Information Invariants for Distributed Manipulation[1]

Bruce Randall Donald, **James Jennings**[*] and **Daniela Rus**[†]
Robotics & Vision Laboratory
Computer Science Department
Cornell University
Ithaca, New York
U.S.A

Abstract

In [Don4], we described a manipulation task for cooperating mobile robots that can push large, heavy objects. There, we asked whether explicit local and global communication between the agents can be removed from a family of pushing protocols. In this paper, we answer in the affirmative. We do so by using the general methods of [Don4] analyzing *information invariants.*

[*]The author's current address is Department of Computer Science, Tulane University, New Orleans, Louisiana, USA.

[†]The author's current address is Department of Computer Science, Dartmouth College, Hanover, New Hampshire, USA.

[1]This paper describes research done in the Robotics and Vision Laboratory at Cornell University. Support for our robotics research is provided in part by the National Science Foundation under grants No. IRI-8802390, IRI-9000532, and by a Presidential Young Investigator award, and in part by the Air Force Office of Sponsored Research, the Mathematical Sciences Institute, Intel Corporation, and AT&T Bell laboratories. The third author has been supported by the Advanced Research Projects Agency of the Defense Department under ONR Contract N00014-92-J-1989, by ONR Contract N00014-92-J-39, and by NSF Contract IRI-9006137.

This paper is a revised version of a paper presented at the Workshop on the Algorithmic Foundations of Robotics (WAFR), San Francisco, Feb. 17, 1993.

We discuss several measures for the information complexity of the task: (a) How much internal state should the robot retain? (b) How many cooperating agents are required, and how much communication between them is necessary? (c) How can the robot change (side-effect) the environment in order to record state or sensory information to perform a task? (d) How much information is provided by sensors? and (e) How much computation is required by the robot? To answer these questions, we develop a notion of information invariants. We develop a technique whereby one sensor can be constructed from others by adding, deleting, and reallocating (a) – (e) among collaborating autonomous agents. We add a resource to (a) – (e) and ask: (f) How much information is provided by the task mechanics? By answering this question, we hope to develop information invariants that explicitly trade-off resource (f) with resources (a) – (e). The protocols we describe here have been implemented in several different forms, and we report on experiments to measure and analyze information invariants using a pair of cooperating mobile robots for manipulation experiments in our laboratory.

1 Introduction

Imagine taking a broad view of robots and robotics. Let us say a *robot* is a computer that can apply forces in the physical world. We insist these forces be (i) external and (ii) programmable (using the computer). (i) rules out clocks (which apply only internal forces), and (ii) excludes videocassette recorders, which can be programmed electronically but not mechanically.

Typically robots also use sensory input (vision, sonar, force-sensing, kinesthetic position sensing) to guide their control strategies and correct for errors in modeling and execution. *Distributed robotics* involves many robots cooperating to perform a task. *Manipulation* tasks require changing the physical world — for example, moving furniture or orienting parts. In *robotic manipulation* a robot applies external forces to perform a manipulation task. When at least two robots simultaneously apply forces to a coupled dynamical system (*e.g.*, they push the same couch), we call it *parallel manipulation.* When a parallel manipulation task is performed by a distributed system, we call it *distributed manipulation.* Hence, distributed manipulation protocols are both kinematically (*i.e.*, spatially) and computationally distributed.

In this paper, we develop and analyze synchronous and asynchronous manipulation protocols for a small team of cooperating mobile robots than can push large boxes. The boxes are typically several robot diameters wide, and 1-2 times the mass of a single robot, although the robots have also pushed couches that are heavier (perhaps 2-4 times the mass, and 8×3 robot diameters in size). We build on the ground-breaking work of [Mason, EM] and others on planar sensorless manipulation. Our work differs from previous work on pushing in several ways. First, the robots and boxes are on a similar dynamic and spatial scale. Second, a single robot is not always strong enough to move the box by itself (specifically, its "strength" depends on the effective lever arm). Third, we do not assume the robots are globally coordinated and controlled. (More precisely, we first develop protocols based on the assumption that local communication is possible, and then we subsequently remove that communication via a series of source-to-source transformations on the protocols). Fourth, our protocols assume neither that the robot has a geometric model of the box, nor that the first moment of the friction distribution is known. Instead, the robot combines sensorimotor experiments and manipulation strategies to infer the necessary information (the experiments have the flavor of [JR]). Finally, the pushing literature generally regards the "pushers" as moving kinematic constraints. In our case, because (i) there are at least two robot pushers and (ii) the robots are less massive than the box, the robots are really "force-appliers" in a system with significant friction. In this sense, our task is in some ways closer in flavor to *dynamic manipulation* [ML], even though the box dynamics are essentially quasi-static.

Of course, our protocols rely on a number of assumptions in order to work. We develop a framework for analysis and synthesis, based on *information invariants* [Don4], to reveal these assumptions and expose the information structure of the task. We believe our theory has implications for the parallelization of manipulation tasks on spatially distributed teams of cooperating robots. To develop a parallel manipulation strategy, first we start with a perfectly synchronous protocol with global coordination and control. Next, in distributing it among cooperating, spatially separated agents, we relax it to an MPMD[2] protocol with local communication and partial synchrony. Finally, we remove all explicit communication. The final protocols are asynchronous, and essentially "uniform," or SPMD[2]—the same program

[2]SPMD (MPMD) = *Single (Multiple) Program, Multiple Data.*

runs on each robot. Ultimately, the robots must be viewed as communicating implicitly through the task dynamics, and this implicit communication confers a certain degree of synchrony on our protocols. Because it is both difficult and important to analyze the information content of this implicit communication and synchronization, we are wielding a fairly heavy hammer, namely the theory of information invariants.

1.1 The Big Picture

Our goal is to investigate the information requirements for robot tasks. This paper uses the theoretical framework introduced by Donald in [Don,Don4]. A central theme to previous work (see the survey article [Don1] for a detailed review) has been to determine what information is required to solve a task, and to direct a robot's actions to acquire that information to solve it. Key questions concern:

1. What information is needed by a particular robot to accomplish a particular task?

2. How may the robot acquire such information?

3. What properties of the world have a great effect on the fragility of a robot plan/program?

4. What are the capabilities of a given robot (in a given environment or class of environments)?

These questions can be difficult. Structured environments, such as those found around industrial robots, contribute towards simplifying the robot's task because a great amount of information is encoded, often *implicitly*, into both the environment and the robot's control program. These encodings (and their effects) are difficult to measure. We wish to quantify the information encoded in the assumption that (say) the mechanics are quasi-static, or that the environment is not dynamic. In addition to determining how much "information" is encoded in the assumptions, we may ask the converse: how much "information" must the control system or planner compute? Successful manipulation strategies often exploit properties of the (external) physical world (eg, compliance) to reduce uncertainty and hence gain information. Often, such strategies exploit mechanical computation, in which the mechanics of the task circumscribes the possible outcomes of an action by dint of physical laws. Executing such strategies may require little or no computation; in contrast, planning or simulating these strategies may be computationally expensive. Since during execution we may witness very little "computation" in the sense of "algorithm," traditional techniques from computer science have been difficult to apply in obtaining meaningful upper and lower bounds on the true task complexity. We hope that a theory of information invariants can be used to measure the sensitivity of plans to particular assumptions about the world, and to minimize those assumptions where possible.

We would like to develop a notion of information invariants for characterizing sensors, tasks, and the complexity of robotics operations. We may view information invariants as a mapping from tasks or sensors to some measure of information. The idea is that this measure characterizes the intrinsic information required to perform the task—if you will, a measure of complexity. For example, in computational geometry, a rather successful measure has been developed for characterizing input sizes and upper and lower bounds for geometric algorithms. Unfortunately, this measure seems less relevant in robotics, although it remains a useful tool. Its apparent diminished relevance in embedded systems reflects a change in the scientific culture. This change represents a paradigm shift from *offline* to *online* algorithms. Increasingly, robotics researchers doubt that we may reasonably assume a strictly offline paradigm. For example, in the offline model, we might assume that the robot, on booting, reads a geometric model of the world from a disk and proceeds to plan. As an alternative, we would also like to consider *online* paradigms where the robot investigates the world and incrementally builds data structures that in some sense represent the

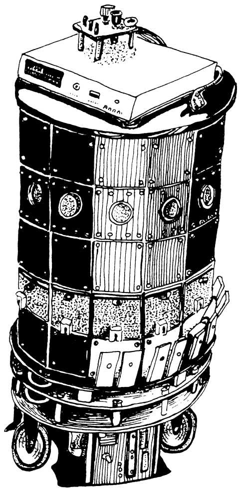

Figure 1: The Cornell mobile robot TOMMY. Note (mounted top to bottom on the cylindrical enclosure) the ring of sonars, the IR Modems, and the bump sensors. LILY is very similar.

Figure 2: TOMMY and LILY pushing a couch in a straight line, using an asynchronous SPMD protocol requiring no explicit communication.

external environment. Typically, online agents are not assumed to have an *a priori* world model when the task begins. Instead, as time evolves, the task effectively forces the agent to move, sense, and (perhaps) build data structures to represent the world. From the online viewpoint, offline questions such as "what is the complexity of plan construction for a known environment, given an *a priori* world model?" often appear secondary, if not artificial. In this paper, we describe two working robots TOMMY and LILY, which may be viewed as online robots. We discuss their capabilities, and how they are programmed. These examples and the papers [JR,Don,Don4,DJR2,RDJ] link our work to the recent but intense interest in online paradigms for situated autonomous agents. In particular, in these papers, we discuss what kind of data structures robots can build to represent the environment. We also discuss the *externalization* of state, and the *distribution* of state through a system of spatially separated agents.

We believe it is profitable to explore online paradigms for autonomous agents and sensorimotor systems. However, the framework remains to be extended in certain crucial directions. In particular, sensing has never been carefully considered or modeled in the online paradigm. The chief *lacuna* in the armamentarium of devices for analyzing online strategies is a principled theory of sensori-computational systems. We attempted to fill this gap in [Don,Don4], where we provided a theory of *situated sensor systems*. We argue this framework is natural for answering certain kinds of important questions about sensors. Our theory is intended to reveal a system's information invariants. When a measure of intrinsic information invariants can be found, then it leads rather naturally to a measure of hardness or difficulty. If these notions are truly intrinsic, then these invariants could serve as "lower bounds" in robotics, in the same way that lower bounds have been developed in computer science.

In our quest for an intrinsic measure of the information requirements of a task, we are inspired by Erdmann's monograph on sensor design [Erd3], and the *information invariants* that Erdmann introduced to the robotics community in 1989 [Erd2]. We also observe that rigorous examples of information invariants can be found in the theoretical literature from as far back as 1978 (see, for example, [BK, Koz]). We note that many interesting lower bounds (in the complexity-theoretic sense) have been obtained for motion planning questions (see, eg, [Reif, HSS, Nat, CR]; see, eg, [Erd1, Don2, Can, Bri] for upper bounds). Rosenschein has developed a theory of synthetic automata which explore the world and build data-structures that are "faithful" to it [Ros]. His theory is set in a logical framework where sensors are logical predicates. Perhaps our theory could be viewed as a geometric attack on a similar problem. This work was motivated by the theoretical attack on perceptual equivalence begun by [DJ] and by the experimental studies of [JR]. Horswill [Hors] has developed a semantics for sensory systems that models and quantifies the kinds of assumptions a sensori-computational program makes about its environment. He also gives source-to-source transformations on sensori-computational "circuits." The paper [Don4] discusses the semantics of sensor systems. This formalism is used to explore some properties of what we call *situated sensor systems.* [Don4] describes a way to transform sensori-computational systems. When one can be transformed into another, we say the latter can be "reduced" to the former, and we call the transformation a "reduction." We also derive algebraic algorithms for reducing one sensor to another. This machinery is only necessary if one wishes to automate the transformation process; it is quite easy to calculate reductions "by hand," using pencil and paper.

In addition to the work discussed here in Section 1, for a detailed bibliographic essay on previous research on the geometric theory of planning under uncertainty, see, *e.g.*, [Don1] or [Don3].

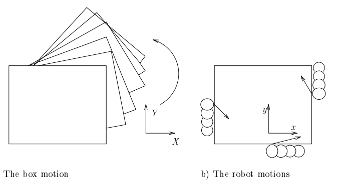

Figure 3: The box is being rotated by three cooperating autonomous agents. (a) The motion of the box viewed in world coordinates. (b) The relative motion of the pushing robots, viewed in a system of coordinates fixed on the box. The arrows illustrate the direction of the applied forces.

The goals outlined here are ambitious and we have only taken a small step towards them. The questions above provide the setting for our inquiry, but we are far from answering them completely. We hope that information invariants can serve as a framework in which to measure the capabilities of robot systems, to quantify their power, and to reduce their fragility with respect to assumptions that are engineered into the control system or the environment. We believe that the equivalences that can be derived between communication, internal state, external state, computation, and sensors, can prove valuable in determining what information is required to solve a task, and how to direct a robot's actions to acquire that information to solve it. There are several things we have learned. We can determine a lot about the information structure of a task by (i) parallelizing it and (ii) attempting to replace explicit communication with communication "through the world" (through the task dynamics). Communication "through the world" takes place when a robot changes the environment and that change can be sensed by another robot. In this paper we give two different protocols (strategies) for a 2-robot pushing task: one protocol uses explicit communication and the other makes use of an encoding in the task mechanics of the same information. Our approach of quantifying the information complexity in the task mechanics involves viewing the world dynamics as a set of mechanically implemented "registers" and "data paths". This permits certain kinds of *de facto* communication between spatially separated robots. This "equivalence" of task mechanics and communication is operational in flavor, and we are still exploring its generality.

	Resource
(a)	How much internal state should the robot retain?
(b)	How many cooperating agents are required, and how much communication between them is necessary?
(c)	How can the robot change (side-effect) the environment in order to record state or sensory information to perform a task?
(d)	How much information is provided by the sensors?
(e)	How much computation is required by the robot?
(f)	How much information is provided by the task mechanics?

Table 1: A list of questions about the resource consumption of robot protocols.

We believe that, by spatially distributing resources among collaborating agents, the information characteristics of a robot task are made explicit. That is, by asking, *How can this task be performed by a team of robots?* one may highlight the information structure. In robotics, the evidence for this is, so far, largely anecdotal. In computer science, one often learns a lot about the structure of an algorithmic problem by parallelizing it; we argue that a similar methodology is useful in robotics.

It is very difficult to analyze the interaction of sensing, computation, communication (a – e) and mechanics (f) (see Table 1) in distributed manipulation tasks. The analyses of [Don4] focus on (a – e), and each analysis is "parameterized" by the task. This paper represents an attempt to integrate a measure of the "information content of the task mechanics" (f) into the theory. Nevertheless, the theory is still biased towards sensing, and it remains to develop a framework that treats action and sensing on an equal footing.

This paper draws extensively on the material reported in the monograph by Donald [Don4], and announced in an abbreviated, preliminary version in [Don]. We reported on our ideas on coordinated manipulation strategies in a preliminary form in [DJR1-2].

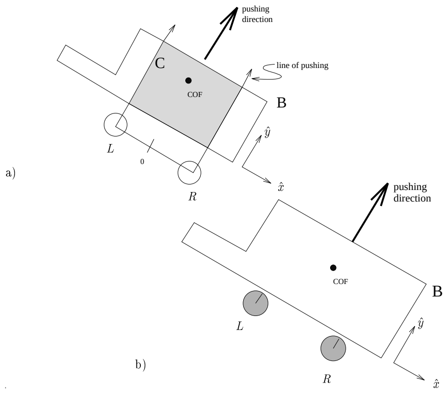

Figure 4: (a) the "two-finger" pushing task vs. (b) the two robot pushing task. The goal is to push the block B in a straight line.

$$\begin{array}{lll} \text{Repeat :} & \text{measure}(\tau) & \\ & \text{case} & (\tau = 0) \Rightarrow \\ & & \quad \text{push}(p) \\ & & (\tau < 0) \Rightarrow \\ & & \quad \text{move}(R) \\ & & (\tau > 0) \Rightarrow \\ & & \quad \text{move}(L) \end{array}$$

Figure 5: Protocol O (for a two-fingered gripper).

1.1.1 Research Agenda

Robot builders make claims about robot performance and resource consumption. In general, it is hard to verify these claims and compare the systems. We really think the key issue is that two robot programs (or sensor systems) for similar (or even identical) tasks may look very different. We discuss why it is hard to compare the "power" of such systems. Our examples are distinguished in that they permit relatively crisp analytical comparisons: they represent the kinds of theorems about information trade-offs that we believe can be proved for sensorimotor systems. The analyses in Section 2 reveal trade-offs in terms of resource consumption. We then ask, is there a general theory quantifying the power gained in such trade-offs? In Section 3, we present a theory, which represents a systematic attempt to make such comparisons based on geometric and physical reasoning. In [Don4], we operationalize our analysis by making it computational; we give effective (albeit theoretical) procedures for computing our comparisons. See Section 7 for a summary.

We wish to rigorously compare embedded sensori-computational systems. To do so, we define a "reduction" $\leq_1$ that attempts to quantify when we can "efficiently" build one sensor out of another (that is, build one sensor using the components of another).[3] Hence, we write $A \leq_1 B$ when we can build A out of B without "adding too much stuff." The last is analogous to "without adding much information complexity." Our measure of information complexity is *relativized* both to the information complexity of the sensori-computational components of B, and to the bandwidth of A. This relativization circumvents some tricky problems in measuring sensor complexity. In this sense, our "components" are analogous to *oracles* in the theory of computation. Hence, we write $A \leq_1 B$ if we can build a sensorimotor system that simulates A, using the components of B, plus "a little rewiring." A and B are modeled as *circuits*, with wires (datapaths) connecting their internal components. However, our sensori-computational systems differ from computation-theoretic (CT) "circuits," in that their spatial configuration—*i.e.*, the spatial location of each component—is as important as their connectivity.

We develop some formal concepts to facilitate the analysis. *Permutation* models the permissible ways to reallocate and reuse resources in building another sensor. Intuitively, it captures the notion of repositioning resources such as the active and passive components of sensor systems (*e.g.*, infra-red emitters and detectors). *Geometric codesignation constraints* further restrict the range of admissible permutations. *I.e.*, we do not allow arbitrary relocation; instead, we can constrain resources to be "installed at the same location", such as on a robot, or at a goal. *Output communication* formalizes our notion of "a little bit of rewiring." When resources are permuted, we often find they must be reconnected using "wires", or data-paths. If we separate previously colocated resources, we will often need to add a communication mechanism to connect the now spatially separate components. Like CT reductions, $A \leq_1 B$ defines an "efficient" transformation on sensors that takes B to A. However, we can give a generic algorithm for synthesizing our reductions (whereas no such algorithm can exist for CT.)[4] Whether such reductions are widely useful or whether there exist better reductions is open; however we try to demonstrate the potential usefulness both through examples and through general claims on algorithmic tractability. We also give a "hierarchy" of reductions, ordered on power, so that the strength of

[3] $\leq_1$ is also called $<_s$ in [Don].

[4] For example: no algorithm exists to decide the existence of a linear-space (or log-space, polynomial time, Turing-computable, etc.) reduction between two CT problems.

our transformations can be quantified.

Our ideas have the following applications:

1. *(Comparison).* Given two sensori-computational systems A and B, we can ask "which is more powerful?" (in the sense of $A \leq_1 B$, above).

2. *(Transformation).* We can also ask "Can B be transformed into A?"

3. *(Design).* Suppose we are given a specification for A, and a "bag of parts" for B. The bag of parts consists of boxes and wires. Each box is a sensori-computational component ("black box") that computes a function of (i) its spatial location or pose and (ii) its inputs. The "wires" have different bandwidths, and they can hook the boxes together. Then, our algorithms decide, can we "embed" the components of B so as to satisfy the spec of A? The algorithms also give the "embedding" (that is, how the boxes should be placed in the world, and how they should be wired together). Hence, we can ask, can the spec of A be implemented using the bag of parts B?

4. *(Universal Reduction).* Consider application 3, above. Suppose that in addition to the spec for A, we are given an encoding of A as a bag of parts, and an "embedding" to implement that spec. Suppose further that $A \leq_1 B$. Since this reduction is relativized both to A and to B, it measures the "power" of the components of A relative to the components in B. By universally quantifying over the configuration of A, we can ask, "can the components of B always do the job of the components of A?"

   More specifically: Let α be a "configuration" of sensorimotor system A. Thus α encodes the spatial embedding of A as well as its wiring connectivity. Similarly, let β be a configuration of system B. Let $A(\alpha)$ and $B(\beta)$ denote systems A and B "installed" at these configurations. The gist of universal reduction is that, we can decide whether or not $(\forall \alpha,\ \exists \beta): \ A(\alpha) \leq_1 B(\beta)$.

1.2 Outline

We discuss the resources (a)—(f) (see Table 1) for an experiment with communicating robots. We consider the task of coordinated manipulation of large objects (particularly, manipulation of a large box (or couch) using a pair of communicating mobile robots). We foreground the task of pushing an object, using two communicating robots who need to infer the position of the first moment of the friction distribution with respect to their lines of pushing (see Figure 4a). In [Don4], we asked whether explicit communication could be removed from this protocol (by "explicit" we mean local communication, such as IR, or global communication, such as RF). In this paper we give a protocol with no explicit communication, and we analyze and compare our protocols using the tools introduced in [Don4]. We believe our methods generalize to other manipulation tasks and to larger teams of robots, but work is still underway; for example, in [DJR1-2,RDJ], we considered the task of coordinated manipulation of large objects (particularly, rotations, or more accurately, *"reorientations"* of a large box using a team of communicating mobile robots). There, we examined an *offline* strategy, in which the robots have a geometric model of the object, and an *online* strategy, in which they do not. In Section 9, we describe these strategies and sketch a general methodology for developing distributed manipulation protocols.

2 Pushing with Two Communicating Mobile Robots

To introduce our ideas we consider a task involving two autonomous mobile robots. The two robots must cooperate to push a box. Now, many issues related to information invariants can be investigated in the setting of a single agent. However, one of our ideas is that, by spatially distributing resources (a) – (f) among collaborating agents, the information characteristics of a task are made explicit. That is, by asking, *How can this task be performed by a team of robots?* one may highlight the information structure.

Here is a preview of how we will proceed. The goal of the *pushing task* is to move an object along a straight line (called the *pushing direction*) with two agents. We first describe a strategy for this task designed to be executed by a manipulator with two rigidly connected fingers and force feedback. We then propose variants of this algorithm that are suitable for execution by autonomous mobile robots. Finally, we compare these strategies with respect to the resources (a)—(f) in Table 1.

2.1 Three Pushing Protocols

Consider the task whose goal is to push a box B in a straight line. Figure 4a depicts one robot (the reader should picture a robot manipulator, or gripper) executing this task. The robot consists of two rigidly connected fingers L and R; for example, they could be the fingers of a parallel-jaw gripper. One complication involves the micro-mechanical variations in the slip of the box on the table [Mas]. This phenomenon is very hard to model, and hence it is difficult to predict the results of a one-fingered push; we will only obtain a straight-line trajectory when the center of friction (COF) lies on the line of pushing. However, with a two-fingered push, the box will translate in a straight line so long as the COF lies between the fingers. The nice thing about this strategy is that the COF can move some and the fingers can keep pushing, since we only need ensure the COF lies in some region C (see Figure 4a), instead of on a line. Second, if the COF moves outside C, then the fingers can move sideways to "capture" it again. We have implemented the control loop described as Protocol O on our PUMA robot's two finger gripper (see Figure 5). The basic idea is to sense the reaction torque τ about the center of friction (COF) in Figure 4a. If $\tau = 0$, push forward in direction $\hat{y}$. If $\tau < 0$ move the fingers in $\hat{x}$; else move the fingers in $-\hat{x}$.

From the mechanics perspective it might appear we are done. However, it is difficult to overstate how critically Protocol O above relies on global communication and control. Now, consider the analogous pushing task in Figure 4b. Each finger is replaced by an autonomous mobile robot such as those described in [RD]. The robots we have in mind are the Cornell mobile robots (see Figure 1), but the details of their construction are not important. The robots can move about by controlling motors attached to wheels. The robots are autonomous and equipped with a ring of 12 simple Polaroid ultrasonic sonar sensors. Each robot has onboard processors for control and programming. The description in [RD] is augmented as follows. (This description characterizes the robots in our lab). We equip each robot with 12 infra-red modems/sensors, arrayed in a ring about the robot body. Each modem consists of an emitter-detector pair. When transmitting or receiving, each modem essentially functions like the remote control for home appliances (*e.g.*, televisions). In addition, each robot has a ring of one-bit contact ("bump") sensors.

We assume the following:

- using bump sensors, robots in contact with a flat object face can measure the relative orientation of that face [JR];
- the robots are on the same flat face of the object;
- both robots know the direction of pushing, p;
- the robots can synchronize their velocities;[5]
- by examining the servo-loop in [RD], it is clear that we can compute a measure of applied force by observing the applied power, the position and velocity of the robot, and the contact sensors.

The pushing strategy described as Protocol 0 can be approximated by observing the following (see Figures 6, 7). Each robot can compute its applied force (f_1 or f_2) and contact mode, and communicate these data to the other. This allows the robots to compute the net torque τ about a point in between the two robots, and from the sign of this quantity, to infer the location of the box. If the center of friction (COF) is between the two lines of pushing, each robot continues pushing along the line p. If there is a positive net torque, the instantaneous center of friction is to the left of both robots. In this situation, the left robot is in contact with the box, while the right robot may or may not be in contact. The left robot can "recapture" the center of friction between the two lines of pushing by moving left along the face of the box (move(L)). If the right robot is not in contact with the box (the predicate `(break?)` returns TRUE) it executes a guarded-move (motion until contact) in the direction p. Otherwise this robot takes the null action, $\emptyset$. The case when the net torque is negative is symmetric. We call this Protocol I (see Figures 6, 7).

A variant of this protocol can be derived for a quasi-static (QS) system. Here, relative displacements along the line of pushing p are measured instead of forces. Figure 9 shows a configuration where the two robots originate at

[5] For the pushing task, it suffices to assume that the robots have identical control systems and can command the same speeds. More generally, velocity synchronization requires that the robots begin and end pushing at the same time; this can be necessary for more complicated manipulation tasks. A protocol for velocity synchronization is not hard to synthesize using our methods; see Section 9.1.

positions $x_1(0)$ and $x_2(0)$, respectively. Their locations at time t are $x_1(t)$ and $x_2(t)$. In this protocol (Figure 8), the initial locations of the robots are communicated to determine their offset Σ_0. Σ_0 "specifies" (or better, *parameterizes*) the pushing task: this offset determines the pushing direction p relative to the initial orientation of the box face. Hence, the computation of Σ_0 can be viewed as a calibration step. The robots exchange location information successively at each loop iteration; this information is used to infer the direction of motion of the box. We call this Protocol I(QS) (see Figures 8 and 9).

We now derive a new protocol from Protocol I(QS) by observing that the information needed to determine the motion of the box (*i.e.* Σ_0 and Σ) is related to the angle θ between the normal to the face of the box n and the direction of pushing p as follows: $2r \tan \theta_0 = \Sigma_0$ and $2r \tan \theta = \Sigma$ (see Figures 9, 11, 10). Moreover, we observe that the tangent function is monotonic and sign preserving; this means we can adapt the control system to servo on θ instead of Σ, without knowing r. Specifically, let n be the normal to the box face. [JR] describe an algorithm for measuring n with bounded error. In our final protocol, the robots measure θ_0 (the initial angle between n and p), and compare this value to the angle $\theta(t)$ measured at time t in order to infer the direction of motion of the box. A negative change in the value of this angle implies a clockwise rotation of the box. A large positive change implies a counterclockwise rotation. The robots adjust their pushing location on the face of the box accordingly. This is an example of how the robots can use the task dynamics instead of explicit communication to determine their next actions. We call this pushing strategy Protocol II (Figure 10).

Left Robot	**Communication**	**Right Robot**
Repeat :		Repeat :
measure(f_1)		measure(f_2)
	$\boxed{L \rightarrow R}$ (f_1)	
		$\tau = r_1 \times f_1 - r_2 \times f_2$
	$\boxed{L \leftarrow R}$ (sign(τ))	
case ($\tau = 0$) $\Rightarrow$		case ($\tau = 0$) $\Rightarrow$
push(p)		push(p)
($\tau < 0$) $\Rightarrow$		($\tau > 0$) $\Rightarrow$
if(`break?`)		if(`break?`)
`guarded-move`(p)		`guarded-move`(p)
else $\emptyset$		else $\emptyset$
($\tau > 0$) $\Rightarrow$		($\tau < 0$) $\Rightarrow$
move(L)		move(R)

Figure 6: Protocol I. The robots synchronize their actuators by communicating their force measurements.

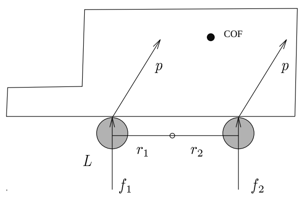

Figure 7: The mechanical observables for Protocol I

Left Robot	**Communication**	**Right Robot**
measure($x_1(0)$)		measure($x_2(0)$)
	$\boxed{L \rightarrow R}$ $(x_1(0))$	
		$\Sigma_0 = x_2(0) - x_1(0)$
Repeat :		Repeat :
measure($x_1(t)$)		measure($x_2(t)$)
	$\boxed{L \rightarrow R}$ $(x_1(t))$	
		$\Sigma = x_2(t) - x_1(t)$
		$s = \text{sign}(\Sigma - \Sigma_0)$
	$\boxed{L \leftarrow R}$ (s)	
case $(s = 0) \Rightarrow$ push(p)		case $(s = 0) \Rightarrow$ push(p)
$(s = -1) \Rightarrow$ move(R)		$(s = -1) \Rightarrow$ move(R)
$(s = 1) \Rightarrow$ move(L)		$(s = 1) \Rightarrow$ move(L)

Figure 8: Protocol I (Quasi-Static). The case for breaking contact is not shown; it can be handled as in Figure 6. The robots synchronize their actions by communicating their odometry.

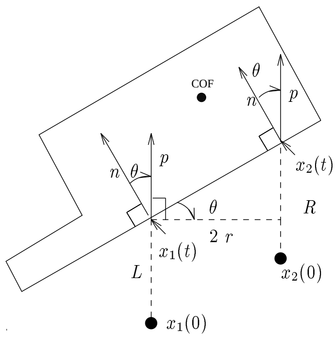

Figure 9: Protocol I(QS). The normal to the box face is denoted by n. The x axis is parallel to the direction of pushing p. The lines of pushing are distance $2r$ apart (perpendicular distance). θ is the angle between n and p.

Left Robot

$\theta_0 \leftarrow \theta(0)$
Repeat :
case $(\texttt{break?}) \Rightarrow$
$\quad\texttt{guarded-move}(p)$
$(\theta(t) \approx \theta_0) \Rightarrow$
$\quad\text{push}(p)$
$(\theta(t) \gg \theta_0) \Rightarrow$
$\quad\text{move}(L)$
$(\theta(t) \ll \theta_0) \Rightarrow$
$(**)\quad\emptyset$

Right Robot

$\theta_0 \leftarrow \theta(0)$
Repeat :
case $(\texttt{break?}) \Rightarrow$
$\quad\texttt{guarded-move}(p)$
$(\theta(t) \approx \theta_0) \Rightarrow$
$\quad\text{push}(p)$
$(\theta(t) \gg \theta_0) \Rightarrow$
$(*)\quad\emptyset$
$(\theta(t) \ll \theta_0) \Rightarrow$
$\quad\text{move}(R)$

Figure 10: Protocol P.II. This protocol is "almost uniform," and can be made uniform by changing the $\emptyset$ lines $(*)$ to $\textsc{Move}(L)$ and $(**)$ to $\textsc{Move}(R)$. Note that "uniform" does not quite imply SIMD, since the protocols run asynchronously. We say these protocols are *SPMD*— Single Program, Multiple Data. There is no explicit communication between the robots.

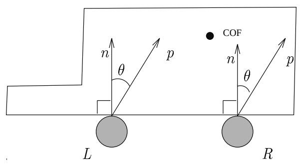

Figure 11: The mechanical observables for Protocol II.

2.2 Comparing the Protocols

Now, we ask, how do protocols P.I, P.I(QS), and P.II compare to one another with respect to the resources (a) – (f) in Table 1? We first note that the three protocols require different sensing capabilities. Protocol P.I relies on force sensing, Protocol P.I(QS) relies on position sensing, and Protocol P.II relies on orientation sensing. Next we observe that the robots must coordinate to find controls that result in stable pushing along p. This coordination is accomplished differently in the three protocols. In Protocol P.I and Protocol P.I(QS) the robots synchronize by exchanging their sensor values.

Robots executing Protocol P.I require communicating $\log \Bbbk(f_1)$ bits to transmit the value of force f_1, and 2 bits to transmit the sign of the torque τ ($\Bbbk(b)$ denotes how many values a variable b may take on). Robots executing Protocol P.I(QS) require $\log \Bbbk(x_1)$ bits to transmit the value of the distance x_1, and 2 bits to transmit the sign of s. In Protocol P.II there is no explicit communication and the synchronization is realized through the world, by monitoring the change in the angle θ between the face normal and the pushing direction. In other words, the robots infer the motion of the object by decoding changes in the task mechanics. Thus, Protocols P.I and P.I(QS) rely on direct communication, while protocol II does not. The internal state requirements of the three strategies are also different. Protocol P.I requires no internal state. Protocol P.I(QS) requires a register to record the value Σ_0. Protocol P.II requires a register to record the value θ_0.

We can get a deeper understanding of the relationship between these protocols by describing them as circuits, and by attempting to "transform" or "reduce" one to the other. We do so below.

3 Reductions and Transformations

We now formalize our model of manipulation protocols by viewing them as "manipulation circuits." (The theory in Sections 3-8 is adapted from [Don4], but the particular examples and application (especially, claim 4.1), the definition of manipulation circuits, and the definition of tradeoffs are new.) We model these circuits as graphs. Vertices correspond to different sensori-computational components of the system (what we will call "resources" below). Edges correspond to "data paths" through which information passes.

Different immersions of these graphs correspond to different spatial allocation of the "resources." Our thesis demands: *What information is added (or lost) in a manipulation circuit when we change its immersion?* and *What information is preserved under all immersions?* We also define an operator + as a way to "combine" manipulation circuits. The operation + is like taking the union of two graphs. We formalize manipulation circuits by first introducing the notion of a *sensori-computational system* (we use the term *"sensor system"* to mean *"sensori-computational system"* where it is mellifluous.) In the end of this section we show how to extend sensor systems to manipulation circuits.

3.1 Situated Sensor Systems

Definition 3.1 *A* labelled graph $\mathcal{G}$ *is a directed graph* (V, E) *with vertices* V *and edges* E*, together with a labelling function that assigns a label to each vertex and edge. Where there is no ambiguity, we denote the labelling function by* ℓ*.*

Definition 3.2 *A* sensor system $\mathcal{S}$ *is represented by a labelled graph* (V, E)*. Each vertex is labelled with a* component*. Each edge is labelled with a* connection.

See Figures 12-13 for illustrations for the circuits—that is, the sensor systems for protocols P.I(QS) and P.II. We will use the terms *protocol* to refer to the computer programs in Figures 8-10, and *circuit* for the sensor systems in Figures 12-13. It is clear that each circuit implements its protocol. Now, in Protocol P.I(QS) (Figure 8), there are three communications operations. The first two ($L \rightarrow R$) use the same datapath; they simply refer to the first use, and to subsequent use, of the same resource. In our circuits, we now restrict the *components* to be the resources such as those described in the figures; however, our definitions could, we feel, be generalized to other resources. In the figures, the components correspond to boxes: $\boxed{\text{ODOM}}$ is an odometer, the "signal"[6] coming out of this box is the odometry reading. The box $\boxed{-}$ performs subtraction, the boxes $\boxed{x_1(0)}$ and $\boxed{\Sigma_0}$ are registers, and they implement internal state. The part of the circuit labelled **case** interfaces to the control

[6]We use *"signal"* as a metaphor; our circuits are strictly digital. Either *message* or *stream* would be better, but both have distracting religious connotations.

part of the circuits, which is the same for both protocols. Intuitively, once the control is added to the sensor system, we obtain a *manipulation circuit*. This augmentation is made precise below.

One interesting resource is $\theta(t)$ in Figure 13—we call this box a θ-*source*—it produces a signal indicating relative orientation. [JR] describe in detail how to implement a bounded-error θ-source. Here is the basic idea. A θ-source is an abstraction of relative normal sensing. It allows the normal of the box to be treated as an external "register" that both robots may read and write. We could implement an (approximate) θ-source as follows. The robot has a ring of 1-bit bump sensors. These are used to implement relative normal sensing [JR]. We specify the pushing direction p, by specifying θ_0 (see Figure 10), the direction of p relative to the box normal $n(0)$ at time 0. More specifically: at the beginning of the task, each robot does a guarded move along p until contact with the box. It then aligns normal to the box, using the bounded-error algorithm of [JR]. Finally, the robot turns by angle θ_0 using pure position control.

INIT is one bit of state, and RUN= $\overline{\text{INIT}}$. The small crossed circles ($\oplus$) that these bits run into are gates; the $\downarrow$ input must be 1 for the $\leftrightarrow$ signal to pass. Thus in robot L in Figure 12, if INIT= 1, then the odometer writes into the register $x_1(0)$. When INIT= 0, then the $\boxed{-}$ component is fed the odometry input. We assume that numbers are represented as signed integers. The $\boxed{\text{SGN}}$ box just selects out the sign bit. We omit the logic to toggle INIT once the register $x_1(0)$ is written. Whereas L requires one bit of state INIT_0, R requires two, due to the asymmetry of the protocol P.I(QS). These are denoted INIT_1 and INIT_2.

Connections are like data-paths in that they carry information; a connection's label represents the information that will be sent along that path. Connections carry data between components. We adopt the convention that two components can communicate without an (explicit) connection when they are spatially colocated. Now, in Figures 12-13, many of the data paths are *internal*, *i.e.*, $L \to L$ or $R \to R$. The most interesting datapaths are the *external* datapaths: the $L \to R$ edge (b) labelled $\text{COMM}(\Delta x_1)$, and the $L \leftarrow R$ edge (b′) labelled $\text{COMM}(s)$. (b) has bandwidth $\log \Bbbk(\Delta x_1)$ bits, and (b′) has bandwidth 2 bits.

Consider Figure 13. Here, there is a θ-source that is external to both robots. There are two interesting external data paths from the θ-source, one to L marked $\text{COMM}(\cdot)$, and one marked (b) to R. Both these datapaths have bandwidth $\log \Bbbk(\theta(t))$ bits.

Definition 3.3 *Let b be a variable that ranges over all possible values that a sensor system can compute. We call b the* output *of the system. Let $\Bbbk(b)$ be the number of values b can take on, and define $\log \Bbbk(b)$ to be both the* size *of b and the* output size *of the sensor. The output size is an upper bound on the bit-complexity of b. For example, if b takes on integer values in the range $[1, q]$, then $\Bbbk(b) = q$, and $\log \Bbbk(b) = \log q$. Now, suppose the information b is communicated over a datapath e. We will assume that the information is communicated repeatedly; without loss of generality, we take the unit of time to be the interval of the occasion to communicate the information. Thus we can take the size of the output b to be the* bandwidth *of e.*

So far, we have defined components and connections operationally. We now give a formal definition. Components and connections are defined by their *simulation functions.* Simulation functions describe the behavior of both components and connections.

Consider a component $\ell(v)$ associated with vertex v. To simulate a component, we need to know (i) its inputs and (ii) its configuration. Suppose a component has r inputs and s outputs, each of which lies in some space R. Let C be the configuration space of the component. A *simulation function* for a component $\ell(v)$ is a map[7] $\Omega_v : R^r \times C \rightarrow R^s$.

Now we connect the components together. Assume for a moment that all the components have the same input-output structure as Ω_v above (*i.e.*, that r and s are fixed throughout the system, but that the components themselves may perform different functions). We model an edge e between vertices v and u by its label, $\ell(e) = b$, and by a pair of integers, (i, j). $\log \Bbbk(b)$ is the *bandwidth* of the edge and the index i (resp. j) specifies to which of the r outputs of $\ell(v)$ (resp., s inputs of $\ell(u)$) we attach e ($1 \leq i \leq r$ and $1 \leq j \leq s$).

Now, a simulation function for this edge e is taken to be a function $\Omega_e : R \rightarrow R$. We will usually restrict the edge functions to be the identify function (but they also check for bandwidth, *i.e.*, that the transmitted data has size no greater than $\log \Bbbk(b)$).

[7] Components that retain state can be modeled by a function $\Omega_v : R^r \times C \times S \rightarrow R^s \times S$, where S is a *store* that records the state. For example, a state element with k bits of state would be modeled with $S = \{0, 1\}^k$. Alternatively, S can be absorbed as a factor subspace in the configuration space of the component.

We also define a resource called the "output device." Each sensor system must have exactly one vertex with this label, called the *ouput vertex*. The output vertex of the sensor system is where the output of the sensor is measured. The simulation function for the output device is the identity function, but the output value of this device defines the output value of the sensor system.

A simulation function $\Omega_{\mathcal{U}}$ for an entire sensor system $\mathcal{U}$, then, is a collection of component simulation functions such as Ω_v and edge simulation functions such as Ω_e. The function $\Omega_{\mathcal{U}}$ simulates all the component simulation functions in the correct configuration, and simulates routing the data between them using the edge simulation functions. We adopt the convention that two components can communicate without an (explicit) connection when they are spatially colocated. When all these component and edge functions are semi-algebraic, then the sensor simulation function $\Omega_{\mathcal{U}}$ is also semi-algebraic (see Section 7).

Definition 3.4 *Consider a sensor system $\mathcal{U}$ with simulation function $\Omega_{\mathcal{U}}$. The* output value *of $\mathcal{U}$ at a particular configuration is the value $\Omega_{\mathcal{U}}$ computes for that configuration. Hence the output value of $\mathcal{U}$ is a function of $\mathcal{U}$'s configuration.*

The notions output value *and* output *(Definition 3.3) are related as follows. The* output *of $\mathcal{U}$ is a variable that ranges over all possible output values of $\mathcal{U}$. Given another sensor system $\mathcal{V}$, we say the* output of $\mathcal{U}$ is the same as the output of $\mathcal{V}$ *when $\Omega_{\mathcal{U}}$ and $\Omega_{\mathcal{V}}$ are identical.*

Under this model, we can simulate trees of embedded sensorimotor computation. It is also possible (in principle) to simulate more general graphs and systems with state, but in this case the value at the output vertex may vary over time (even for a fixed configuration). In this case we need some explicit notion of time and blocking to model the (a)synchronous arrival of data at a component. Such extensions are considered in [Jen94]; for now we restrict our attention to trees, which suffice to model our examples. In general our discussion is restricted to consider one clock-tick; however, generalizations are possible to consider the time-varying behavior of the system [Jen94]. We will treat the circuits P.I(QS) and P.II (below) as effectively being trees, and not graphs, even though there is data flow both from R to L and L to R. To completely model manipulation circuits with feedback we must employ the notion of *weak simulation* and *weak equivalence* defined below.

To summarize: a *component* is a primitive device that computes a a function of (i) its inputs and (ii) its configuration $z \in C$. Each component is installed at a vertex of communication graph with d vertices, whose edges are the *connections* described above. The graph is immersed in a configuration space C^d, and the configuration z of a component is the configuration of its vertex. More generally, components can be *actuators.* An actuator is a component whose output forces the configuration of the graph to change or evolve through a *dynamics equation.* If the configuration of the entire graph is $\mathbf{z} = (z_1, \ldots, z, \ldots, z_d) \in C^d$, then the dynamics equation models a mapping from the actuator component $\ell(v)$'s output at z to the tangent space $T_{\mathbf{z}}C^d$ to the configuration space. See [RDJ,Jen94] for more discussion of actuators. The actuator systems of our circuits are identical and in this paper we do not consider them in detail. This actuator subsystem is represented by the elision $\text{CASE}(s) \cdots$ in Figures 12-13. The circuit for this system would look very much like a traditional plant diagram from control engineering.

Weaker forms of sensori-computational equivalence are possible. If we define the *state* of a sensor system $\mathcal{U}$ to be a pair $(\mathbf{z}, b)$ where $\mathbf{z}$ is the configuration of the system and b is the output value at $\mathbf{z}$, we can examine the equilibrium behavior of $\mathcal{U}$ as it evolves in state space. Consider the following notion of *strong simulation.* Informally, sensor system $\mathcal{U}$ strongly simulates sensor system $\mathcal{V}$ when they have the same output. See Definition 3.6 for the formal definition. By analogy, let us say that a system $\mathcal{U}$ *weakly simulates* another system $\mathcal{V}$ when $\mathcal{U}$ and $\mathcal{V}$ have identical, forward-attracting compact limit sets in state space.[8] If we replace strong simulation ($\cong$ in Definition 3.6) with weak simulation, all of our structural results go through *mutatis mutandis.* The computational results also go through, if we can compute limit sets and their properties (a difficult problem in general). Failing this, if we can derive the properties of limit sets "by hand" then in principle, reductions using weak simulation instead of strong simulation ($\cong$) can also be calculated by hand.

The following concepts define actuators precisely and formalize this notion of weak simulation.

1. *Actuators* are components that take input from sensori-computational circuits, and whose output manifests forces and torques in the configu-

[8]We are grateful to Dan Koditschek, who has suggested this formalism in his papers.

ration space of the system. Like any component, the output (*i.e.*, force exerted) of an actuator depends on its spatial configuration.

Differentially, the output of an actuator at configuration x in a configuration space C can be modeled as a tangent vector (x, f) in the tangent space T_xC to C at x. Informally, this means that the actuator exerts force f at point x.

2. The forces exerted by an actuator change the configuration of a system via a *dynamics equation.* That is, the forces cause the system to evolve in time. For example, Newton's dynamics equation is

$$f = m\ddot{x},$$

where m is mass.

More generally, we can permit actuators to be the more abstract controllers familiar in robotics. For example, a *velocity controller* is a component whose output causes the velocity of the system to change. One easy way to model a velocity controller uses a first-order dynamics equation such as generalized damper dynamics (see [LMT, Erd86, Don89]. The output of the actuator is still manifested as force tangent to the configuration space. However the dynamics equation is changed to

$$f = B\dot{x}$$

where B is a constant chosen in design.

3. A *mobile sensori-computational system* is a sensori-computational circuit (Definition 3.2) containing actuators, when the output of the actuators is used to change the configuration of the system. If the configuration space of the circuit is C_r, then the output of the actuators in the circuit force the configuration $\mathbf{x} \in C_r$ to change over time, via the dynamics equation. For example, consider the two robot system described in Section 2 (see Figure 1). The configuration space for one robot (say, TOMMY) is $C_1 = \mathbb{R}^2 \times S^1$ and a typical configuration $\mathbf{x}$ in C_1 is (x, y, θ) where (x, y) models the center of TOMMY and θ his orientation. When TOMMY executes a `translate` command, his configuration is advanced along the ray anchored at (x, y) and pointing in direction θ. A `rotate` command causes θ to change.

For the two-robot system Lily and Tommy, the configuration space is the product space $C_1 \times C_1$, which has six degrees of freedom. Mobile sensori-computational systems can model navigation, mapmaking, and localization procedures for mobile robots.

4. A *manipulation circuit* is a mobile sensori-computation circuit where the actuators are used to solve a manipulation task. This requires that the forces manifested through the actuators do two things: (i) cause the configuration of the robots to change, and (ii) cause the configuration of some external object to change. For example, in Figures 4b and 2, Tommy and Lily are being used to manipulate a large couch or box. Let C_r be the configuration space of the robots, so that $C_r = C_1 \times C_1$ as above. Let C_m be the configuration space of the couch. Since the couch has three degrees of freedom, $C_m = \mathbb{R}^2 \times S^1$.

   In a manipulation circuit, the forces from the actuators change both the configuration of the robots in C_r, and the configuration of the couch in C_m.

5. *Weak simulation* is an equivalence relation on mobile sensori-computational circuits. When two circuits have the same limit sets (as dynamical systems), we say they *weakly simulate* each other. We use weak simulation for manipulation circuits in place of strong simulation defined for sensori-computational circuits (Definition 3.6). For the rest of the paper we use $\cong$ to denote weak simulation, unless otherwise noted.

   When we are only interested in the behavior of the manipulated object, we can restrict the domain of weak simulation to the configuration space C_m. For example, we can view a set of goal configurations for the couch as a set G in C_m. We say a manipulation circuit *achieves* G when, viewed as a dynamical system, its limit set is G. Any two manipulation circuits that achieve G weakly simulate each other.

In our sensor systems, there is no separate notion of "sensor inputs." Instead, the sensory inputs are encoded in the configuration space.

3.2 Transformations as Reductions

In Section 4, we give a proof (claim 4.1) and an equation (3) relating the circuits P.I(QS) and P.II. Intuitively, Equation (3) indicates that, operationally

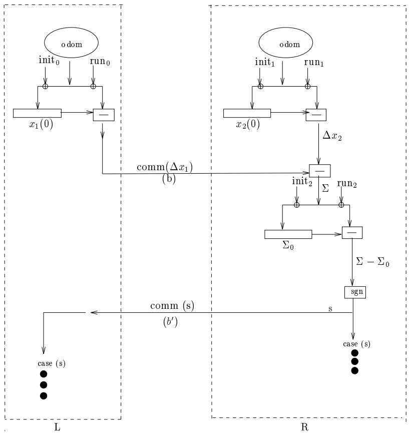

Figure 12: Manipulation circuit P.I(QS). This is a circuit for Protocol I (QS). This circuit shows one possible implementation of the protocol. Figures 12-13 do not show how to handle lost of contact (*i.e.*, the `(break?)` case), but this circuitry is easily added, and is the same for both P.I(QS) and P.II.

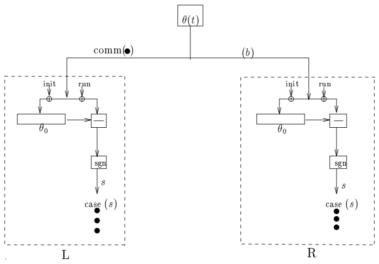

Figure 13: Manipulation circuit P.II. This is a circuit for Protocol II.

speaking, one could transform a system which executes Protocol P.I(QS) into one which executes Protocol P.II by removing the odometry from both robots, and by adding a θ–source, which is a component that senses the orientation of the manipulated object. In describing Protocol P.II, we demonstrated that one implementation of such a circuit involves using some retained state (θ_0), and a relative orientation sensor such as the bumper system described in [JR]. In fact, Claim 4.1 is a precise statement of this engineering fact. Below, we carefully define the operators $+$ and $-$, and formalize the notions of *simulation* and *efficient reduction*, as well as *permutation*, *etc.*

Our reduction involves two concepts. The first is *permutation,* and it involves redistributing resources in a manipulation circuit, without consuming new resources. Surprisingly, a redistribution of resources can add information to the system. In order for permutation to add information, it is necessary for the sensor system to be spatially distributed (as, for example, our circuits are). When permutation gains information, it may be viewed as a way of arranging resources in a configuration of lower entropy.

The second concept is *communication.* It measures resource (b) (from the abstract). We consider adding communication primitives of the form $\textsc{comm}(L \to R,\ \mathit{info})$, which indicates that L sends message *info* to R. Examples of this primitive are COMM(Δx_1) and COMM(s) in Figure 12. Like permutation, communication only makes sense in a spatially distributed manipulation circuit. That is, because spatially colocated components can communicate "for free" in our model, only "external" datapaths add information complexity to the system. In effect, to transform system Protocol O into Protocol I(QS) (see Figures 5, 8), we view it as a system composed of autonomous collaborating agents L and R, each of which has certain resources. The COMM($\cdot$) primitive above we view as shared between L and R. We measure communication by counting the number of agents and the bits required to transmit *info*. This is the only kind of external communication we will consider here (*i.e.*, $L \rightarrow R$ or $L \leftarrow R$); we will abbreviate it by COMM(*info*) when the direction is clear.

We can be sure of getting the semantics of COMM($\cdot$) correct by treating it as a sensor system in its own right (albeit, a small one). Now, COMM(b) defines the graph with vertices $\{u_1, u_o\}$ and a single edge $e = (u_1, u_o)$ with $\ell(e) = b$. The "head" vertex u_o of the edge $e = (u_1, u_o)$, is defined to be the *output vertex* of the sensor system COMM(b). The argument (parameter) b to COMM(b) determines the *bandwidth* of e. Thus, for example, COMM(b) specifies a graph with one edge e whose label is b. This specifies that the edge is a datapath that can carry information b; if b requires $k = \log \mathbb{k}(b)$ bits to encode then k is the *bandwidth* of e. Our model of communication is rather abstract. External communication is probably not possible without buffering by either the sender or the receiver. COMM($\cdot$) should include this buffer to be more realistic about modeling internal state.

We now formalize the ideas above. These definitions generalize sensori-computational systems to manipulation circuits, following Section 3.1. Consider a manipulation circuit with vertices V. For each vertex v in V, we assume there is a configuration space C. A point in this space C represents the configuration of the component.

Definition 3.5 *A* situated *(or* immersed) manipulation circuit $\mathcal{S}$ *is a manipulation circuit* $\mathcal{S} = (V, E)$*, together with an immersion* $\phi : V \rightarrow C$ *of the vertices. If* $v \in V$*, then we call* $\phi(v)$ *the* configuration of the vertex v*. When there is no ambiguity, we also call* $\phi(v)$ *the* configuration of the component $\ell(v)$.

A situated manipulation circuit is essentially an immersed graph. If the map ϕ in Definition 3.5 is injective, then we call ϕ an *embedding*. Immersions need not be injective. Moreover, in order to colocate vertices, it is necessary for immersions to be non-injective.

In Definition 3.5, the immersion ϕ may be a partial (as opposed to total) function, indicating that we do not specify the spatial configuration of those components whose vertices are outside the domain of the immersion. We denote the *domain* of a (partial) immersion $\phi : V \to C$ by $\phi^{-1}C$. We denote its *image* by $\operatorname{im}\phi$.

We can now define what it means for two systems to be equivalent:

Definition 3.6 *Given two sensor systems S and Q, we say Q* simulates *S if the output of Q is the same as the output of S. In this case we write $S \cong Q$. More generally, suppose we write*

$$(\mathcal{S}, \phi) \cong (\mathcal{U}, \psi) \tag{1}$$

for two situated sensor systems. Equation (1) is clearly well-defined when ϕ and ψ are total. Now, suppose that ϕ and ψ are partial, leaving unspecified the configurations of components $\ell(v)$ of $\mathcal{S}$ and $\ell(u)$ of $\mathcal{U}$. Then Equation (1) is taken to mean that $(\mathcal{U}, \psi)$ simulates $(\mathcal{S}, \phi)$ for any *configuration of v and u.*

For Definition 3.6, in the case where (say) ϕ is partial, we operationalize Equation (1) by rewriting it as a statement about all *extensions* $\overline{\phi}$ of ϕ. That is, we define $\operatorname{ex}\phi$ to be the set of all extensions of ϕ. Then, we write: "$\forall\overline{\phi} \in \operatorname{ex}\phi$, Equation (1) holds (with bars placed over the immersions)." We treat ψ similarly, with an inner universal quantifier, although codesignation constraints (Section 3.5) allow us to make the choice of extension $\overline{\psi}$ of ψ depend on the extension $\overline{\phi}$ that is bound by the outer quantifier. For example, Definition 3.6 becomes, "for all configurations $x \in C$ of v, for all configurations $y \in D_{\mathcal{S}}(x)$ of u, Equation (1) holds." Here $D_{\mathcal{S}}(x)$ is a set in C that varies with x; the function $D_{\mathcal{S}}(\cdot)$ models the codesignation constraints. Definition 3.6 can be generalized to any number of "unbound" vertices; see Equation (16) and [Don4].

Definition 3.6 precisely characterizes the equivalence relation of *strong simulation* described in Section 3.1. We now wish to generalize this definition in the case where $\mathcal{S}$ and $\mathcal{Q}$ are manipulation circuits. In this case, the relation strong simulation given by Definition 3.6 still makes formal sense, but in practice it is too fine-grained a relation to obtain proofs of realistic manipulation systems.

For this reason, we will generally employ the equivalence relationship of *weak simulation* (Section 3.1) when analyzing manipulation circuits. This relation compares the behavior of two distributed systems as dynamical systems. In the rest of this paper, we employ the notion of weak simulation, and, specifically, we overload $\cong$ to mean weak simulation, except where noted.

3.3 Permutation

We may consider two orthogonal models of permutation. In both models, the vertex and edge labels $\ell(v)$ and $\ell(e)$ never change. The first model is called *vertex permutation,* and is given in Definition 3.7. In this model, the vertices can move, and they drag the components and wires with them. That is, the vertices move (under permutation), and as they move, the edges follow.

Vertex permutation of a situated manipulation circuit corresponds to the choice of a different immersion with the same domain:

Definition 3.7 *Let $\mathbb{S} = (\mathcal{S}, \phi)$ be a situated manipulation circuit. A* permutation $\mathbb{S}^*$ of $\mathbb{S}$ *is a situated manipulation circuit $(\mathcal{S}, \phi^*)$ such that the domain $\phi^{-1}C$ of ϕ and the domain ${\phi^*}^{-1}C$ of ϕ^* are the same.*

For technical reasons, we also permit a permutation to change which vertex has the "output device" label. Note that the definition of permutation (Definition 3.7) also makes sense for partial immersions. However, see the appendix for a discussion of the semantics of permutation for unsituated sensor systems.

We can also consider an alternate model, called *edge permutation*, where the edge connectivity changes. An edge permutation can be modeled as follows. Consider a graph with vertices V and edges E. Start with any bijection $\sigma : V^2 \to V^2$. We call σ an *edge permutation*, since it induces the restriction map $\sigma_{|_E} : E \to \sigma(E)$ on the edge set E. An edge permutation says nothing about the immersion of a graph.

We can also compose the models. We define a *graph permutation* to be a vertex permutation followed by an edge permutation. In a graph permutation, the vertices and the edges move independently. That is, vertices may move, but in addition, the edge connectivity may change. To illustrate the different models, consider a manipulation circuit $\mathcal{U}$ with three vertices $\{ v_1, v_2, v_3 \}$ with labels $\ell(v_i) = B_i$ $(i = 1, 2, 3)$. $\mathcal{U}$ has one edge $e = (v_1, v_2)$ of bandwidth k that connects B_1 to B_2. So, the B_i are the components of the system, and e is a datapath. A *vertex* permutation $\mathcal{U}^*$ of $\mathcal{U}$ would move the vertices (and therefore the components) spatially, but in $\mathcal{U}^*$, e would still connect v_1 and v_2, (and therefore, B_1 and B_2). An *edge* permutation σ of $\mathcal{U}$ would change the edge connectivity. So, for example, an edge permutation $\sigma(\mathcal{U})$ could be a graph with one edge $\sigma(e) = (v_2, v_3)$, connecting v_2 to v_3 (and hence B_2 to B_3). But in $\sigma(\mathcal{U})$ no edge would connect v_1 and v_2. Finally, consider a *graph* permutation $\mathcal{U}^\star$ of $\mathcal{U}$. Suppose $\mathcal{U}^\star = \sigma(U^*)$, that is, $\mathcal{U}^\star$ is the vertex permutation $\mathcal{U}^*$ followed by the edge permutation σ above. $\mathcal{U}^\star$ has the same edge connectivity as $\sigma(\mathcal{U})$. However, in $\mathcal{U}^\star$, the vertices are immersed as in $\mathcal{U}^*$.

Let $(\mathcal{U}, \phi)$ be a situated manipulation circuit. A graph permutation of $\mathcal{U}$ is given by $\mathcal{U}^\star = (\mathcal{U}, \phi^\star)$ where $\phi^\star = (\phi^*, \sigma)$, ϕ^* is a vertex permutation, and σ is an edge permutation. In practice, we wish to impose some restrictions on edge and graph permutation. For example, suppose we have a sensor system $\mathcal{U}$ containing two cooperating and communicating mobile robots L and R. The sensori-computational systems for L and R are modeled as circuits. The datapaths in the system, in addition to bandwidth, have a *type*, of the form SOURCE→DESTINATION, where both SOURCE and DESTINATION $\in \{ L, R \}$. (Maintaining exactly two physical locations can be done using simple codesignation constraints). When permuting the edges of $\mathcal{U}$ to obtain $\mathcal{U}^\star$, it makes sense to permute only edges of the same type. More generally, we may segregate the edge types into two *classes*, *internal* edges $L \rightarrow L$ and $R \rightarrow R$, and *external* edges $L \rightarrow R$ and $L \leftarrow R$. In constructing $\mathcal{U}^\star$, we may reallocate an internal edge (of sufficient bandwidth) to connect any two components where SOURCE=DESTINATION. External edges (of sufficient bandwidth) can be (re)used when SOURCE≠DESTINATION. Hence, in *class* edge permutation, we permute edges within a type (or class). In this paper we will restrict our edge permutations to this kind of class edge permutation. Class edge permutation leaves unchanged the complexity bounds and the lemmas of [Don4].

To summarize: vertex permutation preserves the graph topology whereas edge permutation can move the edges around. Edge permutation permits arbitrary rewiring (using existing edges). It cannot add new edges, nor can it change their bandwidth. In Section 8, we discuss further the consequences of allowing different kinds of permutation on our model. There we show that graph permutation can be permitted at no additional "cost" and without changing the power of our systems very much.

3.4 Combination

Definition 3.8 *Consider two graphs $\mathcal{G} = (V, E)$ and $\mathcal{G}' = (V', E')$. We define the* combination $\mathcal{G} + \mathcal{G}'$ *of* $\mathcal{G}$ *and* $\mathcal{G}'$ *as follows:*

$$\mathcal{G} + \mathcal{G}' = (V \cup V', E \cup E').$$

We may define $+$ on manipulation circuits (Definition 3.2) by lifting the definition for graphs. We may define $+$ on two immersed graphs whenever the immersions are *compatible.* An immersion ϕ of $\mathcal{G}$ and an immersion ψ of $\mathcal{G}'$ are said to be *compatible* when the two immersions agree on the intersection $V \cap V'$ (for total immersions) or more generally, on $\phi^{-1}C \cap \psi^{-1}C$ (for partial functions). Similarly:

Definition 3.9 *Consider two graphs $\mathcal{G} = (V, E)$ and $\mathcal{G}' = (V', E')$, where $\mathcal{G}'$ is s subgraph of $\mathcal{G}$. We define the* difference $\mathcal{G} - \mathcal{G}'$ *of* $\mathcal{G}$ *and* $\mathcal{G}'$ *as follows:*

$$\mathcal{G} - \mathcal{G}' = (V - V', E - E').$$

Definition 3.9 may also be lifted to (partially) immersed graphs, and hence to situated manipulation circuits.

3.5 Reductions, Calibration, and Codesignation

As we observed in [Don], calibration exploits external state. We wish to order systems on how much information this external state (from calibration) yields, to obtain Definition 3.10, below. *Calibration complexity* is defined formally in [Don4]. Here is the basic idea. Calibration complexity measures how much information we add to a sensor system when we install and calibrate it. Installing a sensor system may require physically establishing some spatial relation between two components of the system. In this case we say the two components *codesignate* by the spatial relation. More generally, we may have to establish a relation between a component and a reference frame in the world. Most generally, when we compare two sensor systems S and

Q, we typically must install and calibrate them in some appropriate relative configuration—again, in a spatial relation. When all these relations are (in)equalities[9] of configuration, we say the system is *simple*. When all the relations are semi-algebraic (s.a.), we say the system is *algebraically codesignated.*

Definition 3.10 *We write $S \leq Q$ when*

1. *Q simulates S ($S \cong Q$),*

2. *S dominates Q in calibration complexity, and*

3. *The maximum bandwidth of S is at least that of Q (Definition 3.12).*

Convention: We will now drop the notation *, and use Q^* to denote any graph permutation of sensor system Q, as described above. The definitions above extend naturally to manipulation circuits to obtain:

Definition 3.11 *We write $S \leq^* Q$ if there exists some (graph) permutation Q^* of manipulation circuit Q such that $S \leq Q^*$.*

3.5.1 Relativized Information Complexity

Consider a sensor system with output z. The complexity of many sensors can essentially be characterized using the size $\log \Bbbk(z)$ of the *output* z. Let us now ask what is the "output" of our protocols, and, more important, what is its "size."

Suppose we suggest that the output of each system is at most 2 bits: the system's output chooses between three motor control states, the actions MOVE(L), MOVE(R), or PUSH(p). In this case, we note that the sensor system has internal bandwidth that is much higher ($\log \Bbbk(\Delta\theta)$ or $\log \Bbbk(\Delta x)$ bits). The output in some sense encodes that information. That is, we may view the protocols as "recognizing" a "model" or a "signal" of size $O(\log \Bbbk(\Delta x))$ bits, and subsequently "hashing" that model to one of three equivalence classes. This argues that perhaps the intrinsic output complexity of the protocols should be more like $\log \Bbbk(\Delta x)$ bits.[10]

[9] *I.e.*, of the form $X = Y$ or $X \neq Y$

[10] To see that instrumenting $\Delta\theta$ and Δx require the same number of bits, requires an argument like the "decalibration" lemmas of [Hors]. For this paper, we can see this from the relation $2r \tan\theta(t) = \Sigma(t)$.

Another idea is to observe that the actuator output p in PUSH(p) would be at a similar resolution to the orientation sensing $\theta(t)$ or odometry $x_i(t)$. This argues that the "output" of the protocol is something more like $O(\log \Bbbk(p))$ bits (since the MOVE(L/R) decision is indeed binary).

This discussion reveals a more general issue with sensor systems. In particular, there are sensor systems whose complexity cannot be well-characterized by the number of bits of output.[11] For example: consider a "grandmother" sensor. Such a sensor looks at a visual field and outputs one bit, returning **#t** if the visual field contains a grandmother and **#f** if it doesn't. Now, one view of the sensor interpretation problem is that of information reduction and identification (compare [DJ], which discusses hierarchies of sensor information). However consider a somewhat different perspective, that views sensors as *model matchers*. So, imagine a computational process that calculates the probability $P(G/V)$ of G (grandmother) given V (the visual field) — *i.e.*, the probability that G is in the data (the visual field itself). The sensor in the former case is something specific only to detecting grandmothers, while the latter prefers to see a grandmother as the model that best explains the current data. The latter is a process that computes over model classes. For example, this sensor might output TIGER (when given a fuzzy picture that is best explained as a tiger).

In short, one may view a sensor system as storing prior distributions. These distributions bias it toward a fixed set of model classes. In principle, the stored distributions may be viewed either as calibration or internal state. To quantify the absolute information complexity of a sensor system, we need to measure the information complexity of model classes stored in the prior distribution of the sensor. This could be very difficult.

Instead, we propose to measure a quantity called the *maximum bandwidth* of a sensor system. Intuitively, this quantity is the maximum over all internal and external edge bandwidths (data-paths). That is:

Definition 3.12 *We define the* internal *(resp.* external*)* bandwidth *of a sensor system $\mathcal{S}$ to be the greatest bandwidth of any internal (resp. external) edge in $\mathcal{S}$. The* output size *of $\mathcal{S}$ is given by Definition 3.3. We define the* maximum bandwidth *to be the greater of the internal bandwidth, external*

[11] This discussion devolves to a suggestion of Sundar Narasimhan [Personal communication], for which we are very grateful.

bandwidth, and the output size of $\mathcal{S}$. We call a sensor system monotonic *if its internal and external bandwidths are bounded above by its output size.*

The maximum bandwidth is an upper bound on the relative intrinsic output complexity (relativized to the information complexity of the components (Sections 3 and 3.5.1)). We explore this notion briefly below.

Maximum bandwidth is a measure of internal information complexity. The bandwidth is a measure of information complexity only *relative* to the sensori-computational components of the system. For example, imagine that we had a sensor system with a single component that outputs one bit when it recognizes a complicated model (say, a grandmother). The only data path in the system has bandwidth one bit, because the single component in the system is very powerful. So, even though the maximum bandwidth is small, the absolute information complexity may be large.

So, some sensors are black boxes. We call a sensor system a *black box* if it is encoded as a single component. The only measure of bandwidth we have for a black box is its output size. For example, the odometry system ODOM and the θ-source system θ in Section 4 are black boxes.

More generally, we call a sensor system *monotonic* if its internal bandwidth is bounded above by its output size. So, black box sensors are trivially monotonic. All the sensor systems in [Don4] are monotonic. If we believe that the output size of our protocols is $O(\log \Bbbk(p))$ bits, then our sensor systems are also monotonic. If we believe the output size is 2 bits, they are not. In any case, the bandwidths of Protocols I(QS) and II are $\log \Bbbk(\Delta x)$ and $\log \Bbbk(\Delta\theta)$ (resp.) Since $2r \tan\theta(t) = \Sigma(t)$, we argue that these two systems have the same maximum bandwidth.

The definitions and arguments in this section generalize *mutatis mutandis* from sensor systems to manipulation circuits.

3.5.2 Reductions using Communication

In light of this discussion, we now define the reduction $\leq_1$ from [Don4], using relativized information complexity. Recall the construction of $\textsc{comm}(\cdot)$ as a sensor system (Section 3.2). First, let S be a monotonic sensor system with output z. In this case, we define $\textsc{comm}(S)$ to be $\textsc{comm}(z)$.

More generally, for (possibly) non-monotonic sensors, we will let $\textsc{comm}(S)$ be $\textsc{comm}(2^k)$ where k is the *relative intrinsic output complexity* of S. Measuring this (k) in general is difficult, but we will treat the *maximum bandwidth* (Definition 3.12) of S as an upper bound on k.

Definition 3.13 *Consider two manipulation circuits S and Q. We say S is* efficiently reducible *to Q if*

$$S \leq^* Q + \text{COMM}(S). \tag{2}$$

In this case we write[12] $S \leq_1 Q$.

Now, permutation (the * operation) and combination (the $+$ operation) "commute" for compatible partial immersions. This is formalized as a "distributive" property in [Don4]. So, for example, for any manipulation circuit $\mathcal{S}$, we have ensured that $\mathcal{S}^* + \text{COMM}(\cdot) = (\mathcal{S} + \text{COMM}(\cdot))^*$, *i.e.*, we can do the permutation and combination in any order. Second we have ensured that the combination operation $+$ is commutative and associative. Third, in the reduction $\leq_1$ we have given the single edge e in $\text{COMM}(\cdot)$ enough bandwidth so that it still works when we switch it (e) around using permutation. Hence, the sensor system $(Q + \text{COMM}(S))^*$ from Equation (2) may be implemented as the sensor system Q permuted in an arbitrary way, plus one extra data path whose bandwidth is that of the largest flow in S.

Observe that even when $\leq^*$ is transitive, it appears that $\leq_1$ is not. To see this, suppose that $A \leq_1 B$ and $B \leq_1 C$. Then it appears that to reduce A to B we require one "extra wire" (namely, $\text{COMM}(A)$), and that to reduce B to C we could require (another) extra wire $\text{COMM}(B)$, and therefore, in the worst case, to reduce A to C we could require *two* extra wires. That is, it could be that A cannot reduce to C with fewer than two extra wires. We have yet to find a non-artificial example of this lower bound but it appears to indicate that $\leq_1$ is not transitive (even for simple sensor systems (Section 3.5)).

Let us summarize. The reduction $\leq_1$ corresponds to a specific circuit transformation. This transformation can be understood as follows. Let S be a monotonic sensor system with output b. Let Q be another sensor system. We imagine Q as a "circuit" embedded in (say) the plane. Let $\text{COMM}(S)$ be a "sensor system" with one datapath e, that has bandwidth $\log \Bbbk(b)$. Then, adding output communication to Q can be viewed as the following transformation on sensor systems: $Q \mapsto Q + \text{COMM}(S)$. The transformation

[12] $\leq_1$ is also called $<_s$ in [Don].

is parameterized by (the bandwidth of) S. The bounded-bandwidth datapath e can be spliced into Q anywhere. We note that this transformation can be composed with permutation (in either order):

$$\begin{array}{ccccc} Q & \mapsto & Q^* & \mapsto & Q^* + \text{COMM}(S) \\ \| & & & & \| \\ Q & \mapsto & Q + \text{COMM}(S) & \mapsto & (Q + \text{COMM}(S))^*. \end{array}$$

The reduction $\leq_1$ (Definition 3.13) is a "1-wire" reduction. It does not appear to be transitive. The reduction $\leq^*$ (Definition 3.11) is a *"0-wire"* reduction. It is transitive for simple sensor systems [Don4]. We could analogously define a 2-wire, or more generally, any *k-wire reduction* $\leq_k$ by modifying Equation (2) in Definition 3.13 to

$$S \leq^* Q + k \cdot \text{COMM}(S), \tag{2'}$$

where $k \cdot \text{COMM}(S)$ denotes $\overbrace{\text{COMM}(S) + \cdots + \text{COMM}(S)}^{k \text{ times}}$.

Since $(\leq^*) = (\leq_0)$, this suggests a hierarchy of reductions, indexed by k. The k-wire reductions $\{\leq_i\}_{i \in \mathbb{N}}$ form a *graded relation.* Even though we believe that $\leq_1$ is not transitive (in the elementary sense), the hierarchy has *graded transitivity* on simple sensor systems [Don4]. This means that for any simple sensor systems S, Q, and U, if $S \leq_i Q$ and $Q \leq_j U$, then $S \leq_{i+j} U$. This follows from a lemma that the 0-wire reduction $\leq_0$ (called $\leq^*$ in Definition 3.11) is elementary transitive for *simple* sensor systems.

Consider the hierarchy of k-wire reductions $\{\leq_i\}_{i \in \mathbb{N}}$. We say such a hierarchy *collapses* if it is isomorphic to an elementary relation. In particular, the hierarchy of k-wire reductions ($k > 0$) collapses if $\leq_1$ is elementary transitive [Don4].

4 Comparing Protocols Using Reductions

The results in Section 4 apply using both strong and weak simulation.

We now apply the ideas above to compare our protocols, P.I(QS) and P.II (the circuits in Figures 8–10). First, we define two black boxes (see sec. 3.5.1). Define the *odometry sensor system* ODOM to have one vertex, whose label is $\boxed{\text{ODOM}}$. It has a single component, an odometer. Similarly, define the *θ-source* sensor system θ to have a one vertex and a single component $\boxed{\theta(t)}$, which is a θ-source. These systems can be installed (or better, spliced) "into" our circuits. Each black box comes with (simple) codesignation constraints. Vertex $\boxed{\text{ODOM}}$ must be installed *on* a robot (either L or R). So, its vertex codesignates with L or R. Vertex $\boxed{\theta(t)}$ must be installed at a location *not* on a robot. So, its vertex cannot codesignate with L or R. We now show:

Claim 4.1 *Let* P.II*,* P.I(QS)*,* ODOM*, and θ be the sensor systems defined as above. Then,*

$$\text{P.II} \leq_k \text{P.I(QS)} - 2\text{ODOM} + \theta \tag{3}$$

for $k = 1$.

Proof: Consider the sensor system $\mathcal{U}$ obtained by removing both odometers from circuit P.I(QS), and adding a θ-source:

$$\mathcal{U} = \text{P.I(QS)} - 2\text{ODOM} + \theta. \tag{4}$$

Now, consider permutations of $\mathcal{U}$, and recall Definition 3.11. We first ask whether P.II can be reduced to $\mathcal{U}$ using $\leq^*$. That is, P.II $\leq^* \mathcal{U}$? First, we note that we can move around the registers and $\boxed{-}$'s from $\mathcal{U}$ to situate all the components of P.II. We also have some leftover components (and wires). P.II requires two sign boxes; however, the $\boxed{\text{SGN}}$ box just selects out the sign bit. To build the extra sign box we can just ignore the other bits, or we can use the leftover hardware from $\mathcal{U}$ to build a small circuit to simulate $\boxed{\text{SGN}}$. We need to argue that a register big enough to hold $x_i(0)$ will also hold θ_0; this follows from $2r \tan\theta(t) = \Sigma(t)$, or from "decalibration" [Hors]. Next, we see that we can permute the internal edges of $\mathcal{U}$ to wire up the components of P.II *in situ*—internally. What about externally?

Permuting the external wiring almost works, but not quite. $\mathcal{U}$ has two external data paths, (b′) and (b), with bandwidth 2 bits and $\log \Bbbk(\Delta x_1)$ bits (resp) (Figure 12). Now, since we only allow class edge permutation (as in Section 3.3), we must permute external edges to external edges and internal edges to internal edges. Therefore, in fig. 13, the edge (b) from $\mathcal{U}$ will suffice as a datapath from $\theta(t)$ to R, since it has adequate bandwidth. However, the datapath (b′) from $\mathcal{U}$ does not have adequate bandwidth to carry information from $\theta(t)$ to L. In order to build P.II from $\mathcal{U}$, we must add another external data path COMM$(\cdot)$ from $\theta(t)$ to L. Now, what is the argument to COMM$(\cdot)$?

This data path must have bandwidth of at least the relative intrinsic output complexity of P.II, or $\log \Bbbk(\Delta\theta)$ bits. Hence we may parameterize this new edge by writing $\textsc{comm}$(P.II), following Section 3.5.1. Hence, we see that

$$\text{P.II} \leq (\mathcal{U} + \textsc{comm}(\text{P.II}))^{*}. \tag{5}$$

Therefore by Definition 3.11,

$$\text{P.II} \leq^{*} \mathcal{U} + \textsc{comm}(\text{P.II}), \tag{6}$$

so using Definition 3.13, we have P.II $\leq_1 \mathcal{U}$. Hence we conclude

$$\text{P.II} \leq_1 \text{P.I(QS)} - 2\textsc{odom} + \theta, \tag{7}$$

which implies Equation (3) as desired. □

This formalizes our intuition that, by removing odometry but adding relative normal sensing, we can accomplish the pushing task without explicit communication. More precisely, we show how to build one circuit P.II "efficiently" out of the other ($\mathcal{U}$). To transform P.I(QS) into P.II, the operators $+$ and $-$ quantify what resources we add and delete. Relative information complexity allows us to measure the effective communication "through the world." The permutation quantifies the redistribution of resources.

5 Tradeoffs and Equivalences

The results in Section 5 hold for both strong and weak simulation.

In Section 4 we have shown how to compare distributed manipulation protocols by analyzing their resource consumption. In this section we describe how the equivalences established between two different protocols for the same task can be used to determine *resource tradeoffs* for the task.

Consider the reduction $\leq_0$, and suppose that $S \leq_0 Q$. If S reduces to Q then there exists a permutation Q^* of Q such that $S \leq Q^*$. Note that $S \leq Q$ implies that $S \leq_0 Q$. This is because the identity permutation (no change) of Q is a valid (if trivial) permutation of Q. Hence, if $S \leq Q$, we take $Q^* = Q$ to obtain $S \leq_0 Q$.

Let A and B be any manipulation circuits. In this section we will think of A and B as "resources"—for example, a primitive action or sensing operation.

Suppose that we know

$$S =_0 Q, \tag{8}$$

and we also have

$$S =_0 (Q - B + A). \tag{9}$$

Equation (9) is shorthand for $S \leq_0 (Q - B + A)$ and $Q \leq_0 S - A + B$.

Equations (8) and (9) together argue that $B - A = 0$, in other words, that B and A trade-off, or are equivalent. One caveat is that "$B = A$" is only valid for the task specified in the codesignation constraints [Don4] of Q and S. For example, we prove resource trade-offs for our two manipulation circuits; but we only claim these trade-offs for the pushing task. For some other task, resources A and B may not function equivalently[13].

This gives the following procedure to prove trade-offs for manipulation circuits X and Y and resources A and B. The key idea in the following is that the relation $=_0$ in equations (8) and (9) can be different—for example, $\equiv$, $=_m$, or $\leq_k$.[14] Even in these cases the resulting information invariant equations provide us with a means of formalizing and proving trade-offs: the catch is that the strength of the trade-off is determined by the strength of the relation.

To prove a trade-off between A and B in the information invariant framework:

1. First establish that XR_nY, where R_n is a relation in the set $\{\equiv, =_n, \leq_n\}$ and n is an integer. Note that the n is not needed for $\equiv$.

2. Then show that $XS_m(Y - A + B)$ where $S_m \in \{\equiv, =_m, \leq_m\}$, m is an integer and $m \geq n$.

3. The information invariants formalism then argues that A and B are equivalent or trade-off. More precisely, $(B - A)T0$, or BTA, where the "strength" of the relation T is defined by the strength of R_n and S_m. That is, $\equiv$ is stronger than $=$ which is stronger than $\leq$, and smaller subscripts are stronger than larger subscripts. This defines a partial order on the space of reduction types $(\{\equiv\} \cup (\{=, \leq\} \times \mathbb{Z}))^2$, and this partial order in turn defines a hierarchy of trade-offs.

[13]To prove trade-offs across a class of tasks, we must use a mechanism called "universal reduction" [Don4], which is beyond the scope of this paper.

[14]When $S \leq_k Q$ and $Q \leq_k S$ we write $S =_k Q$. When $S \leq Q$ and $Q \leq S$ we write $S \equiv Q$.

This argument only constitutes a proof in the case where systems X and Y are *minimal* systems (a system X is *minimal* for a task if there is no resource A that is a subset of that system such that the system $X - A$ can accomplish the task.) We can also obtain a proof in the case resources A and B are indispensible resources even when systems X and Y are not minimal. At present, we can only argue empirically that our protocols are near-minimal. Until a proof of the minimality is obtained, we challenge the reader to design and test manipulation protocols with fewer resources!

We can now apply this notion of tradeoffs to our protocols. The following theorem summarizes the results in terms of resources:

Theorem 5.1 *The following reductions and tradeoffs hold:*

$$\mathrm{P.II} \leq_1 \mathrm{P.I(QS)} - 2\textsc{odom} + \theta,$$

$$\mathrm{P.I(QS)} \leq_0 \mathrm{P.II} - \theta + 2\textsc{odom},$$

$$\mathrm{P.II} =_1 \mathrm{P.I(QS)} - 2\textsc{odom} + \theta.$$

Proof: We prove this theorem by arguing steps (1-3) on Page 41. Section 3.1 defines weak simulation as an equivalence relation on manipulation circuits. The protocols P.II and P.I(QS) solve the same manipulation task, and the configuration of the pushed object evolves identically under each protocol. Hence, circuits P.II and P.I(QS) are equivalent under weak simulation. In other words, $\mathrm{P.II} \equiv \mathrm{P.I(QS)}$ and therefore $\mathrm{P.II} =_0 \mathrm{P.I(QS)}$, mirroring Equation(8), above. This establishes step (1) on Page 41.

Now, by Claim 4.1, we know that

$$\mathrm{P.II} \leq_1 \mathrm{P.I(QS)} - 2\textsc{odom} + \theta. \tag{10}$$

A careful reading of the proof of Claim 4.1 reveals that

$$\mathrm{P.I(QS)} \leq_0 \mathrm{P.II} - \theta + 2\textsc{odom} \tag{11}$$

as well. To see this, observe that the datapath $\textsc{comm}(\cdot)$ (see Figure 13) has sufficient bandwidth to emulate the datapath $\textsc{comm}(s)$ labeled (b') in P.I(QS) (see Figure 12). Next, we observe that Equation(11) implies that

$$\mathrm{P.I(QS)} \leq_1 \mathrm{P.II} - \theta + 2\textsc{odom} \tag{12}$$

since $(\leq_0) \subset (\leq_1)$. Then, using the convention of Equation(9), we combine Equations(10) and(12) to obtain

$$\text{P.II} =_1 \text{P.I(QS)} - 2\textsc{odom} + \theta, \tag{13}$$

as desired.

Finally, using step (3) on Page 41, we argue that the resources θ and $2\textsc{odom}$ tradeoff with strength $(\equiv, =_1)$. □

We can obtain results similar to Theorem 5.1 for other pushing protocols. Starting with Figures 6, 7, it is possible to derive a circuit for Protocol I. We leave this construction as an exercise for the reader. Once this circuit is drawn and analyzed in the information invariant framework, it is possible to prove

$$\text{P.I} =_0 \text{P.I(QS)} - 2\textsc{odom} + 2\textsc{force},$$

where $\textsc{force}$ denotes a sensor system consisting of the force sensor used by protocol P.I.

6 Lower Bounds

We believe it is also possible to show a "lower bound" of the following form:

Conjecture 6.1 *Suppose we require strong simulation so that $\mathcal{S} \leq \mathcal{Q}$ only if $\mathcal{Q}$ strongly simulates $\mathcal{S}$, in the sense of Definition 3.6.*[15] *Then we conjecture that Equation (3) does not hold under strong simulation for $k = 0$.*

Proof (sketch): Conjecture 6.1 can be proven by showing that there is no graph permutation $\mathcal{U}^\star$ of $\mathcal{U}$ that simulates P.II. The proof of Claim 4.1 sketches such an argument for one particular class of permutations; the trick is to show it for all permutations.

As in Section 3.1, we will use the operator * for vertex permutations and * for graph permutations. A graph permutation $\mathcal{U}^\star$ can be specified by a pair (ϕ^*, σ) where ϕ^* is a vertex permutation and σ is an edge permutation. Now,

[15]Hence, $\mathcal{S} \leq_0 Q$ only if there exists a permutation $\mathcal{Q}^*$ such that $\mathcal{Q}^*$ strongly simulates $\mathcal{S}$. See the discussion in Section 3.1.

the codesignation constraints for Protocols P.I(QS) and P.II require that there be exactly two sites (L and R). Each component must be located at exactly one site. Now consider the resources in $\mathcal{U}$. Any vertex permutation of $\mathcal{U}$ is subject to the codesignation constraint that the image of the permutation must inject into the two point set $\{L, R\}$. Hence, any vertex permutation ϕ^* is a *partition* of $\mathcal{U}$'s resources such that some subset is located at L and the rest at R. More precisely:

Proposition 6.2 *Under the codesignation constraint* $\operatorname{im}\phi^* \subset \{L, R\}$*, the partitions of the resources of* $\mathcal{U}$ *and the vertex permutations of* $\mathcal{U}$ *are in one-to-one correspondence.* $\square$

For n components, the number of possible partitions is given by

$$N = \sum_{i=0}^{n} \binom{n}{i}. \tag{14}$$

Therefore, N (Equation (14)) bounds the number of possible vertex permutations. We also observe that the number of edges is at most $O(n^2)$, and hence the number of edge permutations is bounded by $O(n^4)$.

For each graph permutation $\phi^\star = (\phi^*, \sigma)$, we obtain a new manipulation circuit $\mathcal{U}^\star = (\mathcal{U}, \phi^\star)$ whose simulation function is $\Omega_{\mathcal{U}^\star}$. A proof would obtain by enumerating each graph permutation $\phi^\star$, and proving that

$$\Omega_{\mathcal{U}^\star} \neq \Omega_{\text{P.II}} \tag{15}$$

We have checked many reasonable permutations and shown that Equation (15) holds. However, we have not checked them all.

In principle, since there are only a finite number of permutations, they could all be enumerated. Since $\Omega_{\text{P.II}}$ and $\Omega_{\mathcal{U}^\star}$ are semi-algebraic, in principle there is a computer algorithm to prove the lower bound conjectured using the techniques of Section 7. It would be of great value to prove the lower bound, and also of interest to find a more efficient proof technique, since Nn^4 is large.

$\square$

Our conjecture only stands for strong simulation. It is not known whether Equation (3) holds for $k = 0$ under weak simulation. Proving this would require showing that for each graph-permuted system $\mathcal{U}^\star$, the resulting dynamical system has a different limit set and therefore could not emulate the dynamical behavior of Protocol P.II. I believe this is unlikely: it seems plausible that if all the resources of $\mathcal{U}$ were colocated on a single robot, then a reasonable pushing algorithm could be developed. This trick is ruled out by insisting our insistence on strong simulation. Strong simulation (Section 3.1) requires that the output of each manipulation circuit be identical in corresponding configurations. This is impossible to accomplish unless the actuators in $\mathcal{U}$ are distributed across L and R in the same manner as in Protocol P.II.

7 Computing Reductions

Claim 4.1 is a proof done "by hand." That is, we can in principle determine that Equation (3) holds (for $k > 0$) by showing—"by hand"—the existence of a suitable permutation. It is somewhat surprising that for strong simulation (but not for weak simulation) we can in fact automate this process: [Don4] gives algorithms for deciding the relation $\leq_1$. More precisely, given suitable encodings of two sensor systems $\mathcal{S}$ and $\mathcal{U}$, we can computationally decide whether $\mathcal{S} \leq_1 \mathcal{U}$ [Don4]. The algorithm is too complicated to describe here. We examine a special case to give a flavor for it; many details are omitted. The basic idea involves employing the theory of real closed fields with bounded quantification. Let us for a moment restrict our reductions to vertex permutation alone (Definition 3.7). First, suppose that $\mathcal{S}$ and $\mathcal{U}$ each have d vertices. Then an immersion of $\mathcal{S}$ can be encoded as a point in C^d. More generally, a partial immersion ϕ whose domain contains $l \leq d$ vertices can be modeled as a point in C^l. We can "guess" a (vertex) permutation ϕ^* of ϕ by existentially quantifying over the configurations of the l vertices inside ϕ's domain. Hence, the space of permutations of ϕ, denoted $\Sigma(\phi)$, is isomorphic to C^l. Similarly, we can verify a Tarski sentence for all extensions $\bar{\phi}$ of ϕ, by universal quantification over the $d - l$ vertices *outside* the domain of ϕ. Hence, the space of all extensions of ϕ, $\operatorname{ex}\phi$, is isomorphic to C^{d-l}. We will model (algebraic) codesignation constraints as a (possibly constant) semi-algebraic (s.a.) mapping $\bar{\phi} \mapsto D(\bar{\phi})$ taking an immersion $\bar{\phi}$ to a s.a. set $D(\bar{\phi}) \subset C^d$. All these methods generalize to graph permutation as well [Don4]. Now,

Definition 7.1 *A* simulation function $\Omega_{\mathcal{U}}$ for $\mathcal{U}$ *is a map* $\Omega_{\mathcal{U}} : C^d \to R$*, where* R *the space of outputs. We call the value* $\Omega_{\mathcal{U}}(\phi) \in R$ *of* $\Omega_{\mathcal{U}}$ *on a sensor configuration* ϕ *to be the* output value *or* sensor value *at* ϕ*.*

Definition 7.2 *We call a sensor system* $\mathcal{U}$ algebraic *if it is algebraically codesignated (Section 3.5), has an algebraic configuration space* C*, and a semi-algebraic algebraic simulation function* $\Omega_{\mathcal{U}}$*.*

How do we construct and permute simulation functions? Recall the discussion of simulation functions for components and connections above. To decide $\leq^*$ means to deciding whether or not $(\mathcal{S}, \phi) \leq^* (\mathcal{U}, \psi)$. Hence we must decide whether there exists a permutation ψ^* of ψ such that $(\mathcal{S}, \phi) \cong (\mathcal{U}, \psi^*)$. Computationally, this requires deciding the Tarski sentence[16]

$$\begin{gathered}(\exists \psi^* \in \Sigma(\psi),\ \forall \overline{\phi} \in \operatorname{ex} \phi,\ \forall \overline{\psi^*} \in D_{\mathcal{S}}(\overline{\phi}) \cap \operatorname{ex} \psi^*) : \\ \Omega_{\mathcal{S}}(\overline{\phi}) = \Omega_{\mathcal{U}}(\overline{\psi^*}).\end{gathered} \tag{16}$$

Here, $D_{\mathcal{S}}(\cdot)$ models the codesignation constraints; they require the choice of extension $\overline{\psi^*}$ by the inner quantifier to depend on the extension $\overline{\phi}$ selected by the middle quantifier. When comparing two sensor systems $\mathcal{S}$ and $\mathcal{U}$, we typically must install and calibrate them in some appropriate relative configuration—*i.e.*, in the spatial relation that the codesignation constraint $D_{\mathcal{S}}(\cdot)$ encodes.

If we can decide $\leq^*$, we can decide $\leq_1$. Here is why: to decide $\leq_1$, we must determine whether $(\mathcal{S}, \phi) \leq^* (\mathcal{U}, \psi) + \text{COMM}(\mathcal{S})$, (Definition 3.13). Recall the definition of *compatibility* for partial immersions (Section 3.4). We first observe that permutation (the * operation) and combination (the $+$ operation) "commute" for compatible partial immersions [Don4]. Our arguments above for guessing extensions and permutations can be generalized *mutatis mutandis* to compute the combination (Definition 3.8) of two algebraic sensor systems. Since $\text{COMM}(\mathcal{S})$ is a constant-sized sensor system (2 vertices, one edge) with only a constant number of codesignation constraints (at most 2), we may guess how to combine it with a permutation $(\mathcal{U}, \psi^*)$ of $(\mathcal{S}, \psi)$ within the same time bounds given below in lemma 7.3 and Equations (17–18).

[16] We must also be able to compute dominance in calibration complexity (see Definition 3.10). This can be done independently, and much faster than Equation (16) can be decided; see [Don4].

Let us suppose that $\mathcal{U}$ and $\mathcal{S}$ are algebraic. Let us define the *size* d of $\mathcal{U}$ to be the number of vertices in $\mathcal{U}$. Let the *simulation complexity* n_{Ω} be the size of the simulation functions $\Omega_{\mathcal{U}}$ and $\Omega_{\mathcal{S}}$. We let n_{D} measure the complexity of the codesignation constraints $D_{\mathcal{S}}(\cdot)$ in (16). Then, we can decide Equation (16) in the time bounds below:

Lemma 7.3 (Don4) *There is an algorithm for deciding the relations $\leq^*$ and $\leq_1$ for algebraic sensor systems. It runs in time polynomial in the simulation and codesignation complexity $(n_{\Omega} + n_{D})$, and sub-doubly exponential in the size of the sensor systems. That is, if the system has size d the time complexity is*

$$(n_{\Omega} + n_{D})^{(r_c d)^{O(1)}}, \tag{17}$$

where r_c is the dimension of the configuration space C for a single component. □

Let us call $\mathcal{U}$ small if n_{Ω} and n_{D} are only polynomially large in d, *i.e.*, $(n_{\Omega} + n_{D}) = d^{O(1)}$. Note that for "small" sensor systems, Equation (17) becomes

$$d^{(r_c d)^{O(1)}}. \tag{18}$$

Although complex, Equation (16) is simplified for presentation. The full Tarski sentence also contains codesignation constraints for the outer quantifiers, and is given in [Don4]. We must warn that in Section 7 we have examined a special case, where $\mathcal{S}$ and $\mathcal{U}$ are partially situated (that is, the domains of ϕ and ψ are non-empty). A powerful generalization is given in app. A, where the sensor systems can be *unsituated.*

Computing reductions in the case of weak simulation is also possible, if the limit set of a dynamical system and checking equality of limit sets can be computed. This is a difficult algorithmic problem, but it is possible in some cases. Work is underway in extending these methods.

8 Sensitivity of the Model

We wish to explain whether or not our theory has revealed some universal truth about sensori-computational information invariants, or whether the results are sensitive to the particular encodings (circuits) we chose to analyze. How sensitive is our model? We consider two ways to investigate this issue. First, we try changing our model of reduction (specifically, permutation) slightly, to see how that affects our results. Second, we ask, what if the input were reëncoded slightly differently? Would our results change a lot?

Specifically, we ask: how can we compare vertex permutation with graph permutation (Section 3.7)? In particular: (i) if we permit graph permutation instead of vertex permutation, does it change our complexity bounds? and (ii) does graph permutation give us a more powerful reduction than vertex permutation? Question (i) gives us some insight into the sensitivity of our complexity bounds to the model of reduction we use. Question (ii) sheds light on whether we can cheaply and cleverly reëncode a sensor system so as to gain a lot of "power" (information complexity).

[Don4] first derives the complexity bounds in lemma 7.3 and Equation (17–18) for vertex permutation. Next, [Don4] asks: how expensive it is to compute the reductions $\leq^*$ and $\leq_1$ using graph permutation? By extending the configuration space C^d to include all possible edge permutations, we obtain an extended configuration space of sufficiently low dimension that we still obtain the same complexity bounds given in lemma 7.3 and eqs. (17–18), (so long as r and s are constants—see Page 23) [Don4].

We now address question (ii): does graph permutation give us a more powerful reduction? We show:

Lemma 8.1 (The Clone Lemma) *Graph permutation can be simulated using vertex permutation, preceded by a linear time and linear space transformation of the sensor system.*

Proof: Given a sensor system $\mathcal{U}$ we "clone" all its vertices, and attach the edges to the clones. The cloned system simulates the original when each vertex is colocated with its clone. Components remain associated with original vertices. We can move an edge independently of the components it originally connected, by moving its vertices (which are clones). Any graph permutation of $\mathcal{U}$ can be simulated by a vertex permutation of the cloned system.

More specifically: Given a graph $G = (V, E)$ with labelling function ℓ, we construct a new graph $G' = (V', E')$ with labelling function ℓ'. Let the cloning function $\mathrm{cl} : V \hookrightarrow \mathbb{V}$ be an injective map from V into a universe of vertices $\mathbb{V}$, such that $\mathrm{cl}(V) \cap V = \emptyset$. We lift cl to V^2 and then restrict it to E

to obtain $\mathrm{cl} : E \to \mathrm{cl}(V)^2$ as follows: If $e = (u, v)$, then $\mathrm{cl}(e) = (\mathrm{cl}(u), \mathrm{cl}(v))$. Edge labels are defined as follows: $\ell'(cl(e)) = \ell(e)$.

Finally we define $V' = V \cup \mathrm{cl}(V)$ and $E' = \mathrm{cl}(E)$. We define the labelling function ℓ' on V' as follows. $\ell'(v) = \ell(v)$ when $v \in V$. Otherwise, $\ell'(v)$ returns the "identity" component, which can be simulated as the identity function.[17]

Suppose $\mathcal{U}$ has $d = |V|$ vertices and $|E|$ edges. This transformation adds only d vertices and can be computed in time and space $O(d + |E|)$. $\square$

Let us denote by $\mathrm{cl}(\mathcal{U})$ the linear-space clone transformation of $\mathcal{U}$ described in lemma 8.1. Now consider any k-wire reduction $\leq_k$ (Section 3.5.2). We see that:

Corollary 8.2 *Let $k \in \mathbb{N}$. Suppose that for two sensor systems $\mathcal{U}$ and $\mathcal{V}$, we have $\mathcal{V} \leq_k \mathcal{U}$ (using graph permutation). Then $\mathcal{V} \leq_k \mathrm{cl}(\mathcal{U})$ (using only vertex permutation).* $\square$

9 Experiments

Using information invariants, we have presented a formal *post hoc* analysis of two manipulation protocols for a pushing task. However, we are also using these techniques in our laboratory for synthesis, to develop new manipulation protocols and analyze their robustness and information content. We believe that our techniques are useful both for transforming protocols so as to remove assumptions and thereby increase their generality and robustness, and also to develop new protocols for complex manipulation tasks. We give examples of these application below, in Sections 9.1 and 9.2. It is our belief that our methods can give valuable insights into the information structure of the task.

[17] The proof can be strengthened as follows. Recall that two components can communicate without an (explicit) connection when they are spatially colocated. Therefore the proof goes through even if cloned vertices have no associated components, that is, $\ell'(v) = \emptyset$ for $v \notin V$. This version has the appeal of changing the encoding without adding additional physical resources.

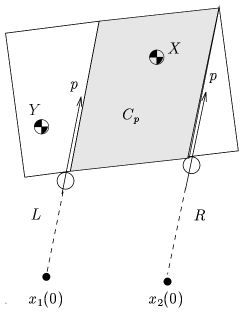

Figure 14: Two robots pushing a box. If their speeds are equal, we can infer that the COF is at Y, outside C_p. However, if the right robot is faster, it is possible that the COF lies between the pushing rays at X.

Left Robot	**Communication**	**Right Robot**
$x_0 \leftarrow$measure($x_1(0)$) $v \leftarrow v_{\text{init}}$		$x_0 \leftarrow$measure($x_2(0)$) $v \leftarrow v_{\text{init}}$
Repeat: measure(Δx_1)		Repeat: measure(Δx_2)
	$\boxed{L \rightarrow R}$ (Δx_1)	
		$\Sigma \leftarrow \Delta x_1 - \Delta x_2$ $s \leftarrow$ sign(Σ)
	$\boxed{L \leftarrow R}$ (s)	
$v \leftarrow v - s\Delta v$		$v \leftarrow v + s\Delta v$

Figure 15: A simplified version of protocol P.V. P.V performs velocity synchronization, using explicit local communication (it does not perform the pushing task!) The velocity of each robot is incremented or decremented by a fixed amount Δv, depending on which on which robot gets ahead. This simplified version assumes that $\Sigma_0 = 0$.

9.1 Removing Assumptions

The protocols we described depend critically on the assumption of velocity synchronization. For example, let v_1 and v_2 be the speeds of the robots. Let C_p be the region between the pushing rays p. Consider the situation shown in Figure 14. One explanation is the the COF is at location Y outside of C_p, and $v_1 = v_2$. But a second explanation is that the COF is at X, inside C_p, and $v_2 > v_1$. Velocity synchronization ($v_1 = v_2$) is key to ensuring that we can infer whether or not the COF is in C_p.

We have used methods similar to those in Sections 2-7 to develop protocols that do not require synchronization. Removing explicit synchronization from manipulation protocols is analogous to removing explicit communication. The ideal protocol we started with assumed global coordination and control—*i.e.*, global velocity synchronization. Next, we developed a velocity synchronization protocol P.V with explicit local communication. Given our analysis above, it is very simple to develop such a protocol, and we give the basic idea in Figure 15. Next, we "composed" it with protocol I(QS) above–this corresponds to "splicing" the circuits for P.V and P.I(QS) together. This is slightly more difficult, but using the techniques outlined in this paper, it it not too hard to do a careful analysis and get all the special cases right. Finally, we removed all explicit communication; again the robots communicate "through the world" (through the task dynamics). We leave it as an exercise for the reader to develop the resulting, communication-free protocol for box pushing without explicit communication or synchronization.[18] It was actually somewhat surprising to us that a uniform asynchronous protocol with no explicit communication can be developed for this task.

9.2 Reorienting Large Objects

Pushing Task	Reorientation Task	
Protocol O	$\triangle_1$	global coordination and control
Protocol I(QS)	$\triangle_2$	local IR comm, partial synchrony, MPMD
Protocol II	$\triangle_3$	uniform, SPMD, asynchronous

Figure 16: Summary of parallel manipulation protocols.

We would like to show that our methods could also be useful in engineering new protocols for difficult, multi-robot manipulation tasks. In this direction, we have also considered three multi-mobot manipulation protocols for box reorientation (see Figure 3 and [DJR1-2,RDJ]). We denote them by $\triangle_1$,$\triangle_2$, etc. For these protocols, we started with the offline algorithm $\triangle_1$ of [Rus], which was designed for multi-fingered robot hands with global coordination. Next, we developed a protocol $\triangle_2$ for three cooperating mobile robots with local IR communication. In this protocol, only one robot moves at a time, so, although the task has been "parallelized", it is not "load-balanced." Finally, $\triangle_3$ we removed all explicit communication between agents, and allowed the robots to perform simultaneous, asynchronous manipulation of the box. Protocol $\triangle_3$ has several advantages over protocol $\triangle_2$. Using protocol $\triangle_3$, two robots (instead of three) suffice to rotate the box. The protocol is "uniform" (SPMD) in that the same program (including the same termination predicate) runs on both robots. More interesting, in $\triangle_3$ it is no longer necessary for the robots to have an *a priori* geometric model of the box— whereas such a model is required for $\triangle_1$ and $\triangle_2$. Of course, various assumptions must hold for the task to succeed—the point

[18]Hint: the easiest way to compose the control loops, is to first add two bits of explicit communication. A one-bit $L \rightarrow R$ datapath tells R when L has broken contact. A symmetric $L \leftarrow R$ datapath tells L when R breaks contact. The need for this communication falls out of a synthesis similar to the one we presented. However, a careful analysis then shows that even these two bits of (explicit) communication can be removed.

of our analyses is to reveal these assumptions. We are currently completing such an analysis.[19]

In terms of program development, synchrony, and communication, we have the following approximate correspondence between these protocols:

We believe that a methodology for developing coordinated manipulation protocols is emerging, based on the tools described in this paper, [DJR1-2,RDJ], and [Don4]:

Developing Parallel Manipulation Protocols

1. Start with a sensorless [EM, EMV] or near-sensorless [Erd4, JR] manipulation protocol requiring global coordination of several "agents" (*e.g.*, fingers [Gol, Rus], or "fences" [PS]). Examples: $\triangle_1$ above [Rus] or protocol O (Figure 5).

2. Distribute the protocol over spatially separated agents. Synchronize and coordinate control using explicit local communication. Examples: Protocol I(QS), or $\triangle_2$ above.

3. Define a *virtual sensor*[20] that measures the key signal we wish to servo on. Example: Σ (or better, $s = \text{sign}(\Sigma - \Sigma_0)$) in P.I(QS) (Figure 8).

4. Find a way to implement this virtual sensor using concrete sensors on mechanical observables. Example: $n(t)$ in P.II (Figure 11).

5. Transform the communication between two agents L and R into shared data structures.

6. Implement the shared data structures as "mechanical registers." Example: *$\theta(t)$ is a "register" that L and R share.*

We believe that our methods are useful for developing parallel manipulation protocols. We think that information invariants can serve as a framework in which to measure the capabilities of robot systems, to quantify their power, and to reduce their fragility with respect to assumptions that are engineered

[19] In particular, we have considerably improved the protocol $\triangle_3$ from [DJR1]. See [RDJ].

[20] We use the term in the sense of [DJ]; others, particularly Henderson have used similar concepts.

Figure 17: TOMMY and LILY reorient a couch using an asynchronous SPMD protocol requiring no explicit communication.

into the control system or the environment. We believe that the equivalences that can be derived between communication, internal state, external state, computation, and sensors, can prove valuable in determining what information is required to solve a task, and how to direct a robot's actions to acquire that information to solve it.

10 Discussion

Our work raises a number of questions. For example, can robots "externalize," or record state in the world? The answer depends not only on the environment, but also upon the dynamics. A juggling robot probably cannot.

On a conveyor belt, it may be possible (suppose "bad" parts are reoriented so that they may be removed later). However, it is certainly possible during quasi-static manipulation by a single agent. In moving towards multi-agent tasks and at least partially dynamic tasks, we are attempting to investigate this question in both an experimental and theoretical setting.

Consider the relation $=_k$ defined in Section 5. $=_k$ is only an equivalence relation for $k = 0$, althought $=_k$ does obey *graded* transitivity [Don4]. By analogy with CT reductions, we may ask, does a given class of sensori-computational systems contain "complete" circuits, to which any member of the class may be reduced? Note that any two complete circuits are "graded equivalent" under $=_k$.

Finally, can we record "programs" in the world in the same way we may externalize state? Is there a "universal" manipulation circuit which can read these programs and perform the correct strategy to accomplish a task? Such a mechanism might lead to a robot which could infer the correct manipulation action by performing sensori-motor experiments.

Appendix

A What is Permutation?

In Section 7 we examined a special case, where $\mathcal{S}$ and $\mathcal{U}$ are partially situated (that is, the domains of ϕ and ψ are non-empty). A powerful generalization is given in [Don4], where the sensor systems can be *unsituated*. Using the ideas in Section 7, we can give an "abstract" version of permutation that is applicable to partially immersed sensor systems with codesignation constraints. Each set of codesignation constraints defines a different arrangement in the space of all immersions. Each cell in the arrangement, in turn, corresponds to a region in C^d.

Permutation corresponds to selecting a different family of immersions, while respecting the codesignation constraints. Since this corresponds to choosing a different region of C^d, the picture of abstract permutation is really not that different from the computational model of situated permutations discussed in Section 7. Suppose a simple sensor system $\mathcal{U}$ has d vertices, two of which are u and v. When there is a codesignation constraint for u

and v, we write that the relation $u \sim v$ must hold. This relation induces a quotient structure on C^d, and the corresponding quotient map $\pi : C^d \to C^d/(u \sim v)$ "identifies" the two vertices u and v. Similarly, we can model a non-codesignation constraint as a "diagonal" $\Delta \subset C^d$ that must be avoided. Abstract permutation of $\mathcal{U}$ can be viewed as follows. Let $D_{\mathcal{U}} = (C^d - \Delta)/(u \sim v)$. $D_{\mathcal{U}}$ is the quotient of $(C^d - \Delta)$ under π . For a partial immersion ψ^* to be chosen compatibly with the codesignation constraints, we view permutation as a bijective self-map of the disjoint equivalence classes

$$\{ \pi(\operatorname{ex} \psi^* - \Delta) \}_{\psi^* \in \Sigma(\psi)}. \tag{19}$$

Thus, in general, the group structure for the permutation must respect the quotient structure for codesignation; correspondingly, we call such permutations *valid*. Below, we define the "diagonal" Δ, precisely.

Now, an unsituated sensor system $\mathcal{U}$ could be modeled using a partial immersion ψ_0 with an empty domain. In this case $\operatorname{ex} \psi_0 = C^d$ and Equation (19) specializes to the single equivalence class $\{ D_{\mathcal{U}} \}$. In this "singular" case, we can take several different approaches to defining unsituated permutation. (i) We may define that $\psi_0^* = \psi_0$. Although consistent with situated permutation, (i) is not very useful. We choose a different definition. For unsituated permutation, we redefine $\Sigma(\psi_0)$ and $\operatorname{ex} \psi_0$ in the special case where ψ_0 has an empty domain. (ii) When $\mathcal{U}$ is simple, we may define $\Sigma(\psi_0)$ to be the set of colocations of vertices of $\mathcal{U}$. That is, let $(x_1, \ldots, x_d)$ be a point in C^d, and define the ij^{th} diagonal $\Delta_{ij} = \{ (x_1, \ldots, x_d) \mid x_i = x_j \}$. Define permutation as a bijective self-map of the cells in the arrangement generated by all $\binom{d}{2}$ such diagonals $\{ \Delta_{ij} \}_{i,j=1,\ldots,d}$. So, $\Sigma(\psi_0)$ is an arrangement in C^d of complexity $O(d^{2dr_c})$, $\operatorname{ex} \psi_0^* \in \Sigma(\psi_0)$ is a cell in the arrangement, and $\psi_0^* \in \operatorname{ex} \psi_0^*$ is a witness point in that cell. Hence ψ_0^* is a representative of the equivalence class $\operatorname{ex} \psi_0^*$. As in situated permutation, unsituated permutation can be viewed as a self-map of the cells $\{ \operatorname{ex} \psi_0^* \}$ or (equivalently) as a self-map of the witnesses $\{ \psi_0^* \}$. Perhaps the cleanest way to model our main examples (Section 4) is to treat all the sensor systems as initially unsituated, yet respecting all the (non)codesignation constraints. This may be done by (1) "algebraically" specifying all the codesignation constraints, (2) letting the domain of each immersion be empty, (3) using (ii) above, choose unsituated permutations that respect the codesignation constraints. The methods of Section 7 can be extended to guess unsituated permutations.

In our examples (Section 4), each guess (*i.e.*, each unsituated permutation) corresponds to a choice of which vertices to colocate.[21]

A.1 Example

Unsituated permutation is quite powerful. Consider deciding Equation (16) (in this example, we only consider vertex permutation of simple sensor systems). In particular, we want to see that (16) makes sense for unsituated permutation, when we replace ψ by ψ_0, ϕ by ϕ_0, etc., to obtain:

$$\begin{array}{c}(\exists \psi_0^* \in \Sigma(\psi_0),\ \forall \overline{\phi_0} \in \operatorname{ex} \phi_0,\ \forall \overline{\psi_0^*} \in D_{\mathcal{S}}(\overline{\phi_0}) \cap \operatorname{ex} \psi_0^*): \\ \Omega_{\mathcal{S}}(\overline{\phi_0}) = \Omega_{\mathcal{U}}(\overline{\psi_0^*}).\end{array} \tag{16$'$}$$

With situated permutation (16), we are restricted to first choosing the partial immersion ϕ, and thereby fixing a number of vertices of $\mathcal{S}$. Next, we can permute $\mathcal{U}$ to be "near" these vertices (this corresponds to the choice of ψ^*). This process gets the colocations right, but at the cost of generality; we would know that for any "topologically equivalent" choice of ϕ, we can choose a permutation ψ^* such that (16) holds. For simple sensor systems, "topologically equivalent" means, "with the same vertex colocations."

Unsituated permutation (16$'$) allows us to do precisely what we want. In place of a partial immersion ϕ for $\mathcal{S}$, we begin with a witness point $\phi_0 \in C^d$. ϕ_0 represents an equivalence class $\operatorname{ex} \phi_0$ of immersions, all of which colocate the same vertices as ϕ_0. So, ϕ_0 says *which* vertices should be colocated, but not *where.* Now, given ϕ_0, the outer existential quantifier in (16$'$) chooses an unsituated permutation ψ_0^* of $\mathcal{U}$. ψ_0^* represents an equivalence class $\operatorname{ex} \psi_0^*$ of immersions of $\mathcal{U}$, all of which colocate the same vertices of $\mathcal{U}$ as ψ_0^* does. The other, disjoint equivalence classes, are also subsets of C^d; each equivalence class colocates different vertices of $\mathcal{U}$, and the set of all such classes is $\Sigma(\psi_0)$ $(= \Sigma(\psi_0^*))$. Choice of ψ_0^* selects which vertices of $\mathcal{U}$ to colocate. The codesignation constraint $D_{\mathcal{S}}(\cdot)$ then enforces that, when measuring the outputs of $\mathcal{S}$ and $\mathcal{U}$, we install them in the same "place." More specifically: ϕ_0 (given as data) determines which vertices of $\mathcal{S}$ to colocate; choice of ψ_0^* determines which vertices of $\mathcal{U}$ are colocated; construction of $D_{\mathcal{S}}(\cdot)$ determines which

[21]The codesignation relation $u \sim v$, the quotient map π, the non-codesignation relation Δ, and definition (ii) of unsituated permutation, can all be extended to algebraic sensor systems using the methods of Section 7.

vertices of $\mathcal{U}$ and $\mathcal{S}$ are colocated. Most specifically, given the configuration $\overline{\phi_0}$ of $\mathcal{S}$, $D_{\mathcal{S}}$ in turn defines a region $D_{\mathcal{S}}(\overline{\phi_0})$ in the configuration space C^d of $\mathcal{U}$. This region constraints the necessary coplacements $\overline{\psi_0^*}$ of $\mathcal{U}$ relative to $(\mathcal{S}, \overline{\phi_0})$.

Perhaps surprisingly, allowing unsituated permutation does not change the complexity bounds of Section 7 [Don4].

Bibliography

[BK] Blum, M. and Kozen, D. *On the power of the compass (or, why mazes are easier to search than graphs),* Proc. 19^{th} Symp. Found. Computer Science, Ann Arbor, MI, pp. 132-42 (1978).

[Bri] Briggs, Amy. *An Efficient Algorithm for One-Step Compliant Motion Planning with Uncertainty*, *Algorithmica*, **8**, (2), 1992. pp. 195-208.

[Can] John Canny *On computability of fine motion plans,* IEEE ICRA, Scottsdale, AZ, (1989)

[CR] Canny, J., and J. Reif, "New Lower Bound Techniques for Robot Motion Planning Problems", *FOCS* (1987).

[Don] Donald, B. R. *Information Invariants in Robotics, Parts I and II,* IEEE International Conference on Robotics and Automation, Atlanta, GA. (1993).

[Don1] Donald, B. R. *Robot Motion Planning,* IEEE Trans. on Robotics and Automation, **(8)**, No. 2. (1992).

[Don2] Donald, B. R. *The Complexity of Planar Compliant Motion Planning with Uncertainty,* Algorithmica, **5** (3), pp. 353-382 (1990).

[Don3] Donald, B. R. *Error Detection and Recovery in Robotics,* Lecture Notes in Computer Science, Vol. 336, Springer-Verlag, New York (1989).

[Don4] Donald, B. R. *Information Invariants in Robotics,* , (1993). *Artificial Intelligence,* 72:217-304, 1995.

[DJ] Donald, B. R. and J. Jennings *Constructive Recognizability for Task-Directed Robot Programming,* Jour. Robotics and Autonomous Systems, **(9),** No. 1, Elsevier/North-Holland pp. 41-74. (1992).

[DJR1] **Donald, B. R., J. Jennings, and D. Rus** *Experimental Information Invariants for Cooperating Autonomous Mobile Robots,* International Joint Conference on Artificial Intelligence (IJCAI) Workshop on Dynamically Interacting Robots. Chambery, France (Aug 28) (1993).

[DJR2] **Donald, B. R., J. Jennings, and D. Rus** *Towards a Theory of Information Invariants for Cooperating Autonomous Mobile Robots,* International Symposium on Robotics Research (ISRR). Hidden Valley, PA (October 2, 1993). (1993).

[Erd1] **Erdmann, M.** *Using Backprojections for Fine Motion Planning with Uncertainty,* IJRR Vol. 5 no. 1 (1986).

[Erd2] **Erdmann, M.** *On Probabilistic Strategies for Robot Tasks,* Ph.D. thesis, MIT Department of EECS, MIT A.I. Lab, Cambridge MIT-AI-TR 1155 (1989).

[Erd3] **Erdmann, M.** *Action Subservient Sensing and Design,* IEEE ICRA, Atlanta. See also the Carnegie-Mellon report CMU-CS-92-116. (1993).

[Erd4] **Erdmann, M.** *Randomization for Robot Tasks: Using Dynamic Programming in the Space of Knowledge States,* Algorithmica Vol. 10, Nos. 2/3/4, Aug/Sept/Oct. pp. 248-291 (1993).

[EM] **Erdmann, M., and M. Mason**, "An Exploration of Sensorless Manipulation", *IEEE International Conference on Robotics and Automation*, San Francisco, April, 1986.

[EMV] **Erdmann, M., M. Mason, and G. Vaněček** *Mechanical Parts Orienting: The Case of a Polyhedron on a Table,* Algorithmica Vol. 10, Nos. 2/3/4, Aug/Sept/Oct. pp. 266-247 (1993).

[Gol] **Goldberg, K. Y** *Orienting Parts without Sensors,* Algorithmica Vol. 10, Nos. 2/3/4, Aug/Sept/Oct. pp. 201-225 (1993).

[HSS] **Hopcroft, J. E., Schwartz, J. T., and Sharir, M.** 1984 On the Complexity of Motion Planning for Multiple Independent Objects; *PSPACE*-Hardness of the "Warehouseman's Problem." *International Journal of Robotics Research.* **3**(4):76–88.

[Hors] **Horswill, I.** *Analysis of Adaptation and Environment,* Submitted to *Artificial Intelligence* (1992).

[JR] Jennings, J. and Rus, D. *Active Model Acquisition for Near-Sensorless Manipulation with Mobile Robots,* The IASTED International Conference on Robotics and Manufacturing, Oxford, UK (1993).

[Koz] Kozen, D. *Automata and Planar Graphs,* Fundamentals of Computing Theory, Proc. Conference on Algebraic, Arithmetic, and categorical methods in Computation Theory (Berlin) ed. L. Budach, Akademie Verlag (1979).

[Lat] Latombe, J.-C. *Robot Motion Planning,* Kluwer: New York (1991).

[LMT] Lozano-Pérez, T., Mason, M. T., and Taylor, R. H. *Automatic Synthesis of Fine-Motion Strategies for Robots,* Int. J. of Robotics Research, Vol 3, no. 1 (1984).

[Mas] Mason, M. T. 1986. Mechanics and Planning of Manipulator Pushing Operations. *International Journal of Robotics Research* **5**(3).

[ML] Mason, M. and Lynch, K. *Dynamic Manipulation,* Proc. of the 1993 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Yokohama, Japan, July 26-30 (1993).

[Nat] Natarajan, B. K. *On Planning Assemblies,* Proc. of the 4th Annual Symposium on Computational Geometry, Urbana, Illinois, June. pp. 299-308. (1988).

[PS] Peshkin, M. *Planning Robotic Manipulation Strategies for Sliding Objects,* Ph.D. dissertation, Department of Physics, Carnegie-Mellon University (1986).

[RD] Rees, J. and B. R. Donald *Program Mobile Robots in Scheme,* Proc. IEEE International Conference on Robotics and Automation, Nice, France (1992).

[Reif] Reif J., "Complexity of the Mover's Problem and Generalizations," Proc. 20th IEEE Symp. FOCS, (1979).

[Ros] Rosenschein, S.J. *Synthesizing Information-Tracking Automata from Environment Descriptions,* Teleos Research TR No. 2 (1989).

[Rus] Rus, D. "Fine Motion Planning for Dexterous Manipulation", PhD. Thesis available as CU-CS-TR 92-1323 (August) from Comp. Sci. Dept., Cornell University, 1992.

[RDJ] D. Rus, Donald, B. R., and J. Jennings *International Symposium on Intelligent Robot Systems (IROS),* , (Moving Furniture with Teams of Mobile Robots). Pittsburgh, PA (August 1995).

Acknowledgment and Historical Note. Many key ideas in this paper arose in discussions with Tomás Lozano-Pérez, Mike Erdmann, Matt Mason, and Ronitt Rubinfeld. Tomás suggested studying the two-finger pushing task in 1987, and laid out a framework for analysis. Many of the ideas in this paper develop suggestions of Mike Erdmann; in particular, he proposed the notion of calibration complexity. Any perspicuous reader will notice our indebtedness to Mike's *weltanschauung*, and to [Erd3]. The robots and experimental devices described herein were built in our lab by Jim Jennings, Russell Brown, Jonathan Rees, Craig Becker, Mark Battisti, and Kevin Newman; these ideas could never have come to light without their help. Thanks to Loretta Pompilio for drawing the illustration in figure 1. Debbie Lee Smith, Amy Briggs, and Karl-Freidrich Böhringer made suggestions and helped draw the other figures, and we are very grateful to them for their help.