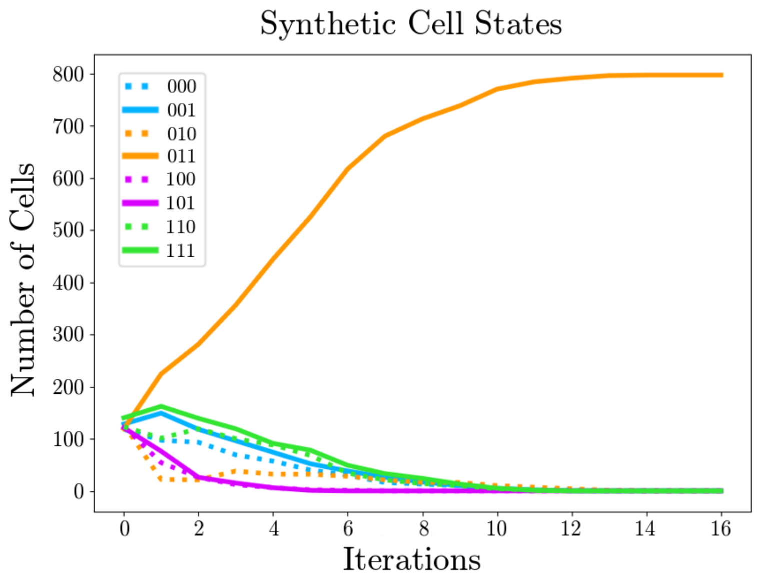


Figure 4.6. Simulated results for 1000 three-bit synthetic cells executing their policies and communicating in the maze from Figure 4.2, with parameters $p_{fp} = 0.2$, $p_{fn} = 0.2$, $p_{lost} = 0.5$, $\rho = 1$, and $r = 0.001$. Around 800 of them converge to the correct policy where they will find the target.

possible control policies are: 01, which takes the synthetic cells past the target; 00 and 11, which both loop the cells around the maze; and 10, which leads the cells down a path that they have probability p_{lost} of never returning from. The goal is for as many cells as possible to end with the 01 policy, where they will all be heading toward the target.

The possibility of getting lost adds further complexity to the system, because not only is the target not reachable with policy 10, but some cells with that policy will not return at all. This could be equivalent to different environmental factors in a body, for example an area with enough acidity to damage or destroy synthetic cells. In practice, I predict that there will be many opportunities for synthetic cells to veer off course and get lost, or to get stuck such that they can no longer contribute to the goals of the group. As the magnitude of p_{lost} increases, more cells get lost and fewer are able to reach the target and combat an invading antigen.

Figure 4.6 shows results for 1000 three-bit synthetic cells exploring the environment shown in Fig. 4.2. All but the lost cells converge to the target policy after 12 iterations.

4.4.1. Particle Filter

A particle filter is a nonparametric implementation of a Bayes filter that represents a distribution using a set of random samples drawn from that distribution [**10**], [**98**]. In a particle filter, the samples from the distribution are called *particles.* I denote these samples $X_n := x_n^1, x_n^2, ..., x_n^L$. Each particle x_n^ℓ is a hypothesis of the true world state at time n—in our example, each particle would be a hypothesis of the policy that leads to the target.

The most basic variant of a particle filter algorithm begins with the particle set X_n and weights w_n, which together represent a prior distribution. This distribution is sampled, resulting in L particles $\bar{x}_{n+1}^1, ..., \bar{x}_{n+1}^L$. The bar indicates that these samples are taken before the measurement has been incorporated. Next, a measurement z_{n+1} is obtained, and it is used to calculate new weights w_{n+1}^ℓ for each particle. The weight is the probability of the measurement given each particle, $w_{n+1}^\ell = P(z_{n+1}|\bar{x}_{n+1}^\ell)$. Lastly, the particle filter resamples the distribution by drawing with replacement L particles from the weighted set $\bar{X}_{n+1}$, where the probability of drawing each particle is given by its weight. The resampled particles $X_{n+1} = x_{n+1}^1, ..., x_{n+1}^L$, along with the weights w_{n+1}, represent the posterior distribution—an updated estimate of which policy leads to the target. Note that in the case of the synthetic cell ensemble, the particle filter is estimating discrete

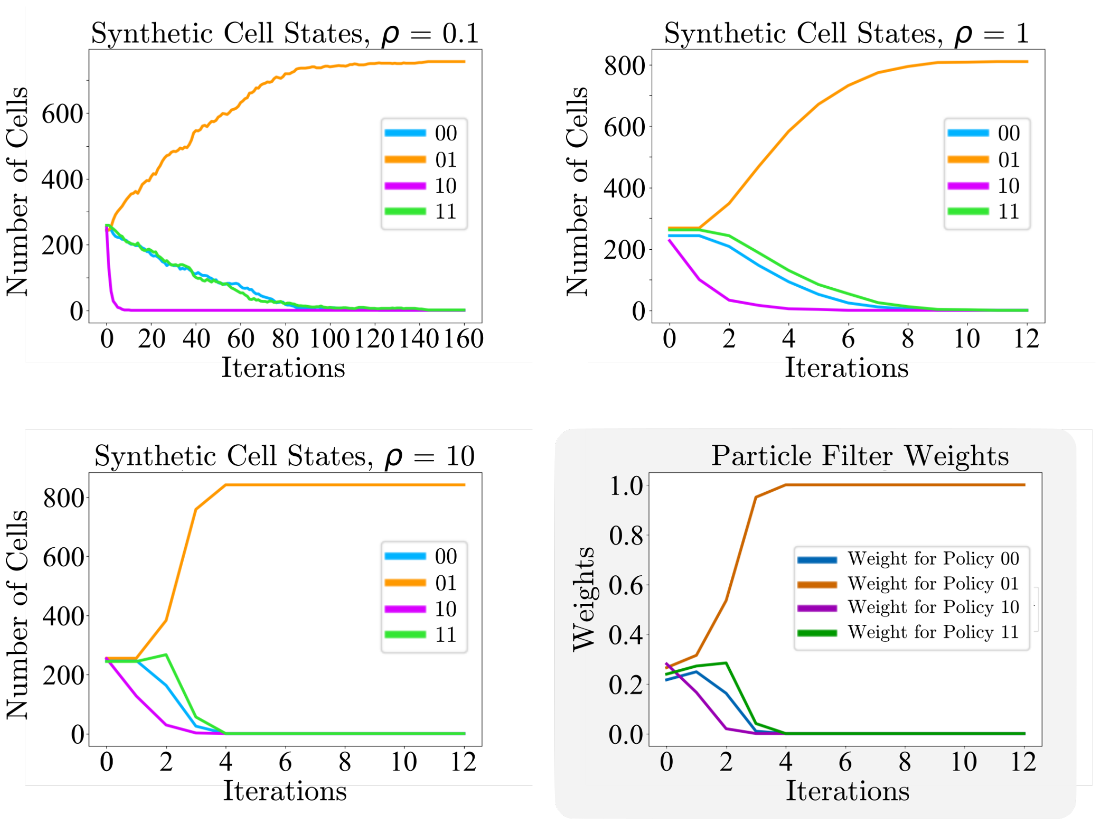


Figure 4.7. Synthetic cell executions for different values of ρ for 1000 synthetic cells and $p_{fp} = 0.1$, $p_{fn} = 0.1$, $p_{lost} = 0.5$ and $r = 0.001$. The bottom right panel shows a particle filter implementation for this system, where the measurements are the current states of the synthetic cells. As ρ increases, it approximates the particle filter. Note that the bottom left plot, where $\rho = 10$, is nearly identical to the plot of the particle filter weights.

states: each particle can only take one of four different values. There are much more than four particles, so many particles will hypothesize that the target is at the same state.

For the synthetic cell implementation, we begin with random policies (and random success bits) on all of the synthetic cells, similar to starting with a uniformly distributed prior distribution. This distribution of M synthetic cells has discrete states $Y_n := y_n^1, y_n^2, ..., y_n^M$. Each of these states are a policy, or a path through the maze that the target might be along, and a success bit, that is on or off. The cells execute their policies and some return with an altered success bit, resulting in new cell states $\bar{y}^1_{n+1}, ..., \bar{y}^M_{n+1}$. The value of the success bit s^m_{n+1} of each cell $\bar{y}^m_{n+1}$ is a binary implementation of the weight w^m_{n+1} in a particle filter. Instead of calculating a conditional probability so that the weights are *between* 0 and 1, the weights are *either* 0 or 1 (before being normalized by the total number of cells). A cell m with state $\bar{y}^m_{n+1}$ with a 0 success bit will not communicate its policy to any other cells, meaning, in particle filter terms, that it will not be sampled from—so its weight is effectively $w^m_{n+1} = 0$.

For example, consider a situation where there are M synthetic cells (and M_{001} synthetic cells with policy 00 and a 1 for a success bit, M_{110} cells with policy 11 and a 0 for a success bit, etc.) and $\rho = 1$, meaning that each cell will communicate with one other cell during each loop around the maze. Cell m (with state $\bar{y}^m_{n+1}$) has a $\frac{\rho}{M}$ probability of communicating with any other cell i (with state $\bar{y}^i_{n+1}$), where $1 \leq i \leq M$, during a given cycle. Cell m's probability of sampling a cell with policy 00 is $\frac{M_{001}}{M}$, its chance of sampling a cell with policy 01 is $\frac{M_{011}}{M}$, and so on. It also has a chance of staying the same, anytime it communicates with a cell with a 0 success bit, which occurs with probability $\frac{M_{000}+M_{010}+M_{100}+M_{110}}{M}$.

This is the main difference between the particle filter algorithm and the synthetic cell implementation: the synthetic cells have some probability of *not* resampling, and just staying the same—unlike particles in a particle filter which are all resampled, every iteration. This difference is demonstrated in Algorithm 4. If each cell *always* communicated with a random successful cell, its behavior would be the same as that of a particle filter. This is illustrated in Fig. 4.7. The bottom right panel of Fig. 4.7 shows a particle filter applied to the synthetic cell system. There are $L = 1000$ particles being randomly

Algorithm 4

```
Particle Filter
1: Prior distribution is described by $L$ particles and their weights
    $X_n, w_n$
2: Distribution is sampled, resulting in $L$ new particles
    $\bar{x}^1_{n+1}, \bar{x}^2_{n+1}, ..., \bar{x}^L_{n+1}$
3: Based on a measurement, weights[1] are assigned to each particle
    $w^\ell_{n+1} = P(z_{n+1}|\bar{x}^\ell_{n+1})$
4: Resample by drawing with replacement $L$ particles from weighted set $\bar{X}_{n+1}$
5: Posterior distribution is described by the resampled particles and their weights
    $X_{n+1}, w_{n+1}$
```

```
Synthetic Cell Implementation
1: Prior distribution is described by $M$ policies and success bits
    $Y_n, s_n$
2: Cells execute their policies, resulting in $M$ new states
    $\bar{y}^1_{n+1}, \bar{y}^2_{n+1}, ..., \bar{y}^M_{n+1}$
3: Based on its success bit, a cell might broadcast its policy
    $s^m_{n+1} = 0$ or 1
4: Each cell $\bar{y}^m_{n+1}$ communicates with $\rho$ other cell(s). This approximates resampling as $\rho \to M$.
5: Posterior distribution is described by the final synthetic cell policies and success bits
    $Y_{n+1}, s_{n+1}$
```

sampled from the synthetic cell distribution (which is also comprised of $M = 1000$ cells), and the weights are being updated based on observations of synthetic cell policies. As ρ increases, the amount of resampling increases, and the synthetic cell behavior is guaranteed to converge to the particle filter behavior. The physical execution of the group of synthetic cells approximates the particle filter algorithm.

Similarly, a particle filter approximates a Bayes filter. The approximation error of a particle filter approaches zero as the number of particles goes to infinity—*the error depends on the number of particles, not on the resampling.* In fact, some particle filter implementations resample very infrequently, to reduce the risk of losing diversity [**98**]. Because the asymptotic guarantee on a particle filter approximating a Bayes filter does not depend on resampling [**98**], it consequently holds for synthetic cells as well. Therefore, since the synthetic cell implementation approximates a particle filter, and a particle filter approximates a Bayesian update, we can conclude that a synthetic cell system using this algorithm approximates a Bayesian update.

This result, which is illustrated in Fig. 4.7, guarantees convergence properties for how synthetic cells will probabilistically behave. These guarantees are valuable because they can be used to reliably predict how synthetic cells will perform in new scenarios, and we can be certain of robustness and safety requirements for physical experiments.

4.4.2. Circulatory System

Many models of the human cardiovascular system exist, including a 36 vessel body tree [**61**], a lumped parameter model [**49**], and a mathematical model featuring both linear and nonlinear constitutive relations [**24**]. In this chapter, I use the model from Hardy et al. [**37**], which clearly defines the 24 different chambers in the circulatory system, as well as the connections going into and out of each one. This model is shown in Fig. 4.8, where each number represents a chamber, as described in the legend, and the connections depict inputs and outputs for blood flow. Figure 4.9 shows how the graphical representation of the circulatory system can be illustrated as the same type of maze that was shown in Figures 4.2 and 4.3.

How much memory does an agent require to navigate in this environment? I have derived the following equation, which computes the number of bits, B, required for any cyclic graph.

$$B = 1 + \sum_{i=1}^{I} ceil(log_2(P_i)) \tag{4.1}$$

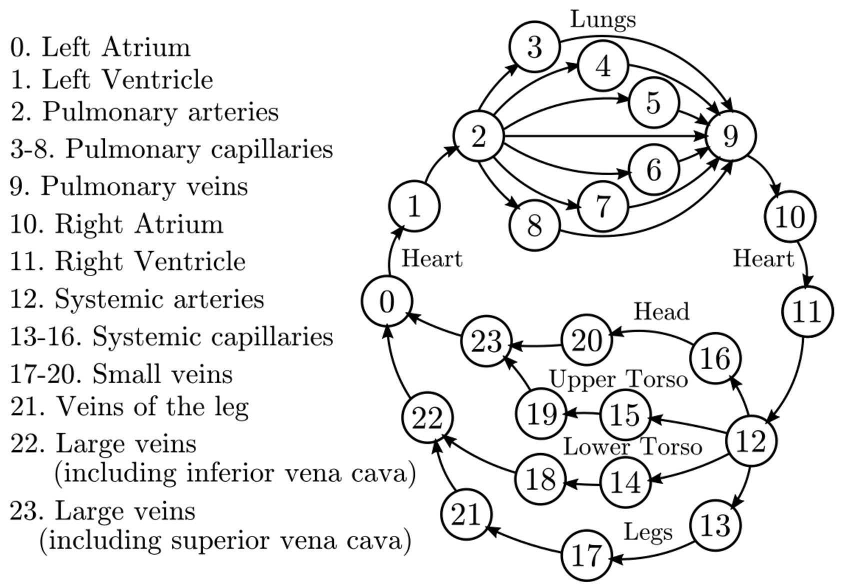


Figure 4.8. Adapted from Figure 1, Figure 2, Figure 3, and Table 2 in Hardy et al. [**37**]. Each number represents a chamber, as described in the legend. The connections between chambers are inputs and outputs illustrating blood flow.

In Eq. 4.1, B is the number of bits required to navigate the graph, I is the number of intersections, or diverging nodes (nodes that have multiple edges leaving them), and P_i is the number of edges leaving each intersection, i. For each intersection $i \in I$, I calculated the binary log of the number of outgoing edges. This gives the number of bits necessary to describe the policy at that intersection. The ceiling function *ceil* rounds up to the nearest integer, as I only consider entire bits. The total number of bits required for each intersection is summed, and one more bit is added to measure success.

The circulatory system shown in Fig. 4.8 has $I = 2$ intersections, at nodes 2 and 12, which have $P_1 = 7$ and $P_2 = 4$ outgoing edges, respectively, and therefore $B = 6$ bits are required to solve the graph. The policy bit organization is shown in Fig. 4.9.

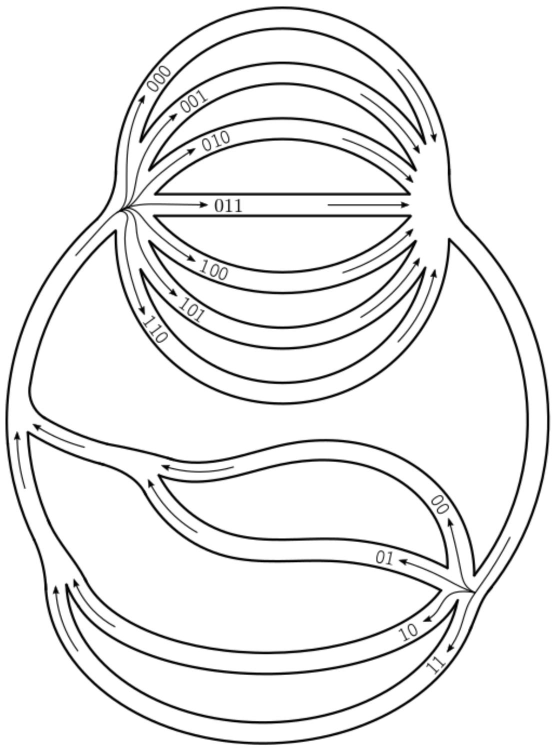


Figure 4.9. A maze, similar to those in Figures 4.2 and 4.3, based on the inputs and outputs of chambers in the circulatory system, shown in Figure 4.8. Bit assignments for each path are also shown, to illustrate the 5 bit policies that describe each of the 28 possible paths through the system.

I simulated synthetic cell executions in this scenario, where the desired target was in the leg, node 13. In this circulatory system model, there are many different policies that will lead to finding the target. It doesn't matter how the cells pass through the pulmonary system (states $2-9$ in Fig. 4.8 or the top part of the maze in Fig. 4.9), as long as they reach the leg in the end. The results of this simulation are shown in Fig. 4.10, and an animation of these simulations can be found at https://sites.google.com/u.northwestern.edu/bayesianparticles.

In the previous example, shown in Figs. 4.2 and 4.6, I simulated a probability of getting lost along one of the paths. This was to acknowledge that in practice, unexpected

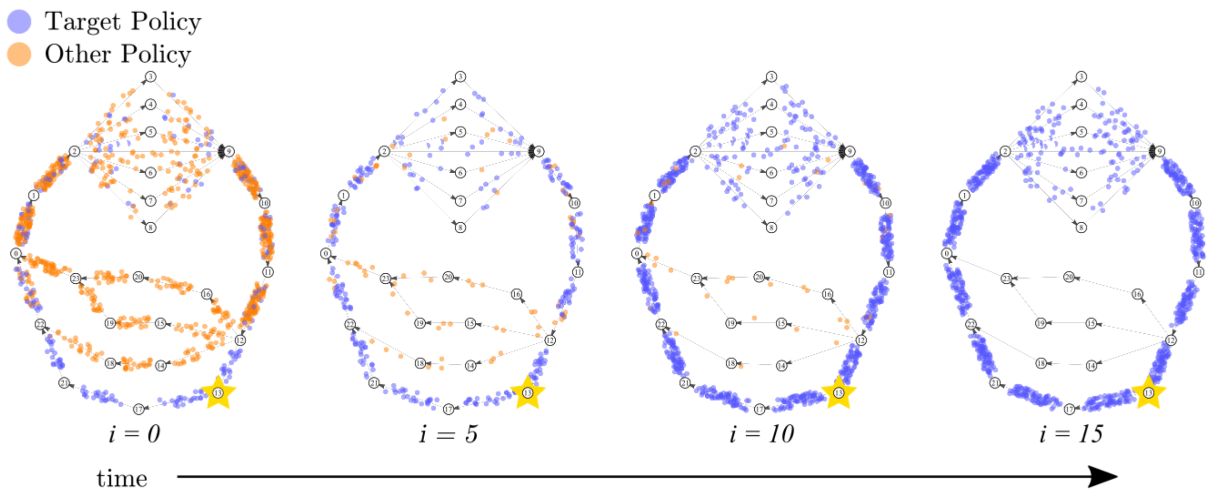


Figure 4.10. Density plots illustrating the distribution of 1000 six-bit synthetic cells executing their policies in the maze from Figure 4.9, with parameters $p_{fp} = 0.1$, $p_{fn} = 0.1$, $\rho = 1.0$ and $r = 0.001$. The target is shown by a yellow star and the cells with policies that pass by it are shown in blue.

events can happen where some synthetic cells will get lost, stuck, destroyed, or otherwise do not contribute to the group's estimate of the target location. We recognize that this can happen no matter where the cell is, so in this example we implemented a small p_{lost} on every execution of every synthetic cell. All of the cells, besides the ones that have been lost to the environment due to p_{lost}, converge to the correct policy by about the twenty sixth loop around the maze.

4.4.3. Moving Targets

In earlier sections, our algorithm was shown to enable all simulated synthetic cells to converge to a policy that passed by a *stationary* target. In this section, I use the same algorithm to show that *moving* targets can be found and communicated, without any additional prior knowledge.

To find a moving target, synthetic cells will have to rely on a small amount of random exploration and decaying memory. I model the random exploration as the probability r of each unsuccessful agent generating a new random policy each time they pass the heart node, as shown in Algorithm 3. Decaying memory enables a cell's success bit to turn off (back to 0) after some amount of time has passed since it last detected a target. We know from [**50, 57**] that this is physically feasible, given variable chemical decay rates and reactions that act similarly to capacitors with a decaying charge.

Figure 4.11 shows simulated results for 1000 synthetic cells navigating through an environment and learning the policies to keep finding the new location of a target which moves from node 13 to node 3, and finally to node 20.

4.5. Experiments

To validate the simulated results, physical robots were constructed. The design of the robots and execution of the experiments were made possible by my exceptional collaborators Karalyn Baird and Joshua Cohen.

Each robot, shown in Fig. 4.12)(a), consists of a Pololu Zumo chassis, which is 95 mm long on each side, and uses infrared reflectance sensors facing down toward the ground to follow lines, micro gear motors to drive a pair of silicone tracks, and four AA batteries for power. The robots use LEDs to display their states and they have two range sensors: one facing straight forward and one at 45° to the right to avoid collisions with other robots. We also custom designed a PCB to connect the Zumo circuitry to a TI CC1310 microcontroller, which enables inter-robot communication via a radio frequency (RF) transceiver.

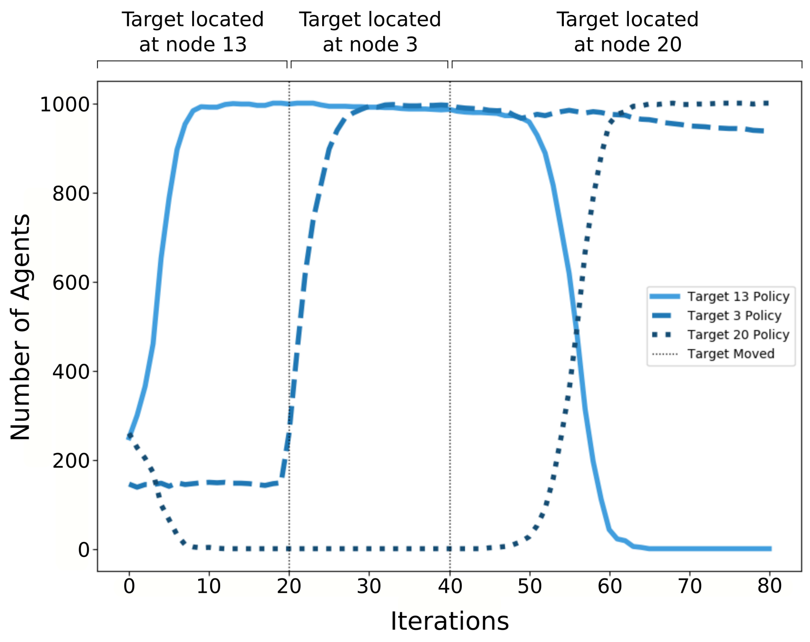
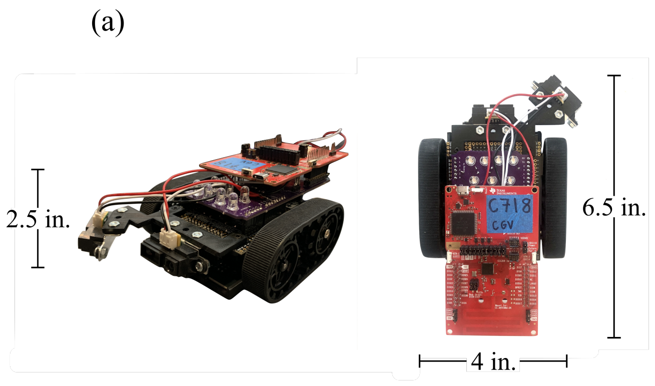


Figure 4.11. Simulated results for 1000 six-bit synthetic cells executing their policies in the maze from Figure 4.9, with parameters $p_{fp} = 0.1$, $p_{fn} = 0.1$, $\rho = 1.0$, and $r = 0.001$ as they search for a target that moves from node 13 to node 3 to node 20. Because nodes 13 and 3 are *in series*, meaning that an agent can have a policy that passes by both of those nodes, the agents converge relatively quickly to the new target policy (about 10 iterations). Conversely, at iteration 40, when the target moves to node 20, all agents are still passing by node 13 which is mutually exclusive with node 20 – agents can't pass by both nodes 13 and 20 because they are *in parallel*, when the target moves to node 20, the agents take much longer to converge to the new target location (about 25 iterations).

The maze, shown in Fig. 4.12, was printed on posters totaling 7 ft by 7 ft. It is composed of black paths associated with policies and white triangles (preceded by short, orthogonal grey lines) indicating intersections. The sensors on the robot that are directed toward the ground record the reflectance of the different colors on the printed posters, and enable the robots to drive along the lines.

When placed on the maze, the robots behave according to the same algorithm as the simulated Bayesian particles, shown in Algorithm 3, and their resulting behavior was very



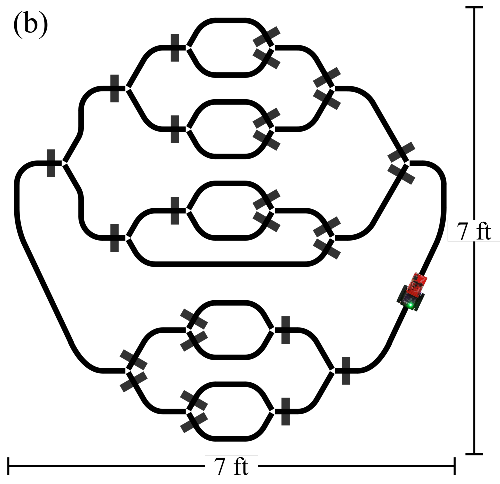


Figure 4.12. Two views of the robots are shown in (a). The robots are 2.5 inches tall by 4 inches wide by 6.5 inches long. A few visible components are the robots' treads, the red CC1310 microcontroller on top, and the two IR sensors – one facing straight forward and one 45 degrees to the right. The circulatory maze used for experiments is shown in (b). This was printed on a 7 foot by 7 foot poster, which was placed on the ground for the robots to drive on. A robot is shown on the right side of the maze for scale.

similar. The agents began with randomized policies on the heart node of the map, and then proceeded to execute their policies, turning left or right at each intersection, while searching for the target. The target was a small mirror placed on either side of the path, which the robots detected using their downward-facing infrared reflectance sensors. After a robot successfully (or falsely) detected the target, it would broadcast its policy using its RF transceiver to any other robot on the same path (between the same two intersections) as itself. The heart node area consists of a long path which every robot, regardless of policy, must pass through. This is where the most useful communications occurred, and where the robots generated a new random policy if they had been unsuccessful on their previous lap. For experiments, I used random policy generation rate $r = 1$, so the agents generated a new policy each unsuccessful lap.

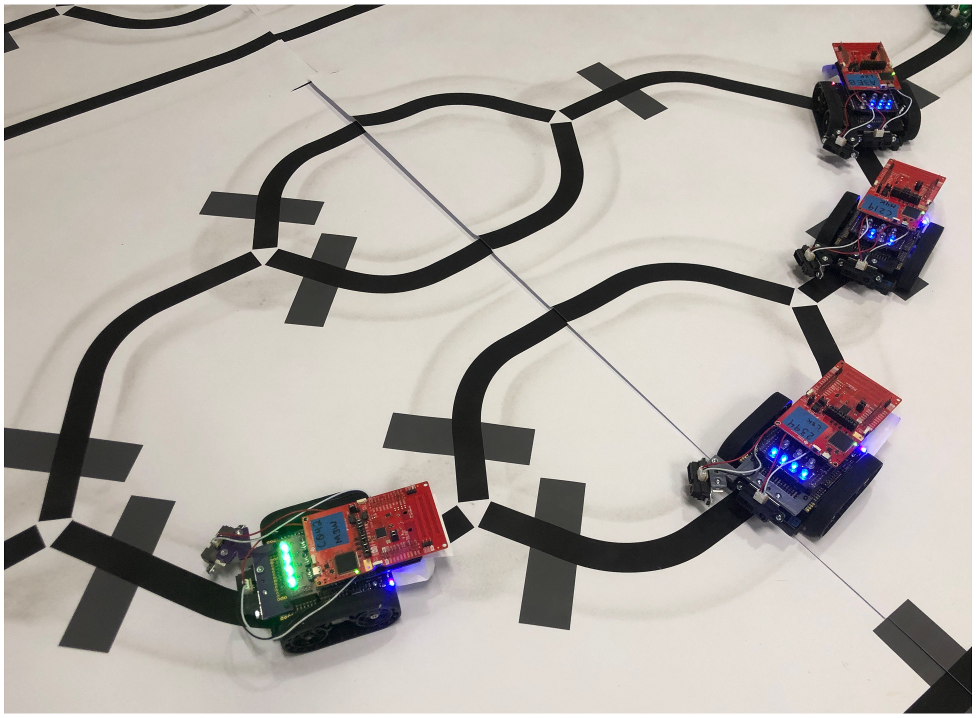


Figure 4.13. The robots use infrared sensors pointing down to observe the printed path below them, and to navigate through the intersections.

Dozens of hours of experimental data was recorded, although only about 8 hours of data are presented here. This is for conciseness, as well as some robot sensor errors. For example, in some experiments a robot would fail to detect an intersection and drive off of the line it had been following. Most of these errors could be attributed to lighting and reflectance issues, and none were due to algorithmic problems. Figure 4.13 shows the robots driving through the maze, and a video of the experiments can be found at https://sites.google.com/u.northwestern.edu/bayesianparticles.

The left side of Fig. 4.14 shows the convergence times and results over 15 experimental trials with 4 agents on the circulatory graph. Ten communication trials are shown in green, overlaid with a dark green plot of the average of the trials. The five trials done without communication are shown in orange, with the average of those five trials overlaid

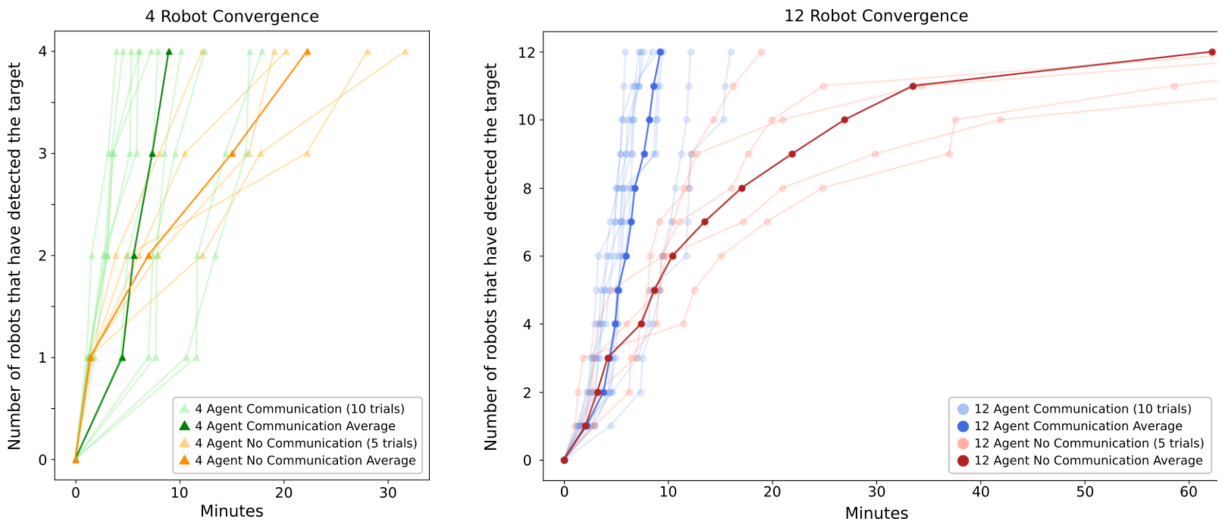


Figure 4.14. Over time, the robots converge to the target policy. On the left, results of experiments with 4 robots are shown. The green lines show the 10 experimental trials performed where the robots used the communication algorithm introduced in this chapter, and the orange lines show the 5 trials where the robots did not communicate. On the right, results of experiments with 12 robots are shown. The blue lines show the 10 trials where the robots used communication, and the red lines show the 5 trials in which they did not. Notably, the averages of both the 4 and 12 robot communication trials (shown in dark green and dark blue, respectively) exhibit extremely similar results.

in dark orange. The right side of Fig. 4.14 shows the experimental results for 15 trials using 12 robots. Communication trials are shown in blue, and trials without communication are shown in red, with their averages similarly overlaid. Notably, the trials with communication converged, on average, in about 8.5 minutes – with both 4 robots and 12 robots. The trials without communication took around 21 minutes on average for 4 robots, and nearly 60 minutes with 12 robots (and many trials went over an hour).

On average, it took 2 minutes and 38 seconds (2.64 minutes) for each robot to complete a lap around the maze. There was a total of 483 minutes of data (slightly over 8 hours). A total of 6 false positives occurred during data collection, and only 1 false negative.

The results of the experimental robots closely mirror those of the simulated agents. These validate our conclusions and demonstrate a physical implementation of the particle filter analogy.

4.6. Robustness

We show that coverage guarantees can arise from the topology of an environment and Bayesian inference, leading to the implication that the *structure* of the circulatory system plays a role in the resulting immune response. A key observation in this work is that the type of environment, which we define as a uniformly oriented (UO) graph, is a requirement for the guarantees, and that animal circulatory systems happen to meet this requirement. The work in this section was done in collaboration with Professor Andrea Richa's group at Arizona State University, including Jamison Weber, Anya Chaturvedi, and Rebecca Martin.

4.6.0.1. Uniformly Oriented Graphs. A uniformly oriented graph is a specific class of directed random graphs, in which there exists one vertex h that is an element of every directed cycle. A directed cycle is one lap around the graph, beginning and ending at the heart node h. The graphs have up to k degree and c' path length. These requirements ensure some probability of interaction between different agents, even in worst-case scenarios like trees and graphs of low conductance.

Conductance is a measure of a graph's connectivity. High conductance corresponds to a well-connected graph on which agents experience high mobility, whereas low conductance indicates that an agent is likely to be trapped in certain areas of the state space.

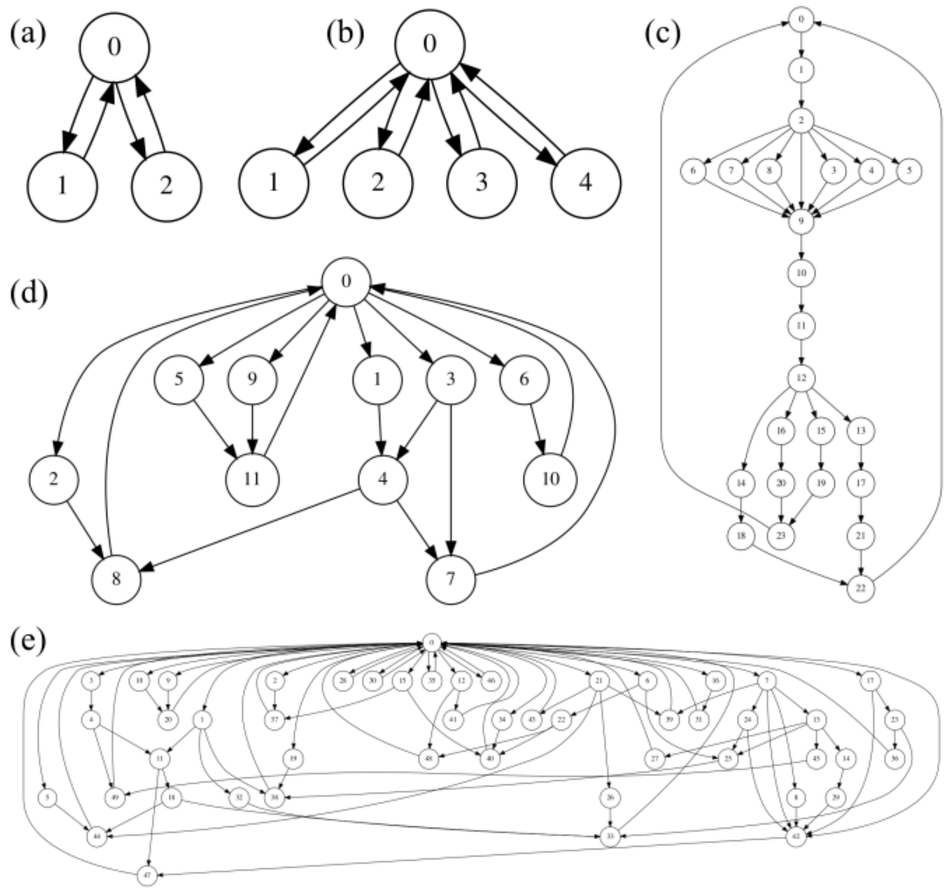


Figure 4.15. Examples of Uniformly Oriented (UO) graphs. (a) is a figure 8 graph, (b) is a four path graph, (c) is the circulatory graph, and (d) and (e) are randomly generated UO graphs with 12 and 50 nodes, respectively. Notice the heart node 0 in each graph, which appears in each cycle. Also note that each node only points to nodes of larger value. Each of these examples satisfies conditions of UO graphs.

Figure 4.15 shows examples of UO graphs, ranging from relatively simple (a) and (b) to randomly generated (d) and (e). Figure 4.15(c) shows a model of the circulatory system, with a different organization than shown earlier in Fig. 4.8.

4.6.0.2. Convergence. For a UO graph with n nodes, path length parameter $c = c'/\log n$ where c' is the longest path length, the probability β of moving forward in the graph (i.e., not self-looping on the same node), and a constant γ' corresponding to the number of particles, we are able to show that all particles will have converged to the target within

$$t_c := c\gamma' \log^6 n + \frac{c}{2\beta^2} \log^2 n \log^2 \left(\frac{c}{4} \log n\right) + c \log^3 n = \mathcal{O}(\log^6 n)$$

time steps with high probability [**78**].

We take this analysis further by drawing conclusions not only about high probability results, but also the expectation (or average) convergence time of agents on a UO graph. We can bound the expectation for the convergence time

$$\mathbb{E}[conv] \leq b \cdot c\gamma' \log^2 n + \frac{c}{2\beta^2} \log^2 n \log^2 \left(\frac{c}{4} \log n\right) + b \cdot c \log n = \mathcal{O}(\log^2 n \log^2 \log n)$$

where $b = 1/\beta$. These poly-log bounds are proven and discussed in detail in [**78**]. The results are also verified in my simulations of high dimensional graphs, discussed next.

4.6.0.3. High Dimensional Trees. To emphasize the effectiveness of the algorithms introduced in this chapter, they were simulated on large trees. Trees are graphs that start from a single root node (analogous to the heart node required for UO graphs) and branch out at some *degree* to some number of levels or *height*. The nodes at the bottom of the tree are called leaves. To turn a tree into a UO graph, we connect edges leading from each of the leaves back to the root node, which is also the heart node, to make the graph cyclical. An example is shown in Fig. 4.16, which has a height of 5 and degree of 2. The grey arrows point from the leaf nodes to the root node. This tree has a total of 63 nodes.

Simulated results are shown in Fig. 4.17. Simulations were run on various trees with different rates of false negatives and false positives. The false rate fr is the probability

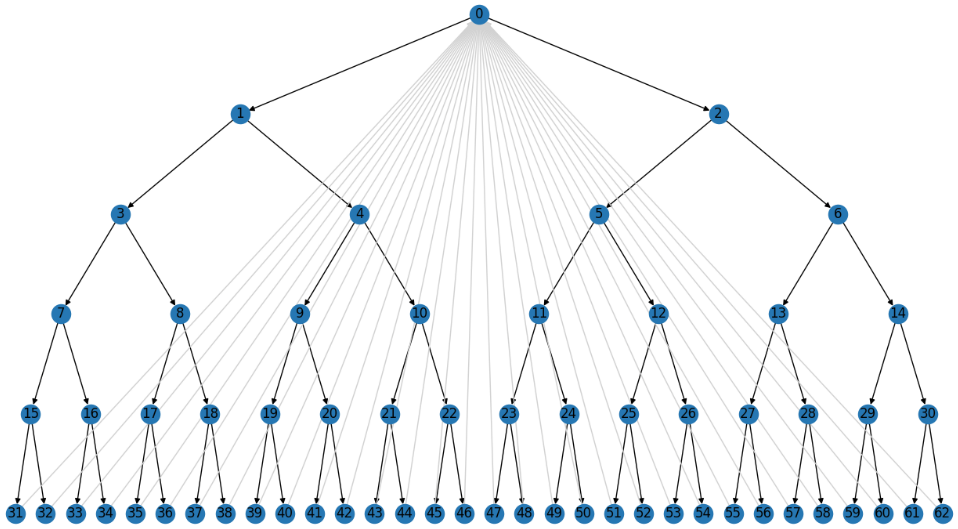


Figure 4.16. A tree of height 5 and degree 2. Each node has two edges leaving it, resulting in two child nodes. The last level of nodes (the leaves) each have one edge (shown in grey) leading back to node 0.

of a false negative or false positive occurring for each agent (depending on whether they have the target policy, either a false positive or a false negative is possible) during each cycle. The red lines on the plots in Fig. 4.17 indicate results without any false positives or negatives. Each line on the plots is an average of 10 trials run with those conditions.

The three trees that were simulated all had a height of 5. The plot on top of Fig. 4.17 shows results of simulations done on a tree with degree 2 (the same tree that was shown in Fig. 4.16), which means it had 63 nodes. The plot on the bottom left of Fig. 4.17 shows results from simulations on a tree with degree 3, and therefore 364 nodes. And the plot on the bottom right are the results of simulations on a tree with degree 4, resulting in 1365 nodes. For each of the simulations, the number of agents interacting on the graph was equivalent to the number of nodes. For the bottom right plot, with degree 4, only three different false rates were tested.

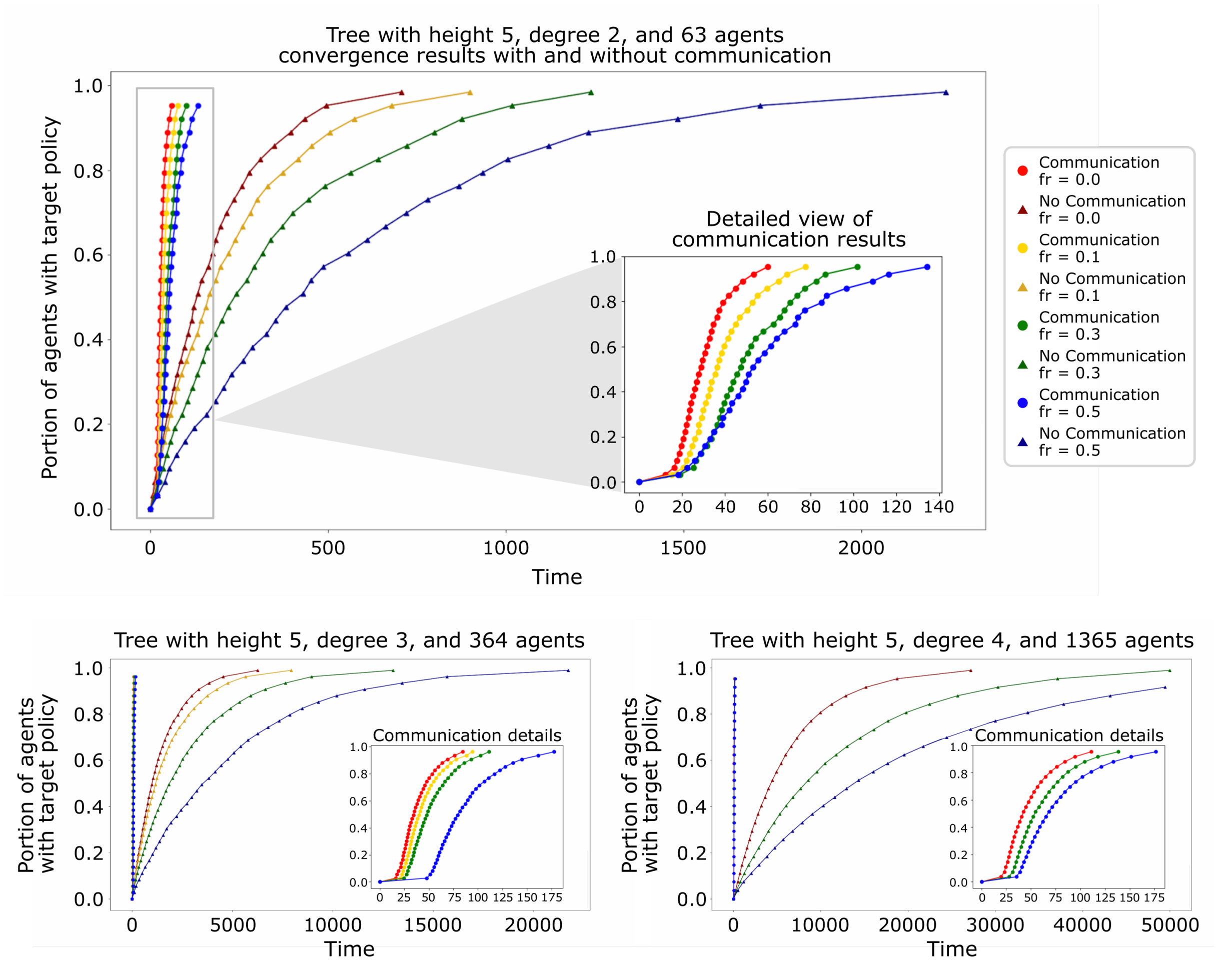


Figure 4.17. Simulations were run on different size trees with varying false rates (fr). The false rates are the probability of detecting a false positive or negative each cycle through the graph. As the false rates increase, the agents take longer to converge to the target policy. The top plot shows the portion of agents that have the target policy with respect to time, with and without communication, for a tree of height 5 and degree 2 (resulting in 63 nodes). The communication trials converged much faster than the trials without communication, so an inset plot shows a zoomed in picture of the communication results. The bottom left plot illustrates the same results for a tree of height 5 and degree 3 (resulting in 364 nodes). The bottom right plot shows results for a tree of height 5 and degree 4 (resulting in 1365 nodes). Each line on the plots is an average of 10 trials run with those conditions.

The results demonstrate the significant benefits of simple agents employing our communication algorithm on UO graphs. As the environments become more complicated and the graphs have higher numbers of nodes, the performance differences between communication and no communication become drastic. The trends in the simulated results in Fig. 4.17 closely resemble the trends in the plots of the experimental data shown in Fig. 4.14, especially the results of the simulations with 12 robots.

4.7. Discussion

This work was inspired by cooperation and self-organization behaviors that naturally occur in microorganisms. This is relevant to understanding collective intelligence in the biological response at the cellular level. It is also critical to the co-design of micro- and nano-robotic systems that exist at scales where computation and actuation are not possible in classical senses. In this work, I demonstrated a scalable framework for theoretical analysis of cyclic environments and the capabilities of minimal agents within them. I proved that the simple behaviors shown in Algorithm 3 enable agents to collectively perform a Bayesian update, and I validated this algorithm in simulation and in a macro-scale robotic system. In simulation, Bayesian particles cover their randomly generated environments and converge to unknown targets within them. In hardware, robotic agents demonstrate coverage and convergence on a model of the topology of the human circulatory system.

The consequence of this work is that environmentally-driven coverage combined with analysis of Bayesian inference implies that extremely simple systems can complete tasks with complex specification. Moreover, this method inherits the robustness and convergence guarantees provided by Bayesian inference which can lead to stable and reliable implementations of microrobots (or other simple agents) providing coverage of and searching for targets in complex environments. This work relates structure at extremely small scales to structure at extremely large scales and how they interact to create a functional organism.

CHAPTER 5

Future Direction: Flexible Tool Design

Robot design does not only consist of selecting sensors and actuators, but also designing geometry. In this chapter, I present a framework for designing a flexible tool. I use the extended example of a bendable wire being manipulated by a two-armed robot, in which the robot bends the tool into a desired shape before using it to attempt a task.

5.1. Introduction

It is most straightforward to analyze robot design in minimal robots, which typically exist at small scales, and is what most of this thesis has been about. But the principles discussed in the previous chapters need not only apply to very simple, microscopic robots. In this chapter, I will present a pipeline for the design of a macroscopic soft tool, like a bendable wire.

This project was inspired by the common use of a wire coat hanger as a multi-purpose tool—not just to hang something, but as a hook, as a frame, to open an older-model locked car—this example enables a single, malleable object to accomplish multitudes of possible tasks. For a robot to shape its own tool to accomplish a task, it must design the geometry of the tool and the strategy for using it simultaneously.

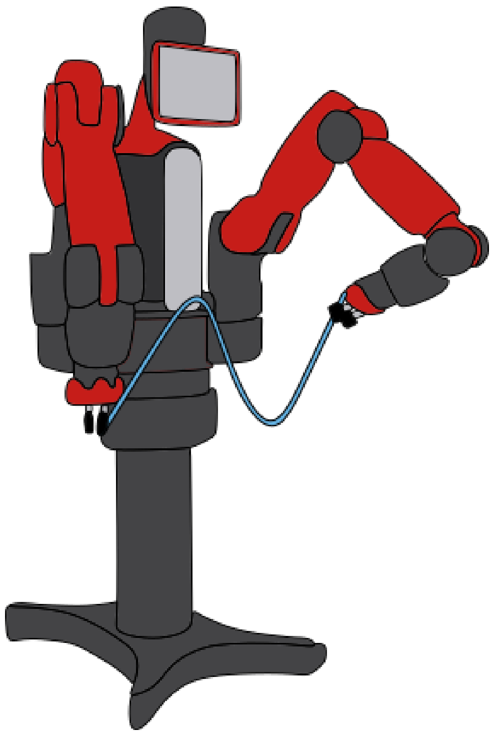


Figure 5.1. An illustration of a two-armed robot manipulating a flexible tool – in this case, into an 'N' shape.

5.2. Related Work

We begin by considering the problem posed in [**13**], which consists of a thin, flexible wire of fixed length that is held at each end by a robotic gripper. This is illustrated in Fig. 5.1. With the endpoints of the rod located anywhere and with any orientation in reachable space (subject to the length of the wire, L), the rod settles into an equilibrium configuration that can be characterized by a geometric optimal control problem.

Assuming that the rod is thin and inextensible, and that it is stiff enough to not be affected by gravity along any unsupported length, we can describe the shape of the rod by a continuous map $q : [0, 1] \to G$, where $G = SE(3)$ [**13**]. The map is required to satisfy

$$\dot{q} = q(u_1 X_1 + u_2 X_2 + u_3 X_3 + X_4)$$

for some $u : [0,1] \to U$, where $U = \mathbb{R}^3$ and X is a basis defined in [**13**]. Here q is a 4×4 transformation matrix, and the function u_1 is the twisting strain and u_2 and u_3 are bending strains along the rod.

The robotic grippers hold the ends of the rod in place at $q(0)$ and $q(1)$ and the rest of the rod rests at its lowest energy equilibrium position, depending on its material properties including the torsional stiffness $c_1 > 0$ and bending stiffnesses $c_2, c_3 > 0$ of the rod. Assuming that the rod is Kirchoff elastic [**9**], its total elastic energy is given by

$$\frac{1}{2}\int_0^1 (c_1 u_1^2 + c_2 u_2^2 + c_3 u_3^2)dt.$$

In a configuration with fixed endpoints, the rod will be motionless only if its configuration locally minimizes this elastic energy. A configuration (q, u) is in a static equilibrium configuration if

$$\begin{aligned} \min_{q,u} \quad & \frac{1}{2}\int_0^1 (c_1 u_1^2 + c_2 u_2^2 + c_3 u_3^2)dt \\ \text{subject to} \quad & \dot{q} = q(u_1 X_1 + u_2 X_2 + u_3 X_3 + X_4) \\ & q(0) = e,\ q(1) = b \end{aligned} \tag{5.1}$$

for some b that is in the set of physically reachable points from e, which we often take to be I, the identity element of $SE(3)$.

Finally, Bretl et al. [**13**] show that the set of equilibrium configurations for the rod is a finite-dimensional smooth manifold that can be parameterized by a six-dimensional coordinate chart. These coordinates describe all possible configurations of the elastic rod and the smooth, quasi-static transitions between them [**13**]. To define them, they formalize a space $\mathcal{A}$ which is $\mathbb{R}^6$ with a two-dimensional plane removed.

$$\mathcal{A} = \{a \in \mathbb{R}^6 : (a_2, a_3, a_5, a_6) \neq (0, 0, 0, 0)\} \tag{5.2}$$

Next they define a system of differential equations in $\mu : [0, L] \rightarrow \mathbb{R}^6$, where the function μ is the vector of internal forces and torques along the rod

$$\begin{aligned} \dot{\mu}_1 &= u_3\mu_2 - u_2\mu_3 & \dot{\mu}_4 &= u_3\mu_5 - u_2\mu_6 \\ \dot{\mu}_2 &= \mu_6 + u_1\mu_3 - u_3\mu_1 & \dot{\mu}_5 &= u_1\mu_6 - u_3\mu_4 \\ \dot{\mu}_3 &= -\mu_5 + u_2\mu_1 - u_1\mu_2 & \dot{\mu}_6 &= u_2\mu_4 - u_1\mu_5 \end{aligned} \tag{5.3}$$

subject to initial conditions $\mu(0) = a$ [**13, 91**]. In Theorem 5 of [**13**], it is shown that for each (q, u) there exists a unique $a \in \mathcal{A}$.

Planning in $\mathcal{A}$ is useful because any point in $\mathcal{A}$ uniquely defines an equilibrium configuration of the rod, and any two stable equilibrium configurations in $\mathcal{A}$ are path connected [**91**]. The contributions in [**91**] detail a manipulation strategy in which a roadmap consisting of $\mathcal{A}$ configurations and the corresponding (q, u) solutions is pre-computed and stored in memory. The roadmap encodes a stable subspace of $\mathcal{A}$, connects the nearby points, and builds paths to rapidly connect start and goal configurations of the rod [**91**].

Bretl et al. [**13**] also extended their work to cases with multiple grippers. The work in [**65**] demonstrated planar manipulation of an elastic rod and manipulation planning when both ends of the rod were fixed. We build off of these conclusions in the following sections.

5.3. Results

5.3.1. Neural Network Model

My collaborator Bowen Feng and I extended the results of Bretl et al. [**13, 65, 91**] by training a neural network to relate the (q, u) rod configuration to the $\mathcal{A}$ representation instead of using a roadmap to do so. I generated $16,000$ data points consisting of random values for $a = (a_1, a_2, a_3, a_4, a_5, a_6)$ between 0 and 1 and solved for the corresponding (q, u) functions using the mathematical framework outlined in [**13**]. We then constructed a simple neural network containing a long short-term memory (LSTM) architecture. The LSTM included a 16-dimensional input (the flattened q transformation matrix), two layers, and a 32-dimensional output, which were then passed through a linear layer to compress the output to the 6-dimensional a.

We trained this network with a batch size of 1000 data points and an annealing learning rate starting at 0.01 for 1000 epochs. This resulted in performance with an error of under 1%, as shown on the left of Fig. 5.2. The error, or loss, was calculated as the mean squared error (MSE) between the actual a values and the values that our model predicted. The model was evaluated on a test set of data that the learning agent had never seen before and increased its accuracy and performance over time to end up with an error rate of less than 1%.

This model enables us to quickly and accurately change the representation of our rod for efficient calculations of stability and nearby equilibrium configurations during manipulation of the rod. Unlike a roadmap, the neural network model can precisely predict the coordinates of configurations it has never seen before. This feature will be

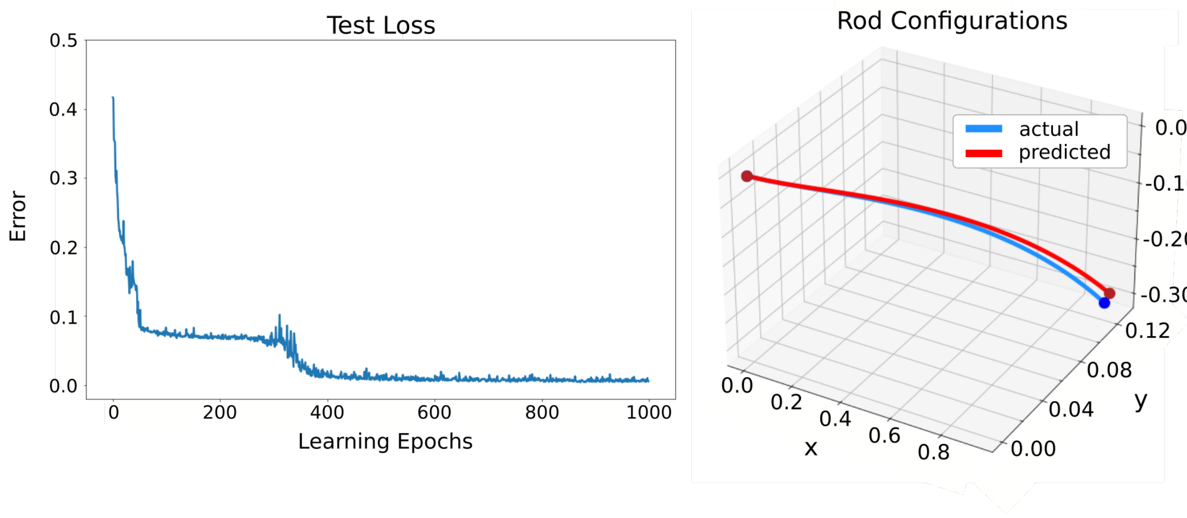


Figure 5.2. Left: The mean squared error between the actual a values and the predicted values on the test dataset over time, as the agent learns. Right: Predicted results of the learned model compared to the actual configuration.

beneficial in future simulations and experiments, when the robot is tinkering with the rod and needs to know how to represent novel configurations.

5.3.2. Grasping and Bending

In this section I introduce a framework for two robot arms grasping, releasing, and regrasping a rod to bend it into a complex shape. The agent attempting this task will be allowed a limited number of actions for affecting the rod and a limited number of sensory measurements for observing the rod. The agent holds onto one end of the rod with one end effector, and its action space consists of selecting where, ℓ, along the length of the rod its other gripper will grasp and then where, p, in the reachable region of $SE(3)$ that gripper (and the section of the rod that it is holding) will move to. Here $\ell : [0, 1] \rightarrow L$, where $L = \mathbb{R}$, is the location, defined as a distance from the beginning of the rod, that the robot will grip and $p = [x, y, z, x_\phi, y_\phi, z_\phi, \omega_\phi]$ represents the new location in $SE(3)$ to which the gripper will move the section of the rod. The first three values of p describe its position and the last four encode its orientation using a quaternion of the form $x_\phi\hat{\mathbf{i}}+y_\phi\hat{\mathbf{j}}+z_\phi\hat{\mathbf{k}}+\omega_\phi$. The robot is able to take noisy observations $(x_{obs}, y_{obs}, z_{obs})$ of the locations of the rod along its length. In experiment, this will be approximated using two cameras at different angles.

With the location ℓ, the position and orientation p, and the formulation introduced by Bretl et al. [**13**], [**91**], we can solve for the curve of the rod segment between two grippers. This final shape depends on the endpoints of the rod segment, and also on the initial configuration of the rod. The rod segment will settle into an equilibrium that is a local minimum of Eq. (5.1), so the final configuration is significantly dependent on the initial configuration and the path that the robot end effectors take to move the endpoints of the rod.

After grasping and moving the rod, the robot has a piece-wise description of the new rod configuration. The robot can re-grasp and re-calculate the expected rod shape by using the new piece-wise description of the rod as its initial configuration and then re-grasp and bend the rod as it wishes. Note that since we made the assumption that the rod is not affected by gravity, any segments of the rod *not* between the grippers will hold their previous shape.

I simulated this system using a custom-built environment in which an initial configuration $q_{initial}$, ℓ, and p can be inputted, and the resulting final configuration is determined. Figure 5.3 shows the initial rod configuration on the left, in this case the identity I, which is a straight line in the x direction. The inputted grasp location was $\ell = \frac{5}{8}$ (note that

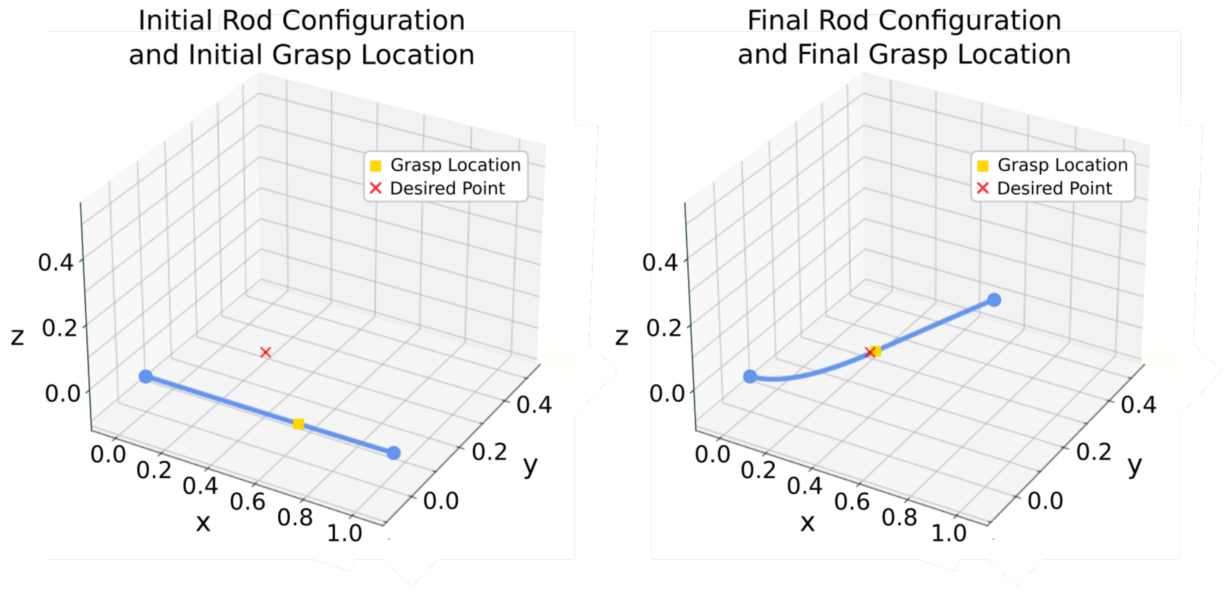


Figure 5.3. Left: Initial rod configuration shown in blue, with the grasp location, $\ell = \frac{5}{8}L$, shown in yellow, and the desired grasp final location and orientation, $p = (x, y, z, x_\phi, y_\phi, z_\phi, \omega_\phi) = (0.3, 0.2, 0, 0, 0, \frac{1}{2}, \frac{\sqrt{3}}{2})$, shown in red. Right: Final rod configuration with the grasped section at its final location.

I use a unit length rod, $L = 1$), the spatial location was $(x, y, z) = (0.3, 0.2, 0)$, and the orientation was a $\frac{\pi}{3}$ radian rotation around the z axis, or $(x_\phi, y_\phi, z_\phi, \omega_\phi) = (0, 0, \frac{1}{2}, \frac{\sqrt{3}}{2})$. On the right of Fig. 5.3 is the final configuration of the rod, after the agent moves the grasp location of the rod to the desired point, and the rest of the rod remains in its initial configuration.

5.4. Discussion

In future work, we plan to use this simulated environment and a reinforcement learning (RL) algorithm to bend the rod into desired shapes. The learning agent will able to select ℓ and p, subject to physical feasibility, and then observe the resulting $(x_{obs}, y_{obs}, z_{obs})$, which

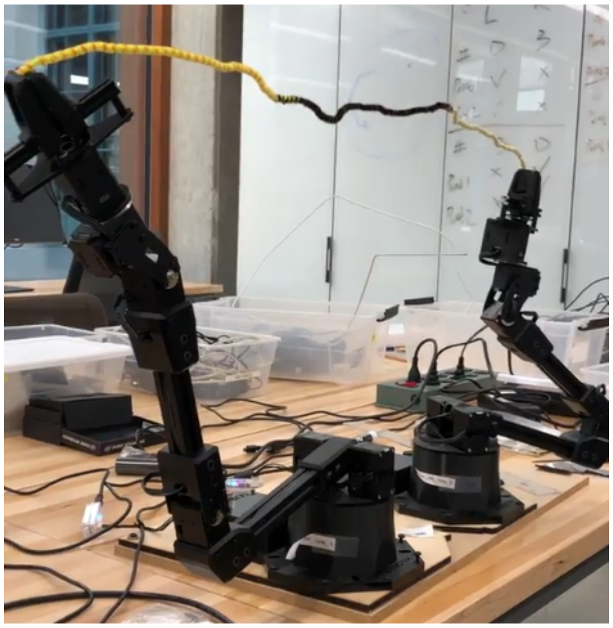


Figure 5.4. Two WidowX 200 robots grip the ends of a pipe cleaner and reconfigure it in physical experiments.

is a subset of the entire configuration (q, u) of the rod. The robot can then bend the rod again, according to the error between the desired shape and the observed configuration.

We have already begun working on an experimental set up with two WidowX 200 robots, each with five degrees of freedom and grippers with soft pads, shown in Fig. 5.4. We use a pipe cleaner as our rod, which is stiff enough to hold its shape despite gravity while being lightweight and bendable enough for the robots to bend and manipulate. Pipe cleaners also have the added benefit of a soft surface with friction to assist in gripping.

Equipping a robot with the ability to bend a malleable tool into any desired shape will allow it to tinker with objects in its environment and attempt tasks with various tools – all made out of one initial piece. This would open up a world of possibilities for automated robot design – especially in real-world experiments.

References

[1] T. Alam, L. Bobadilla, and D. A. Shell. Minimalist robot navigation and coverage using a dynamical system approach. In *IEEE Int. Conf. Rob. Comp.*, 2017.

[2] K. Anand and G. Bianconi. Entropy measures for networks: Toward an information theory of complex topologies. *Physical Review E*, October 2009.

[3] V. N. M. Arelekatti and A. Winter. Design and preliminary field validation of a fully passive prosthetic knee mechanism for users with transfemoral amputation in india. *Journal of Mechanisms and Robotics*, pages 350–356, 2015.

[4] L. Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, pages 1–3, 1966.

[5] D. Banarse, Y. Bachrach, S. Liu, G. Lever, N. Heess, C. Fernando, P. Kohli, and T. Graepel. The body is not a given: Joint agent policy learning and morphology evolution. *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2019.

[6] A. Bayuelo, T. Alam, L. Bobadilla, L. F. Niño, and R. N. Smith. Computing feedback plans from dynamical system composition. In *IEEE International Conference on Automation Science and Engineering (CASE)*, pages 1175–1180, 2019.

[7] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas. Symbolic planning and control of robot motion [grand challenges of robotics. *IEEE Robotics Automation Magazine*, pages 61–70, March 2007.

[8] S. C. Bengea and R. A. DeCarlo. Optimal control of switching systems. *Automatica*, pages 11–27, January 2005.

[9] J. Biggs, W. Holderbaum, and V. Jurdjevic. Singularities of optimal control problems on some six dimensional lie groups. *IEEE Transactions on Automatic Control*, 52(6):1027–1038, 2007.

[10] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, Berlin, Heidelberg, 2006.

[11] R. P. Boisseau, D. Vogel, and A. Dussutour. Habituation in non-neural organisms: evidence from slime moulds. *Proceedings of the Royal Society B: Biological Sciences*, 283, April 2016.

[12] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, January 2013.

[13] T. Bretl and Z. McCarthy. Quasi-static manipulation of a kirchhoff elastic rod based on a geometric analysis of equilibrium configurations. *The International Journal of Robotics Research*, 33(1):48–68, 2014.

[14] R. A. Brooks. New approaches to robotics. *Science*, September 1991.

[15] C. P. Burgess, P. Hayman, M. Williams, and L. Zalavári. Point-particle effective field theory i: classical renormalization and the inverse-square potential. *Journal of High Energy Physics*, 2017(106), 2017.

[16] T. M. Caldwell and T. D. Murphey. Projection-based optimal mode scheduling. *IEEE Conference on Decision and Control*, December 2013.

[17] T. M. Caldwell and T. D. Murphey. Projection-based iterative mode scheduling for switched systems. *Nonlinear Analysis: Hybrid Systems*, pages 59–83, August 2016.

[18] A. Censi. A mathematical theory of co-design. *ArXiv e-print*, 2015.

[19] A. Censi. A class of co-design problems with cyclic constraints and their solution. *IEEE Robotics and Automation Letters*, January 2016.

[20] A. Censi. Uncertainty in monotone co-design problems. *IEEE Robotics and Automation Letters*, February 2017.

[21] R. Chambers, H. B. Fell, and W. B. Hardy. Micro-operations on cells in tissue cultures. *Proceedings of the Royal Society of London*, 109(763):380–403, 1931.

[22] D. Chaplin. Overview of the immune response. *The Journal of Allergy and Clinical Immunology*, 125:S3–23, February 2010.

[23] S. Collins, A. Ruina, R. Tedrake, and M. Wisse. Efficient bipedal robots based on passive-dynamic walkers. *Science*, pages 1082–1085, 2005.

[24] M. J. Conlon, D. Russell, and T. Mussivand. Development of a mathematical model of the human circulatory system. *Annals of Biomedical Engineering*, 34:1400–1413, October 2006.

[25] B. Esteban-Fernández de Ávila, P. Angsantikul, D. E. Ramírez-Herrera, F. Soto, H. Teymourian, D. Dehaini, Y. Chen, L. Zhang, and J. Wang. Hybrid biomembrane functionalized nanorobots for concurrent removal of pathogenic bacteria and toxins. *Science Robotics*, 2018.

[26] M. A. Montes de Oca, E. Ferrante, A. Scheidler, C. Pinciroli, M. Birattari, and M. Dorigo. Majority-rule opinion dynamics with differential latency: A mechanism for self-organized collective decision-making. *Swarm Intelligence*, 5:305–327, 2011.

[27] M. Dehmer and A. Mowshowitz. A history of graph entropy measures. *Journal of Information Science*, pages 57–78, January 2011.

[28] M. H. Dickinson, C. T. Farley, R. J. Full, M. A. R. Koehl, R. Kram, and S. Lehman. How animals move: An integrative view. *Science*, pages 100–106, April 2000.

[29] J. Ding, V. R. Challa, M. G. Prasad, and F. T. Fisher. *Vibration Energy Harvesting and Its Application for Nano- and Microrobotics*. Springer, New York, NY, 2013.

[30] T. Ding, V. K. Valev, A. R. Salmon, C. J. Forman, S. K. Smoukov, O. A. Scherman, D. Frenkel, and J. J. Baumberg. Light-induced actuating nanotransducers. *Proceedings of the National Academy of Sciences*, 113(20):5503–5507, 2016.

[31] B. R. Donald, J. Jennings, and D. Rus. Information invariants for distributed manipulation. *Int. J. Rob. Res.*, 16(5):673–702, 1997.

[32] S. M. Douglas, I. Bachelet, and G. M. Church. A logic-gated nanorobot for targeted transport of molecular payloads. *Science*, 335(6070):831–834, 2012.

[33] M. Egerstedt, Y. Wardi, , and H. Axelsson. Optimal control of switching times in hybrid systems. *International Conference on Methods and Models in Automation and Robotics*, 2003.

[34] M. Egerstedt, Y. Wardi, and H. Axelsson. Transition-time optimization for switched-mode dynamical systems. *IEEE Transactions on Automatic Control*, pages 110–115, January 2006.

[35] M. Erdmann. Understanding action and sensing by designing action-based sensors. *The International Journal of Robotics Research*, pages 483–509, 1995.

[36] M. Guix, C. C. Mayorga-Martinez, and A. Merkoçi. Nano/micromotors in (bio)chemical science applications. *Chemical Reviews*, 114(12):6285–6322, 2014.

[37] H. H. Hardy, R. E. Collins, and R. E. Calvert. A digital computer model of the human circulatory system. *Medical and Biological Engineering and Computing*, 20(5):550–564, Sep. 1982.

[38] Y. He, J. Wu, and Y. Zhao. Designing catalytic nanomotors by dynamic shadowing growth. *Nano Letters*, 7(5):1369–1375, 2007.

[39] D. Howard, A. E. Eiben, D. F. Kennedy, J. B. Mouret, P. Valencia, and D. Winkler. Evolving embodied intelligence from materials to machines. *Nature Machine Intelligence*, 2019.

[40] J. R. Howse, R. A. L. Jones, A. J. Ryan, T. Gough, R. Vafabakhsh, and R. Golestanian. Self-motile colloidal particles: From directed propulsion to random walk. *Physical Review Letters*, 99:048102, July 2007.

[41] U. Huzaifa, C. Bernier, Z. Calhoun, G. Heddy, C. Kohout, B. Libowitz, A. Moenning, J. Ye, C. Maguire, and A. LaViers. Embodied movement strategies for development of a core-located actuation walker. *016 6th IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pages 176–181, June 2016.

[42] M. Ilton, M. S. Bhamla, X. Ma, S. M. Cox, L. L. Fitchett, Y. Kim, J. Koh, D. Krishnamurthy, C. Kuo, F. Z. Temel, A. J. Crosby, M. Prakash, G. P. Sutton, R. J. Wood, E. Azizi, S. Bergbreiter, and S. N. Patek. The principles of cascading power limits in small, fast biological and engineered systems. *Science*, 360(6387), 2018.

[43] H. Jiang, N. Yoshinaga, and M. Sano. Active motion of a janus particle by self-thermophoresis in a defocused laser beam. *Physical Review Letters*, 105, December 2010.

[44] C. Kaewsaneha, P. Tangboriboonrat, D. Polpanich, and A. Elaissari. Multifunctional fluorescent-magnetic polymeric colloidal particles: Preparations and bioanalytical applications. *ACS Applied Materials & Interfaces*, 7(42):23373–23386, 2015.

[45] N. Kamaly, B. Yameen, J. Wu, and O. C. Farokhzad. Degradable controlled-release polymers and polymeric nanoparticles: Mechanisms of controlling drug release. *Chemical Reviews*, 116(4):2602–2663, 2016.

[46] V. Kantsler, J. Dunkel, M. Polin, and R. E. Goldstein. Ciliary contact interactions dominate surface scattering of swimming eukaryotes. *Proceedings of the National Academy of Sciences*, 110(4):1187–1192, 2013.

[47] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, pages 846–894, 2011.

[48] J. Kim and D. A. Shell. A new model for self-organized robotic clustering: Understanding boundary induced densities and cluster compactness. In *IEEE Int. Conf. Rob. Auto. (ICRA)*, pages 5858–5863, May 2015.

[49] Y.-T. Kim, J. S. Lee, C.-H. Youn, J.-S. Choi, and E. B. Shim. An integrative model of the cardiovascular system coupling heart cellular mechanics with arterial network hemodynamics. *Journal of Korean Medical Science*, 28(8):1161–1168, Aug. 2013.

[50] V. B. Koman, P. Liu, D. Kozawa, A. T. Liu, A. L. Cottrill, Y. Son, J. A. Lebron, and M. S. Strano. Colloidal nanoelectronic state machines based on 2d materials for aerosolizable electronics. *Nature Nanotechnology*, 13:819–827, Sep. 2018.

[51] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, USA, 2006.

[52] S. Lee, A. J. Cortese, P. Trexel, E. R. Agger, P. L. McEuen, and A. C. Molnar. A 330μm x 90μm opto-electronically integrated wireless system-on-chip for recording of neural activities. In *2018 IEEE International Solid-State Circuits Conference, ISSCC*, pages 292–294, 2018.

[53] S. C. Lenaghan, S. Nwandu-Vincent, B. E. Reese, and M. Zhang. Unlocking the secrets of multi-flagellated propulsion: drawing insights from tritrichomonas foetus. *Journal of the Royal Society, Interface*, 11, April 2014.

[54] J. Li, B. Esteban-Fernández de Ávila, W. Gao, L. Zhang, and J. Wang. Micro/nanorobots for biomedicine: Delivery, surgery, sensing, and detoxification. *Science Robotics*, 2(4), 2017.

[55] S. Li, Q. Jiang, S. Liu, Y. Zhang, Y. Tian, C. Song, J. Wang, Y. Zou, G. J. Anderson, J. Han, Y. Chang, Y .Liu, C. Zhang, L. Chen, G. Zhou, G. Nie, H. Yan, B. Ding, and Y. Zhao. A dna nanorobot functions as a cancer therapeutic in response to a molecular trigger in vivo. *Nature Biotechnology*, pages 258–264, 2018.

[56] A. T. Liu, M. Hempel, J. F. Yang, A. Pervan, V. B. Koman, G. Zhang, D. Kozawa, T. D. Murphey, T. Palacios, and M. S. Strano. Colloidal robots: Autonomous particles with on-board computation. *Nature Materials?*, 2021.

[57] A. T. Liu, J. F. Yang, L. N. LeMar, G. Zhang, A. Pervan, T. D. Murphey, and M. Strano. Autoperforation of two-dimensional materials to generate colloidal state machines capable of locomotion. *Faraday Discussions*, 2020.

[58] P. Liu, A. L. Cottrill, D. Kozawa, V. B. Koman, D. Parviz, A. T. Liu, J. F. Yang, T. Q. Tran, M. H. Wong, S. Wang, and M. S. Strano. Emerging trends in 2d nanotechnology that are redefining our understanding of "nanocomposites". *Nature Today*, 2018.

[59] P. Liu, A. T. Liu, D. Kozawa, J. Dong, J. F. Yang, V. B. Koman, M. Saccone, S. Wang, Y. Son, M. H. Wong, and M. S. Strano. Autoperforation of 2d materials for generating two-terminal memristive janus particles. *Nature Materials*, pages 1005–1012, 2018.

[60] A. Mavrommati and T. D. Murphey. Automatic synthesis of control alphabet policies. *EEE International Conference on Automation Science and Engineering (CASE)*, August 2016.

[61] M. Mirzaee, O. Ghasemalizadeh, and B. Firoozabadi. Modeling vessels in human cardiovascular system and calculating pressure wastes using lumped method. *The Internet Journal of Bioengineering*, 3, 2008.

[62] J. L. Moran. *Robotic Systems and Autonomous Platforms*. Woodhead Publishing, 2019.

[63] J. L. Moran and J. D. Posner. Locomotion of electrocatalytic nanomotors due to reaction induced charge auto-electrophoresis. *Physical Review E*, 81, June 2010.

[64] J. L. Moran, P. M. Wheat, and J. D. Posner. Phoretic self-propulsion. *Annual Review of Fluid Mechanics*, 49:511–540, 2017.

[65] M. Mukadam, A. Borum, and T. Bretl. Quasi-static manipulation of a planar elastic rod using multiple robotic grippers. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, September 2014.

[66] T. Newman. How the immune system works. *Medical News Today*, Jan. 2018.

[67] A. Q. Nilles, Y. Ren, I. Becerra, and S. M. LaValle. A visibility-based approach to computing nondeterministic bouncing strategies. *Int. Work. Alg. Found. Rob.*, Dec. 2018.

[68] J. M. O'Kane and S. M. LaValle. Comparing the power of robots. *Int. J. Rob. Res.*, 27(1):5–23, 2008.

[69] J. F. M. Oudenhoven, L. Baggetto, and P. H. L. Notten. All-solid-state lithium-ion microbatteries: A review of various three-dimensional concepts. *Adv. Energy Mat.*, 1(1):10–33, 2011.

[70] J. M. O'Kane and S. M. LaValle. Comparing the power of robots. *The International Journal of Robotics Research*, pages 5–23, January 2008.

[71] J. M. O'Kane and D. A. Shell. Concise planning and filtering: Hardness and algorithms. *IEEE Transactions on Automation Science and Engineering*, pages 1666–1681, October 2017.

[72] W. F. Paxton, K. C. Kistler, C. C. Olmeda, A. Sen, S. K. St. Angelo, Y. Cao, T. E. Mallouk, P.E. Lammert, and V. H. Crespi. Catalytic nanomotors: autonomous movement of striped nanorods. *Journal of the American Chemical Society*, 126:13424–13431, 2004.

[73] A. Pervan and T. D. Murphey. Low complexity control policy synthesis for embodied computation in synthetic cells. *International Workshop on the Algorithmic Foundations of Robotics*, December 2018.

[74] A. Pervan and T. D. Murphey. 9 - algorithmic materials: Embedding computation within material properties for autonomy. In S. M. Walsh and M. S. Strano, editors, *Robotic Systems and Autonomous Platforms*, Woodhead Publishing in Materials, pages 197 – 221. Woodhead Publishing, 2019.

[75] A. Pervan and T. D. Murphey. Bayesian particles on cyclic graphs. *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[76] A. Pervan and T. D. Murphey. Algorithmic design for embodied intelligence in synthetic cells. *Transactions on Automation Science and Engineering (T-ASE)*, 2021.

[77] A. Pervan, A. Q. Nilles, T. A. Berrueta, T. D. Murphey, and S. M. LaValle. Information requirements of collision-based micromanipulation. *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2020.

[78] A. Pervan, J. Weber, K. Baird, J. Cohen, A. Chaturvedi, R. Martin, A. Richa, and T. D. Murphey. Bayesian particles in uniformly oriented environments. *In Preparation*, 2021.

[79] R. Pfeifer and J. C. Bongard. *How the Body Shapes the Way We Think*. MIT Press, 2007.

[80] R. Pfeifer, M. Lungarella, and F. Iida. Self-organization, embodiment, and biologically inspired robotics. *Science*, pages 1088–1093, November 2007.

[81] G. Pola, A. Girard, and P. Tabuada. Approximately bisimilar symbolic models for nonlinear control systems. *Automatica*, pages 2508–2516, 2008.

[82] B. Price and C. Boutilier. A bayesian approach to imitation in reinforcement learning. *International Joint Conferences on Artificial Intelligence*, 2003.

[83] D. Ramachandran and E. Amir. Bayesian inverse reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 2586–2591, 2007.

[84] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, pages 206–230, 1987.

[85] C. Reid, T. Latty, A. Dussutour, and M. Beekman. Slime mold uses an externalized spatial "memory" to navigate in complex environments. *Proceedings of the National Academy of Sciences of the United States of America*, Oct. 2012.

[86] A. Reina, R. Miletitch, M. Dorigo, and V. Trianni. A quantitative micro-macro link for collective decision: The shortest path discovery/selection example. *Swarm Intelligence*, 2015.

[87] F. Z. Saberifar, J. M. O'Kane, and D. A. Shell. The hardness of minimizing design cost subject to planning problems. *International Workshop on the Algorithmic Foundations of Robotics*, Dec. 2018.

[88] W. Savoie, T. A. Berrueta, Z. Jackson, A. Pervan, R. Warkentin, S. Li, T. D. Murphey, K. Wiesenfeld, and D. I. Goldman. A robot made of robots: Emergent transport and control of a smarticle ensemble. *Science Robotics*, 4(34), 2019.

[89] D. Seo, R. M. Neely, K. Shen, U. Singhal, E. Alon, J. M. Rabaey, J. M. Carmena, and M. M. Maharbiz. Wireless Recording in the Peripheral Nervous System with Ultrasonic Neural Dust. *Neuron*, 91(3):529–539, 2016.

[90] M. Shahid Shaikh and P. E. Caines. *On the optimal control of hybrid systems: Optimization of trajectories, switching times, and location schedules.* Springer Berlin Heidelberg, 2003.

[91] A. Sintov, S. Macenski, A. Borum, and T. Bretl. Motion planning for dual-arm manipulation of elastic rods. *IEEE Robotics and Automation Letters*, 5(4):6065–6072, 2020.

[92] M. Sitti. Voyage of the microrobots. *Nature*, 458:1121–1122, April 2009.

[93] M. Sitti, H. Ceylan, W. Hu, J. Giltinan, M. Turan, S. Yim, and E. Diller. Biomedical applications of untethered mobile milli/microrobots. *Proceedings of IEEE*, 103(2):205–224, February 2015.

[94] F. Soto and R. Chrostowski. Frontiers of medical micro/nanorobotics: in vivo applications and commercialization perspectives toward clinical uses. *Front. in Bioeng. and Biotech.*, 6:170–170, Nov. 2018.

[95] J. C. Spagna, D. I. Goldman, P. C. Lin, D. E. Koditschek, and R. J. Full. Distributed mechanical feedback in arthropods and robots simplifies control of rapid running on challenging terrain. *Bioinspiration & Biomimetics*, January 2007.

[96] S. E. Spagnolie, C. Wahl, J. Lukasik, and J. Thiffeault. Microorganism billiards. *Phys. D*, 341:33–44, 2016.

[97] R. Tedrake, T. W. Zhang, M. Fong, and H. S. Seung. Actuating a simple 3d passive dynamic walker. *IEEE International Conference on Robotics and Automation*, April 2004.

[98] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Sec. 4.3)*. The MIT Press, 2005.

[99] G. Valentini, E. Ferrante, H. Hamann, and M. Dorigo. Collective decision with 100 kilobots: Speed versus accuracy in binary discrimination problems. *Autonomous Agents and Multi-Agent Systems*, 2016.

[100] E. Ferrante Valentini G. and M. Dorigo. The best-of-n problem in robot swarms: Formalization, state of the art, and novel perspectives. *Frontiers in Robotics and AI*, 2016.

[101] J. Wainer, D. J. Feil-Seifer, D. A. Shell, and M. J. Mataric. Embodiment and human-robot interaction: A task-based perspective. *IEEE International Symposium on Robot and Human Interactive Communication*, August 2007.

[102] R. J. Wood. The first takeoff of a biologically inspired at-scale robotic insect. *IEEE Transactions on Robotics*, 2008.

[103] X. Wu, I. Lee, Q. Dong, K. Yang, D. Kim, J. Wang, Y. Peng, Y. Zhang, M. Saliganc, M. Yasuda, K. Kumeno, F. Ohno, S. Miyoshi, M. Kawaminami, D. Sylvester, and D. Blaauw. A 0.04mm316nw wireless and batteryless sensor system with integrated cortex-m0+ processor and optical communication for cellular temperature measurement. In *2018 IEEE Symposium on VLSI Circuits*, June 2018.

[104] T. Xu, J. Yu, X. Yan, H. Choi, and L. Zhang. Magnetic actuation based motion control for microrobots: An overview. *Micromachines*, 6(9):1346–1364, Sept. 2015.

[105] T. Xu, J. Zhang, M. Salehizadeh, O. Onaizah, and E. Diller. Millimeter-scale flexible robots with programmable three-dimensional magnetization and motions. *Science Robotics*, 4(29), 2019.

[106] X. Xu and P. J. Antsaklis. Optimal control of switched systems via non-linear optimization based on direct differentiations of value functions. *International Journal of Control*, pages 1406–1426, 2002.

[107] G. Z. Yang, J. Bellingham, P. E. Dupont, P. Fischer, L. Floridi, R. Full, N. Jacobstein, V. Kumar, M. McNutt, R. Merrifield, B. J. Nelson, B. Scassellati, M. Taddeo, R. Taylor, M. Veloso, Z. L. Wang, and R. Wood. The grand challenges of science robotics. *Science Robotics*, 3(14), 2018.

[108] J. Yao, H. Yan, S. Das, J. F. Klemic, J. C. Ellenbogen, and C. M. Lieber. Nanowire nanocomputer as a finite-state machine. *Proceedings of the National Academy of Sciences*, 111(7):2431–2435, 2014.

[109] H. P. Zhang, Avraham Be'er, E.-L. Florin, and Harry L. Swinney. Collective motion and density fluctuations in bacterial colonies. *Proceedings of the National Academy of Sciences of the United States of America*, 107(31):13626–30, 2010.

[110] Li Zhang, Jake J. Abbott, Lixin Dong, Bradley E. Kratochvil, Dominik Bell, and Bradley J. Nelson. Artificial bacterial flagella: Fabrication and magnetic control. *Applied Physics Letters*, 94(6):064107, 2009.

APPENDIX A

From Primitives to Policies

Appendix pertaining to subroutines and robot policies from Primitives and Logic for Robot Comparisons chapter.

Subroutines

We define how the subroutines introduced in Section 2.5 can be achieved. Subroutines for wall following (in a random direction) for u_L steps, observing the object y_R, y_B, y_Y, and orienting in the direction of the blue side of the object are shown below, in Algorithm 5.

For wall following, the robot continuously rotates a small angle (using the P_A primitive) until it detects that there is nothing in front of it (when the range detecting primitive P_R reads ∞), and is therefore facing a direction parallel to the wall. Then the robot uses primitive P_L to move forward u_L steps.

For observing the object, the robot continuously rotates until it has either detected the blue side of the object, detected the yellow side of the object, or completed a full rotation. If it has detected a color, it records the distance to the object and the color detected (if a color was detected). If the robot completes a full rotation without detecting either color, it returns a range of ∞, to encode that no color was found. It is possible to call a subset of measurements from this subroutine, for example in Algorithm 7 the *Middle* substrategy only queries the observe_object() subroutine for y_B and y_Y.

Algorithm 5 Subroutines

```
**wall_follow** (input $u_L$)
while $P_R(y_R \neq \infty)$                          (while the robot is not parallel to the wall)
    $P_A(u_A = \phi)$                                  (rotate a small angle $\phi$)
$P_L(u_L)$                                             (step forward a distance $u_L$)

observe_object (output $y_R$, $y_B$, $y_Y$)
$inc = 0$
while $P_B(y_B = 0)$ and $P_Y(y_Y = 0)$ and $inc \leq 360°$    (while neither yellow nor blue
                                                       are detected, and the robot has
                                                       not completed a full rotation)
    $P_A(u_A = \phi)$                                  (rotate a small angle $\phi$)
    $inc{+}{+}$
if $y_B = 1$ or $y_Y = 1$                              (if blue or yellow are detected)
    $y_R = P_R$                                        (measure the distance to the color)
else
    $y_R = \infty$                                     (otherwise, return $\infty$ to encode 'no color')

aim_toward_blue ()
while $P_B(y_B = 0)$                                   (while blue is not detected)
    $P_A(u_A = \phi)$                                  (rotate a small angle $\phi$)
```

For aiming toward blue, the robot will rotate in place (using P_A) until it detects the color blue (using P_B).

Robot 0 Policy

Robot 0 uses the primitive P_O, and requires knowledge of the width of the channel, ℓ, and the length of the object, $2s$. First, in its initial state, it observes its own position (x_r, y_r) and the position of the object (x_o, y_o). If the robot, x_r, is to the left of object's left edge, $(x_o - s)$ (where x_o is the center of mass of the object and s is half the length of the object), then the robot should execute substrategy *Left*.

If robot is to the right of the object's edge, it will go straight up, $u_{O_y} = \ell/\Delta t_k$, to either the top wall of the channel, or the underside of the object (if the robot happened to start below the object), and then move left, $u_{O_x} = ((x_o - s) - x_r)/\Delta t_k$, until it has passed the object. Then the robot transitions to the *Left* substrategy.

In the *Left* substrategy, the robot translates to the left side of the object, and then pushes it a set distance ϵ to the right. It increases the *count* variable with each subsequent push.

Algorithm 6 Robot 0 Policy: π_0

```
Requires
Primitive: $P_O$
Parameters: $s$ is half of the length of the object, $\ell$ is the width of the channel

Initial
$count = 0$
$x_r, y_r, x_o, y_o = P_O()$                          (read the positions of the robot and the object)
if $x_r < (x_o - s)$                        (if the robot is to the left of the object's left edge)
    Switch to Left
else
    $P_O(u_{O_x} = 0, u_{O_y} = \ell/\Delta t_k)$                      (move up, until the object or wall)
    $P_O(u_{O_x} = ((x_o - s) - x_r)/\Delta t_k, u_{O_y} = 0)$            (move to the left of the object)
    Switch to Left


Left
$P_O(u_{O_x} = ((x_o - s) - x_r)/\Delta t_k, u_{O_y} = (y_o - y_r)/\Delta t_k)$        (go to the center of the
                                                                          left side of the object)
$P_O(u_{O_x} = \epsilon/\Delta t_k, u_{O_y} = 0)$                (push the object a distance of $\epsilon$ to the right)
$count++$
```

Robot 1 Policy

Robot 1 uses the primitives P_A, P_T, P_B, P_R, P_L, and P_Y, and requires knowledge of the set of distances w from the object that allow the robot to fall into the limit cycle. First, in its initial state, R_1 measures the distance between it and the object, and the color directly in front of it (if any). It uses this knowledge to switch to a substrategy: either *Limit Cycle*, *Left*, *Right*, or *Middle*.

In the *Limit Cycle* substrategy, as shown in Fig. 2.3, the robot translates forward, rotates, and repeats – continuously executing the limit cycle and counting how many times it bumps the object forward.

In the *Left* substrategy, the robot orients itself so that it is facing the blue side of the object, then switches to *Limit Cycle*, where it will translate toward the object, rotate, and enter the limit cycle.

In the *Right* substrategy, R_1 will measure the distance to the object y_{R_old}, move along the wall a small distance δ, then measure the distance to the object again y_R, and compare the two distances. If the distance to the object increased, then the robot is moving toward the right and must turn around, using primitive P_A. Otherwise, the robot is moving toward the left, and will continue in that direction until it detects the blue side of the object and switches to the *Left* substrategy.

In the *Middle* substrategy, the robot is directly beneath or above the object, and cannot tell which direction is which. It chooses a random direction to follow the wall, until it detects either the blue or the yellow side of the object. If it detects blue it switches to the *Left* substrategy, and if it detects yellow it switches to the *Right* substrategy.

Algorithm 7 Robot 1 Policy: π_1

```
Requires
Primitives: $P_A$, $P_T$, $P_B$, $P_R$, $P_L$, $P_Y$
Parameters: $w$ is the range of distances that are attracted by the limit cycle

Initial
$count = 0$
$y_R, y_B, y_Y =$ observe_object()                (read object distance and color)
if $y_B = 1$ and $y_R \in w$                       (if blue was detected at a distance in $w$)
   Switch to Limit Cycle
else if $y_B = 1$ and $y_R \notin w$               (if blue was detected at a distance not in $w$)
   Switch to Left
else if $y_Y = 1$                                  (if yellow was detected)
   Switch to Right
else
   Switch to Middle


Limit Cycle
$P_T$                                              (translate forward to an obstacle)
$P_A(u_A = \theta)$                                (rotate $\theta$)
$P_T$                                              (translate forward to an obstacle)
$P_A(u_A = \theta)$                                (rotate $\theta$)
$P_T$                                              (translate forward to an obstacle)
$P_A(u_A = \theta)$                                (rotate $\theta$)
$count++$


Left (bounce off of blue side of object)
aim_toward_blue()
Switch to Limit Cycle
```

Algorithm continued on next page.

Robot 1 Policy: π_1 cont.

```
Right (wall follow toward object)
$y_{R\_old}$ = observe_object()                    (read object distance)
wall_follow($u_L = \delta$)                         (step forward a small distance $\delta$)
$y_R$ = observe_object()                           (read object distance)
if $y_R > y_{R\_old}$                              (if the distance to the object increased)
  $P_A(u_A = 180°)$                                (turn around)
while $y_B = 0$                                    (while blue has not been detected)
  wall_follow($u_L = \delta$)                       (step forward a small distance $\delta$)
  $y_B$ = observe_object()                         (scan for object and record color)
Switch to Left


Middle (wall follow until blue or yellow detected)
while $y_B = 0$ and $y_Y = 0$                      (while neither blue nor
                                                    yellow have been detected)
  wall_follow($u_L = \delta$)                       (step forward a small distance $\delta$)
  $y_B, y_Y$ = observe_object()                    (scan for object and check if blue)
if $y_B = 1$                                       (if blue has been detected)
  Switch to Left
else if $y_Y = 1$                                  (if yellow has been detected)
  Switch to Right
```

Robot 2 Policy

Robot 2 uses the primitives P_A, P_T, P_B, and P_R, and requires knowledge of the set of distances w from the object that allow the robot to fall into the limit cycle. First, in its initial state, R_2 measures the distance between it and the object, and the color directly in front of it (if any). It uses this knowledge to switch to a substrategy: either *Limit Cycle*, *Left*, or *Lost*.

In the *Limit Cycle* substrategy, as shown in Fig. 2.3, the robot translates forward, rotates, and repeats – continuously executing the limit cycle and counting how many times it bumps the object forward.

In the *Left* substrategy, the robot orients itself so that it is facing the blue side of the object, then switches to *Limit Cycle*, where it will translate toward the object, rotate, and enter the limit cycle.

In the *Lost* substrategy, R_2 translates forward to a wall or the object, and then stays still. It can never recover from this state.

Algorithm 8 Robot 2 Policy: π_2

```
Requires
Primitives: $P_A$, $P_T$, $P_B$, $P_R$
Parameters: $w$ is the range of distances that are attracted by the limit cycle

Initial
$count = 0$
$y_R, y_B$ = observe_object()                      (read object distance and color)
if $y_B = 1$ and $y_R \in w$                  (if blue was detected at a distance in $w$)
    Switch to Limit Cycle
else if $y_B = 1$ and $y_R \notin w$      (if blue was detected at a distance not in $w$)
    Switch to Left
else
    Switch to Lost

Limit Cycle
$P_T$                                          (translate forward to an obstacle)
$P_A(u_A = \theta)$                                                    (rotate $\theta$)
$P_T$                                          (translate forward to an obstacle)
$P_A(u_A = \theta)$                                                    (rotate $\theta$)
$P_T$                                          (translate forward to an obstacle)
$P_A(u_A = \theta)$                                                    (rotate $\theta$)
$count++$

Left (bounce off of blue side of object)
aim_toward_blue()
Switch to Limit Cycle

Lost
$P_T$                                          (translate forward to an obstacle)
```

Robot 3 Policy

Robot 3 uses the primitives P_A, P_T, and P_B. In its initial state, R_3 attempts to execute the limit cycle (repeating P_T and P_A) six times – which, if R_3 started in the limit cycle, would translate to two cycles and the robot would detect blue twice during those two cycles (each time it bumped the object). If this is the case, the robot increases its *count* to 2 and switches to the *Limit Cycle* substrategy. If the robot attempts to execute two limit cycles and does *not* detect blue exactly twice, that means it did not start with the correct initial conditions, and switches to the *Lost* substrategy.

In the *Limit Cycle* substrategy, as shown in Fig. 2.3, the robot translates forward, rotates, and repeats – continuously executing the limit cycle and counting how many times it bumps the object forward.

In the *Lost* substrategy, R_3 translates forward to a wall or the object, and then stays still. It can never recover from this state.

Algorithm 9 Robot 3 Policy: π_3

```
**Requires**
Primitives: $P_A$, $P_T$, $P_B$

**Initial**
$count = 0$
$inc = 0$                                        (variable for counting bounces)
$B = 0$                          (variable for counting instances of blue detection)
**while** $B < 2$ and $inc < 6$        (while blue has been detected less than twice
                                  and fewer than six bounces have been attempted)
    $inc++$
    **if** $P_B(y_B = 1)$                             (if blue has been detected)
        $B++$                                      (count one blue detection)
    $P_T$                                  (translate forward to an obstacle)
    $P_A(u_A = \theta)$                                          (rotate $\theta$)
**if** $B = 2$                                (if blue has been detected twice)
    $count = 2$
    Switch to **Limit Cycle**
**else if** $inc \geq 6$                    (if six bounces have been attempted)
    Switch to **Lost**


**Limit Cycle**
$P_T$                                      (translate forward to an obstacle)
$P_A(u_A = \theta)$                                              (rotate $\theta$)
$P_T$                                      (translate forward to an obstacle)
$P_A(u_A = \theta)$                                              (rotate $\theta$)
$P_T$                                      (translate forward to an obstacle)
$P_A(u_A = \theta)$                                              (rotate $\theta$)
$count++$


**Lost**
$P_T$                                      (translate forward to an obstacle)
```