Information Requirements of Collision-Based Micromanipulation

Alexandra Q. Nilles*,1, Ana Pervan*,2, Thomas A. Berrueta*,2,
Todd D. Murphey2 and Steven M. LaValle1,3

1Department of Computer Science,
University of Illinois at Urbana-Champaign, Urbana, IL, USA
2Department of Mechanical Engineering,
Northwestern University, Evanston, IL, USA
3Faculty of Information Technology and Electrical Engineering,
University of Oulu, Oulu, Finland

Abstract. We present a task-centered formal analysis of the relative power of several robot designs, inspired by the unique properties and constraints of micro-scale robotic systems. Our 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, we 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*. We 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. Using an information space framework and a hierarchical controller, we compare several robot designs, emphasizing the information requirements of goal completion under different initial conditions, as well as what is required to recognize irreparable task failure. Finally, we present a physically-motivated model of boundary interactions, and analyze the robustness and dynamical properties of resulting trajectories.

1 Introduction

Robots at the micro-scale have unique constraints on the amount of possible onboard 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 [23,24]. While certain robotic systems have succeeded at these tasks with assistance from external sensors and actuators [26], 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

*These authors contributed equally to the work.
e-mails: `nilles2@illinois.edu`, `anapervan@u.northwestern.edu`, `tberrueta@u.northwestern.edu`, `t-murphey@northwestern.edu`, `steven.lavalle@oulu.fi`

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 [4]. The work presented in this manuscript 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, we analyze the information requirements of tasks [8].

Particularly, we focus on the task of *micromanipulation*, which is a fundamental task primitive underlying more complex, finely-tuned procedures such as drug delivery and cell transplantation [16]. In this paper, we investigate a variety of micro-robot motion strategies and hierarchically describe their performance. In particular, we focus on boundary interactions: controlling what the robot does when it encounters a boundary of its environment. Our models are deliberately abstract, yet physically motivated, in an attempt to define abstractions that roughly cover the set of possible physical realizations of autonomous micro-robots. For each class of designs, we present results on minimal requirements to accomplish the task of micromanipulation, with respect to controller complexity, sensor power, and onboard memory. Identifying minimal information requirements is essential for rigorously reasoning about robot performance, and is also a first step toward automating co-design of robots and their policies.

1.1 Background and Related Work

The scientific potential of precise manipulation at microscopic scales has been appreciated for close to a century [5]. As micro/nano-robots have become increasingly sophisticated, biomedical applications such as drug delivery [16] and minimally-invasive surgery [24] have emerged as grand challenges in the field [28].

To formally analyze the information requirements of planar micromanipulation, we deliberately abstract many physically-motivated actuators and sensors, such as odometry, range-sensing, differential-drive locomotion, and others. Additionally, similar to the work in [8], we avoid tool-specific manipulation by purposefully abstracting robot-object contacts in an effort to be more general. However, we focus 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 [12,17,22]. In particular, we will use the example of a robot pushing a box in a "corridor" with parallel walls. While this environment is simple (and our solution depends on the environment being known beforehand), many applications in microrobotics allow for the environment to be known or even designed in tandem with the robots. In fact, a large amount of research is currently investigating how to design environment geometry for directed transport, control, and rectification of micro-robot motion [6,13,15]. Thus, through deliberate abstraction we develop and analyze a model of collision-based micromanipulation that can serve as a test-bed for micro-robotic designs.

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 [10]. Furthermore, battery capacities and efficiencies [19], as well as charging and power harvesting [7], experience diminishing performance at small scales, which restrict electrical designs of micromachines. Additionally, at these length-scales the physics of an environment—*e.g.*, a fluid or granular medium—can dominate the locomotor capabilities of a robot [2,21]. These mechanical, electrical, and environmental 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 nanorobot is capable of capturing and releasing molecular payloads in response to a binary stimulus [9]. 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. On the other hand, top-down approaches use the formulation of high-level control policies to guide the physical design of robots. For example, in [20], the authors developed a projection operator that maps continuous optimal control policies to finite state machines capable of encoding the policy at varying degrees of fidelity. The resulting state machine can then be directly mapped to physical implementations of a design.

A unifying theory of robotic capabilities was established in [18]. 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 we are capable of mapping the individual information histories of different robot designs into the task information space, the performance of robots may be compared. We expand upon this framework in the following sections.

2 Model and Definitions

Here we introduce relevant abstractions for characterizing robots generally, as well as their capabilities for given tasks. We largely follow from the work of [14,18].

2.1 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 [18], we 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 $h_i : X \to Y_i$ is the observation function. Primitives may correspond to use of either a sensor or an actuator, or both if their use is simultaneous (see [14] for examples). We will model time as proceeding in discrete stages. At stage k the robot observes sensor reading y_k and must choose its next action u_k. 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, we occasionally write robot definitions as $R = \{P_1,\ ...,\ P_N\}$ when robots share the same configuration space to emphasize differences between robot capabilities.

2.2 Information Spaces

A useful abstraction to reason about robot behavior in the proposed framework is the information 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 $(u_0, y_1, \ldots, u_{k-1}, y_k)$. The history, combined with initial conditions, yields the *history information state*, $\eta_k = (\cdot, y_0, u_0, y_1, \ldots, u_{k-1}, y_k)$. In this framework, initial conditions may either be known exactly, or be 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 a function $\kappa : \mathcal{I}_{hist} \to \mathcal{I}_{der}$ 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 us to directly compare 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, we introduce an *information preference relation* to distinguish between derived information states [18]. 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 we can assess "preference" over information states, notated as $\kappa(\eta_1) \preceq \kappa(\eta_2)$ if η_2 is preferred over η_1.

2.3 Robot Dominance

Extending on the definition of information preference relations in the previous section, we can define a similar relation to compare robots. Here, we define a relation 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. We additionally define a function F that iteratively applies a policy to update an information history. The updated history I-state is given by $\eta_{m+k} = F^m(\eta_k, \pi, x_k)$.

Definition 1. (Robot dominance from [18]) 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 $\kappa^{(1)} : \mathcal{I}_{hist}^{(1)} \to \mathcal{I}_{der}$ and $\kappa^{(2)} : \mathcal{I}_{hist}^{(2)} \to \mathcal{I}_{der}$, if for all: $\eta_1 \in \mathcal{I}_{hist}^{(1)}$ and $\eta_2 \in \mathcal{I}_{hist}^{(2)}$ for which $\kappa^{(1)}(\eta_1) \preceq \kappa^{(2)}(\eta_2)$; and $u_1 \in U^{(1)}$; there exists a policy, defined as $\pi_2 : \mathcal{I}_{hist}^{(2)} \to U^{(2)}$, generating actions for R_2 such that for all $x_1 \in X^{(1)}$ consistent with η_1 and all $x_2 \in X^{(2)}$ consistent with η_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 $\kappa^{(1)}$ and $\kappa^{(2)}$, 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 $R_1 \equiv R_2$. Lastly, we introduce the following lemma:

Lemma 1. (from [18]) 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:

(a) $R_1 \trianglelefteq R_1 \cup R_3$ (adding primitives never hurts);
(b) $R_2 \equiv R_2 \cup R_1$ (redundancy does not help);
(c) $R_1 \cup R_3 \trianglelefteq R_2 \cup R_3$ (no unexpected interactions).

3 Manipulating a Cart in a Long Corridor

We begin our analysis of the information requirements of micro-scale object manipulation by introducing a simple, yet rich, problem of interest. Consider a long corridor, containing a rectangular object, as shown in Fig. 1(a). The object may only translate left or right down the corridor, and cannot translate toward the corridor walls or rotate. We can abstract this object 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. 1(a)). We believe that 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, we assume 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. We also assume 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.

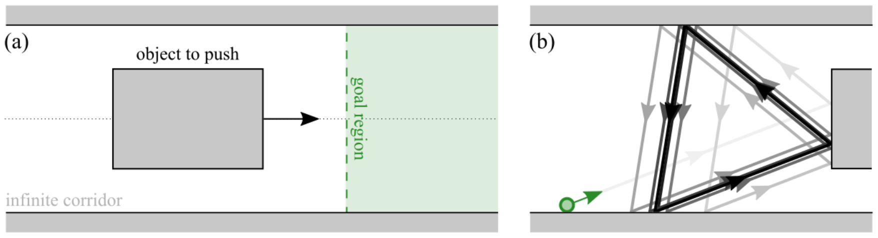


Fig. 1: (a) A rectangular object in a long corridor that can only translate to the left or right, like a cart on a track. The robot's task is to move the object into the green goal region. (b) The robot, shown in green, executes a trajectory in which it rotates the same relative angle each time it collides.

Here, we investigate the requirements of *minimal* strategies for object manipulation, given the constraints of micro-robotic control strategies. In the spirit of minimality, we make use of all interesting dynamical behaviors of our system prior to the specification of increasingly complex control policies. To this end, we have chosen to study "bouncing robots," which have been shown to exhibit several behaviors, such as highly robust limit cycles, chaotic behavior, and large basins of attraction [1,17,25]. Once discovered, these behaviors can be chained together and leveraged towards solving robotic tasks such as coverage and localization without exceedingly complex control strategies [3]. 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 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 Section 5.

4 A Formal Comparison of Several Robot Designs

In order to achieve the goal of manipulating an object in a long corridor, we will introduce several robot designs and then define policies that each robot might use to 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 and succeed at the task.

Prior to describing the robot designs, we introduce the relevant primitives that will be used in composing the robots. Figure 2 shows four robotic primitives taken directly from [18] (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, we define P_R 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 we refer the reader to [18]. The chosen prim-

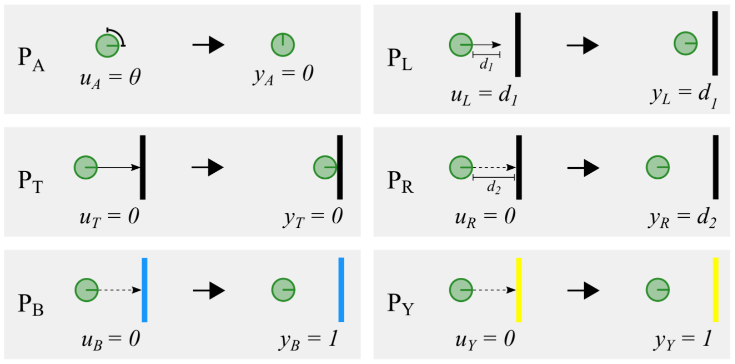


Fig. 2: Simple robotic behaviors called primitives. P_A is a local rotation, P_T is a forward translation to an obstacle, P_L is a forward translation a set distance, P_R is a range sensor, P_B senses the color blue, and P_Y senses the color yellow.

itives were largely selected based on their feasibility of implementation at the micro-scale, as observed in many biological systems [25,11].

To differentiate different facets of the object being manipulated, we employ two primitives that are sensitive to the signature of each side of the target. 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 the observation functions h_B and h_Y return 1 when the appropriate signature is in front of the robot. We deliberately make the "blue" and "yellow" sensors 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 3.

These six simple primitives, shown in Fig. 2 can be combined in different ways to produce robots of differing capabilities. Notably, here 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 our proposed robots and their constructed policies are explored in the following subsections.

4.1 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 [16,26,27]. Given the prevalence of such powerful robots capable of perfect information and actuation in the literature, we introduce this non-minimal design for the purposes of comparison to the minimal robots we present in the following sections.

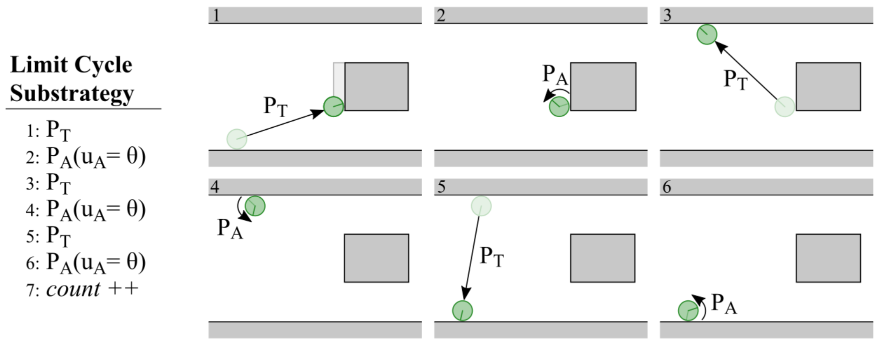


Fig. 3: The *Limit Cycle* strategy uses the primitive P_T, which moves forward until coming into contact with an object, and P_A, which rotates relative to the robot heading. Once a limit cycle is completed, the *count* variable is incremented.

We can specify the primitive for an all-capable robot 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 $R_0 = \{P_O\}$. 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, we provide an example of such a policy π_0 that completes the task in Algorithm 2 in the Appendix.

4.2 Robot 1: Complex

The underlying design principle behind each of the following minimal robotic designs is modularity. By considering the available set of primitives, we construct modular control policies to best accomplish the manipulation task. To this end, we use our available primitives to develop *subroutines* corresponding to useful behaviors such as wall following, measuring the distance to an object, and orienting a robot in the direction of the blue side of the cart. These subroutines are specified in the Appendix. Moreover, through these subroutines we construct meta-level behaviors, which we term *substrategies*, that can be combined to ac-

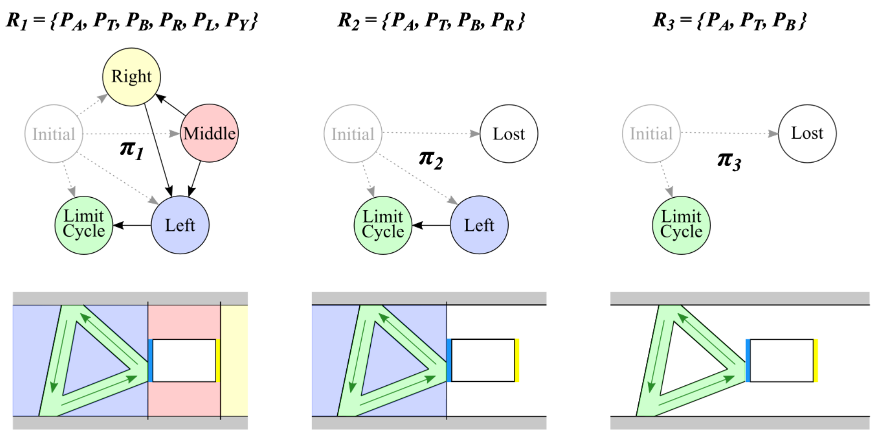


Fig. 4: (Left) A complex robot (composed of 6 primitives) can successfully achieve its goal no matter its initial conditions. (Middle) A simple robot (composed of 4 primitives) can only be successful if its initial conditions are on the left side of the object. (Right) A minimal robot (composed of 3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

complish the task. We represent the complete robot policy π_i as a combination of these substrategies in the form of a finite state machine (FSM).

The key substrategy that enables the success of our minimal robot designs is the *Limit Cycle* substrategy (shown in Fig. 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 Section 5). All other substrategies employed in the design of our robots serve the purpose of positioning the robot into a configuration where it is capable of carrying out the limit cycle substrategy.

Hence, we define our complex (yet minimal relative to R_0) robot 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. Its corresponding policy π_1, as represented by an FSM, is shown in Fig. 4. The details describing this policy are in Algorithm 3 in the Appendix, but it is important to note 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. 4. Due to the structure of the FSM and the task at hand, the robot always knows that if it is in the *Limit Cycle* state it is succeeding 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. 3) the robot does not require memory to perform this substrategy and we have only included it for facilitating analysis.

4.3 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. 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 4 in the Appendix.

4.4 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 5 in the Appendix), 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.

4.5 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. 3 and in the algorithms in the Appendix). 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.

We express the goal for the task of manipulating the cart in a long corridor as $\mathcal{I}_G \subseteq \mathcal{I}_{der}$, where $\mathcal{I}_G$ is an open subset of the nonnegative integers. In this set up, as illustrated in Fig. 1(a), the robot must push the object some N times, corresponding to a net distance traveled, to succeed. More formally, we express our information preference relation 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.

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 our modular 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, we state 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. 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 $\eta_1 \in \mathcal{I}^{(1)}_{hist}, \eta_2 \in \mathcal{I}^{(2)}_{hist}, \eta_3 \in \mathcal{I}^{(3)}_{hist}$, and define $\mathcal{I}$-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 is 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 we can deduce the rest of the hierarchies presented in this section.

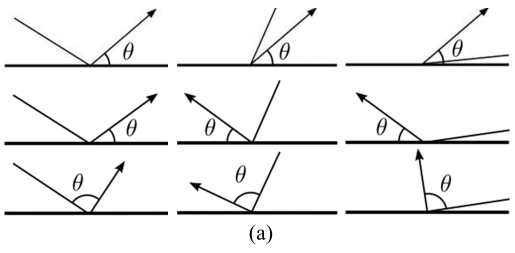
Comparing R_0 and R_1 To compare R_0 and R_1 we may continue with a similar reachability analysis as in the previous subsection. We note that from any $x \in X$, R_1 and R_0 are capable of reaching the object and nudging it. This means that given that the domain X is bounded and information history states $\eta_0 \in \mathcal{I}^{(0)}_{hist}, \eta_1 \in \mathcal{I}^{(1)}_{hist}$ 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 $\kappa^{(1)}(\eta_1) \preceq \kappa^{(0)}(F^l(\eta_0, \pi_0, x))$. So we have that $R_1 \trianglelefteq R_0$ 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.

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

Lemma 2. (Transitive property) Given three robots R_0, R_1, R_2, and knowledge that $R_2 \trianglelefteq R_1$ and $R_1 \equiv R_0$, then $R_2 \trianglelefteq R_0$ is also true.

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, we see 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. In the following sections we discuss the necessary conditions for establishing such



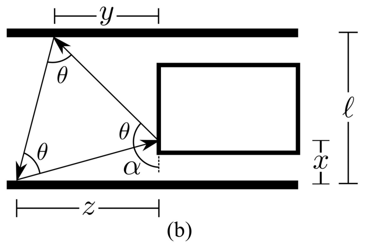


Fig. 5: (a) The three types of bounce rules considered. The top row depicts *fixed* bounce rules, where the robot leaves the boundary at a fixed angle regardless of the incoming trajectory. The second row shows *fixed monotonic* bounce rules preserving the horizontal direction of motion but keep the absolute angle between the boundary and the outgoing trajectory fixed. The third row shows *relative* bounce rules rotating the robot through an angle relative to its previous heading. (b) The geometric setup for analyzing dynamics of triangular trajectories formed by repeated single-instruction relative bounce rules.

cycles, as well as the robustness properties of limit cycle behavior, which are important for extending this work to less idealized and deterministic settings.

5 Feasibility and Dynamics of Cyclic Motion Strategies

In this section we will derive and analyze limit cycle motion strategies that can be used to manipulate objects through incidental collisions. The goal is to engineer *flows*: patterns in the robot's trajectory that are useful for this task. When looking to move an object down the corridor, three boundaries are present: the two walls of the corridor, and the object itself. An ideal motion strategy would use collisions with the two walls to direct the robot to collide with the object in a repeatable pattern. Is it possible to do so with a single instruction to the robot that it repeats indefinitely, every time it encounters a boundary? This single-instruction strategy would lend itself well to the design of a micro-robot, so that the robot robustly performs the correct boundary interaction each time.

We consider three types of boundary actions, as seen in Fig. 5(a). The first and second types (fixed bounce rules) could be implemented through alignment with the boundary (mechanical or otherwise) such that forward propulsion occurs at the correct heading. This measurement and reorientation can be done compliantly, and does not necessarily require traditional onboard measurement and computation. See, for example, similar movement profiles of microorganisms resulting from body morphology and ciliary contact interactions [11,25]. The third type of boundary interaction (relative bounce rule) requires a rotation relative to the robot's prior heading, implying the need for rotational odometry or a fixed motion pattern triggered upon collision.

Here, we analyze the relative power of these actions for the task of pushing an object down a hallway, without considering the broader context of initial conditions or localization, which were considered in Section 4. Particularly, we consider the system of the robot, hallway and object as a purely dynamical system, to establish the feasibility of trajectories resulting from minimal policies.

For the rest of this section, suppose that the bouncing robot navigates in a corridor with parallel walls and a rectangular object as described in Section 3.

Proposition 1. *For all bouncing robot strategies consisting of a single repeated fixed bounce rule, there does not exist a strategy that results in a triangular trajectory that makes contact with the rectangular object.*

Proof. A fixed bounce rule in this environment will result in the robot bouncing back and forth between the two parallel walls forever after at most one contact with the object. A monotonic fixed bounce rule would result in the robot bouncing down the corridor away from the object after at most one contact.

Remark 1. The feasibility of fixed (monotonic) bounce rule strategies in environments without parallel walls is unknown and possibly of interest.

Proposition 2. *There exist an infinite number of strategies consisting of two fixed bounce rules that result in a triangular trajectory that makes contact with the rectangular object. The robot must also be able to distinguish the object from a static boundary, or must know its initial conditions.*

Proof. Geometrically, an infinite number of triangles exist that can be executed by a robot with the choice between two different fixed bounce rules. It is necessary that the robot be able to determine when it has encountered the first corridor wall during its cycle, in order to switch bounce rules to avoid the situation described in the proof of Proposition 2. One sufficient condition is that the robot knows the type of boundary at first contact, and has one bit of memory to track when it encounters the first corridor wall and should switch actions. Equivalently, a strategy could use a sensor distinguishing the object from a static boundary at collision time, along with one bit of memory.

Proposition 3. *There exists a strategy consisting of a single repeated relative bounce rule that results in a triangular trajectory that makes contact with the rectangular object. Moreover, this strategy is robust to small perturbations in the rotation θ and the initial angle α.*

Proof. See Fig. 5(b) for the geometric setup. Here we will provide exact expressions for the quantities x, y and z as a function of the initial conditions, position, and orientation of the robot on its first collision with the object.

First assume x_t is given, as the point of impact of the robot with the object at stage t. Let α_t be the angle indicated in Fig. 5(b), the angle between the incoming trajectory at x_t and the object face. Then, using simple trigonometry, $y = (\ell - x_t)\tan(\pi - \theta - \alpha_t)$ where θ is the interior angle of the robot's rotation. To create an equiangular triangle, $\theta = \frac{\pi}{3}$, but we will leave θ symbolic for now to enable sensitivity analysis. To compute z, we consider the horizontal offset due to the transition from the top to the bottom of the hallway. This leads to $z = y + \ell\cot(\frac{3\pi}{2} - 2\theta - \alpha_t)$. Finally, we can compute the coordinate of where the

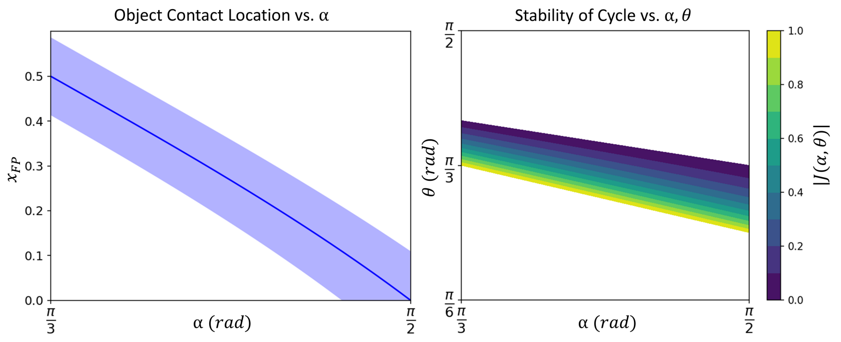


Fig. 6: (Left) The location of the impact on the object is nearly a linear function of α for $\theta = \frac{\pi}{3} \pm 0.1$. Only the counterclockwise cycle is shown; the clockwise cycle follows from symmetry. (Right) The Jacobian of this system indicates robustness to small perturbations in θ (white regions represent instability or infeasibility).

robot will return to the object, $x_{t+3} = z\tan(\frac{3\pi}{2} - \alpha - 3\theta)$. Solving for the fixed point of this dynamical system, $x_{FP} = x_t = x_{t+3}$ gives

$$x_{FP} = \ell\left(\frac{\tan(\alpha+\theta) - \tan(\alpha+2\theta)}{\tan(\alpha+\theta) - \tan(\alpha+3\theta)}\right).$$

Note that this x_{FP} expression is not valid for all values of (α, θ), but only for values leading to inscribed triangles in the environment such that $x_{FP} > 0$.

For $\theta = \frac{\pi}{3}$, Fig. 6 (Left) shows the location of the fixed point as a function of the angle α, indicating that an infinite number of stable cycles exist for this motion strategy, and every point of the object in the bottom half of the corridor is contactable with counterclockwise cycles. The top half of the object can be reached using clockwise cycles.

Since $x_{t+3} = F(x_t)$ is linear in x_t, the Jacobian is $J = -\tan(\alpha+\theta)\tan(\alpha + 3\theta - \frac{\pi}{2})$. Figure 6 (Right) shows the value of the Jacobian as a function of α and θ. All the shaded regions have an absolute value less than one, indicating robustness to small perturbations of α and θ over a large domain.

Remark 2. We note that Propositions 2 and 3 are equivalent in terms of memory requirements, both requiring the robot to "remember" either its previous heading or its previous bounce rule at each stage of the strategy. However, the fixed bounce strategy implies that the robot has more knowledge of its surroundings than the relative bounce strategy, as it must measure the plane of the boundary it encounters and orient itself appropriately. The relative bounce strategy requires only information within the robot's own reference frame.

6 Conclusion

In this paper, we have designed robust motion strategies for minimal robots that have great promise for micromanipulation. We have also 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 our example of a rectangular obstacle in a corridor is simple, we think of this as *robust directed transport*, a key building block for future work.

6.1 Future Directions

Outside of our 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, we focus on derived I-spaces that encode information about what *will happen* to the robot if it continues with its strategy. The setting of micro-robotics provides motivation for the approach; at the micro-scale, coarse high-level controllers that can be applied to a collection of many micro-robots are much easier to implement than fine-grained individual controllers. This approach requires formal reasoning about the space of possible trajectories, in order to funnel the system into states that allow for task completion. Abstracting the system as bouncing robots in planar geometric environments allows us to leverage knowledge of the environment and dynamics of the robots to construct dynamically predictive derived information states. Beyond dynamical analysis, this approach allows for sensor models to be integrated naturally. Recent work in formal methods such as hybrid systems, probabilistic programming, and symbolic reasoning hints at an exciting unification of I-space analysis and techniques for automated analysis of dynamical systems. If we can automatically identify system dynamics that reduce state-space uncertainty, and have sensor models that allow for discretization of possible states through preimage analysis, the combination can be used to design I-spaces that guarantee a strong level of robustness.

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. Y. Alapan, O. Yasa, B. Yigit, I. C. Yasa, P. Erkoc, and M. Sitti. Microrobotics and microorganisms: Biohybrid autonomous cellular robots. *Annu. Rev. Control Rob. Auto. Sys.*, 2(1):205–230, 2019.
3. A. Bayuelo, T. Alam, L. Bobadilla, L. F. Niño, and R. N. Smith. Computing feedback plans from dynamical system composition. In *IEEE Int. Conf. Auto. Sci. Eng.*, pages 1175–1180, 2019.
4. A. Censi. A mathematical theory of co-design. Technical report, September 2016. Conditionally accepted to IEEE Trans. Rob.
5. R. Chambers, H. B. Fell, and W. B. Hardy. Micro-operations on cells in tissue cultures. *Proc. Royal Soc. of London.*, 109(763):380–403, 1931.
6. R Di Leonardo, L Angelani, D Dell'Arciprete, Giancarlo Ruocco, V Iebba, S Schippa, MP Conte, F Mecarini, F De Angelis, and E Di Fabrizio. Bacterial ratchet motors. *Proceedings of the National Academy of Sciences*, 107(21), 2010.
7. 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.
8. B. R. Donald, J. Jennings, and D. Rus. Information invariants for distributed manipulation. *Int. J. Rob. Res.*, 16(5):673–702, 1997.
9. 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.

10. 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.
11. Vasily Kantsler, Jörn Dunkel, Marco Polin, and Raymond E Goldstein. Ciliary contact interactions dominate surface scattering of swimming eukaryotes. *Proc. Natl. Acad. Sci.*, 110(4):1187–1192, 2013.
12. 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.
13. N Koumakis, C Maggi, and R Di Leonardo. Directed transport of active particles over asymmetric energy barriers. *Soft matter*, 10(31):5695–5701, 2014.
14. S. M. LaValle. *Planning Algorithms.* Cambridge University Press, USA, 2006.
15. He Li and HP Zhang. Asymmetric gear rectifies random robot motion. *EPL (Europhysics Letters)*, 102(5):50007, 2013.
16. J. Li, B. Esteban-Fernández de Ávila, W. Gao, L. Zhang, and J. Wang. Micro/nanorobots for biomedicine: Delivery, surgery, sensing, and detoxification. *Sci. Rob.*, 2(4), 2017.
17. 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.
18. J. M. O'Kane and S. M. LaValle. Comparing the power of robots. *Int. J. Rob. Res.*, 27(1):5–23, 2008.
19. 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.
20. A. Pervan and T. D. Murphey. Low complexity control policy synthesis for embodied computation in synthetic cells. *Int. Work. Alg. Found. Rob.*, Dec 2018.
21. E. M. Purcell. Life at low Reynolds number. *Amer. J. Phys.*, 45(1):3–11, 1977.
22. 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. *Sci. Rob.*, 4(34), 2019.
23. M. Sitti, H. Ceylan, W. Hu, J. Giltinan, M. Turan, S. Yim, and E. Diller. Biomedical applications of untethered mobile milli/microrobots. *Proc. of IEEE*, 103(2):205–224, Feb 2015.
24. 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.
25. S. E. Spagnolie, C. Wahl, J. Lukasik, and J. Thiffeault. Microorganism billiards. *Phys. D*, 341:33–44.
26. 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, Sep 2015.
27. T. Xu, J. Zhang, M. Salehizadeh, O. Onaizah, and E. Diller. Millimeter-scale flexible robots with programmable three-dimensional magnetization and motions. *Sci. Rob.*, 4(29), 2019.
28. 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. *Sci. Rob.*, 3(14), 2018.

Appendix

Subroutines

We define how the subroutines introduced in Section 4 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 1.

Algorithm 1 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 $inc \leq 360^\circ$ (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 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 3 the *Middle* substrategy only queries the observe_object() subroutine for y_B and y_Y.

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 2 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. 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 3 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

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. 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 4 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. 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 5 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)
```