ELSEVIER

Artificial Intelligence 72 (1995) 217–304

Artificial Intelligence

On information invariants in robotics ★

Bruce Randall Donald *

Computer Science Department, Cornell University, 4130 Upson Hall, Ithaca, NY 14853-7501, USA

Received September 1992; revised April 1994

Abstract

We consider the problem of determining the information requirements to perform robot tasks, using the concept of *information invariants*. This paper represents our attempt to characterize a family of complicated and subtle issues concerned with measuring robot task complexity. We also provide a first approximation to a purely operational theory that addresses a narrow but interesting special case.

We discuss several measures for the information complexity of a 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? We consider how one might develop a kind of "calculus" on (a)–(e) in order to compare the power of sensor systems analytically. To this end, we attempt to develop a notion of information invariants. We develop a theory whereby one sensor can be "reduced" to another (much in the spirit of computation-theoretic reductions), by adding, deleting, and reallocating (a)–(e) among collaborating autonomous agents.

★ 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, IRI-9201699, and by a Presidential Young Investigator award to Bruce Donald, and in part by the Air Force Office of Sponsored Research, the Mathematical Sciences Institute, Intel Corporation, and AT&T Bell laboratories.

* E-mail: brd@cs.cornell.edu.

SSDI 0004-3702(94)00024-U

Part I—State, communication, and side-effects

1. Introduction

In this paper we investigate the information requirements for robot tasks. Our work takes as its inspiration the *information invariants* that Erdmann [1] introduced to the robotics community in 1989 [24], although rigorous examples of information invariants can be found in the theoretical literature from as far back as 1978 (see, for example, [1, 35]).

Part I of this paper develops the basic concepts and tools behind information invariants in plain language. Therein, we develop a number of motivating examples. In Part II, we provide a fairly detailed analysis. In particular, we admit more sophisticated models of sensors and computation. This analysis will call for some machinery whose complexity is best deferred until that time.

A central theme to previous work (see the survey article [11] 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 (e.g., 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

[1] Erdmann introduced the notion of measuring task complexity in *bit-seconds*; the example is important but somewhat complicated; the interested reader is referred to [24].

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 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 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 Part I of 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. We also consider formal models of online robots, foregrounding the *situated automata* of [1]. The examples in Part I link our work to the recent but intense interest in online paradigms for situated autonomous agents. In particular, 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 attempt to fill this gap in Part II, where we provide a theory of *situated sensor systems*. We argue that 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 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 a measure of the intrinsic information requirements of a task, we are inspired by Erdmann's monograph on sensor design [25]. Also, we note that many interesting lower bounds (in the complexity-theoretic sense) have been obtained for motion planning questions (see, e.g., [9,30,43,45]; for upper bounds see, e.g., [4,

6, 12]). Rosenschein has developed a theory of synthetic automata which explore the world and build data structures that are "faithful" to it [46]. 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 inspired by the theoretical attack on perceptual equivalence begun by Donald and Jennings [14] and by the experimental studies of Jennings and Rus [33]. Horswill [32] 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". 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., [11] or [13].

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. This paper is intended to raise issues concerning information invariants, survey some relevant literature and tools, and take a first stab at a theory. Part I of this paper (Sections 1–3) provides some practical and theoretical motivations for our approach. In part II (Sections 4–9) we describe one particular and very operational theory. This theory contains a notion of *sensor equivalence*, together with a notion of *reductions* that may be performed between sensors. Part II contains an example which is intended to illustrate the potential of a such a theory. We make an analogy between our "reductions" and the reductions used in complexity theory. Readers interested especially in the four questions above will find a discussion of "installation complexity" and the role of calibration in comparing sensors in Section 5 below. Section 8 discusses the semantics of sensor systems precisely; as such this section is mathematically formal, and contains a number of claims and lemmata. This formalism is used to explore some properties of what we call *situated sensor systems.* We also examine the semantics of our "reductions". The results of Section 8 are then used in Section 9 to derive algebraic algorithms for reducing one sensor to another.

1.1. Research contributions and applications

Robot builders make claims about robot performance and resource consumption. In general, it is hard to verify these claims and compare the systems. I really think that the key issue is that two robot programs (or sensor systems) for similar (or even identical) tasks may look very different. Part I of this paper attempts to demonstrate how very different systems can accomplish similar tasks. We also discuss why it is hard to compare the "power" of such systems. The examples in Part I are distinguished in that they permit relatively crisp analytical comparisons. We present these examples so as to demonstrate the standard of crispness to which we aspire: these are the kinds of theorems about information tradeoffs that we believe can be proved for sensorimotor systems. The analyses in Part I are illuminating but ad hoc. In Part II, we present our theory, which represents a systematic attempt to make such comparisons based on geometric and physical reasoning. Finally, we try to operationalize our analysis by making it computational; we give effective (albeit theoretical) procedures for computing our comparisons. Our algorithms are exact and combinatorially precise.

We wish to rigorously compare embedded sensori-computational systems. To do so, we define a "reduction" $\leqslant_1$ that attempts to quantify when we can "efficiently" build one sensor system out of another (that is, build one sensor using the components of another). Hence, we write $A \leqslant_1 B$ when we can build system A out of system B without "adding too much stuff". The last phrase 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 $A \leqslant_1 B$ 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 (data-paths) 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, they must be reconnected using "wires", or data-paths. If we separate previously colocated resources, we will usually need to add a communication mechanism to connect the now spatially separate components. Like CT reductions, $A \leqslant_1 B$ 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.)[2] 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 our transformations can be quantified.

We foresee the following potential for application of these ideas:

(1) (*Comparison*). Given two sensori-computational systems A and B, we can ask "which is more powerful?" (in the sense of $A \leqslant_1 B$, 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 specification of A? The algorithms also

[2] 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.

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 specification of A be implemented using the bag of parts B?"

(4) (*Universal reduction*). Consider application 3, above. Suppose that in addition to the specification for A, we are given an encoding of A as a bag of parts, and an "embedding" to implement that specification. Suppose further that $A \leqslant_1 B$. 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?"

Our paper represents a first stab at these problems, and there are a number of issues that our formalism does not currently consider. We discuss and acknowledge these issues in Section 12.1.

2. Examples

2.1. A following task

2.1.1. A method of inquiry

To introduce our ideas we consider a task involving two autonomous mobile robots. One robot must follow the other. Now, many issues related to information invariants can be investigated in the setting of a single agent. We wish, however, to relate our discussion to the results of Blum and Kozen (in Section 2.2 below), who consider multiple agents. Second, one of our ideas is that, by spatially distributing resources 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. In robotics, the evidence for this is, so far, largely anecdotal. In computer science, often one learns a lot about the structure of an algorithmic problem by parallelizing it; we would eventually like to argue that a similar methodology is useful in robotics.

Here is a simple preview of how we will proceed. We first note that it is possible to write a servo loop by which a mobile robot can track (follow) a nearby moving object, using sonar sensing for range calculations, and servoing so as to maintain a constant nominal following distance. A robot running this program will follow a nearby object. In particular, it will not "prefer" any particular kind of object to track. If we wish to program a task where one robot follows another, we may consider adding local infrared communication between the robots, enabling them to transmit and receive messages. This kind of communication allows one robot to lead and the other to follow. It provides an experimental setting in which to investigate the concept of information invariants.

2.1.2. Details of the following task

We now discuss the task of following in some more detail. Consider two autonomous mobile robots, such as those described in [44]. The robots we have in mind are the Cornell mobile robots [44], but the details of their construction are not important.

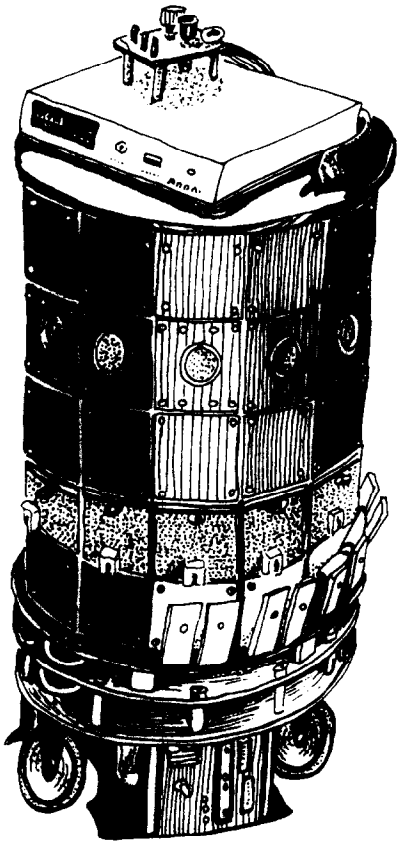


Fig. 1. The Cornell mobile robot TOMMY. Note (mounted top to bottom on the cylindrical enclosure) the ring of sonars, the IR Modems, and the bump sensors. LILY is very similar.

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 an onboard processor for control and programming.

We wish to consider a task in which one robot called LILY must follow another robot called TOMMY. It is possible to write such a control loop using only sonar readings and position/force control alone.

We now augment the robots described in [44] 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 (see Fig. 1). 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., TVs).[3] Experiments with our initial design [5] seemed to indicate that the communication bandwidth we could expect was roughly 2400 baud-feet. That is, at a distance of 1 foot between LILY and TOMMY, we

[3] The IR modems can time-slice between collision detection and communication; moreover, nearby modems (on the same robot) can "stagger" their broadcasts so as not to interfere with each other.

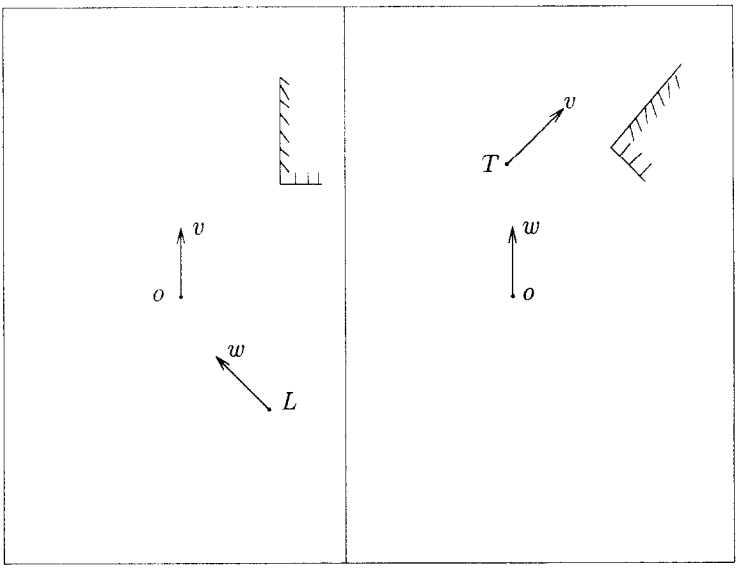


Fig. 2. The "radar screens" of TOMMY and LILY. TOMMY (T) is approaching a wall (on his right) at speed v, while LILY (L) follows at speed w.

could expect to communicate at 2400 baud; at 2 feet, the reliable communication rate drops to 1200 baud, and so forth.

We pause for a moment to note that this simple, experimentally-determined quantity is our first example of an information invariant.

Now, modem i is mounted so as to be at a fixed angle from the front of the robot base, and hence it is at a fixed angle θ_i from the direction of forward motion, which is defined to be 0.

Now, suppose that TOMMY is traveling at a commanded speed of v (note v need not be positive). For the task of following, each modem panel i on TOMMY transmits a unique identifier (e.g., `'Tommy`), the angle θ_i, and the speed v. That is, he transmits the following triple:[4] $\langle id, \theta_i, v \rangle$.

In this task, LILY transmits the same information, with a different *id* of course. This means that when the robots are in communication each can "detect" the position (using sonars and IRs), the heading, and the name of the other robot.[5] In effect each robot can construct a virtual "radar screen" like those used by air traffic controllers, on which it notes other robots, their position and heading, as well as obstacles and features of

[4] The identifier is necessary for applications involving more than two robots. Also, using the *id* a robot can disambiguate other robots' broadcasts from its own IR broadcast (e.g., reflections off white walls).

[5] This data is noisy, but since an adequate servo loop for following can be constructed using sonars alone [44], the IRs only add information to the task. The IR information does not measurably slow down the robot, since the IR processing is distributed and is not done by the Scheme controller.

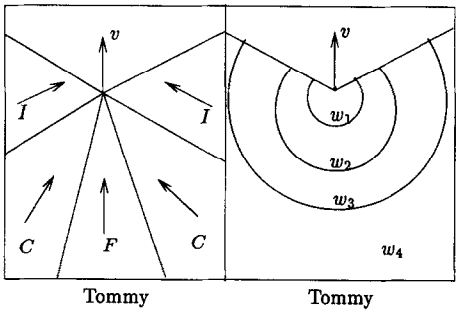


Fig. 3. The statespace "radar screen" of TOMMY is partitioned to indicate the control for LILY. (For the task of following, we could partition LILY's screen instead, but this is clearer for exposition). On the left is LILY's direction control; and the regions are F (follow), C (correct), and I (intercept). The commanded motion direction is shown as an arrow. On the right is LILY's speed control, with w_1 being very slow, w_4 fast, and $w_1 < w_2 < w_3 < w_4$. This control partition is conditioned on TOMMY's speed v.

the environment. The screen (see Fig. 2) is in local coordinates for each robot.[6] It is important to realize that although Fig. 2 "looks" like a pair of maps, in fact, each is simply a local reconstruction of sensor data. Moreover, these "local maps" are updated at each iteration through the servo loop, and so little retained state is necessary.

Now, robotics employs the notion of *configuration space*[7] [37] to describe control and planning algorithms. The *configuration* of one of our robots is its position and heading. *Configuration space* is the set of all configurations. In our case, the configuration space of one robot is the space $\mathbb{R}^2 \times S^1$. A related notion is *state space*, which is the space of configurations and velocities of the robot. After some reflection, it may be seen that Fig. 2 is a geometric depiction of a state space for the robot task of following (it is actually a representation of the mutual configuration spaces of the robots). Depending on where the robots are in Fig. 2, each must take a different control (servo) action. The points where one robot takes the same (parameterized) action may be grouped together to form an equivalence class. Essentially, we partition the state space in Fig. 2 into regions where the same action is required. This is a common way of synthesizing a feedback control loop. See Fig. 3.

The point is that in this analysis, we may ask, "What state must the robot LILY retain?". After some thought, the answer is, *very little*, since the "radar screens" in Fig. 2 may be drawn again from new sensor readings at each iteration. That is, no state must be retained between servo loop iterations, because in an iteration we only need some local state to process the sensor information and draw the information in Fig. 2. (We do not address whatever state TOMMY would need to figure out "where to lead", only how he should modify his control so as not to lose LILY.) One consequence of this kind of "stateless" following is that if communication is broken, or one robot

[6] In the language of [14], the sonar sensors, plus the IR communication, represent *concrete* sensors, out of which the *virtual* sensors shown in Fig. 2 can be constructed. The construction essentially involves adding the IR information above to the servo loop for following using sonar given in [44]. The details are not particularly important to this discussion.

[7] See [36] for a good introduction.

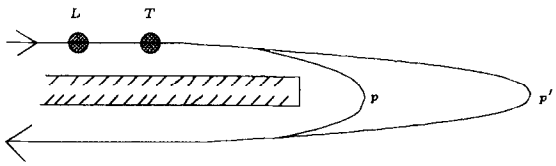


Fig. 4. Following around a wall. The shorter path p is quicker by Δt than p', but it cannot be executed without more communication or state.

is obscured from the other, then the robots have no provision (no *information*) from the past on which to base a strategy to reacquire contact. They can certainly go into a search mode, but this mode is stateless in the sense that it is not based on retained state (data) built up from before, before the break in contact. In short, at one time-step, LILY and TOMMY wake up and look at their radar screens. Based on what they see, they act. If one cannot see the other, perhaps it can begin a search or broadcast a cry for help. This is an essential feature of statelessness, or reactivity. Let us call a situation in which the robots maintain communication *preserving the control loop*. If they break communication it *breaks* the control loop.

Now, suppose that TOMMY has to go around a wall, as in Fig. 4. Suppose TOMMY has a geometric model of the wall (from a map or through reconstruction). Then it is not hard for TOMMY to calculate that if he takes a quick turn around the wall (as shown in trajectory p), that the line of sight between the robots may be broken. Since LILY is "stateless" as described above, when communication is broken the following task will fail, unless LILY can reacquire TOMMY. It is difficult to write a general such "search and reacquire" procedure, and it would certainly delay the task.

For this reason, we may prefer TOMMY to predict when line-of-sight communication would be broken, and to prefer a trajectory like p' (Fig. 4). When executed slowly enough, such trajectories as p' will allow the robots to maintain communication, and hence allow the following task to proceed. However, there is a cost: for example, we may reasonably assume that taking p' will take Δt longer than p. Now, let p^* denote the trajectory that follows the same path as p, but slowed-down so it takes the same time as p'[8]. It might also be reasonable to assume that if TOMMY slowed down enough to follow p^*, the robots could also maintain communication.

Hence, in this example, the quantity Δt is a measure of the "cost" of maintaining communication. It is a kind of invariant. But we can be more precise.

In particular, TOMMY has more choices to preserve the control loop. The distance at which LILY servos to TOMMY is controlled by a constant, which we will call the *following distance*[9] d. Hence, TOMMY, could transmit an additional message to LILY, containing the a new following distance d'. The meaning of this message would be "tighten up"—that is, to tell LILY to servo at a closer distance. Note that the message $\langle \texttt{heel}, d' \rangle$ essentially encodes a *plan* D—a new servo loop—for LILY. In this case, LILY will servo to follow TOMMY at the closer distance d', which will successfully permit the robots to navigate p while maintaining contact.

[8] So, p^* is the time-rescaled trajectory from p [19].

[9] For an explicit use of this constant in an actual servo loop, see, for example, [44].

Another possibility is that we could allow LILY to retain some state, and allow TOMMY to broadcast an *encoding* of the trajectory p. This encoding could be via points on the path, or a control program—essentially, by transmitting the message $\langle p \rangle$, TOMMY transmits a plan—a motion plan—for LILY. In this case, after losing contact with TOMMY, LILY will follow the path (or plan) p open loop, until TOMMY is reacquired.

In both these cases, we must allow LILY to retain enough state to store d or p. Since LILY already stores some value for d (see [44]), we need merely replace that. However, the storage for the plan (or path) p could be significant, depending on the detail.

Finally, we could imagine a scenario where LILY retains some amount of state over time to "track" TOMMY. For example, by observing TOMMY's trajectory before the break in communication, it may be possible to extrapolate future positions (one could, for example, use forward projections [23] or a kalman filter). Based on these extrapolations, LILY could seek TOMMY in the region of highest expectation. I will not detail this method here, but, it is not too difficult to see that it requires some amount of state for LILY to do this computation; let us call this amount s.

There is a tradeoff between execution time (Δt), communication (transmitting $\langle d' \rangle$ or $\langle p \rangle$), and internal state (storage for p or s). What is this relationship? Here is a conjecture one would like to prove about this relationship. For a path or a control program p or D, we denote its information complexity by $|p|$. For example, $|p|$ could measure the number of via points on p times their *bit-complexity* (the number of bits required to encode a single point).

Idea 2.1. There is an *information invariant* c for the task of following, whose units are bit-seconds. In particular,

$$c = |p|\, t_p = |D|\, t_D = s t_s, \tag{1}$$

where t_p, t_D, and t_s are the execution times for the three strategies above.

Eq. (1) should be interpreted as a lower bound—like the Heisenberg principle. It is no coincidence that Erdmann's information invariants are also in bit-seconds. An information invariant such as (1) quantifies the tradeoff between speed, communication, and storage. Currently, to prove such crisp results we must first make a number of assumptions about dynamics and geometry (see Appendix F.1). Moreover, the methods we describe below typically yield results using "order" notation ($O(\cdot)$ or big-theta $\Theta(\cdot)$) instead of strict equality.

One example of provable information invariants is given in the *kinodynamic* literature [8, 19, 20]. This work is concerned with provable planning algorithms for robots with dynamics. We give some details in Appendix F.1. Here we note that Xavier, in [21, 49], developed "trade-offs" similar in flavor to Eq. (1). Both Erdmann and Xavier obtain "trade-offs" between information and execution speed. Their methods appear to require a performance measure (e.g., the "cost" of a control strategy). One might view our work (and also [1], below) as investigating information invariants in the absence of a performance measure. In this case, we cannot directly measure absolute information

complexity in bit-seconds. Instead, we develop a way to relativize (or reduce) one sensori-computational system to another, in order to quantify their (relative) power. See Appendix F.1 for more details on information invariants with performance measures.

To summarize: the ambition of this work is to define the notions in Idea 2.1 so that they can be measured directly. Previous work [21,25,49] has required a performance measure in order to obtain a common currency for information invariance. In order not to use this crutch, we first define a set of transformations on sensori-computational systems. Second, we propose understanding the information invariants in terms of what these transformations preserve.

2.2. *The power of the compass*

In 1978, Blum and Kozen wrote a ground-breaking paper on maze-searching automata [1,35]. This section is devoted to a discussion of their paper, *On the power of the compass* [1], and we interpret their results in the context of autonomous mobile robots and information invariants.

In 1990, we posed the following question together with Jim Jennings:

Question 2.2 ([15]). "Let us consider a rational reconstruction of mobile robot programming. There is a task we wish the mobile robot to perform, and the task is specified in terms of external (e.g., human-specified) perceptual categories. For example, these terms might be "concepts" like *wall*, *door*, *hallway*, or *Professor Hopcroft*. The task may be specified in these terms by imagining the robot has *virtual* sensors which can recognize these objects (e.g., a wall sensor) and their "parameters" (e.g., length, orientation, etc.). Now, of course the physical robot is not equipped with such sensors, but instead is armed with certain *concrete* physical sensors, plus the power to retain history and to compute. The task-level programming problem lies in implementing the virtual sensors in terms of the concrete robot capabilities. We imagine this implementation as a tree of computation, in which the vertices are control and sensing actions, computation, and state retention. A particular kind of state consists of geometric constructions; in short, we imagine the mobile robot as an automaton, connected to physical sensors and actuators, which can move and interrogate the world through its sensors while taking notes by making geometric constructions on "scratch paper". But what should these constructions be? What program runs on the robot? How may these computation trees be synthesized?"

Let us consider this question of state, namely, what should the robot record on its scratch paper? In robotics, the answer is frequently either "nothing" (i.e., the robot is reactive, and should not build any representations), or "a map" (namely, the robot should build a geometric model of the entire environment). In particular, even schemes such as [39] require a worst-case linear amount of storage (in the geometric complexity n of the environment). Can one do better? Is there a sufficient representation that is between 0 and $O(n)$?

Blum and Kozen provide precise answers to these questions in the setting of theoretical, situated automata. This section didactically adopts the rhetorical "we" to compactly

interpret their results. While these results are theoretical, we believe they provide insight into Question 2.2 above.

We define a *maze* to be a finite, two-dimensional obstructed checkerboard. A finite automaton (DFA) in the maze may, in addition to its automaton transitions, transit on each move to an adjacent unobstructed square in the N, S, E, or W direction. We say an automaton can *search* a maze if eventually it will visit each square. It need not halt, and it may revisit squares. Hence, this kind of "searching" is the theoretical analog of the "exploration" task that many modern mobile robots are programmed to perform. However, note that in this entire section there is no control or sensing uncertainty.

We can consider augmenting an automaton with a single *counter*; using this counter it can record state. (Two counters would not be an interesting enhancement, because then we obtain the power of a Turing machine).[10]

We say two (or more) automata *search a maze together* as follows. The automata move synchronously, in lock-step. This synchronization could be effected using global control, or with synchronized clocks. When two automata land on the same square, each transmits its internal state to the other.

Finally, we may *externalize* and *distribute* the state. Instead of a counter, we may consider equipping an automaton with *pebbles*, which it can drop and pick up. Each pebble is uniquely identifiable to any automaton in the maze. On moving to a square, an automaton senses what pebbles are on the square, plus what pebbles it is carrying. It may then drop or pick up any pebbles.

Hence, a pure automaton is a theoretical model of a "reactive", robot-like creature. (Many simple physical robot controllers are based on DFAs). The exchange of state between two automata models local communication between autonomous agents. The pebbles model the "beacons" often used by mobile robots, or, more generally, the ability to side-effect the environment (as opposed to the robot's internal state) in order to perform tasks. Finally, the single counter models a limited form of state (storage). It is much more restrictive than the tape of a Turing machine. We believe that quantifying communication between collaborating mobile robots is a fundamental information-theoretic question. In manipulation, the ability to structure the environment through the actions of the robot (see, e.g., [13]) or the mechanics of the task (see, e.g., [40]) seems a fundamental paradigm. How do these techniques compare in power?

We call automata with these extra features *enhanced*, and we will assume that automata are not enhanced unless noted. Given these assumptions, Blum and Kozen demonstrate the following results. First, they note a result of Budach that a single automaton cannot search all mazes.[11] Next they prove the following:

[10] A *counter* is like a register. A DFA with a *counter* can keep a count in the register, increment or decrement it, and test for zero. A single counter DFA (introduced by Fischer [28] in 1966) can be viewed as a special kind of push-down (stack) automaton (PDA) that has only one stack symbol (except for a top of the stack marker). This means we should not expect a single-counter machine to be more powerful than a PDA, which, in turn, is considerably weaker than a Turing machine (see, e.g., [31, Chapter 5]). The proof that a two-counter DFA can simulate a Turing machine was first given by Papert and McNaughton in 1961 [41] but shorter proofs are now given in many textbooks, for example, see [31, Theorem 7.9].

[11] See [1] for references.

(1) There are two (unenhanced) automata that together can search all mazes.
(2) There is a two-pebble automaton that can search all mazes.
(3) There is a one-counter automaton that can search all mazes.

These results are crisp information invariants. It is clear that a Turing machine could build (a perfect) map of the maze, that would be linear in the size of the maze. This they term the naïve linear-space algorithm. This is the theoretical analog of most map-building mobile robots—even those that build "topological" maps still build a linear-space geometric data structure on their "scratch paper". But (3) implies that there is a *log-space* algorithm to search mazes—that is, using only an amount of storage that is logarithmic in the complexity of the world, the maze can be searched. [12] This is a precise answer to part of our Question 2.2.

However, points (1–3) also demonstrate interesting information invariants. (1) = (2) demonstrates the equivalence (in the sense of information) of beacons and communication. Hence side-effecting the environment is equivalent to collaborating with an autonomous co-agent. The equivalence of (1) and (2) to (3) suggests an equivalence (in this case) and a tradeoff (in general) between communication, state, and side-effecting the environment. Hence we may credit [1] with a excellent example of information invariance.

2.2.1. *The power of randomization*

Erdmann's Ph.D. Thesis is an investigation of the power of randomization in robotic strategies [24]. The idea is similar to that of randomized algorithms—by permitting the robot to randomly perturb initial conditions (the environment), its own internal state, or to randomly choose among actions, one may enhance the performance and capabilities of robots, and derive probabilistic bounds on expected performance. [13] This lesson should not be lost in the context of the information invariants above. For example, as Erdmann points out, one finite automaton can search any maze if we permit it to randomly select among the unobstructed directions. The probability that such an automaton will eventually visit any particular maze square is one. Randomization also helps in finite 3D mazes (see Section 2.2.2 for more on the problems that deterministic (as opposed to randomized) finite automata have in searching 3D mazes), although the expected time for the search increases somewhat.

These observations about randomizing automata can be even extended to *unbounded* mazes (the mazes we have considered are finite). However, in a 2D unbounded maze, although the automaton will eventually visit any particular maze square with probability

[12] Here is the idea. First, [1] show how to write a program whereby an unenhanced DFA can traverse the boundary of any single connected component of obstacle squares. Now, suppose the DFA could "remember" the southwesternmost corner (in a lexicographic order) of the obstacle. Next, [1] show how all the free space can then be systematically searched. It suffices for a DFA with a single counter to record the y-coordinate $y_{\min}$ of this corner. We now imagine simulating this algorithm (as efficiently as possible) using a Turing machine, and we measure the bit-complexity. If there are n free squares in the environment then $y_{\min} \leqslant n$, and the algorithm consumes $O(\log n)$ bits of storage. For details, see [1].

[13] While the power of randomization has long been known in the context of algorithms for maze exploration, Erdmann was able to lift these results to the robotics domain. In particular, one challenge was to consider continuous state spaces (as opposed to graphs).

one, the expected time to visit it is infinite. In 3D, however, things are worse: in 3D unbounded mazes, the probability that any given "cube" will be visited drops from one to about 0.37.

2.2.2. What does a compass give you?

Thus we have given precise examples of information invariants for *tasks* (or for one task, namely, searching, or "exploration"). However, it may be less clear what the information invariants for a *sensor* would be. Again, Blum and Kozen provide a fundamental insight. We motivate their result with the following

Question 2.3. Suppose we have two mobile robots, TOMMY and LILY, configured as described in Section 2.1. Suppose we put a flux-gate magnetic compass on LILY (but not on TOMMY). How much more "powerful" has LILY become? What tasks can LILY now perform that TOMMY cannot?

Now, any robot engineer knows compasses are useful. But what we want in answer to Question 2.3 is a precise, provable answer. Happily, in the case where the compass is relatively accurate,[14] [1] provide some insight:

Consider an automaton (of any kind) in a maze. Such an automaton effectively has a compass, since it can tell N,S,E,W apart. That is, on landing on a square, it can interrogate the neighboring N,S,E,W squares to find out which are unobstructed, and it can then accurately move one square in any unobstructed compass direction.

By contrast, consider an automaton in a graph (that need not be a maze). Such an automaton has no compass; on landing on a vertex, there are some number $g \geqslant 0$ of edges leading to "free" other vertices, and the automaton must choose one.

Hence, as Blum and Kozen point out, "*Mazes and regular planar graphs appear similar on the surface, but in fact differ substantially. The primary difference is that an automaton in a maze has a* compass: *it can distinguish N,S,E,W. A compass can provide the automaton with valuable information, as shown by the second of our results*" [1]. Recall point (1) in Section 2.2. Blum and Kozen show, that in contrast, to (1), no two automata together can search all finite planar cubic graphs (in a *cubic* graph, all vertices have degree $g = 3$). They then prove that no three automata suffice. Later, Kozen showed that four automata do not suffice [35]. Moreover, if we relax the planarity assumption but restrict our cubic graphs to be 3D mazes, it is known that no finite set of finite automata can search all such finite 3D mazes [3].

Hence, [1,35] provide a lower bound to the question, "What information does a compass provide?" We close by mentioning that in the flavor of Section 2.2.1, there is a large literature on randomized search algorithms for graphs. As in Section 2.2.1, randomization can improve the capability and performance of the search automata.

[14] In considering how a very accurate sensor can aid a robot in accomplishing a task, this methodology is closely allied with Erdmann's work on developing "minimal" sensors [25].

3. Discussion: measuring information

We have described the basic tools and concepts behind information invariants. We illustrated by example how such invariants can be analyzed and derived. We made a conceptual connection between information invariants and tradeoffs. In previous work, tradeoffs arose naturally in kinodynamic situations, in which performance measures, planning complexity, and robustness (in the sense of resistance to control uncertainty) are traded-off. We noted that Erdmann's invariants are of this ilk [24].

However, without a performance (cost) measure, it is more difficult to develop information invariants. We believe measures of *information complexity* are fundamentally different from *performance measures*. Our interest here is in the former; we will not discuss performance measures again until Appendix F.1. Here are some measures of the information complexity of a robotic 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*? and (c) *How can the robot change (side-effect) the environment in order to record state or sensory information to perform a task*? Examples of these categories include: (a) space considerations for computer memory, (b) local IR communication between collaborating autonomous mobile robots, and (c) dropable beacons. With regard to (a), we note that, of course, memory chips are cheap, but in the mobile robot design space, most investigations seem to fall at the ends of the design spectrum. For example, (near) reactive systems use (almost) no state, while "map builders" and model-based approaches use a very large (linear) amount. Natarajan [43] has considered an invariant complexity measure analogous to (b), namely the number of robot "hands" required to perform an assembly task. This quantifies the interference kinematics of the assembly task, and assumes global synchronous control. With regard to (c), the most easily imagined physical realization consists of coded IR beacons; however, "external" side-effects could be as exotic as chalking notes on the environment (as parking police do on tires), or assembling a collection of objects into a configuration of lower "entropy" (and hence, greater information). *Calibration* is an important form of external state, which we explore in Part II.

In Part I, we exploited automata-theoretic results to explore invariants that trade-off internal state, communication, and external state. While Part I concentrates on information invariants for *tasks*, we did touch on how information invariants for *sensors* can be integrated into the discussion. In particular, we reviewed a precise way to measure the information that a compass gives an autonomous mobile robot. Somewhat surprisingly, trading-off the measures (a)–(c) proves sufficient to quantify the information a compass supplies.

The compass invariant illustrates the kind of result that we would like to prove for more general sensors. That is, we could add a fourth measure, (d) *How much information is provided by sensors*? While the examples we presented are perhaps didactically satisfying, we must introduce some more machinery in order to extend our discussion to include two additional important measures of the information complexity of a robotic task: (d), and (e) *How much computation is required of the robot*? In Part II we explore these issues in some detail. In particular, we describe how one might develop a kind of "calculus" on measures (a)–(e) in order to compare the power of

sensor systems analytically. To this end, we develop a theory whereby one sensori-computational system can be "reduced" to another (much in the spirit of computation-theoretic reductions), by adding, deleting, and reallocating (a)–(e) among collaborating autonomous agents.

Part II—Sensors and computation

4. Sensors

Intuitively, we can imagine a *sensor system* being implemented as a tree of sensori-computational elements, in which the vertices are controllers and sensors, computing devices, and state elements. Such a system is called a *virtual sensor* by [14]. In a virtual sensor, outputs are computed from the outputs of other sensors in the same device. Given two sensor systems E and H, we would like to be able to quantify the information the sensors provide. In particular, suppose E and H are different "implementations" (in a sense we shall soon make precise) of superficially similar sensor systems. We would like to be able to determine whether the two systems are "equivalent" in the sense that they deliver "equivalent" information, that is, whether $E \cong H$. More generally, we would like to be able to write an "equation" like

$$E \cong H + \boxdot \tag{2}$$

where we can rigorously specify what box $\boxdot$ we need to "add" to H to make sensor E. For example, the box could represent some new sensing, or some computation on existing sensory and stored data. In Part II we discuss some methods for achieving these goals. To illustrate our techniques, we describe two sensors, the *radial* sensor [25], and the *beacon*, or *lighthouse* sensor. We then develop methods to compare the sensors and their information invariants. These sensors bear some relation to the *compass* discussed in Part I; it is our goal here to quantify this relationship precisely. In the beginning, we will allow informal definitions, which suffice for building intuition. The following concepts will be defined precisely in Section 8: the term *simulate*, the *output* of a sensor, a sensori-computational *resource*, the relation $\cong$, and the operator $+$. We begin as follows:

Definition 4.1 (*Informal*). [15] For 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 $S \cong Q$.

The operator $+$ in Eq. (2) represents "adding" something to H. Informally, this "something" is what we would like to call a *resource* (later, in Section 4.2.1). We will later see that $\cong$ is an equivalence relation.

Here is a preview of the formalism we will develop. We view sensor systems as "circuits". We model these circuits as graphs. Vertices correspond to different sensori-computational components of the system (what we will call "resources" below). Edges

[15] This definition is formalized in Section 8.1.

correspond to "data-paths" through which information passes. Different embeddings of these graphs correspond to different spatial allocation of the "resources". We also permit resources to be *colocated*. This requires that we consider graph *immersions* as well as graph embeddings. *Immersions* are like embeddings, but they need not be injective. Under this model, the concepts above are easily formalized. For example, the operation + turns out to be like taking the union of two graphs.

One key idea involves asking: "What information is added (or lost) in a sensor system when we change its immersion?" and "What information is preserved under all immersions?". Our goal will be to determine what classes of immersions preserve information. Sections 4.1–7 explore this idea through an example.

4.1. The radial sensor

We begin with a didactic example. In [25] Erdmann demonstrates a method for synthesizing sensors from task specifications. The sensors have the property of being "optimal" or "minimal" in the sense that they convey exactly the information required for the control system to perform the task. For our purposes, it is sufficient to examine a particular sensor, called the *radial sensor*, which is the output of one of his examples. The radial sensor arises by considering manipulation strategies in which the robot must achieve a goal despite uncertainty.

The radial sensor works as follows. Consider a small robot in the plane. Suppose there is a goal region G which is a small disc in the plane. See Fig. 5. The robot is at some configuration $\boldsymbol{x} \in \mathbb{R}^2$, and at some heading $h \in S^1$. Both these state variables are unknown to the robot. The robot can only command *relative* motions (relative to the local coordinate system specified by $(\boldsymbol{x}, h)$). Thus, it would command a velocity $v_{\Delta\theta}$, and the robot would move in *relative* direction $\Delta\theta$, which is *global* direction $h + \Delta\theta$. The radial sensor returns the angle θ_r which is the angle between h and the ray between $\boldsymbol{x}$ and the goal. The robot need only command v_{θ_r} to reduce its distance to the goal.[16] This example easily generalizes to the case where there is uncertainty in the robot's control system (that is, the "aim" of $v_{\Delta\theta}$) see [25, 38]. It is plausible (and indeed, Erdmann proves) that this sensor is necessary and sufficient to write a feedback loop that provably attains the goal.

To summarize: the radial sensor returns information that encodes the relative heading θ_r of the goal G—relative to the robot's current heading h. See Fig. 5. We emphasize that the radial sensor does not reveal the configuration $(\boldsymbol{x}, h)$ of the robot beyond this. We will not describe possible physical implementations of the radial sensor, but see [25] for a discussion.[17]

[16] In the language of [14], the perceptual equivalence classes for this sensor are the rays emanating at $\boldsymbol{x}$.

[17] Erdmann emphasizes the special cases where the robot always knows its heading, or, where the robot's heading is always fixed (say, due North, so that h is always identically zero). In these cases, the radial sensor returns the *global* heading to the goal. This special case arises in the domain of manipulation with a robot arm, which, of course, is why it is natural for Erdmann's theory. The radial sensor we present is just slightly generalized for the mobile robot domain.

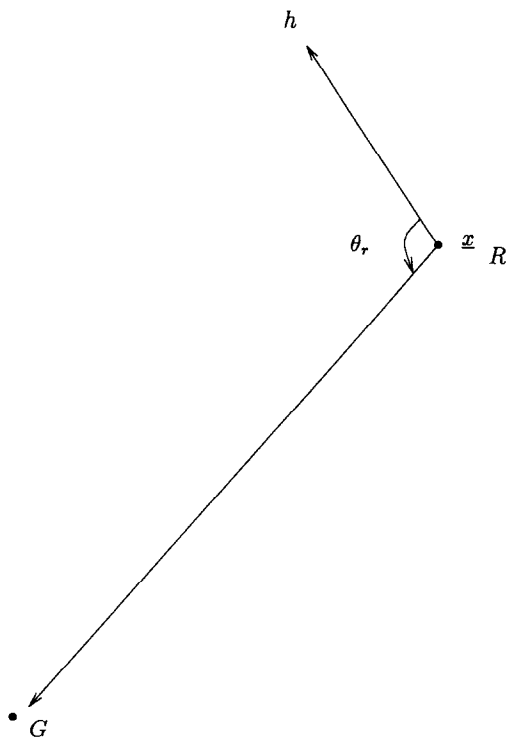


Fig. 5. The radial sensor E, showing heading h and relative goal direction θ_r.

4.2. *Lighthouses, beacons, ships, and airplanes*

We now describe another sensor. Our goal is to compare this sensor to the radial sensor using information invariants. See Fig. 6. We call this a *lighthouse* sensor system. We call this a sensor *system* since as described, it involves two physically separated "agents". We motivate this sensor as follows. Consider two mobile robots, which we denote L and R (see Fig. 6). L will be the "lighthouse" (beacon) and R will be the "ship". The robots live in the plane. In introducing the lighthouse system, we will informally introduce machinery to describe sensori-computational *resources*.

4.2.1. *Resources*

Now, to analyze the information invariants, we must be careful about the implementation of the sensor system, and, in particular, we must be careful to *count* how resources (a)–(e) (Section 3) are consumed and allocated—much the same way that one must be careful in performing a complexity analysis for an algorithm. Let us catalog the following kinds of *resources*:

- *Emitters.* On L, there are two lights which we call *physical emitters.* There is a unidirectional green light $\boxed{g}$ that rotates at a constant angular velocity. That is, the green light shines along a ray that is anchored (at its origin) at L. The ray sweeps

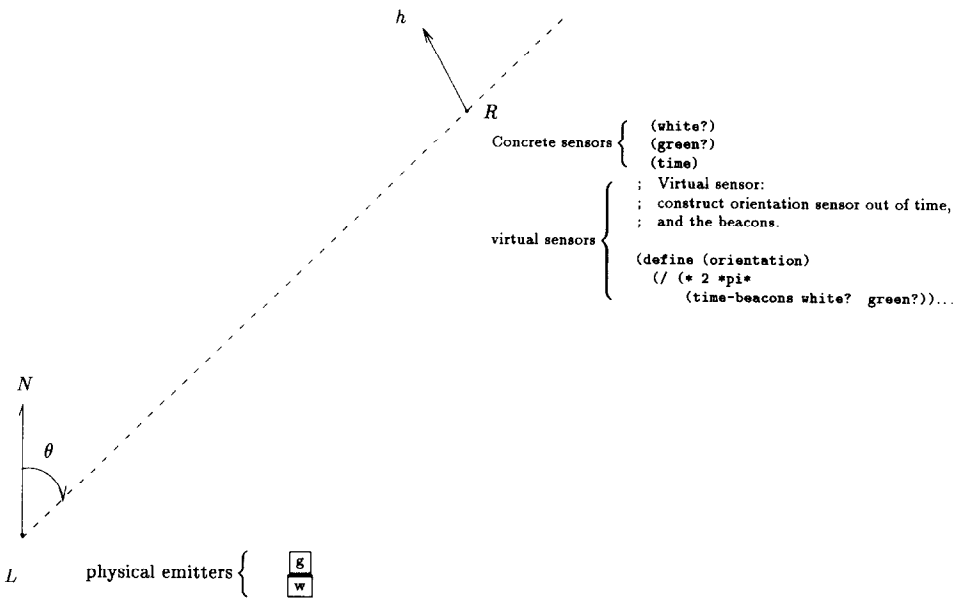


Fig. 6. The "beacon" sensor H, which is based on the same principle employed by lighthouses.

(rotates) about L. The green light can only be seen by points on that ray. Second, there is an omnidirectional white light [w] that flashes whenever the green light is pointing due North. That is, the white light can be seen from all directions.

- *Concrete sensors.* On R, there is a photo-electric sensor that detects when a white light illuminates R. Another sensor detects green light. There is also a clock on R.
- *Computation.* There is a computer on R that we can program in Scheme, following [44]. The concrete sensors above are interfaced to Scheme via library functions (as in [44]). The functions `(white?)` and `(green?)` are of type UNIT $\rightarrow$ BOOL, and return `#t` when light is sensed and `#f` otherwise. The clock is available as the function `(time)`, which returns the time measured in small units. We can measure the time and space requirements of a computation using standard techniques. Furthermore, we may quantify the amount of sensor information consumed by counting the number of calls to `(white?)`, `(green?)`, and `(time)` and the number of bits returned.

  Now, here is how lighthouses work. See Fig. 6. The "ship" R times the period t_W between white flashes. Then it measures the time t between a white flash and the next green flash. Clearly the "angle" θ of the robot—the angle between North and the ray from L to R—can be computed as $\theta = 2\pi t/t_W$. (Assuming the ship is moving slowly, relative to t_W).
- *Virtual sensors.* We can implement this as a *virtual sensor* [14] called `(orientation)` shown immediately below. The `orientation` sensor is specified as a computation that (i) calls concrete sensors, (ii) retains some local state (`T0`), and (iii) does some computation (`*`, `/`, etc.). It is easy to measure the time

and space requirements of the "circuit" that computes θ. Hence, we can implement certain *virtual sensors* to compute orientation. We detail this implementation below:

Given the resources above, we can implement the following virtual sensors "on" R:[18]

```
;  Virtual sensor:
;  construct orientation sensor out of time,
;  and the beacons.

(define (orientation)
  (/ (* 2 *pi*
        (time-beacons white? green?))
     (time-beacons white? white?)))

;  time between beacons
;   event1  and event2  are type UNIT → BOOL.%
[19]
(define (time-beacons event1 event2)
  (sleep-until event1)
  (let ((T0 (time)))
     (sleep-until event2)
     (- (time) T0)))

:  utility in scheme48 [44].
; sleep-until  waits until thunk  returns #t,
;  and then returns.
(define (sleep-until thunk) ....)
```

- *Resources R does not have.* Let us contrast our exemplar robot ship R with an *enhanced* version R' that corresponds to a real ship navigating at sea using lighthouse sensors. We should not confuse R with a real ship. A real ship R' has a map, on which are located *a priori* features, including a point which R' will assume corresponds to the location of L. True North is indicated on the map. R' computes θ as above (see Fig. 6), and draws a ray on the map, anchored at L, that is θ degrees from North. R' now knows that it is on that ray. In addition to possessing a map, and knowing the map coordinates of L, a real ship often has a compass. In the robotics domain, orientation odometry could approximate an accurate compass. Real ships also have communication devices like radios. We observe *communication* resources compare roughly to (b) in Section 3. Our *unenhanced* robot R, however, is not a real ship, and it has none of these resources.

Modern aircraft navigate using two sensors similar to the radial and lighthouse sensors. An *Automatic Direction Finder* (*ADF*) is a radial sensor. An ADF is simply a needle

[18] We must make some assumptions to prove this real-time program is correct. For example, we must assume the clock and the processor are very fast relative to the green light (and the ship).

[19] Objects of type UNIT → BOOL are called *boolean thunks*.

that points to a ground radio transmitter, in relative airplane coordinates. You do not need to know where you are or which way you are headed. You simply make the needle point straight ahead, by turning the airplane. So it is a radial sensor, and you track into the goal. A *VOR* (*VHF Omnirange*) is a lighthouse sensor. The VOR ground transmitter has the equivalent of a green and white light arrangement. The radio receiver in the plane decodes it, and then tells you the radial direction from the transmitter, in global coordinates. Then, if you actually want to fly to the VOR you have to have a compass, look at it, and turn the plane to fly in the same direction as your radio indicates. The VOR uses a clock, just like in the lighthouse. The "green emitter" in the VOR rotates at 30 Hz, and the white "North" light flashes 30 times a second. The receiver in the plane decodes the difference, just like in the lighthouse example, to give a direction. VORs do not use light, but they broadcast in the Megahertz range instead of the visual range.

To follow a radial sensor you only need to make the source be straight ahead of you; to follow a lighthouse sensor you need a compass. The radial sensor is in local coordinates and the lighthouse sensor is in global coordinates.

The ADF requires fewer instruments, but pilots tend to use the VOR. Why? Because that way you can look up your position on a chart, which is often what you care about (one VOR gives you a line; two give you your location). But if you just want to get somewhere, all you need is the ADF.[20]

5. Reduction of sensors

5.1. Comparing the power of sensors

Let us call the radial sensor E and the (unenhanced) lighthouse system H. The sensors are, of course, superficially similar: both have components at two spatially separated locations. Both sensors measure angles. Of course, they measure *different* angles. We cannot transform the information delivered by H into the information specification of E, without consuming more resources. These sensors deliver *incomparable* information, in that neither delivers strictly more information than the other.

We wish to be able to compare two sensors even when they deliver incomparable information. To do this, we introduce a mechanism called *reduction*, which allows us to compare the power of two sensor systems such as E and H. Hence, even though neither E nor H delivers strictly more information, they are comparable under a partial order induced by our reduction.

5.2. Sensor reduction

The analytic goal of sensor reduction is to be able to write "equations" like Eq. (2). The operational goal is to build one sensor out of another, and to measure the "power"

[20] There are some other reasons for using VORs, such as the fact that VORs are VHF while ADFs are LF/MF, so ADF reception gets blocked by thunderstorms while VOR reception does fine. On the other hand, VORs require line-of sight, whereas ADFs will work over the horizon.

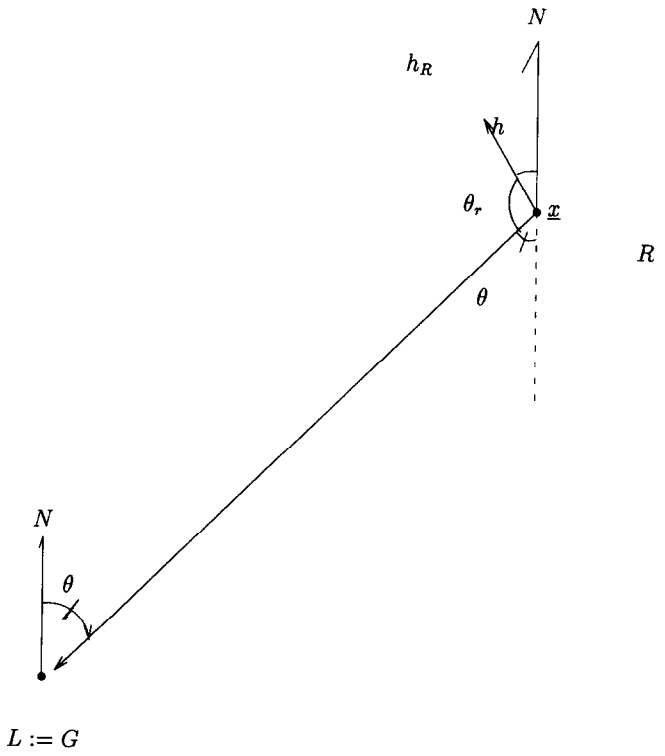


Fig. 7. Reduction using a compass h_R.

of the construction by a careful accounting for the resources we add. To illustrate the concept, we give two ways of constructing sensor E from sensor H. First, following Section 4.1, we assume that R is located at $x \in \mathbb{R}^2$ and has heading $h \in S^1$. However, R cannot sense these state variables and it does not know its configuration (x, h). Before we begin we stress the following: our goal is to change sensor H (by adding resources) so as to simulate sensor E. We have accomplished this task when R knows the angle θ_r, which is shown in Figs. 5, 7, and 8.

5.2.1. A reduction by adding a compass

We sketch a way to construct sensor E from H. This way is easy since it involves adding a powerful resource, namely a *compass*, to H. We will model this reduction as a function $\underline{S}$ from sensors to sensors. The reduction contains the following steps , which we denote S_1, S_2, and S_3 (see Fig. 7):

(S_1) We place the beacon L at the goal G.

(S_2) We add a concrete sensor called a *compass*[21] to R. The compass senses the heading h.

(S_3) The devices on R compute θ using the function `(orientation)` above, and then compute $\theta_r = \pi - h - \theta$. (See Fig. 7).

The reduction also adds a small amount of computation (but only a constant amount—two subtractions). We handle this by defining the compass to include this computation. Specifically, we define a sensor h_R to be a device that (i) computes the heading h, (ii) takes the output value of θ from `(orientation)` as an input, and (iii) outputs θ_r as specified in Step S_3. h_R could be implemented by a compass plus a small "circuit" to compute the value θ_r given h and θ. The subscript R of h_R denotes that it is installed on R. We will continue to refer call h_R a "compass" even though it is really a compass plus a small amount of computation.

In this reduction all the changes are made to R; L remains the same. Now, recall Eq. (2). Intuitively, we can substitute h_R for the box $\boxdot$ in this equation, and define the $+$ operator to encode how h_R is added to H, as specified in Steps $S_1, \ldots, S_3$ above.

5.2.2. *Reduction using permutation and communication*

The reduction in Section 5.2.1 requires adding new resources (the compass h_R). The next reduction we consider involves two new concepts. The first is *permutation*, and it involves redistributing resources in a sensor system, 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, H is; see Fig. 6). 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) in Section 3. We consider adding communication primitives of the form COMM($L \rightarrow R$, *info*), which indicates that L sends message *info* to R. Like permutation, communication only makes sense in a spatially distributed sensor system. That is, because spatially colocated components can communicate "for free" in our model, only "external" data-paths add information complexity to the system. *Internal* data-paths have the same (spatial) source and destination. *External* data-paths have a different (spatial) source and destination. Hence, permutation (alone) can change the information complexity of a system by "externalizing" internal data-paths. To analyze a system like H, 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 communication we will consider here (i.e., $L \rightarrow R$), and so we will henceforth abbreviate it by COMM(*info*).

[21] In using the term "compass" we make no commitment to a particular technology for implementation (such as sensing magnetic fields). In particular, the "compass" is an *orientation sensor* that could in principle be implemented using odometry or dead-reckoning, plus some initial calibration. Moreover, "North" N can be any fixed direction for our purposes, and need not be "true North". In the language of [38], the compass senses the projection of a perfect position sensor $p^* \in \mathbb{R}^2 \times S^1$ onto S^1.

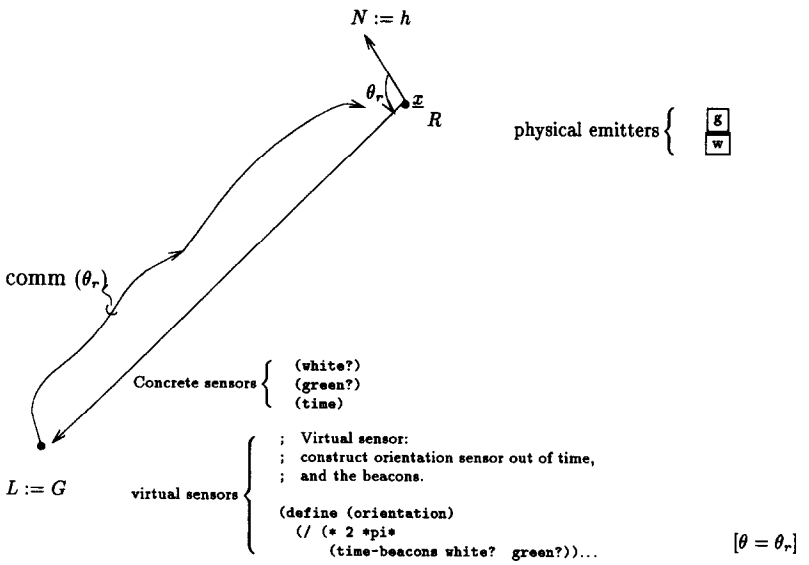


Fig. 8. Reduction using permutation and communication.

Given these concepts, we can sketch another reduction $\underline{Y}$. See Fig. 8. The reduction contains the following steps, which we denote Y_1, Y_2, and so forth.

(Y_1) As before, we place L at the goal G.

(Y_2) We move the physical emitters $\boxed{\text{g}}$ $\boxed{\text{w}}$ from L to R (i.e., we mount them on the robot). "North" for the emitters should be installed in the direction of R's heading. That is, the white light flashes when the green light passes the local (to R) North, which is defined to be the robot's heading, h.

(Y_3) We move the concrete sensors `(green?)`, `(white?)`, and `(time)` from R to L.

(Y_4) We move the virtual sensor `(orientation)` coded above to L. That is, now this program will run on L.

See Fig. 8. Given $(Y_1, \ldots, Y_4)$, by calling the procedure `(orientation)`, L can now compute the value of the angle θ_r shown in the figure. However, although L now knows θ_r, R does not. We solve this problem by allowing L to communicate the value θ_r to R using the COMM(·) primitive described above:

(Y_5) L communicates the value of θ_r to R using the primitive COMM(θ_r).

Note that the permutation steps $(Y_2, \ldots, Y_4)$ require no new resources. They merely require permuting the sensors and emitters. We do not view the relocation of the virtual sensor as "moving the computer to L". Instead, we view the virtual sensor `(orientation)` as a computational circuit; we move that circuit to L.

5.3. Installation notes

Crucial to installing a sensor is describing how the various physical resources should be lined up. We call these alignments *calibrations*. Since these calibrations constrain the spatial relationships among the various resources, as opposed to leaving them arbitrary, they effectively add information to the system. A calibration is some spatial relationship that is locked into place at the outset. This relationship may (or may not) change over time. Even when it does change, the initial calibration may still add information to the system, since the system can measure relative distances to the initial setting. Hence, calibration introduces an invariant that persists (at best) for the lifetime of the system. For example, by eliminating uncertainty at installation, we perform a kind of calibration, thereby eradicating that uncertainty for the duration of the calibration. Hence, calibration can displace the task of dealing with sensor uncertainty from the execution phase to the installation or layout phase. The purpose of this section is to introduce formal means for describing these calibrations, which we call *installation notes*. To make this more concrete, let us consider the calibrations necessary to permute and install sensor system H in the two reductions $\underline{\mathrm{S}}$ (Section 5.2.1) and $\underline{\mathrm{Y}}$ (Section 5.2.2).[22]

The installation notes are numbered I_1, I_2, and so forth.

Note I_1 (*Step* S_1) *and Note* I_2 (*Step* Y_1). The installation notes for Steps S_1 and Y_1 are identical. When installing L at G, we must make sure that L and G line up perfectly; otherwise, the angle measured will not be exactly θ_r.

Note I_3. When installing the physical emitters on L, we must make sure that "North" for the emitters line up perfectly with true North. Compare Note I_5, below.

Note I_4 (Step S_2). When installing the compass, we must make sure that it lines up perfectly with the heading of the robot.

Note I_5 (Step Y_2). We want the white light to flash when the green light passes through R's heading h. Hence, when installing the physical emitters on R, we must make sure that "relative North" for the emitters line up perfectly with the robot's heading h.

5.3.1. Calibration complexity

It is difficult to precisely measure the information gained in calibration. However, we note the following. First, the calibrations in I_3, I_4, and I_5 each add an equivalent amount of information to the system: each installation requires calibration of two 1 degree of freedom (1DOF) systems, each of which has configuration space S^1. Hence we say that I_3, I_4, and I_5 are *equivalent installation calibrations*.

Now let us consider calibrations I_1 and I_2 above. This installation requires a careful calibration of two 2DOF systems. To calibrate H so that at L is located at the point G clearly adds information. More precisely, note that we have so far considered the radial sensor E at a fixed goal G in the plane. Let us denote this particular installation by E_G. More generally, for a point y in the plane, we make the dependence explicit by writing E_y; thus we obtain a family of sensors $\{E_y\}$ parameterized by $y \in \mathbb{R}^2$.

[22] This section devolves to a suggestion of Mike Erdmann [22], for which we are very grateful.

Similarly, let us denote by H_y the sensor system H installed so that L is located at the point y. Now, our goal is to approximate one *particular* E_G using some H_y. Clearly, we could consider the case $G \neq y$; however in specifying E_G we specify G, and so this information is given. That is, it is no more work to locate H at G than to locate E at G, and the latter is unavoidable; it is the only way to implement E_G. Hence, we should be allowed to do at least this much work in installing H. In other words, merely in order to specify the sensor task, it is necessary to calibrate a 2DOF system to G—there is a sense in which the problem of approximating E cannot be specified without calibrating to some $y \in \mathbb{R}^2$. This argument is similar to saying that certain algorithms must at least read all their input. In this case, we say that the calibrations I_1 and I_2 are *necessary to specify the sensor* E. That is, the calibration required to install H_G is *necessary to specify* E_G. When the calibration parameter (the subscript G in this case) is understood, we will drop it.

Definition 5.1 (*Informal*). Consider two sensor systems S and Q. When S and Q require equivalent installation calibrations, and when the calibrations required to install Q are necessary to specify S, we say that S *dominates* Q *in calibration complexity.*

In Section 5.2.1 we described a reduction using a compass that yields a new sensor system from H. In Section 5.2.2 we described a reduction using permutation and communication, obtaining a different new sensor system from H. From the preceding discussion (Section 5.3), we conclude that E dominates both of these new sensor systems in calibration complexity.

Now it is clear that calibration is a source of information. We view calibration as a measure of the *external state* (see resource (c), Section 3) required for the task. Quantifying external state is tricky, since the time at which the resource is allocated (e.g., the time of calibration) may be much earlier than the time of the task execution. We developed the relatively sophisticated perspective of calibration complexity in this section, precisely to deal with this problem. Finally, it is worth noting the special role of *time* in this analysis (in that calibration and execution may be distant in time). We found it surprising that time would appear so crucial not only here, but also in the virtual sensor `(orientation)`.

5.4. Comments on power

The reduction in Section 5.2.1 requires adding an orientation sensor (which may be implemented using a compass or odometry). The reduction in Section 5.2.2 requires permuting resources (sensors and emitters). It also requires adding communication, since L must now communicate θ_r to R.

Let H^* denote the permutation of H described in Steps $(\mathrm{Y}_2, \ldots, \mathrm{Y}_4)$ in Section 5.2.2. Thus, in H^*, L has not been assigned any particular location, and while L knows θ_r, R does not. By installing H^* so that L is assigned the location G, we obtain a sensor called H^*_G. Now, recall the orientation sensor h_R for R, described in Section 5.2.1. Thus, in the language of Eq. (2), we have sketched how

$$E_G \cong H_G + h_R,$$
$$E_G \cong H_G^* + \text{COMM}(\theta_r). \tag{3}$$

Eq. (3) holds for all G. The operator $+$ denotes "combining" the two sensor subsystems. If this sounds somewhat operational, we will give a more analytic discussion below in Section 6 and a formal definition in Section 8, where we describe the semantics of our sensor model in detail.

5.4.1. Output communication

The term $\text{COMM}(\theta_r)$ in Eq. (3) says that we permit the permuted system H_G^* to route the information θ_r from one subsystem of H_G^* to another, spatially removed subsystem (these subsystems happen to be L and R in our case). First, note that θ_r is exactly the desired output of the sensor E_G. Hence the term $\text{COMM}(\theta_r)$ denotes an internal rerouting $(L \rightarrow R)$ of this information within the permuted sensor system H_G^*. Let us generalize this construction.

Definition 5.2. 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 $\mathbb{K}(b)$ be the number of values b can take on, and define $\log \mathbb{K}(b)$ 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 $\mathbb{K}(b) = q$, and $\log \mathbb{K}(b) = \log q$. In our example, θ_r is the output of E_G; the quantity $\log \mathbb{K}(\theta_r)$ is the output size of E_G. Now, suppose the information b is communicated over a data-path 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.

To return to our example, it is clear that we can make the permuted sensor system H_G^* satisfy the information specification of E_G if we merely add one internal re-routing operation of bandwidth $\log \mathbb{K}(\theta_r)$. In this case, we say we have added *output communication* to the permuted sensor system. [23]

More precisely, let S be a sensor system with output b. Let Q be another sensor system. We imagine Q as a "circuit" embedded in (say) the plane. Let $\text{COMM}(b)$ be a "sensor system" with one data-path e, that has bandwidth $\log \mathbb{K}(b)$. Then, adding output communication to Q can be viewed as the following transformation on sensor systems: $Q \mapsto Q + \text{COMM}(b)$. The transformation is parameterized by (the bandwidth of) S. The bounded-bandwidth data-path e can be spliced into Q anywhere. We note that this transformation can be composed with permutation (in either order):

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

[23] To borrow a UNIX metaphor, this transformation allows the system to do an internal `rcp`, but not RPC—that is, it can copy information between subsystems, but it cannot request arbitrary remote evaluations.

We give a fully formal, graph-theoretic model of this transformation in Section 8.7.2.

6. A hierarchy of sensors

The examples above illustrate a general principle. This principle is analogous to the notion of *reduction* in the theory of computation. We would like our notion of reduction to do work analogous to the work done by computation-theoretic reductions. Consider two sensor systems S and Q. Recall the definitions of *simulation* (Definition 4.1) and *calibration complexity* from Section 5.3.1.

Definition 6.1. 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 5.2. We define the *maximum bandwidth* $\mathrm{mb}(\mathcal{S})$ to be the greater of the internal bandwidth, external 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.

Definition 6.2. We write $S \leqslant Q$ when
(1) Q simulates S ($S \cong Q$),
(2) S dominates Q in calibration complexity, and
(3) $\mathrm{mb}(Q)$ is bounded above by $\mathrm{mb}(S)$.

Calibration exploits external state. Definition 6.2 allows us to order systems on how much information this external state (from calibration) yields. We will complete the formalization and analysis of calibration complexity later, in Sections 8.7.4 and A.1. 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 of configuration, we say the system is *simple*. When all the relations are semi-algebraic (s.a.), we say the system is *algebraically codesignated.*

Now, let Q^* denote a permutation of sensor system Q, as described in Section 5.2.2. (For a formal definition, see Definition 8.6.)

Definition 6.3. We write $S \leqslant^* Q$ if there exists some permutation Q^* of sensor system Q such that $S \leqslant Q^*$.

Recall the meaning of COMM(*info*) from Sections 5.2.2 and 5.4.1. Finally,

Definition 6.4. Given two sensor systems S and Q, choose b such that $\log \mathbb{K}(b) = \mathrm{mb}(S)$. We say S is *efficiently reducible* to Q if

$$S \leqslant^{*} Q + \mathrm{COMM}(b). \tag{4}$$

In this case we write $S \leqslant_{\mathrm{I}} Q$.

For monotonic sensor systems, it suffices to take b to be the output of S (see Appendix B). This special case motivates the construction on the right-hand side of (4), where we add "output communication" to the sensor system Q (Section 5.4.1).

We now recap a couple of crisp results using reductions:

Claim 6.5.
(a) $E_G \leqslant H_G + h_R$, *and*
(b) $E_G \leqslant H_G^{*} + \mathrm{COMM}(\theta_r)$.

Proof. Recall the discussion from Section 5.3.1 on calibration complexity. To obtain (a), we use the reduction that employs a compass (Section 5.2.1). The proof of (b) is obtained by the reduction using permutation and communication (Section 5.2.2). □

Now, recall Eq. (3). The relation $E_G \cong H_G + h_R$, which derives from the compass reduction in Section 5.2.1, does *not* imply efficient reducibility, since adding a new concrete sensor h_R is too powerful to imply efficient reducibility. However, by the reduction in Section 5.2.2:

Proposition 6.6. *Erdmann's radial sensor E is efficiently reducible to the lighthouse sensor system H, that is $E \leqslant_{\mathrm{I}} H$.*

Proof. Recall from Eq. (3) that $E_G \cong H_G^{*} + \mathrm{COMM}(\theta_r)$, and that θ_r is the output of E_G. From this, and Claim 6.5(b), we conclude that $E \leqslant_{\mathrm{I}} H$. □

7. Information invariants

The relation $\leqslant_{\mathrm{I}}$ defines a hierarchy of sensors. Compare the perceptual lattice of [14], who propose a geometric program for the analysis and synthesis of sensors based on their perceptual equivalence classes. The relation $\leqslant_{\mathrm{I}}$ orders sensor systems on the complexity of their information invariants.[24]

[24] It is possible to develop a geometric account of information invariance by pursuing the direction of [14]. For more on this connection, see Appendix D. The account we give in Section 8 is also geometric but with a different flavor. Appendix D deals with the geometry of lattices, where an element of the lattice represents (essentially) a knowledge state. In Section 8 we examine different immersions of sensor systems. "Permutations" or "automorphisms" of the function space of immersions that preserve the sensor functionality are viewed as a kind of information-preserving transformation, and, hence, a model of information invariance.

At this point it would be useful to review the particular information invariants in our example. Here is the basic idea. The invariants may be analyzed by first examining Eq. (3). Since $\cong$ is an equivalence relation, we obtain the peculiar equation

$$H_G + h_R \cong H_G^* + \text{COMM}(\theta_r). \tag{5}$$

Now, what exactly does Eq. (5) mean? We understand that at present, this equation is not yet formal. Our goal is to understand this intriguing result. To do so, we must give a formal account of the colocation of resources. Here is a general idea of how we will proceed:

Recall the transformation described in Section 5.4.1 and Definition 6.4, where we added output communication to a sensor system. Recalling that h_R denotes the compass, at first glance, we would appear to obtain the following information invariant: *a compass is equivalent to permutation plus output communication.* This idea is tantalizing because it seems to define an information equivalence between normally unapposed categories: it yields an information invariant relating sensors, communication, and resource permutation. The invariant (5) is valid. However, it appears that this invariant is critically conditioned on the *type* of information being rerouted by the output communication. Output communication permits us to transform between local and global coordinates; however, if some form of orientation sensing (at L) is not present before the output communication step, then no amount of permutation and communication can simulate a global compass.[25] In Section 8.8, we address the generality of Eq. (5). There we model the colocation of resources as geometric *codesignation constraints.* This colocation can be modeled as a quotient map, and in Section 8.8 we discuss its relationship to information invariance.

8. On the semantics of situated sensor systems

In this section, we formalize our model of sensor systems. We give formal definitions of the reductions using permutation, and by "combining" sensor systems and "adding" resources. Below, we use the term "*sensor system*" to mean "*sensori-computational system*" where it is mellifluous.

8.1. Situated sensor systems

We formalize our model of sensor systems using a concept similar to the *communication graph* from distributed systems [27].

Definition 8.1. 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.

[25] In the language of [14], communication and permutation permit us to map between the perceptual equivalence classes (PECs) of E (the rays described in Section 4.1) and the PECs of H.

Definition 8.2. 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*.

In Section 4.2.1 we 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[26] $\Omega_v : R^r \times C \to R^s$.

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). $\log \mathbb{K}(b)$ is the *bandwidth* of the edge (Section 5.4.1) 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 ($1 \leqslant i \leqslant r$ and $1 \leqslant j \leqslant s$).

Now, a simulation function for this edge e is taken to be a function $\Omega_e : R \to R$. 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 $\log \mathbb{K}(b)$).

We also define a resource called the "output device". Each sensor system must have exactly one vertex with this label, called the *output 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. In the examples introduced in Section 4 (the radial sensor E, lighthouse sensor system H, and the permuted lighthouse sensor H^*), we locate the output vertex on the "ship" at R.

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 9). These concepts will be used to implement our notions of a "specification" for a sensor system (Section 1.1, application 3) and "universal reductions" (Appendix A.4).

Definition 8.3. 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.

[26] Components that retain state can be modeled by a function $\Omega_v : R^r \times C \times S \to R^s \times S$, 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.

Hence the output value of $\mathcal{U}$ is a function of $\mathcal{U}$'s configuration.

The notions *output value* and *output* (Definition 5.2) 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 [34]; for now we restrict our attention to trees, which suffice to model our examples.[27] In general our discussion is restricted to consider one clock-tick; however, generalizations are possible to consider the time-varying behavior of the system [34].

Let us relate these new definitions to the examples from Part I. Examples of components are given by the resources described in Section 4.2.1. *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. One common connection is specified using the COMM(*info*) primitive defined in Section 5.4.1. For example, recall the permuted sensor system H^* introduced in Section 5.4. Next, recall Eq. (3):

$$\begin{aligned} E_G &\cong H_G + h_R, \\ E_G &\cong H_G^* + \mathrm{COMM}(\theta_r). \end{aligned} \tag{3}$$

Consider the sensor system specified by the bottom right-hand side of Eq. (3):

$$H_G^* + \mathrm{COMM}(\theta_r). \tag{$*$}$$

In the graph representation of $(*)$, the edge from the virtual `(orientation)` sensor at G to the output device at R, is labelled "θ_r".

Now, for each vertex v in V, we assume there is a configuration space C_v. A point in this space C_v represents a possible configuration of the component. Some components have configurations that change during the operation of the system (for example, in the lighthouse sensor system, all components mounted on the ship change configuration as the ship moves). Others are installed at fixed configurations. For example, the *emitters* g w in the lighthouse example, are installed at a specific position (L) and orientation (the white light flashes when the green light points North). So, the configuration space C for these emitters is $\mathbb{R}^2 \times S^1$. For convenience, let us assume that all components have the same configuration space C, and so $C = C_v$ (for all $v \in V$).

To summarize: a *component* is a primitive device that computes a function of (i) its inputs and (ii) its configuration $z \in C$. Each component is installed at a vertex

[27] Note the sensor system $H_G^* + \mathrm{COMM}(\theta_r)$ in Eq. (3) is effectively a tree, and not a graph, even though there is data flow both from R to L and L to R. This is because the output vertex u_0 on R does not feed back into the system.

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 $z = (z_1, \ldots, z, \ldots, z_d) \in C^d$, then the dynamics equation models a mapping from the actuator component $\ell(v)$'s output at z to the tangent space $T_z C^d$ to the configuration space. See [17,34] for more discussion of actuators.

Now, we give

Definition 8.4. A *situated* (or *immersed*) *sensor system* S is a sensor system $S = (V, E)$, together with an immersion $\phi : V \to C$ 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 sensor system is modeled by an immersed graph. If the map ϕ in Definition 8.4 is injective, then we call ϕ an *embedding*. Immersions need not be injective. In particular, in order to colocate vertices, it is necessary for immersions to be non-injective.

In Definition 8.4, 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 $\operatorname{im}\phi$.

Example 8.5. H_G is a situated sensor system (H, ψ). H_G^* is a different immersion ψ^* of the same sensor system H, and so $H_G^* = (H, \psi^*)$.

This example illustrates a general concept: *permutation* of a situated sensor system corresponds to the choice of a different immersion with the same domain. Formally:

Definition 8.6. Let $\mathbb{S} = (S, \phi)$ be a situated sensor system. A *permutation* $\mathbb{S}^*$ *of* $\mathbb{S}$ is a situated sensor system (S, ϕ^*) such that the domain $\phi^{-1}C$ of ϕ and the domain ${\phi^*}^{-1}C$ of ϕ^* are the same.[28]

Furthermore, for technical reasons, we also permit a permutation to change which vertex has the "output device" label. See Section C.2.

We can now formalize Definition 4.1 to say precisely what it means for two partially situated sensor systems to be equivalent:

Definition 4.1 (*Formalized*). 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

[28] Technically, there are two kinds of permutation. Definition 8.6 is called *vertex permutation*; in Appendix A.2.1 we discuss a more general model called *graph permutation*. Vertex permutation suffices for all examples in this paper, but our results go through for graph permutation as well.

$$(\mathcal{S}, \phi) \cong (\mathcal{U}, \psi) \tag{6}$$

for two situated sensor systems. Eq. (6) 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 Eq. (6) is taken to mean that $(\mathcal{U}, \psi)$ simulates $(\mathcal{S}, \phi)$ for *any* configuration of v and u.

For Definition 4.1, in the case where (say) ϕ is partial, we operationalize Eq. (6) 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: "$\forall \overline{\phi} \in \operatorname{ex}\phi$, Eq. (6) holds" (with bars placed over the immersions). We treat ψ similarly, with an inner universal quantifier, although codesignation constraints (Sections 8.3 and 8.5.1) 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 4.1 becomes, "for all configurations $x \in C$ of v, for all configurations $y \in D_{\mathcal{S}}(x)$ of u, Eq. (6) 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 4.1 can be generalized to any number of "unbound" vertices; see Eq. (34) in Section 9.

Definition 4.1 uses a strong notion of simulation (in which the outputs of the sensor systems must be identical). A weaker notion, which merely requires the same equilibrium behavior, is introduced in Section 12.

8.2. *Pointed sensor systems*

Suppose we wish to consider a sensor system $\mathcal{S} = (V, E)$, where one component $\ell(v)$ for $v \in V$ is in a particular configuration $G_0 \in C$. This corresponds to immersion via the partial function ϕ with domain $\{v\}$ and range $\{G_0\}$. We may abbreviate the situated system $(\mathcal{S}, \phi)$ by writing $\mathcal{S}_{G_0}$, to distinguish it from the unsituated system $\mathcal{S}$. This is the notation we use in Section 5.3.1 and after. Of course, for this notation to capture all the information above about v, we must specify the preimage[29] of G_0 under ϕ, but we did that in Section 5.3.1 when we wrote down

> "... *let us denote by* H_G *the sensor system* H *installed with* $L = G$".

We now explain the notation used in Example 8.5. First, we formalize our discussion of $\mathcal{S}_{G_0}$, above:

Definition 8.7. A *pointed immersion* of a sensor system $\mathcal{S} = (V, E)$ is a pair (ϕ, G) where $\phi : V \to C$ is an immersion of the vertices of $\mathcal{S}$, and $G \in \operatorname{im}\phi$. G is called the *base point*. An *extension* of a partial pointed immersion (ϕ, G) is any total pointed immersion $(\overline{\phi}, G)$ where $\overline{\phi}$ is an extension of ϕ.[30]

Definition 8.8. A *pointed sensor system* is a triple $(\mathcal{S}, \phi, G)$ where $(\mathcal{S}, \phi)$ is a situated sensor system and (ϕ, G) is a pointed immersion (Definition 8.7) of $\mathcal{S}$. We abbreviate $(\mathcal{S}, \phi, G)$ by $\mathcal{S}_G$.

---

[29] More precisely: we must write down that the preimage of G_0 under the immersion ϕ contains v.

[30] (ϕ, G) is called *weakly pointed* if ϕ is partial and G is not necessarily contained in $\operatorname{im}\phi$.

Hence, H_G in Example 8.5 is a pointed sensor system. Next,

Definition 8.9. A *pointed permutation* of a sensor system $(\mathcal{S}, \phi)$ is a pointed sensor system $(\mathcal{S}, \phi^*, G)$, where ϕ^* is a permutation of ϕ.

Hence, H_G^* in Example 8.5 is a pointed permutation of the pointed sensor system H_G. In general, if $\mathcal{S}_G^*$ is a pointed permutation of $\mathcal{S}_G$, then $\mathcal{S}_G$ is a pointed permutation of $\mathcal{S}_G^*$.

8.3. *Codesignation: basic concepts*

If we view the configurations of components in a sensory system as "variables", then Convention 4.1 gives a "default" for determining which variables are "free" and which are "bound". Here is another view:

The partial immersion specifies which variables are specialized to be *constants*. These are the vertices in the domain of the immersion. Their configurations correspond to bound variables (constants). The configuration variables for vertices outside the domain of the immersion are not yet specialized, and hence are free.

We now have two concepts to define and investigate. First, we show how to specify systems which contain some constant configuration variables. After that, we must find a way to make two free variables *codesignate* (see [7]). Two vertices r and u *codesignate* under an immersion ϕ when $\phi(r) = \phi(u)$. More generally, r and u codesignate under *different* immersions ϕ and ψ when $\phi(v) = \psi(u)$. We now proceed with these two tasks.

Recall our example of a pointed sensor system $\mathcal{S}_{G_0}$ from Section 8.2 above. Recall $\mathcal{S}_{G_0} = (\mathcal{S}, \phi, G_0)$, and $\mathcal{S} = (V, E)$. The domain of ϕ is the single vertex $v \in V$. Now, to continue, suppose that $r \in V$ is the vertex of component $\ell(r)$, and that $r \neq v$ so that ϕ does not specify how to immerse r. Consider a different sensor system $\mathcal{U}$, with at least one vertex u. We wish to consider "combining" $\mathcal{U}$ and $\mathcal{S}$ by saying something like this:

> Immerse $\mathcal{S}$ with vertex v at G_0. Now, vertex r of $\mathcal{S}$ will be somewhere, say, R; but we want to immerse $\mathcal{U}$ so that u is at R also.

Hence, we don't care where R is, save that we wish to colocate r and u. To do this, we make r and u codesignate under the immersions of $\mathcal{S}$ and $\mathcal{U}$. We call this a *codesignation constraint* after [7]. Here is how we may say this more precisely:

> Let $\mathcal{S}_{G_0}$ denote sensor system $\mathcal{S}$ immersed with vertex v at G_0 (as above). Immerse the rest of $\mathcal{S}$ in any consistent manner, and denote this immersion by $\overline{\phi}$. Thus $\overline{\phi}$ is the *extension* of ϕ so that the restriction $\overline{\phi}|_{\{v\}}$ of $\overline{\phi}$ to $\{v\}$ is identical to ϕ. Now, let $R \in C$ be the configuration of r under $\overline{\phi}$, i.e., $R = \overline{\phi}(r)$. Denote by ψ the (partial) immersion of $\mathcal{U}$ defined as follows. ψ sends vertex u of $\mathcal{U}$ to R. Note that G_0 is a "constant" and R is a "free variable", in the sense that R depends on which extension $\overline{\phi}$ of ϕ we choose, whereas G_0 does not.

In Eqs. (2)–(5), we abbreviated this construction as follows:

$$\mathcal{S}_{G_0} + \mathcal{U}_R \tag{7}$$

which is short for $(\mathcal{S}, \phi) + (\mathcal{U}, \psi)$ with ϕ and ψ defined as above. Note that (7) is not sufficient to specify the desired (partial) immersion unless we also note that the preimage (under the immersion ϕ) of G_0 contains vertex v of $\mathcal{S}$, and that

$$\overline{\phi}(r) = R = \psi(u). \tag{*}$$

(*) represents a codesignation constraint; we will define such constraints formally below in Section 8.5.1. We must also specify that G_0 is a constant and R is a free variable. The notation explained in (7) is used in the body of the paper, for example, in Eq. (3).

It remains for us to define precisely the + operator we just used, and we do so in Definitions 8.10–8.12 below.

8.4. Combining sensor systems

The + operator is defined on two graphs as a way of taking their union. Specifically:

Definition 8.10. 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 sensor systems (Definition 8.2) 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). Given Definition 8.10, we have:

Claim 8.11. *The operator + defined in Definition 8.10 is associative and commutative.*

Proof. Definitional. □

8.5. The general case

Let $(\mathcal{S}, \phi)$ and $(\mathcal{U}, \psi)$ be two situated sensor systems. Let V denote the vertices of $\mathcal{S}$ and U the vertices of $\mathcal{U}$. Our notation above ($\mathcal{S}_G$, $\mathcal{U}_R$, H_G, h_R, etc.) is effective when the image of each partial immersion is a singleton, e.g., $\phi(V) = \{G\}$ and $\psi(U) = \{R\}$. In these cases it suffices to abbreviate

$$\mathcal{S}_G = (\mathcal{S}, \phi) \quad \text{and} \quad \mathcal{U}_R = (\mathcal{U}, \psi),$$

and to specify which (if any) of the configurations G and R is constant and which (if any) is free. We now generalize this notation for more complicated partial immersions.

Suppose $(\mathcal{S}, \phi)$ and $(\mathcal{U}, \psi)$ have compatible partial immersions. Now, $\phi(V)$ and $\psi(U)$ (which need not be singletons, in general) represent the "constant" configuration

bindings of vertices (analogous to the singleton G above). We now consider codesignation constraints. All the codesignation constraints we have seen so far in Section 8 have this form: each was a pair $(v,u) \in V \times U$. A codesignation constraint is *compatible* with the immersions ϕ and ψ if one of the following is true:

(1) v is not in the domain $\phi^{-1}C$ of ϕ;

(2) u is not in the domain $\psi^{-1}C$ of ψ;

(3) $\phi(v) = \psi(u)$.

This definition is not quite general enough; we must also be able to specify (a) that two vertices of U (resp. V) codesignate—this means two components of $\mathcal{S}$ must be colocated. (b) we must also be able to specify that that two vertices *not* codesignate, for example, that $\phi(v) \neq \psi(u)$. The general definition is complicated and is given in Definition 8.14 below.

However, putting off the formal definitions for a moment, we can see what a combined sensor system really is. In summary: the immersions ψ and ϕ specify which component configurations are to be held constant. The codesignation constraints specify which components are to be co-located.

Definition 8.12. Let $(\mathcal{S},\phi)$ and $(\mathcal{U},\psi)$ be two situated sensor systems with compatible partial immersions. The *combined sensor system*

$$(\mathcal{S},\phi) + (\mathcal{U},\psi) \tag{8}$$

is specified by (8), together with a set of codesignation constraints compatible with ϕ and ψ. We say the combination (8) is *defined* when the partial immersions ϕ and ψ are compatible.

Now, consider two sensor systems $\mathcal{S}$ and $\mathcal{U}$. Both have output vertices, say, v_0 and u_0 resp. If $v_0 = u_0$ then this vertex remains the output vertex of $\mathcal{S}+\mathcal{U}$. In the case where $v_0 \neq u_0$, we must naturally specify which is the unique output vertex of the new, combined sensor system. By convention we will declare it to be either v_0 or u_0 (we must say which).[31] We adopt one default convention for this choice in Section 8.7.3. For more on output vertices, see Appendices C.1–C.2.

Definition 8.12 specializes to the particular cases such as Eq. (3) we have considered, by appropriate choice of partial immersions and codesignation constraints. To illustrate these choices, we give an example below, in Section 8.6. The operator $+$ is associative and commutative (see Claim 8.11 and Appendix C).

8.5.1. Codesignation constraints

Throughout this section, we let $(\mathcal{S},\phi)$ and $(\mathcal{U},\psi)$ be two situated sensor systems with compatible partial immersions $\phi: V \to C$ and $\psi: U \to C$.

Definition 8.13. Define the partial immersion $\phi + \psi$ as follows:

[31] This is not a severe restriction when we are considering permutations like $(\mathcal{S}+\mathcal{U})^*$ of $\mathcal{S}+\mathcal{U}$. See Appendix C.2.

$$\phi + \psi : \rightarrow C,$$
$$x \mapsto \begin{cases} \phi(x), & \text{if } x \in V, \\ \psi(x), & \text{if } x \in U. \end{cases}$$

We say the map $\phi+\psi$ is *defined* when the partial immersions ϕ and ψ are compatible.

Definition 8.14. A *codesignation constraint* is a pair $(x, y) \in (V \cup U)^2$.

Definition 8.15. We say a codesignation constraint (x, y) is *compatible with the partial immersions ϕ and ψ* if one of the following is true:

(1) x is not in the domain $(\phi + \psi)^{-1}C$ of $(\phi + \psi)$;
(2) y is not in the domain $(\phi + \psi)^{-1}C$ of $(\phi + \psi)$;
(3) $(\phi + \psi)(x) = (\phi + \psi)(y)$.

Noncodesignation constraints are modeled symmetrically to codesignation constraints. A codesignation constraint (x, y) indicates that we require that for any total immersion $\overline{\phi + \psi}$ that extends $\phi + \psi$,

$$(\overline{\phi + \psi})(x) = (\overline{\phi + \psi})(y) \qquad (*)$$

holds. A *noncodesignation* constraint requires *inequality* (instead of equality) in $(*)$. Definitions 8.14–8.15 handle a single constraint. For sets of constraints, you have to employ the machinery of Appendix A, which generalizes these definitions.

8.6. Example: the basic idea

As an example, let us interpret Eq. (3). We give it again:

$$\begin{aligned} E_G &\cong H_G + h_R, \\ E_G &\cong H_G^* + \text{COMM}(\theta_r). \end{aligned} \qquad (3)$$

Recall E_G and H_G are situated sensor systems. E_G is the radial sensor located at $G \in \mathbb{R}^2$. H_G is the lighthouse sensor with the emitters $\boxed{\text{g}}$ $\boxed{\text{w}}$ located at G and oriented Northward.

When H is situated at G as above to obtain H_G, the immersion is partial, leaving the position R of the ship, unspecified in H_G. h_R denotes the compass installed at R, calibrated towards North. Eq. (3) (top) holds for any ship's position R so long as the sensor system h_R is co-located at R. Compare the right-hand side of Eq. (3) to (7). As in (7), in Eq. (3), once the preimages (under the immersion) of G and R are specified, the immersion of the combined sensor system becomes clear.

Now, H_G^* defines a new immersion of H (by "new" we mean different from H_G). The immersion depends on R but Eq. (3) holds for any R. COMM(θ_r) defines a graph with exactly one edge e. e is an edge with label $\ell(e) = \theta_r$, from the virtual sensor `(orientation)` to the ship (the output vertex) at R. Thus, e is an edge between two

vertices of H^* (or H_G^*) but note that e is not part of the graph H^* (nor H_G^*); e is only present in the combination $H_G^* + \text{COMM}(\theta_r)$.

Finally, by convention, Eq. (3) (by itself) only holds for G. But, we specify in the sentence below Eq. (3) that it holds for any G. This is equivalent to placing the symbols "$\forall G$" before Eq. (3). This effectively "frees" G. The appearance of G as a subscript both on the left- and right-hand side of Eq. (3) indicates a codesignation constraint.

8.7. *Example (continued) : a formal treatment*

8.7.1. *The top of Eq.* (3)

We now rewrite Eq. (3) using the general notation of Section 8.5. In this example we do not explicitly consider orientation of components. However, the discussion can be generalized by taking the configurations G and R to lie in the configuration space $\mathbb{R}^2 \times S^1$.

Let ϕ be a partial immersion of E. Let ϕ_G be a partial immersion of E that installs it at G, so that $E_G = (E, \phi_G)$.

Let ψ be a partial immersion of H. Let ψ_G be a partial immersion of H that installs the emitters $\boxed{\text{g}}$ $\boxed{\text{w}}$ at G, so that $H_G = (H, \psi_G)$. We will define codesignation constraints so that all the concrete and virtual sensors are installed on the ship (i.e., at R).

Let v_1 and v_2 be the vertices of H such that $\ell(v_1) = \boxed{\text{g}}$, and $\ell(v_2) = \boxed{\text{w}}$.

Let $u_1, \ldots, u_k$ be the vertices of H corresponding to the concrete and virtual sensors described in Section 4.2.1. In particular, u_1 is the vertex of the virtual sensor `(orientation)`.

Let u_0 be the output vertex of H.

Let ρ be a partial immersion of the compass h. Let w be the vertex of the compass in h. Then we can rewrite the top of Eq. (3) as:

$$(E, \phi_G) \cong (H, \psi_G) + (h, \rho) \qquad (3\text{–top})$$

together with the codesignation constraints[32]

$$\{(u_1, u_i)\}_{1<i\leqslant k} \cup \{(v_1, v_2), (u_0, u_1), (u_1, w)\}. \qquad (9)$$

8.7.2. *The bottom of Eq.* (3): *the sensor system* COMM(·)

Now, H^* denotes a different immersion of H. Call this immersion ψ^*. Let ψ_G^* denote the partial immersion that installs the concrete and virtual sensors at G. We will define codesignation constraints so that the emitters are installed on the ship. We must now precisely define what COMM(·) means.

We can be sure of getting the semantics of COMM(·) correct by treating it as a sensor system in its own right (albeit, a small one). Now, $\text{COMM}(\theta_r)$ defines the graph with

[32] A careful analysis will show that, while it is necessary that the rotating emitter $\boxed{\text{g}}$ be located at G, the omnidirectional $\boxed{\text{w}}$ can be anywhere. Hence the codesignation constraint (v_1, v_2) is unnecessary. However, by removing it, we are left with the problem of synchronizing $\boxed{\text{g}}$ and $\boxed{\text{w}}$. Either we must add communication, or else calibrate the emitters and give $\boxed{\text{w}}$ a clock. These issues complicate the example and so we will not deal with them further.

vertices[33] $\{u_1, u_o\}$ and a single edge $e = (u_1, u_o)$ with $\ell(e) = \theta_r$. We observe that the transformation on sensor systems whereby we add output communication (Section 5.4.1 and Definition 6.4) implies the following:

> *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(θ_r).

Our model of communication is fairly abstract. External communication is probably not possible without some form of buffering by either the sender or the receiver. COMM($\cdot$) should include this buffer to be more realistic about modeling internal state.

Hence the bottom half of Eq. (3) may be written:

$$(E, \phi_G) \cong (H, \psi_G^*) + \text{COMM}(\theta_r) \qquad (3\text{–bot})$$

together with the codesignation constraints

$$\{(u_1, u_i)\}_{1<i\leqslant k} \cup \{(v_1, v_2)\}. \qquad (10)$$

Hence the bottom codesignation constraints (10) for (3-bot) are different from the top codesignation constraints (9) for (3-top), in that in the bottom constraints, w does not appear (since it is associated with the compass). Second, in the bottom equation, the output vertex is not constrained to be colocated with the virtual sensor `(orientation)`. Thus the codesignation constraint (u_1, u_o) disappears.

8.7.3. Bandwidth and output vertices

We have defined COMM($\cdot$) as a graph with a single edge e. 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 data-path that can carry information b; if b requires $k = \log \mathbb{K}(b)$ bits to encode then k is the *bandwidth* of e.

Now recall the discussion on how to choose output vertices in combined sensor systems (Section 8.5). Here, (Section 8.7.2, Eq. (3-bot)) we have u_o as the output vertex of both H^* and COMM(θ_r), and so it unambiguously remains the output vertex of the combined system $H^* + \text{COMM}(\theta_r)$. More generally, we adopt the following

Convention 8.16. Let $\mathcal{S}$ be a sensor system. Unless otherwise stated, we take the output vertex of the combined sensor system $\mathcal{S} + \text{COMM}(\cdot)$ to be the output vertex u_o of COMM($\cdot$).

For more on bandwidth, see Appendix B; for more on output vertices under permutation, see Appendices C.1–C.2.

8.7.4. Calibration complexity and codesignation

The size of the set (9) or (10) (number of codesignation constraints) is one measure of *calibration complexity* (see Section 5.3.1). However, this should be only part of the

[33] In this example, the vertices of COMM($\cdot$) are also vertices of H^*; but more generally the vertex sets can be disjoint.

measure. One reason that the number of codesignation constraints, alone, is not a good measure, is that one sensor system (say H, for argument) could have a single component that functions in the place of several colocated components in another sensor system (say, $\mathcal{V}$). For example, we could build a sensor $\mathcal{V}$ as follows: consider the emitter [g] in H. Break up the emitter [g] into all its tiny wires, power supply, filaments, rotating actuator, etc. All these components must then be colocated. This would result in more codesignation constraints for $\mathcal{V}$ than for H and thus, a spuriously high measure of calibration or installation complexity.

Instead, in order to measure calibration complexity we should compare "size" using something like order (Big-Oh $O(\cdot)$) notation. This is the basic idea we use, but there are some additional subtleties that we defer to Appendix A.1. There we propose a measure of calibration complexity that is more reasonable. This measure retains, however, one useful property: it is easy to compute it (in fact, like "size" above, it can be computed in the same time it takes to read the input).

8.7.5. Noncodesignation constraints and parametric codesignation constraints

To complete our model for this example, we must also introduce noncodesignation constraints so that $G \neq R$; this is necessary for our sensors to work. Suppose the radial sensor E has two vertices, t_0 and t_1, where t_0 is the output vertex, and t_1 is the "central vertex" of E (this is the vertex located at G in Fig. 5). The *non*codesignation constraints for both (3-top) and (3-bot) are

$$\{(u_1, v_1), (t_0, t_1)\}. \tag{11}$$

The former is a constraint on H (and H^*). The latter is a constraint on E. Finally, we require the *co*designation constraint

$$(t_0, u_0). \tag{12}$$

Eq. (12) is called a *parametric* codesignation constraint; it ensures that for all extensions $\overline{\phi_G}$, $\overline{\psi_G}$, and $\overline{\psi_G^*}$ of ϕ_G, ψ_G, and ψ_G^* resp., we have $\overline{\psi_G}(u_0) = \overline{\phi_G}(t_0) = \overline{\psi_G^*}(u_0)$. Parametric codesignation constraints are discussed further in Appendix A.3.

This completes our detailed discussion of the sensor systems in Eq. (3). The example is designed to explain most facets of our theory in a simple setting. Let us sketch how to make this analysis computationally effective. We choose two arbitrary points G and R in C. We begin with the two pointed immersions ϕ_G and ψ_G, with domains $\{t_1, t_0\}$ and $\{v_1, u_1\}$ resp. (So, ϕ_G is total and ψ_G is partial.) These functions and the desired permutation ψ_G^* are:

	t_1	t_0	u_1	v_1
ϕ_G	G	R		
ψ_G			R	G
ψ_G^*			G	R

We want our analysis to be true for any R and G (with $R \neq G$) and not just the ones we chose. To do this, we in effect wish to universally quantify over R and G and

treat these configurations as variables. To do this carefully and computationally requires the quantification machinery from Section 9. Here, we give the basic idea. Now, after our first use of Eq. (3), we wrote

"Eq. (3) holds for all G."

This sentence effectively adds "$\forall G$" to the front of Eq. (3), and hence to Eqs. (3-top) and (3-bot). We call this *freeing* G. To obtain this effect, we rewrite Eqs. (3-top) and (3-bot) as follows: remove the G subscripts: that is, replace ϕ_G by any immersion ϕ of E. Similarly, replace ψ_G by ψ and ψ_G^* by ψ^*. (See Section 10 for more details). We have chosen this notation because our constructions are parameterized by the task, and the task is specified by G. The notation leaves this parameterization explicit. As we shall see below, perhaps the cleanest way to model this example is to treat all the sensor systems as initially unsituated, yet respecting all the (non)codesignation constraints above. This may be done using the tools developed in the sequel (Sections 8.8–10).

8.8. Generality and codesignation

Consider a sensor system $\mathcal{S}$ with d vertices V, immersed via a map $\phi: V \to C$. The configuration space of this sensor system can be viewed as C^d, since any immersion ϕ can be represented as a point in[34] C^d. Consider a codesignation constraint (u, v) for $u, v \in V$. This specifies a new immersion of $\mathcal{S}$ in a quotient $C^d/(u \sim v)$ of C^d in which the images of u and v are identified. This quotient construction can be used to analyze information equivalence in certain cases. We give an example below.

In Section 7, we discussed how general Eqs. (3) and (5) are. We can now address this question more precisely by noting that the top and bottom of Eq. (3) have different codesignation constraints. This means that equivalence only holds under the appropriate spatial identifications. (Recall that each codesignation constraint specifies such an identification.) Hence, Eq. (5) is a relation that holds only on a quotient of configuration space. It is analogous to a "projective invariant" in geometry: an invariant relation that holds for projective space but not for affine space. To see this analogy, recall that, for example, real projective space $\mathbb{RP}^2$ is obtained as a quotient of real Euclidean space $\mathbb{R}^3$ by identifying all nonzero points on a line through the origin to a single point. There exist projective relationships in $\mathbb{RP}^2$ (for example, invariants in projective geometry) that do not hold in $\mathbb{R}^3$. In our case, it seems that by investigating the structure of these quotient relations one may measure the generality of information invariants, and, more generally, information-preserving transformations (e.g., reductions and immersions) on sensor systems.

It is interesting to note that the geometric structure of noncodesignation constraints is different from the quotient construction given above. The quotient construction can be viewed as follows. Let $\pi: C^d \to C^{d-1}$ be the projection of C^d onto C^{d-1}. This map models the quotient construction since C^{d-1} is isomorphic to $C^d/(u \sim v)$. Hence π models the identification of u and v. π then induces a new immersion $\tilde{\phi} = \pi(\phi)$:

[34] This just says that the function space C^V is isomorphic to C^d.

$$\begin{array}{ccc} \phi & \in & C^d \\ & & \downarrow \pi \\ \tilde{\phi} = \pi(\phi) & \in & C/(u \sim v). \end{array} \tag{13}$$

One the other hand, noncodesignation constraints are essentially a kind of genericity requirement. To see this, let us assume that u and v are the first and second of the d vertices of V. We then consider an immersion to be "generic" when it sends u and v to different values. Define the diagonal $\Delta = \{(z, z) \in C^2 \mid z \in C\}$. Then the noncodesignation constraint insists that we avoid the embedded diagonal, that is, we must have an immersion

$$\phi' \in (C^2 - \Delta) \times C^{d-2}. \tag{14}$$

Combining (13) and (14) gives the general form for the configuration space of the sensor.

8.9. More general codesignation relations

8.9.1. The semantics of codesignation constraints

The codesignation constraints we have encountered so far model the necessary equality of images of vertices under immersions. For example,

$$\phi(u) = \psi(v) \tag{15}$$

for (some particular) $u \in U$ and $v \in V$:

$$\begin{array}{ccc} U & \overset{\phi}{\searrow} & \\ & & C. \\ V & \overset{\psi}{\nearrow} & \end{array} \tag{16}$$

Let us call this simple kind of codesignation constraints in (15), *equality* codesignation constraints.

More generally, we could consider relations of the form "The three points z, $\phi(u)$, and $\psi(v)$ are colinear" or "$\phi(u)$ is within distance d of $\psi(v)$", etc. This other kind of codesignation constraints could be called *general codesignation relations.* We could model such a relation as follows: consider a triple (u, v, Φ) where Φ is a semi-algebraic predicate on $C \times C$. So far, in considering equality codesignation constraints, all the predicates we have used have been diagonals:[35]

$$\Phi(x, y) \quad \text{iff} \quad x = y. \tag{17}$$

This choice (Eq. (17)) explicitly encodes the assumption that all working sensor configurations can be specified using colocation (or noncolocation). For example, for the lighthouse sensor H it is necessary for the green and white lights $\boxed{\text{g}}$ $\boxed{\text{w}}$ to be colocated.

[35] For a noncodesignation constraint, we complement the diagonal.

Similarly, the sensor only works when the ship R is not at G. These statements give geometric constraints on the sensor semantics: the (non)codesignation constraints specify what (non)colocations must occur for the sensor to function properly. Hence, equality codesignation constraints such as Eq. (17) encode the assumption that the only geometric characteristic that affects sensor semantics is the colocation of components. Obviously this is not true for all sensors, but it is true for the sensors we have considered in this paper. We call such sensors *simple*, and they are worth a definition (Definition 8.17) below.

More generally, we could, in principle, require general codesignation relations to hold between component configurations—or, more generally, it may be true that there exist relationships other than (in)equality that must hold for the sensors to function properly. In this paper, we primarily discuss simple sensor systems, and only in Sections 8–9 do we consider the ramifications of such extensions. However, we feel our framework could (and should) be extended to handle at least restricted algebraic codesignation. To see how this would go, assume for a moment semi-algebraic predicates for general codesignation relations. The effect of general codesignation relations would be (geometrically) as follows. First, for a noncodesignation constraints, the "forbidden diagonal" would generalize to an arbitrary variety Y in C^d; Y would be characterized by some polynomial inequalities, and immersions $\phi \in Y$ would be forbidden. For general codesignation relations, we would construct a quotient whereby points in C^d would be identified via an algebraic map (a polynomial equation). The geometry of such spaces can be complicated; however, from a theoretical point of view, a line of attack can be seen.

We can summarize this discussion with a definition that captures the kind of sensor systems this paper addresses:

Definition 8.17. A sensor system that can be specified using only a finite number of equality codesignation (and noncodesignation) constraints is called *simple*. A sensor system that can be specified using only a finite number of semi-algebraic predicates in its general codesignation (and noncodesignation) constraints is called *algebraically codesignated*.

Since (17) is algebraically codesignated, all simple systems are algebraic codesignated. We consider only simple sensor systems in Sections 1–7. However, the algorithms in Section 9 apply to all algebraically codesignated systems.

8.9.2. The semantics of permutation

The semantics of permutations is intimately bound up in the semantics of codesignation. We now discuss the connection. The results of this section not only clarify our semantics, but also lead to a computational result, which we describe later in Section 9.

The meaning of a permutation (see Definition 8.6) is clear for a totally situated sensor system (i.e., a sensor system with a total immersion). Recall from Section 8.8 that we can view an immersion ϕ and its permutation ϕ^* as elements of the configuration

space[36] C^d. Now, suppose, for a moment, that for every immersion $\phi \in C^d$ it is possible to choose[37] a permutation ϕ^* satisfying Definition 8.6. Imagine that for each $\phi_0 \in C^d$, we build a sequence of such choices, $\{\phi_0, \phi_1, \phi_2, \phi_3, \ldots\} \subset C^d$, where $\phi_{i+1} = \phi_i^*$. This defines a map

$$\begin{array}{ccccccc} C^d & \rightarrow & C^d & \rightarrow & C^d & \rightarrow & \cdots \\ \phi_0 & \mapsto & \phi_1 & \mapsto & \phi_2 & \mapsto & \cdots . \end{array} \tag{18}$$

Hence, a permutation can be viewed as a way of "permuting" the components of a sensori-computational system, or, it may be viewed as a kind of automorphism of sensor configuration space.

Now, suppose we now allow ϕ to be a *partial* immersion. Then by a permutation ϕ^* of ϕ we mean a different partial immersion with the same domain (Definition 8.6 still applies).

Permutations of a partial immersion have a structure that is related to codesignation constraints, in that each can be characterized geometrically via regions in C^d. Consider a partial immersion ϕ. Given ϕ we can define the set of *extensions* of ϕ:

$$\mathrm{ex}\,\phi = \{\overline{\phi} \in C^d \mid \overline{\phi}|_{\phi^{-1}C} = \phi\},$$

which is a region in C^d. A permutation ϕ^* of ϕ corresponds to selecting a new region $\mathrm{ex}\,\phi^*$ of C^d, with this property:

$$\phi^{-1}C = \phi^{*-1}C. \tag{19}$$

Now, it would be convenient if we could treat the regions $\mathrm{ex}\,\phi$ and $\mathrm{ex}\,\phi^*$ like "equivalence classes" in C^d. That way we could view ϕ and ϕ^* as the "generators" of different classes of immersions. A partial function then corresponds to a region in C^d, and permutation corresponds to choice of a different region in C^d. To take this view, we need the following:

Proposition 8.18. *Let ϕ^* be a permutation of ϕ. Then* $\mathrm{ex}\,\phi$ *and* $\mathrm{ex}\,\phi^*$ *are disjoint, unless $\phi = \phi^*$.*

Proof. Let $\overline{\phi} \in \mathrm{ex}\,\phi \cap \mathrm{ex}\,\phi^*$. Since $\overline{\phi}$ is an extension of both ϕ and ϕ^*, we have

$$\overline{\phi}|_{\phi^{-1}C} = \phi, \qquad \overline{\phi}|_{\phi^{*-1}C} = \phi^*.$$

But ϕ^* is a permutation of ϕ, which implies that ϕ and ϕ^* have the same domain (Definition 8.6). Since $\phi^{*-1}C = \phi^{-1}C$, therefore $\phi = \phi^*$. □

Let $\Sigma(\phi)$ denote all permutations of ϕ. Essentially, Proposition 8.18 tells us that the map $\mathrm{ex}: \Sigma(\phi) \rightarrow \{\text{Regions in } C^d\}$ has an injection-like property: the images of distinct permutations under ex do not intersect. The map ex also has a surjection-like property which we characterize as follows:

[36] We defer the necessity of quotienting C^d and removing diagonals, until Section 10.

[37] The choice will not, in general, be unique.

Claim 8.19. *Let $\phi, \psi : V \to C$ where ϕ is partial and ψ is total. Then there exists a permutation ϕ^* of ϕ such that ψ is an extension of ϕ^*.*

Proof. Take $\phi^* = \psi|_{\phi^{-1}C}$. □

Proposition 8.20. *Fix a partial immersion ϕ. The images of* ex : $\Sigma(\phi) \to$ {Regions in C^d} *cover C^d, that is,*

$$\bigcup_{\phi^* \in \Sigma(\phi)} \operatorname{ex} \phi^* = C^d.$$

Proof. Immediate from Claim 8.19. □

We can summarize this as follows: we have viewed permutation as a bijective self-map of $\Sigma(\phi)$. It is equivalent to view permutation as a bijective self-map of the disjoint "equivalence" classes

$$\{\operatorname{ex} \phi^*\}$$

(for all permutations ϕ^* of ϕ) in C^d. This viewpoint is justified by the following two claims:

Proposition 8.21. *The map*

$$\begin{aligned} p_\phi : C^d &\to \Sigma(\phi) \\ \psi &\mapsto \phi^* \qquad \text{s.t. } \psi \in \operatorname{ex} \phi^* \end{aligned} \tag{20}$$

is well-defined.

Proof. Observe that $p_\phi(\psi) = \psi|_{\phi^{-1}C}$ (see Claim 8.19). The map p_ϕ is defined for every $\psi \in C^d$, by Propositions 8.19 and 8.20. That $p_\phi(\psi)$ is uniquely defined by (20), we see from Proposition 8.18. □

Now, suppose the domain $\phi^{-1}C$ of ϕ contains k vertices, $1 \leqslant k \leqslant d$. We can represent any permutation ϕ^* of ϕ by the k images $(z_1, \ldots, z_k)$ of the vertices of $\phi^{-1}C$ under ϕ. That is, we can represent any such permutation ϕ^* by a point in C^k. Conversely, any point in C^k defines a permutation ϕ^*.

Lemma 8.22. *The following properties hold:*[38]

(1) $\Sigma(\phi) \simeq C^k$.

(2) *The map p_ϕ is a projection and we can give it in C-coordinates as:*

$$\begin{aligned} p_\phi : \quad C^d &\to C^k \\ (z_1, \ldots, z_k, \ldots, z_d) &\mapsto (z_1, \ldots, z_k). \end{aligned} \tag{21}$$

[38] We use $\simeq$ to denote isomorphism.

(3) *Let* ϕ^* *be a permutation of* ϕ. *Then* $\operatorname{ex}\phi^* \subset C^d$ *is a cylinder over* $\phi^* \in C^k$, *and* $\operatorname{ex}\phi^* = p_\phi{}^{-1}\phi^*$.
(4) *The map* p_ϕ *is a quotient map.*
(5) $C^d/p_\phi \simeq C^k$.

Proof. Definitional. □

Finally, we note that our discussion of permutation for partially immersed sensor systems can be specialized to pointed sensor systems and pointed permutation (with the same base point). If ϕ^* is a pointed permutation of ϕ with point G, then the classes $\operatorname{ex}\phi$ and $\operatorname{ex}\phi^*$ have these additional properties (see Definition 8.7):[39]

$$G \in \operatorname{im}\phi = \bigcap_{\overline{\phi} \in \operatorname{ex}\phi} \operatorname{im}\overline{\phi}, \qquad G \in \operatorname{im}\phi^* = \bigcap_{\overline{\phi}^* \in \operatorname{ex}\phi^*} \operatorname{im}\overline{\phi}^*. \tag{22}$$

Thus for (partially) immersed systems, we have a handle on permutation, and now we know more precisely what the difference between (e.g.) H_G and H_G^* is, (see Section 5.3.1) in terms of permutation. Permutation corresponds to choosing a different equivalence class of C^d. For most of this paper we examine a special case, where the sensor systems are partially situated (that is, the domains of the immersions are nonempty). A powerful generalization is given in Section 10, where the sensor systems can be *unsituated*. This will allow us to understand the unsituated sensor system H^* precisely as a permutation of the (unsituated) system H.

8.10. The semantics of reductions

Recall the definition of *efficiently reducible* (Definition 6.4). To explore this notion, we first turn to the question of whether or not the relation $\leqslant^*$ in Definition 6.3 is transitive.

Consider three sensor systems, $\mathcal{U}$, $\mathcal{V}$, and $\mathcal{W}$, and their permutations:[40]

Sensor system	Vertices	Immersion	Permutation 1	Permutation 2
$\mathcal{U}$	$U = \{u_0, u_1, \ldots\}$	$\mathcal{U} = (U, \alpha)$		
$\mathcal{V}$	$V = \{v_0, v_1, \ldots\}$	$\mathcal{V} = (V, \beta)$	$\mathcal{V}^* = (\mathcal{V}, \beta^*)$	
$\mathcal{W}$	$W = \{w_0, w_1, \ldots\}$	$\mathcal{W} = (W, \gamma)$	$\mathcal{W}^* = (\mathcal{W}, \gamma^*)$	$\mathcal{W}^+ = (\mathcal{W}, \gamma^+)$.

(23)

If $\leqslant^*$ is transitive, then if $\mathcal{U} \leqslant^* \mathcal{V}$ and $\mathcal{V} \leqslant^* \mathcal{W}$, then $\mathcal{U} \leqslant^* \mathcal{W}$. We explore when this property holds. From Definitions 6.3 and 6.4 we can see that dominance in calibration complexity (Definition 5.1) is transitive, and so we will concentrate here on the less obvious aspects of transitivity.[41] To simplify the discussion we only deal with codesignation constraints, but the argument generalizes *mutatis mutandis* for noncodesignation constraints.

[39] For pointed sensor systems, the surjection-like properties (Propositions 8.19 and 8.20) only hold for the class of pointed immersions (with the same base point).

[40] Other permutations are possible, only a couple are shown.

[41] See Sections 8.7.4, and A.1 for more on computational calibration complexity.

8.10.1. Weak transitivity

First, let us observe that $\leqslant^*$ always obeys a property that is like transitivity, but "weaker". We now elaborate. Suppose $\mathcal{U} \leqslant^* \mathcal{V}$. Then (Definition 6.3), there exists some permutation $\mathcal{V}^* = (\mathcal{V}, \beta^*)$ of $\mathcal{V}$ such that $\mathcal{U} \leqslant \mathcal{V}^*$ (see Definition 6.2 for the definition of $\leqslant$). So, we have

$$(\mathcal{U}, \alpha) \leqslant (\mathcal{V}, \beta^*). \tag{24}$$

Now, suppose $(\mathcal{V}, \beta^*) \leqslant^* \mathcal{W}$. Then there exists a permutation $\mathcal{W}^* = (\mathcal{W}, \gamma^*)$ such that

$$(\mathcal{V}, \beta^*) \leqslant (\mathcal{W}, \gamma^*). \tag{25}$$

From Eqs. (24) and (25), and the definition of $\leqslant$ (Definitions 5.1, 6.3) we have

$$(\mathcal{U}, \alpha) \leqslant (\mathcal{W}, \gamma^*), \tag{26}$$

and therefore $\mathcal{U} \leqslant^* \mathcal{W}$. This property we call *weak transitivity*.

8.10.2. Strong transitivity for simple sensor systems

Simple sensor systems (Definition 8.17) obey *strong* transitivity, so long as all permutations are chosen to obey their codesignation constraints. Suppose $\mathcal{U}$, $\mathcal{V}$, and $\mathcal{W}$ are all simple. If $\leqslant^*$ is transitive: then, if $\mathcal{U} \leqslant^* \mathcal{V}$ and $\mathcal{V} \leqslant^* \mathcal{W}$, then $\mathcal{U} \leqslant^* \mathcal{W}$. In other words:

> Suppose $\mathcal{U} \leqslant^* \mathcal{V}$ and $\mathcal{V} \leqslant^* \mathcal{W}$. Then there exist permutations $\mathcal{V}^* = (\mathcal{V}, \beta^*)$ of $\mathcal{V}$ and $\mathcal{W}^* = (\mathcal{W}, \gamma^*)$ of $\mathcal{W}$ such that
>
> $$(\mathcal{U}, \alpha) \leqslant (\mathcal{V}, \beta^*) \tag{24}$$
>
> and
>
> $$(\mathcal{V}, \beta) \leqslant (\mathcal{W}, \gamma^*). \tag{27}$$
>
> (Compare (27) with (25)). Then if $\leqslant^*$ is transitive, then there exists another permutation $\mathcal{W}^+ = (\mathcal{W}, \gamma^+)$ of $\mathcal{W}$, such that
>
> $$(\mathcal{U}, \alpha) \leqslant (\mathcal{W}, \gamma^+). \tag{28}$$

Strong transitivity is a much stricter condition than weak transitivity. It requires that we be able to "compose" the immersions β^*, β, and γ^* to somehow construct the immersion γ^+. This may not, in general, be possible. However, it is possible for simple sensor systems, in which only equality codesignation constraints are employed to specify the system (Definition 8.17).

In order for strong transitivity to hold, we must make sure that both the permutations β and β^* for $\mathcal{V}$ and $\mathcal{V}^*$ respect the codesignation constraints for $\mathcal{V}$'s semantics. This is because we cannot expect *any* permutation of $\mathcal{W}$ to simulate $\mathcal{U}$ if either β or β^* are faulty configurations of $\mathcal{V}$. We call an immersion β^* of $\mathcal{V}$ *valid* if β^* respects the codesignation constraints for $\mathcal{V}$. This corresponds to restricting β^* to the valid regions of sensor configuration space C^d, as in Sections 8.9.2 and 10. We call a permutation

$\mathcal{V}^* = (\mathcal{V}, \beta^*)$ of $\mathcal{V}$ *valid* if its immersion β^* is valid. In this case we also say that the sensor system $\mathcal{V}^*$ is *valid*.

Lemma 8.23. *The relation* $\leqslant^*$ *(Definition 6.3) is transitive for valid simple sensor systems (Definition 8.17).*

Proof. Assume there exist valid permutations α, β, β^*, and γ^* so that (24) and (27) hold as above. We construct an immersion γ^+ so that (28) holds.

The picture we have is as follows:

$$\begin{array}{ccc} & & V \\ & & \beta \downarrow \quad \downarrow \beta^* \\ U & \xrightarrow{\alpha} & C \\ & & \gamma^* \uparrow \quad \uparrow \gamma^{-1} \\ & & W. \end{array} \tag{29}$$

Consider Fig. 9. Certain vertices (for example v_0 and u_1) are colocated. Codesignation implies colocation, but the converse is not necessarily true. In constructing a new immersion we must simulate all colocations, because that way we will be sure to reproduce all codesignation constraints accurately in the new immersion. Because (only) colocation affects sensor semantics for simple sensor systems (Definition 8.17), this suffices to ensure that the new immersion preserves the sensor semantics. In short, colocation is evidence for codesignation.

We want to construct γ^+ as follows (see Fig. 9):

$$\begin{aligned} \gamma^+ : W &\to C \\ w_i &\mapsto \beta^*(v_j) \quad \text{if } \beta(v_j) = \gamma^*(w_i). \end{aligned}$$

The general form of the set of colocations that γ^+ must simulate, is $\gamma^*(W) \cap \beta(V)$. This construction is general, and can be expressed as follows. Let

$$\begin{aligned} f : \gamma^{*-1}(\gamma^*(W) \cap \beta(V)) &\to C \\ w_i &\mapsto \beta^*(\beta^{-1}(\gamma^*(w_i))). \end{aligned}$$

The map f is almost the map we want. When the image of f is a one-point set $\{z\}$, we define $\gamma^+(w_i) = z$. If $\beta^{-1}(\gamma^*(w_i)) \subset V$ is not a singleton (see Fig. 10), then we have a choice in the construction of γ^+. In this case we know that $\gamma^+(w_i) \in f(w_i)$. Since $f(w_i)$ is finite, we can enumerate all possible candidates for γ^+; one of them will be the correct one. □

We note that our proof is not constructive: we only prove there exists a permutation $\mathcal{W}^+$. However, we can give a procedure for enumerating the finite number of candidates for the permutation γ^+. It is possible to check which is the correct one, by applying the results of the next section (Section 9).

We do not believe that the relation $\leqslant^*$ holds for arbitrary algebraically codesignated sensors. This is because the algebraic constraints may be incompatible. It would be of

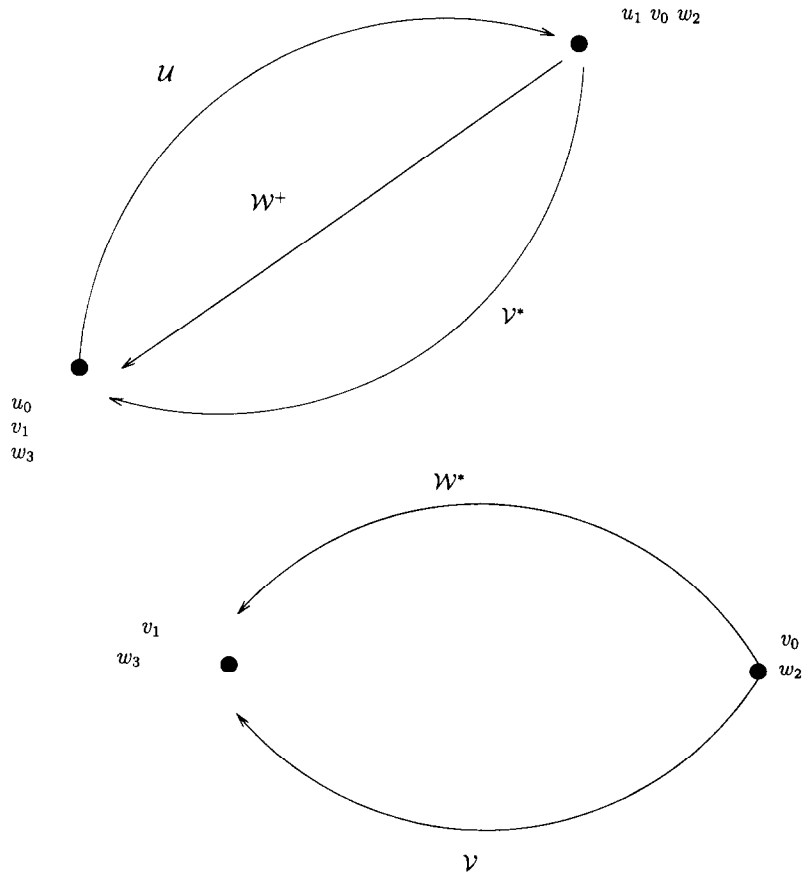


Fig. 9. The situated sensor systems $\mathcal{U} = (U, \alpha)$, $\mathcal{V} = (V, \beta)$, $\mathcal{V}^* = (\mathcal{V}, \beta^*)$, $\mathcal{W}^* = (\mathcal{W}, \gamma^*)$, and $\mathcal{W}^+ = (\mathcal{W}, \gamma^+)$ for Lemma 8.23. The vertices of $\mathcal{U}$, $\mathcal{V}$, and $\mathcal{W}$ are $U = \{u_0, u_1, \ldots\}$, $V = \{v_0, v_1, \ldots\}$ and $W = \{w_0, w_1, \ldots\}$, resp. Not all vertices are shown. $\gamma^+(w_2) = \beta^*(v_0) = \alpha(u_1)$ and $\gamma^+(w_3) = \beta^*(v_1) = \alpha(u_0)$. $\beta(v_1) = \gamma^*(w_3)$ and $\beta(v_0) = \gamma^*(w_2)$.

interest to find a restricted class that is larger than equality codesignation, for which transitivity holds.

8.10.3. A hierarchy of reductions

We now use our study of $\leqslant^*$'s transitivity to understand the reduction $\leqslant_1$ (Definition 6.4).[42] Now, even when $\leqslant^*$ is transitive, it appears that $\leqslant_1$ is not. To see this, suppose that $A \leqslant_1 B$ and $B \leqslant_1 C$. Then it appears that to reduce A to B we require one "extra wire" (namely, COMM(A)), and that to reduce B to C we could require (another) extra wire 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

[42] I would like to thank Ronitt Rubinfeld for contributing key insights to this discussion of k-wire reductions.

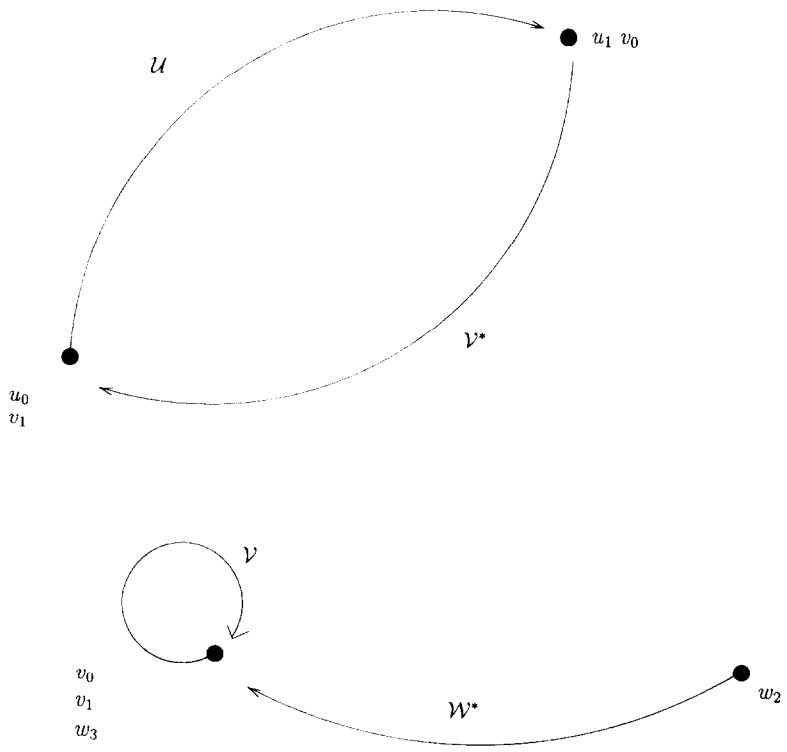


Fig. 10. The case where $\beta^{-1}(\gamma^*(w_i))$ is not a singleton (in this case, it is $\{v_0, v_1\} \subset V$). In this example, $\beta(v_0) = \beta(v_1) = \gamma^*(w_3)$. Now, we note that v_0 and v_1 colocate under β but not under β^*. However, this difference cannot be semantic (i.e., it cannot affect the sensor function), since we assume that both permutations are chosen to be valid with respect to the codesignation constraints for $\mathcal{V}$. In other words, (v_0, v_1) is not a codesignation constraint for $\mathcal{V}$ in this example.

than two extra wires. We have yet to find a non-artificial example of this lower bound but it appears to indicate that $\leqslant_1$ is not transitive (even for simple sensor systems).

Let us summarize. The reduction $\leqslant_1$ (Definition 6.4) is a "1-wire" reduction. It does not appear to be transitive. The reduction $\leqslant^*$ (Definition 6.3) is a "0-*wire*" reduction. It is transitive for simple sensor systems (Lemma 8.23). We could analogously define a 2-wire, or more generally, any *k-wire reduction* $\leqslant_k$ by modifying Eq. (4) in Definition 6.4 to

$$S \leqslant^* Q + k \cdot \mathrm{COMM}(b), \tag{4'}$$

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

Since $(\leqslant^*) = (\leqslant_0)$, this suggests a hierarchy of reductions, indexed by k. In general, we have the following:

Definition 8.24. We say a relation $\succ$ is *transitive* when $x \succ y$ and $y \succ z$ always implies

$x \succ z$. To distinguish this from *graded transitivity* (below), we call this *elementary transitivity* when necessary.[43]

We say a map $\mathcal{F}: \mathbb{N} \to 2^{X \times X}$, with $\mathcal{F}(i) = (\succ_i)$, is a *graded relation on* $X \times X$, when each $\succ_i$ is a relation on $X \times X$. We also write $\mathcal{F}$ as $\{\succ_i\}_{i \in \mathbb{N}}$.

We say that $\mathcal{F}$ *has graded transitivity* (or *is graded transitive*) if the following property holds: for every $x, y, z \in X$, if $x \succ_i y$ and $y \succ_j z$, then $x \succ_{i+j} z$.

Clearly, the k-wire reductions $\{\leqslant_i\}_{i \in \mathbb{N}}$ form a graded relation.

Corollary 8.25.

(a) *The 0-wire reduction* $\leqslant_0$ *(called* $\leqslant^*$ *in Definition 6.3) is elementary transitive for simple sensor systems.*

(b) *The k-wire reductions* $\{\leqslant_i\}_{i \in \mathbb{N}}$ *are graded transitive (Definition 8.24) for simple sensor systems.*

Proof. (a) is definitional from Lemma 8.23. To see (b), we use Lemma 8.23, and recall Definition 6.4, and that the $+$ operator is associative and commutative (Claim 8.11). To complete the argument, we also need a technical lemma, given by the "distributive" property[44] of Proposition C.3. □

We call the k-wire reductions $\{\leqslant_i\}_{i \in \mathbb{N}}$ a *hierarchy* of reductions. We say such a hierarchy (i.e., any graded relation on X^2) *collapses* if it is isomorphic to an elementary relation (i.e., to a single subset of X^2).

Corollary 8.26. *The hierarchy of k-wire reductions* $(k > 0)$ *on simple sensor systems collapses if* $\leqslant_1$ *is elementary transitive (on simple sensor systems).*

Proof. Suppose $X \leqslant_k Z$ $(k > 1)$. To collapse the hierarchy, it suffices to show that $X \leqslant_1 Z$. (This follows from Lemma 8.23, by observing that the $+$ operator is commutative and associative, and by the "distributive" property of Proposition C.3).

Now, construct k sensor systems, $Y_i = Z^* + i \cdot \text{COMM}(b_X)$, where $\log \mathbb{K}(b_X) = \text{mb}(X)$ (for $1 = i, \ldots, k$). Hence each of the i "extra wires" in Y_i has bandwidth $\log \mathbb{K}(b_X)$ (see Sections 5.4.1, 8.7.3 and Definition 6.1; to see that this yields sufficient bandwidth, see Definition 6.2(3)). So, there exist k simple sensor systems $Y_1, Y_2, \ldots, Y_k$ with $1, 2, \ldots, k$ more wires than Z resp., such that $X \leqslant_0 Y_k \leqslant_1 Y_{k-1} \leqslant_1 \cdots \leqslant_1 Y_1 \leqslant_1 Z$. Recall that $(\leqslant^*) = (\leqslant_0)$, and observe that $(\leqslant_0) \subset (\leqslant_1)$. If $\leqslant_1$ and $\leqslant^*$ are transitive, then $X \leqslant_1 Z$. □

For monotonic sensor systems, we can simply take b_X to be the output of X (see Section 6). Corollary 8.26 is stated for simple sensor systems, but it holds for the more general algebraic systems (in which case each Y_i is algebraic but not necessarily simple).

[43] Elementary transitivity is the sense used in Lemma 8.23.

[44] See Appendix C.

8.10.4. A partial order on simple sensor systems

In this section, all sensor systems are assumed to be simple.

Definition 8.27. We write $\mathcal{U} \leqslant_\infty \mathcal{V}$ if there exists some integer k such that $\mathcal{U} \leqslant_k \mathcal{V}$.

As a reduction, $\leqslant_\infty$ corresponds to adding an arbitrary amount of global, point-to-point communication. It is easy to see that $\leqslant_\infty$ is elementary transitive for simple sensor systems.

In a multi-agent sensor system, it makes sense to allow the "size" (i.e., number of components) of the system to grow, and to consider reductions parameterized by that size. For example, given a sensor system $\mathcal{U}$, we can use the notation $i \cdot \mathcal{U}$ to denote "i copies" of $\mathcal{U}$. Now, even if for another sensor system $\mathcal{V}$ we have $\mathcal{U} \leqslant_1 \mathcal{V}$, it is unlikely that we will have $i \cdot \mathcal{U} \leqslant_1 i \cdot \mathcal{V}$, for all $i \in \mathbb{N}$. However, it is easy to see the following

Claim 8.28. *If $\mathcal{U} \leqslant_k \mathcal{V}$, then for any $i, j \in \mathbb{N}$, $i \cdot \mathcal{U} \leqslant_\infty j \cdot \mathcal{V}$.*

The family $\{i \cdot \mathcal{U}\}_{i \in \mathbb{N}}$ is just one example of such a system; we could imagine other examples where the number of components, number of agents, or number of sensors varies with i. Our emphasis has changed slightly from the preceding. Before, we asked, what $k \in \mathbb{N}$ suffices such that $\mathcal{U} \leqslant_k \mathcal{V}$? Now, we ask to find that *k as a function of the size of $\mathcal{U}$ and $\mathcal{V}$.*

Now, we might deem it unfair to add arbitrary communication to the system. Let us instead consider adding only a polynomial amount of communication. In Definition 8.29, $\mathcal{U}$ and $\mathcal{V}$ are data and q is a fixed polynomial. n is the size of $\mathcal{U}$ and $\mathcal{V}$ (e.g., take n to be the total number of components). $q(n)$ (a function of n), is the amount of communication sufficient to reduce $\mathcal{U}$ to $\mathcal{V}$.

Definition 8.29. Let $\mathcal{U}$ and $\mathcal{V}$ be sensor systems. We write $\mathcal{U} \leqslant_\mathcal{P} \mathcal{V}$ if there exists some fixed polynomial function $q(n)$ of the size n of $\mathcal{U}$ and $\mathcal{V}$, such that $\mathcal{U} \leqslant_{q(n)} \mathcal{V}$ for all sizes n.

So, the assertion "$\mathcal{U} \leqslant_\mathcal{P} \mathcal{V}$" is a statement about a family of sensor systems. It says that $\mathcal{U}$ reduces to $\mathcal{V}$ by permuting $\mathcal{V}$ and adding an amount of communication that is polynomial in the size of $\mathcal{U}$ and $\mathcal{V}$. In particular, note that if $\mathcal{U} \leqslant_\mathcal{P} \mathcal{V}$, then for any $i, j \in \mathbb{N}$, $i \cdot \mathcal{U} \leqslant_\mathcal{P} j \cdot \mathcal{V}$. However, we can say something stronger:

Lemma 8.30 (Completeness of polynomial communication). *$\mathcal{U} \leqslant_\mathcal{P} \mathcal{V}$ if, and only if, $\mathcal{U} \leqslant_\infty \mathcal{V}$.*

Proof. "If" is trivial; we show the "only if" direction. If $\mathcal{U}$ and $\mathcal{V}$ have at most n vertices, then global point-to-point communication can be implemented by adding $O(n^2)$ new data-paths. Hence it is always true that $\mathcal{U} \leqslant_{O(n^2)} \mathcal{V}$. Any additional communication would be superfluous and would not add power to the system. □

It follows that $\leqslant_{\mathcal{P}}$ is elementary transitive on simple sensor systems. Therefore it is a partial order on simple sensor systems.

9. Computational properties

In this section, we give a computational model of simulation (Definition 4.1), and discuss an algorithm for deciding the relations $\leqslant^*$ and $\leqslant_1$. This section relies heavily on the results of Section 8. Readers unfamiliar with algebraic decision procedures may wish to consult the review in Appendix A, where we review some basic facts about semi-algebraic sets. This section also yields benefits in terms of clarity. For example, pointed immersions are a somewhat awkward way to specify codesignation constraints; the machinery of this section enables us to dispense with them in an elegant matter.

9.1. Algebraic sensor systems

The algorithms in this section are algebraic and use the theory of real closed fields. In the first-order theory of real closed fields, we can quantify over real variables, but not over functions. This might seem to imply that we cannot quantify over immersions of sensor systems, since these immersions are functions. However, since our immersions have a finite domain, each immersion function can be represented as a point in a configuration space C^d. Therefore we can quantify over them in our algebraic theory. We now proceed to use this fact.

Definition 9.1. We say a function is semi-algebraic when its graph is semi-algebraic.

Consider a situated sensor system $(\mathcal{U}, \phi)$, and for the moment assume that ϕ is a valid immersion that is semi-algebraic and total. Let us define the *size* d of $\mathcal{U}$ to be the number of vertices in $\mathcal{U}$. Now,

Definition 9.2. A *simulation function* $\Omega_{\mathcal{U}}$ *for* $\mathcal{U}$ is a map $\Omega_{\mathcal{U}}: C^d \to R$, where R is a ring. We call the value $\Omega_{\mathcal{U}}(\phi) \in R$ of $\Omega_{\mathcal{U}}$ on a sensor configuration ϕ to be the *sensor value* or *output value* at ϕ.

Simulation functions compute the value of the sensor given a configuration of the sensor. The idea is that we can apply a simulation function to determine what value the sensor will return—what the sensor will compute in configuration ϕ. Definition 9.2 also formalizes our notion of a "specification" for a sensor system, alluded to in the context of design (Section 1.1, application 3). See Sections 8.1 and A.2 for more on simulation functions.

Example 9.3. Recall the "lighthouse" sensor system H (Fig. 6). A simulation function Ω_H for H computes the value of θ. We imagine Ω_H works by simulating the `(orientation)` sensor (see Section 4.2.1). Other, equivalent, simulations are also possible ("equivalent" means they compute the same value for θ). For example: let

$(x, h) \in \mathbb{R}^2 \times S^1$ be the configuration of the ship R. Let $(L, \theta_0) \in \mathbb{R}^2 \times S^1$ be the configuration of the "lighthouse". Then $\theta = \theta_0 + \tan^{-1}(x - L)$. We note that this simulation function is not algebraic (because arctangent is not algebraic). See Example 9.7, below.

Now, if the configuration space C is algebraic, then so is the function space C^d. Hence, every immersion ϕ of $\mathcal{U}$ with algebraic coordinates can be represented as an algebraic point in C^d. So ϕ is algebraic exactly when it can be represented as such an algebraic point.

Now, let $\mathbb{T}$ be a predicate on C^d in the theory of real closed fields. Then $\mathbb{T}(\phi)$ is either true or false, and we can decide it by applying $\mathbb{T}$ to ϕ.

Next, suppose we now permit ϕ to be partial. We call a partial function ϕ *semi-algebraic* when its restriction $\phi|_{\phi^{-1}C}$ to its domain $\phi^{-1}C$ is semi-algebraic. If ϕ is semi-algebraic, then the set of its extensions $\operatorname{ex}\phi \subset C^d$ is also semi-algebraic. We then observe that the expression denoting "for all extensions (resp., there exists an extension) $\overline{\phi}$ of ϕ, $\mathbb{T}(\overline{\phi})$ holds" namely

$$\Diamond\overline{\phi} \in \operatorname{ex}\phi\colon \mathbb{T}(\overline{\phi})$$

is also semi-algebraic ($\Diamond \in \{\forall, \exists\}$). To quantify over all extensions $\overline{\phi}$ of ϕ, we simply quantify over the configurations of the vertices *outside* the domain of ϕ. By Section 8.9.2 we can also "guess" permutations of ϕ—that is, it is possible to existentially quantify over permutations and hence to decide sentences of the form[45]

$$\exists\phi^* \in \Sigma(\phi)\colon \mathbb{T}(\phi^*)$$

which means, "there exists a permutation ϕ^* of ϕ, such that for any extension $\overline{\phi^*}$ of ϕ^*, $\mathbb{T}(\overline{\phi^*})$ holds". That is,

$$\exists\phi^* \in \Sigma(\phi),\ \forall\overline{\phi^*} \in \operatorname{ex}\phi^*\colon \mathbb{T}(\overline{\phi^*}). \tag{30}$$

To guess a permutation of ϕ we existentially quantify over the configurations of vertices *inside* the domain of ϕ.

Example 9.4. Let C be an algebraic configuration space. Let V be a set of three vertices, $V = \{v_1, v_2, v_3\}$. Now, we can encode any algebraic function $\psi\colon V \to C$ semi-algebraically, e.g., by a set of three ordered pairs $\{(v_1, z_1), (v_2, z_2), (v_3, z_3)\}$, where $\psi(v_i) = z_i$, $(i = 1, 2, 3)$. Let us call such an s.a. representation of ψ by the name $\sigma(z_1, z_2, z_3)$:

$$\sigma(z_1, z_2, z_3) = \{\psi\colon V \to C \mid \psi(v_i) = z_i,\ (i = 1, 2, 3)\}.$$

Now, consider a partial immersion $\phi\colon V \to C$ with domain $\{v_1\}$, such that $\phi(v_1) = G$, where G is algebraic. We can encode ϕ as

$$\exists z_2 \exists z_3 : \sigma(G, z_2, z_3).$$

[45] We call the existential quantification "guessing", since deciding a predicate in the existential theory of the reals is like guessing a witness to make the predicate true.

We can also encode the extensions and permutations of ϕ semi-algebraically. Specifically, we can encode any permutation ϕ^* of ϕ by a single point z_1 (the image of v_1); we can encode any extension $\overline{\phi^*}$ of ϕ^* by a pair (z_2, z_3) (the images of v_2 and v_3, respectively).

Thus, we can rewrite (30) as

$$\exists z_1 \; \forall z_2 \; \forall z_3 : \mathbb{T}(\sigma(z_1, z_2, z_3)). \tag{31}$$

If C has dimension r_c, then the formula (31) is a Tarski sentence in $3r_c$ variables.

We summarize:

Proposition 9.5. *If $\phi : V \to C$ is a semi-algebraic partial function, then the set* ex ϕ *(ϕ's extensions) and the set $\Sigma(\phi)$ (ϕ's permutations) are also semi-algebraic.*

To guess a *valid* permutation, (Definition 8.10.2) we restrict the configurations to lie within the (algebraic) codesignation constraints, as described in Sections 8.9.2 and 10. (We are simply using algebraic decision procedures to make these choices effective.) Any s.a. codesignation constraints for an algebraically codesignated sensor system can be represented by an s.a. set $D \subset \mathbb{R}^r$. The structure of the region D, especially in relation to the region ex ϕ^*, is discussed in Sections 8.9.2 and 10. We must restrict our choice of permutation to D. To guess a valid permutation, we modify (30) to be:

$$\exists \phi^* \in \Sigma(\phi), \; \forall \overline{\phi^*} \in D \cap \text{ex}\, \phi^* : \mathbb{T}(\overline{\phi^*}). \tag{32}$$

Definition 9.6. We call a sensor system $\mathcal{U}$ *algebraic* if it is algebraically codesignated (Definition 8.17), has an algebraic configuration space C, and a semi-algebraic algebraic simulation function $\Omega_{\mathcal{U}}$ (Definition 9.2).

Example 9.7. Recall Example 9.3, above. The simulation function Ω_H in Example 9.3 is not algebraic. However, we can define a (semi-)algebraic simulation function that encodes the same information, and is adequate, in the sense that we can use it to compare the sensor H's function to another orientation sensor. The algebraic simulation function we give now is adequate to decide the relation $\leqslant^*$.

To construct an algebraic version of Ω_H, we use a simple trick from calculus (also used in kinematics; see, for example [18]). Let ϕ be a configuration of sensor system H (Fig. 6). Define $\Omega'_H(\phi) = (\tan(\theta/2), q)$, where $\theta = \Omega_H(\phi)$ (see Example 9.3), and $q \in 4$ denotes which quadrant R is in relative to L. Ω'_H encodes the same information as Ω_H, but it is semi-algebraic.

We will not prove Ω'_H is algebraic but here is a brief argument. Substitute $u = \tan(\theta/2)$. Then we have $\sin\theta = (1 - u^2)/(1 + u^2)$ and $\cos\theta = 2u/(1 + u^2)$, and our simulation function is a rational function. By clearing denominators we obtain an algebraic function. See [18] for details. Essentially the graph of Ω'_H is an s.a. set in correspondence with the graph of the non-algebraic map Ω_H. The correspondence is given by $\theta \mapsto u$.

9.2. Computing the reductions $\leqslant^*$ and $\leqslant_1$

Now, suppose we have two algebraic sensor systems $\mathcal{U}$ and $\mathcal{V}$. We wish to decide whether $\mathcal{U} \leqslant^* \mathcal{V}$. If $\mathcal{U} = (\mathcal{U}, \alpha)$ and $\mathcal{V} = (\mathcal{V}, \beta)$, then we wish to decide whether there exists a permutation β^* such that

$$(\mathcal{U}, \alpha) \leqslant (\mathcal{V}, \beta^*).$$

(Here in Section 9.2 the relation $\leqslant$ is used as in Definition 6.2). That is, we wish to decide the following (assume that α and β are partial):

$$\left(\exists \beta^* \in \Sigma(\beta), \forall \overline{\alpha} \in \operatorname{ex} \alpha, \forall \overline{\beta^*} \in \operatorname{ex} \beta^*\right) : \ \Omega_{\mathcal{U}}(\overline{\alpha}) = \Omega_{\mathcal{V}}(\overline{\beta^*}). \tag{33}$$

Eq. (33) does not incorporate the codesignation constraints. Since $\mathcal{U}$ and $\mathcal{V}$ are algebraically codesignated, their codesignation constraints may be represented as semi-algebraic sets $D_{\mathcal{U}}$, $D_{\mathcal{V}}$, and $D_{\mathcal{V}\mathcal{U}}(\overline{\alpha})$ in C^d. So (33) becomes:

$$\begin{aligned}&\left(\exists \beta^* \in (\Sigma(\beta) \cap D_{\mathcal{V}}), \forall \overline{\alpha} \in (\operatorname{ex} \alpha \cap D_{\mathcal{U}}), \forall \overline{\beta^*} \in (\operatorname{ex} \beta^* \cap D_{\mathcal{V}\mathcal{U}}(\overline{\alpha}))\right) : \\ &\quad \Omega_{\mathcal{U}}(\overline{\alpha}) = \Omega_{\mathcal{V}}(\overline{\beta^*}).\end{aligned} \tag{34}$$

Note that $\mathcal{V}$'s codesignation constraints depend on $\overline{\alpha}$: that is, the s.a. set $D_{\mathcal{V}\mathcal{U}}(\overline{\alpha})$ is an s.a. function of $\overline{\alpha}$. This technicality is necessary to allow for sufficient generality in specifying codesignation, and is explained further in Section A.3.

Using Grigoryev's algorithm (Theorem A.1) we can decide (34) in the following time. (We use (A.5) below to compute the time bound). Let n_Ω be the size of the simulation functions $\Omega_{\mathcal{U}}$ and $\Omega_{\mathcal{V}}$. Let r_c be the dimension of C. Let n_D be the size of the s.a. predicates for the codesignation constraints $D_{\mathcal{U}}$, $D_{\mathcal{V}}$, and $D_{\mathcal{V}\mathcal{U}}$. In (34), the outer existential quantifier binds some number $k \leqslant d$ vertices of $\mathcal{V}$ that are in the domain of ϕ. The inner universal quantifier binds the remaining $d-k$ vertices of $\mathcal{V}$. The middle universal quantifier binds up to d vertices of $\mathcal{U}$. Hence, we see there are at most $r = 2r_c d$ variables, and there are $a = 2$ alternations. Let us treat the maximum degree δ as a constant. The predicate has size $m = 2(n_\Omega + n_D)$. Therefore we can decide (34) in time

$$(m\delta)^{O(r)^{4a-2}} = (n_\Omega + n_D)^{O(r_c d)^6}. \tag{35}$$

Definition 9.8. Consider an algebraic sensor system $\mathcal{U}$, with d vertices. Recall we call d the *size* of $\mathcal{U}$. We call the size n_Ω of a sensor simulation function $\Omega_{\mathcal{U}}$ the *simulation complexity*. We call the size n_D of the codesignation constraints for $\mathcal{U}$ the *codesignation complexity*. We call $\mathcal{U}$ *small* if n_Ω and n_D are only polynomially large in d, i.e., $(n_\Omega + n_D) = d^{O(1)}$.

Now, let us assume that it is possible to compute dominance in calibration complexity (see Definition 6.2) in a time that much faster than (35) (see Section A.1 for how). Then we see the following

Lemma 9.9. *There is an algorithm for deciding the relation $\leqslant^*$ (Definition 6.3) 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*:

$$(n_\Omega + n_D)^{(r_c d)^{O(1)}}, \tag{36}$$

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

Corollary 9.10. *For small*[46] *sensor systems (Definition 9.8) of size d, there is an algorithm to decide the relation* $\leqslant^*$ *in time*

$$d^{(r_c d)^{O(1)}}. \tag{37}$$

Corollary 9.11. *For algebraic sensor systems, the relation* $\leqslant_1$ *(Definition 6.4) can be decided in the same time bounds as in Lemma 9.9 and Corollary 9.10.*

Proof. Consider deciding $S \leqslant^* Q + \text{COMM}(b)$, as in Definition 6.4. Recall the definition of *compatibility* for partial immersions (Section 8.4). We first observe that permutation (the * operation) and combination (the + operation) "commute" for compatible partial immersions. This is formalized as the "distributive" property[47] shown in Proposition C.3. We have already shown how to guess a permutation Q^* of Q. Our arguments above for guessing extensions and permutations can be generalized *mutatis mutandis* to compute the combination (Definition 8.10) of two algebraic sensor systems. Since COMM(b) is a constant-sized sensor system (two vertices, one edge) with only a constant number of codesignation constraints (at most 2), we may guess how to combine it with a permutation Q^* of Q within the same time bounds given in Lemma 9.9 and Corollary 9.10. To complete the proof we require a technical argument (given in Appendix A.2) on how to simulate a permuted sensor system. □

10. Unsituated permutation

In Section 9 we examined a special case, where $\mathcal{U}$ and $\mathcal{V}$ are partially situated (that is, the domains of ϕ and ψ are non-empty). We now give a powerful generalization in which the sensor systems can be *unsituated*. Using the ideas in Sections 8.9.2 and 9, 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 9. Suppose a simple sensor system $\mathcal{U}$ has d vertices, two of which are u and v. When there is a codesignation constraint

[46] Recall all small systems are algebraic.

[47] See Appendix C.

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

$$\{\pi(\operatorname{ex}\psi^* - \Delta)\}_{\psi^* \in \Sigma(\psi)}. \tag{38}$$

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 Eq. (38) 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 ijth 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 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 9 can be extended to guess unsituated permutations. In our examples, each guess (i.e., each unsituated permutation) corresponds to a choice of which vertices to colocate.[48] All our computational results (including our bounds) in Section 9 can be shown to hold for unsituated permutation by a simple extension of the arguments above.

10.1. Example of unsituated permutation

Unsituated permutation is quite powerful. Consider deciding Eq. (34) (in this example, we only consider vertex permutation of simple sensor systems). In particular, we

[48] 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 9.

want to see that (33) makes sense for unsituated permutation, when we replace β by β_0, α by α_0, etc., to obtain:

$$\begin{aligned}&\big(\exists \beta_0^* \in (\Sigma(\beta_0) \cap D_{\mathcal{V}}),\\ &\forall \overline{\alpha_0} \in (\mathrm{ex}\,\alpha_0 \cap D_{\mathcal{U}}),\\ &\forall \overline{\beta_0^*} \in (\mathrm{ex}\,\beta_0^* \cap D_{\mathcal{VU}}(\overline{\alpha_0}))\big) :\\ &\quad \Omega_{\mathcal{U}}(\overline{\alpha_0}) = \Omega_{\mathcal{V}}(\overline{\beta_0^*}).\end{aligned} \tag{34'}$$

With situated permutation (34), we are restricted to first choosing the partial immersion α, 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 β^*). This process gets the colocations right, but at the cost of generality; we would know that for any "topologically equivalent" choice of α, we can choose a permutation β^* such that (34) holds. For simple sensor systems, "topologically equivalent" means, "with the same vertex colocations".

Unsituated permutation (34′) allows us to do precisely what we want. In place of a partial immersion α for $\mathcal{S}$, we begin with a witness point $\alpha_0 \in C^d$. α_0 represents an equivalence class $\mathrm{ex}\,\alpha_0$ of immersions, all of which colocate the same vertices as α_0. So, α_0 says *which* vertices should be colocated, but not *where*. Now, given α_0, the outer existential quantifier in (34′) chooses an unsituated permutation β_0^* of $\mathcal{U}$. β_0^* represents an equivalence class $\mathrm{ex}\,\beta_0^*$ of immersions of $\mathcal{U}$, all of which colocate the same vertices of $\mathcal{U}$ as β_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 $\Sigma(\beta_0)$ $(= \Sigma(\beta_0^*))$. Choice of β_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: α_0 (given as data) determines which vertices of $\mathcal{S}$ to colocate; the choice of β_0^* determines which vertices of $\mathcal{U}$ are colocated; construction of $D_{\mathcal{S}}(\cdot)$ determines which vertices of $\mathcal{U}$ and $\mathcal{S}$ are colocated. Most specifically, given the configuration $\overline{\alpha_0}$ of $\mathcal{S}$, $D_{\mathcal{S}}$ in turn defines a region $D_{\mathcal{S}}(\overline{\alpha_0})$ in the configuration space C^d of $\mathcal{U}$. This region constraints the necessary coplacements $\overline{\beta_0^*}$ of $\mathcal{U}$ relative to $(\mathcal{S}, \overline{\alpha_0})$.

11. Application and experiments

We now describe an application of the theory in this paper, presented in [17]. This work is still preliminary, but we describe it here to give some feeling for the potential of our theory. The paper [17] relies heavily on the results and methods introduced here. Donald et al. [17] quantify a new resource: (f) *How much information is encoded or provided by the task mechanics*? The theme of exploiting task mechanics is important in previous work.[49] One could define "exploiting task mechanics" for robot manipulation as: *taking advantage of the mechanical and physical laws that govern how objects move and change*. Currently, in our framework the mechanics are embedded in the geometry

[49] For example, see the discussion of [25,26,40] in Part I.

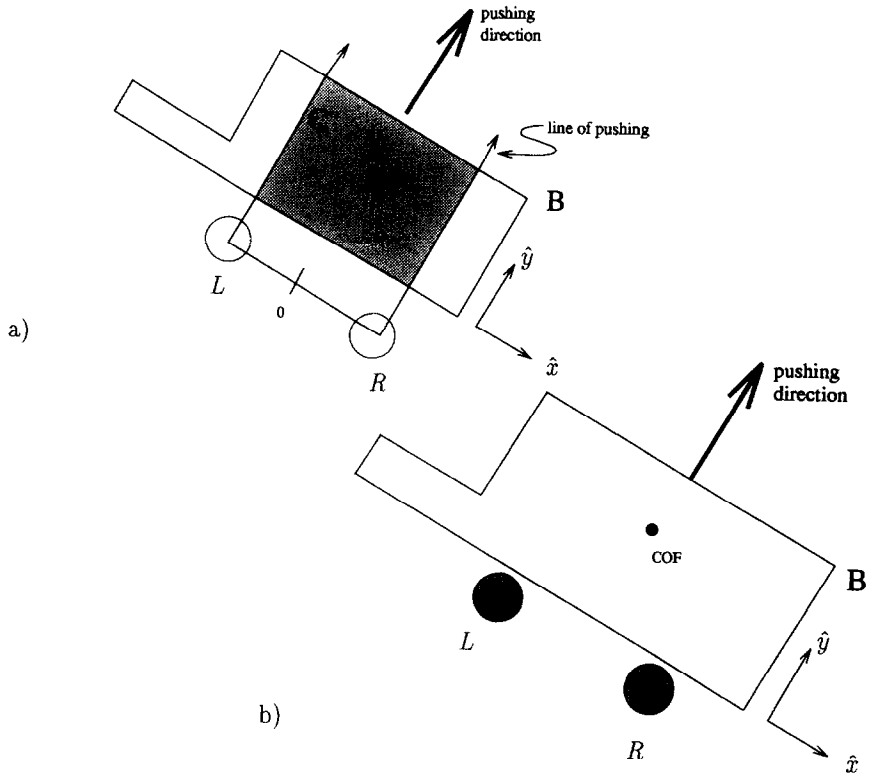


Fig. 11. (a) the "two-finger" pushing task versus (b) the two robot pushing task. The goal is to push the block B in a straight line.

of the system. In [17], we developed information invariants that explicitly trade-off (f) with resources (a)–(e) from the abstract, in the style of the preceding. Developing such invariants is quite challenging. We close with an example. This example opens up a host of new issues; see [17] for details.

Fig. 11(a) depicts a two-finger planar pushing task. The goal is to push the box B in a straight line (pure translation). The two fingers f_1 and f_2 are rigidly connected; 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. 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 somewhat and the fingers can keep pushing, since we only need ensure the COF lies in some region C (see Fig. 11(a)), instead of on a line. Second, if the COF moves outside C, then the fingers can move sideways to capture it again. For example, in [17] we implement the following control loop on our PUMA: *Sense the reaction torque τ about the point 0 in Fig. 11(a). If $\tau = 0$, push forward in*

```
(do-forever
  (let ((τ (measure-torque)))
    (cond ((zero? τ) (push ŷ))
          ((negative? τ) (move +x̂))
          ((positive? τ) (move −x̂)))))
```

Fig. 12. Protocol P1 (for a two-fingered gripper).

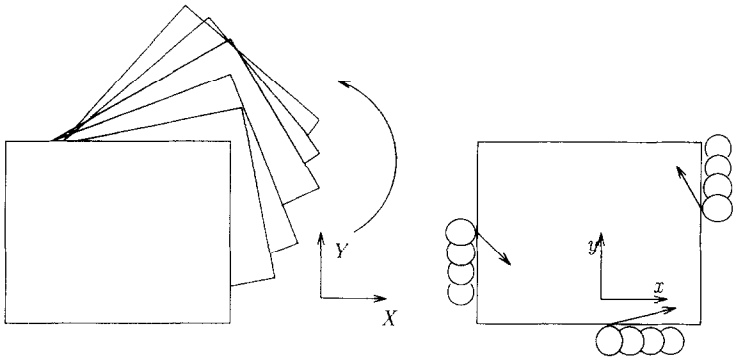
direction $\hat{y}$. *If* $\tau < 0$ *move the fingers in* $\hat{x}$; *else move the fingers in* $-\hat{x}$. See Fig. 12.

From the mechanics perspective it might appear we are done. However, it is difficult to overstate how critically the control loop (Fig. 12) relies on global communication and control. Now, consider the analogous pushing task in Fig. 11(b). Each finger is replaced by an autonomous mobile robot with only local communication, configured as described in Section 2.1 of Part I. Each robot has a ring of one-bit contact ("bump") sensors. In addition, by examining the servo-loop in [44], 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.

Now, we ask, how can the system in Fig. 11(b) approximate the pushing strategy (Fig. 12), above? We observe the following. Each robot can compute its applied force and contact mode, and communicate these data to the other. The robots together must perform a control strategy (move in $\hat{y}$, move in $\pm\hat{x}$, etc.). Since the robots are not rigidly linked, there are five qualitative choices on how to implement a move in $\pm\hat{x}$. Our experiments suggest these strategies are aided by the ability to sense the box's surface normal, and to compliantly align to it. The IR-Modem mechanism described in Part I allows the communication of the following information: each robot's identity, orientation, and speed. In addition here are several kinds of information a robot might transmit for the pushing task: whether it is in contact with the box, the contact "bearing" (where the contact is on the bumper ring), the power being applied to the motors, and the local surface normal of the box. Next, a robot could communicate the message "Do this strategy: ..." or else "I am about to do this strategy: ..." Finally, the robots may have to transmit communication primitives like "Wait" and "Acknowledged".

While it is possible to specify and indeed implement sufficient communication to perform this task robustly, it is difficult to convince oneself that some particular communication scheme is optimal, or indeed, even necessary.

In [17], we analyze information invariants for manipulation tasks using the formalism presented here. For example, it is clear that the surface normal computation requires some internal state, and the compliant align can be viewed as consuming external state or as temporary calibration. Communication appears fundamental to performing the task in Fig. 11(b). So we ask: what communication is necessary between the robots to accomplish the (2-robot) pushing task? How many messages and what information is required? In [17] we use the methods introduced here to compare and contrast pushing protocols, and to answer these questions. First, we precisely describe two manipulation tasks for cooperating mobile robots that can push large, heavy objects. One task is shown



a) The box motion b) The robot motions

Fig. 13. The task is to rotate the box by a specified angular amount. Here we illustrate the box 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. From [17].

in Fig. 11(b), the other in Fig. 13. More specifically, we ask: *Can all explicit local and global communication between the agents be removed from a family of pushing protocols for these tasks?*[50] Donald et al. [17] answer in the affirmative—a surprising result—by using the general methods introduced here for analyzing information invariants.

11.1. Using circuits and reductions to analyze information invariants

In [17], 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 one to two times the mass of a single robot, although the robots have also pushed couches that are heavier (perhaps two to four times the mass, and 8×3 robot diameters in size). We build on the ground-breaking work of Mason and Erdmann [26,40] 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 (the COF) is known. Instead, the robot combines sensorimotor experiments and manipulation strategies to infer the necessary information (the experiments have the flavor of [33]). 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.

[50] This question was first posed as an open problem in a 1992 draft of this paper.

Of course, our protocols rely on a number of assumptions in order to work. We use the theory of information invariants developed here, 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[51] protocol with local communication and partial synchrony. Finally, we remove all explicit communication. The final protocols are asynchronous, and essentially "uniform", or SPMD[51]—the same program 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 believe that using our theory of information invariants is justified.

The manipulation protocols in [17] are first modeled as circuits, using the formalism developed in Section 8. Source-to-source transformations on these protocols are then represented as circuit transformations. The circuit transformations are modeled using the reductions described in this paper. For the task in Fig. 11b, [17] consider three pushing protocols P1, P2, and P3, and their interreducibility under $\leqslant_1$. In particular, we transform an MPMD pushing protocol P2 with explicit IR communication to an asynchronous SPMD protocol P3 with no explicit communication. This transformation is then analyzed as an instance of reducing the latter to the former, using $\leqslant_1$. 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. For example, protocol P2 uses explicit communication and protocol P3 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.

In [17], we also consider three protocols R1, R2, and R3, for a reorientation task (see Fig. 13). A transformational approach to developing these protocols is viewed as a series of reductions. The final protocol R3 has several advantages over the initial protocols R1 and R2. Using protocol R3, 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 R3 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 R1 and R2.

In terms of program development, synchrony, and communication, we have a correspondence between these protocols, shown in Fig. 14. We believe that a methodology for developing coordinated manipulation protocols is emerging, based on the tools described

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

Pushing task	Reorientation task	
P1	R1	global coordination and control
P2	R2	local IR communication, partial synchrony, MPMD
P3	R3	uniform (SPMD), asynchronous, no explicit communication

Fig. 14. Summary of parallel manipulation protocols from [17].

here. This methodology helps transform an offline, synchronous manipulation strategy (e.g., P1 or R1) with global coordination and control, into an online, asynchronous, distributed strategy (P3 or R3) for the same task:

Developing parallel manipulation protocols [17]

(1) Start with a sensorless [26] or near-sensorless [33,47] manipulation protocol requiring global coordination of several "agents" (e.g., parallel-jaw fingers, or fingers of a dexterous hand).
(2) Distribute the protocol over spatially separated agents. Synchronize and coordinate control using explicit local communication.
(3) Define *virtual sensors*[52] for the quantities Step (2) measures.
(4) Implement each virtual sensor using concrete sensors on mechanical observables.
(5) Transform the communication between two agents L and R into shared data structures.
(6) Implement the shared data structures as "mechanical registers".

Our circuits model the protocols in the steps above. Our reductions model the transformations between steps. By the results of Section 9, these reductions can be effectively computed. Therefore, in principle, the transformations in [17] could be synthesized automatically. We believe that our methods are useful for developing parallel manipulation protocols. We have implemented and tested our asynchronous, distributed, SPMD manipulation protocols using TOMMY and LILY, and found them robust and efficient. See [17] for a full discussion.

12. Conclusions

In this paper we suggested a theory of information invariance that includes sensors and computation. Our results generalize the work of [1]; first, we consider fairly detailed yet abstract models of physical autonomous mobile robots; second, we consider generalizations and variations on compasses and orientation sensors; third, we develop a generalized and stratified theory of the "power" of such sensori-computational devices. As such, perhaps our work could be called *On the generalized power of generalized compasses.*

[52] We use the term in the sense of [14]; others, particularly Henderson have used similar concepts. See Section 4.2.1 for examples of virtual and concrete sensors.

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 into the control system or the environment. We believe that the equivalencies 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. Our paper proposes a beachhead on information invariance from which, we hope, such goals may be obtained. There are several things we have learned. First, we were surprised by how important *time* and *communication* become in invariant analysis. Much insight can be gained by asking *How can this sensor be simulated by a simpler system with a clock* (*resp. communication*)? Time-based sensors are ubiquitous in modern aircraft navigation systems (compare Section 4.2.1). In "DMEs" (*distance measuring equipment*) a ground station and the plane talk to each other, and measure differences in timing pulses to estimate their distance apart. GPS, which was approved in July, 1993 for use in airplanes, also operates on timing principles.

Robot builders make claims about robot performance and resource consumption. In general, it is hard to verify these claims and compare the systems. One reason is calibration: pre-calibration can add a great deal of information to the system. In order to quantify the "use" of external state, we suggested a theory of calibration complexity. Our theory represents a systematic attempt to make such comparisons based on geometric and physical reasoning. Finally, we try to operationalize our analysis by making it computational; we give effective (albeit theoretical) procedures for computing our comparisons. Our algorithms are exact and combinatorially precise.

Our reduction $\leqslant_{\mathrm{I}}$ (Definition 6.4) attempts to quantify when we can "efficiently" build one sensor out of another (that is, build one sensor using the components of another). Hence, we write $A \leqslant_{\mathrm{I}} B$ 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 (see Appendix B). In this sense, our "components" are analogous to *oracles* in the theory of computation. Hence, we write $A \leqslant_{\mathrm{I}} B$ if we can build a sensor that simulates A, using the components of B, plus "a little rewiring". A and B are modeled as *circuits*, with wires (data-paths) 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.

Permutation models the permissible ways to reallocate and reuse resources in building another sensor. *Codesignation constraints* further restrict the range of admissible permutations. *Output communication* formalizes our notion of "a little bit of rewiring". Like CT reductions, $A \leqslant_{\mathrm{I}} B$ defines an "efficient" transformation on sensors that takes B to A. However, we give a generic algorithm for synthesizing our reductions (whereas

no such algorithm can exist for CT[53]). Whether such reductions are widely useful or whether there exist better reductions (e.g., our *"k-wire"* reductions in Section 8.10.3) is open; however in our laboratory we are using $\leqslant_1$ to design manipulation protocols [17] for multiple mobile robots. We also give a "hierarchy" of reductions, ordered on power, so that the strength of our transformations can be quantified. See Appendix A.4 for a discussion of "universal reduction" as per Section 1.1. See Appendices B and C.3 for more on relativized information complexity.

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. We discuss these issues further in [17].

By analogy with CT reductions, we may define an equivalence relation $=_k$, such that $A =_k B$ when $A \leqslant_k B$ and $B \leqslant_k A$. We may also ask, does a given class of sensori-computational systems contain "complete" circuits, to which any member of the class may be reduced? Note that the relation $=_k$ holds between any two complete circuits.

Weaker forms of sensori-computational equivalence are possible. If we define the *state* of a sensor system $\mathcal{U}$ to be a pair (z, b) where z is the configuration of the system and b is the output value at z, we can examine the equilibrium behavior of $\mathcal{U}$ as it evolves in state space. Recall Definition 4.1; let us call this *strong simulation*. 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.[54] If we replace strong simulation ($\cong$ in Definition 4.1) 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.

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 sensorimotor experiments.

12.1. Future research

This paper represents a first stab at several difficult problems, and there are a number of issues that our formalism does not currently consider. We now acknowledge some of these issues here.

[53] 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.

[54] I am grateful to Dan Koditschek, who has suggested this formalism in his papers.

Our theory allows us to compare members of a certain class of sensor systems, and, moreover, to transform one system into another. However, it does not permit one to judge which system is "simpler" or "better" or "cheaper". In particular, for a given measurement problem, it does not permit a "simplest" sensor system to be identified. There are several reasons for this. The first is that there are inherent limitations on comparing absolute sensor complexity—and these problems represent structural barriers to obtaining good notions of "better" or "simpler". The theory is designed, in part, to get around some of these limitations. We discuss these problems—which are quite deep—in Appendix B at some length. Second, such comparisons would require an explicit performance measure. We discussed such measures as speed (or execution time) in Section 2.1.2. In Section F.1, we argue that such performance measures allow us to apply *kinodynamic* analysis tools [21,49]. There is no doubt that external performance measures such as "simpler" and "better" and "cheaper" could be used with our framework—but we don't know what exactly these measures are. It appears that efficient algorithms for exploiting these measures will have to take advantage of their structure.

Instead of investigating performance measures, we have argued that it is very hard to even measure or compare the "power"of sensorimotor systems. To address this problem, we developed our reductions. To make our stance clear, consider as an analogy the theory of computation (CT). CT does not tell us which algorithms are more "simple", but it does tell us which are more powerful (i.e., which can compute more). In our theory, as in CT, we can define transformations or reductions that we consider "fair", and then discuss equivalence of systems up to these transformations. Now, in CT, given performance measures (e.g., asymptotic complexity) we can also compare the performance of algorithms—although there are many different measures to choose from. But in CT, "faster" does not necessarily mean "simpler" in any sense. Our reductions are analogous to CT reductions. Execution time or speed is analogous to computational complexity. Finally, as in CT, notions of "simplicity" are orthogonal to notions of either reduction or performance.

However, the notion of performance measures opens up a host of practical issues.[55] Certainly some simple scheme of looking up the "cost" of components in a table could be used in conjunction with our system. An instrumentation engineer, confronted by a problem where one measurement strategy is ineffective, may choose to measure some other property to solve the problem rather than reconfigure the sensori-computational components of the system (for example, measuring the temperature rather than the pressure of a fixed volume of gas). This approach is not envisaged by our theorems, although the power of two given strategies could be compared. Furthermore, distinct measurement strategies have costs other than those considered here—for example, the cost of transducers, the effect on the measurement noise of measuring one observable and inferring another from its value, noise properties of transducers and common mode effects (for example, in positioning strain gauges). These issues should be considered in future work.

There is much to be done. Our model of reduction is very operational and others should be attempted. In addition to measuring the information complexity of commu-

[55] I would like to thank the anonymous referees for suggesting these issues and the wording to describe them.

nication, it may be valuable to quantify the *distance* messages must be sent. Similarly, it may make sense to measure the "size" of a resource permutation, or how far resources are moved. All these ideas remain to be explored. Finally, we have approached this problem by investigating information *invariance*, that is, the kind of information-preserving equivalencies that can be derived among systems containing the resources (a)–(e) (Section 3). An alternative would be to look at information *variance*, that is, it would be valuable to have a truly uniform measure of information that would apply across heterogeneous resource categories.

In the appendices we present a number of important extensions, and attempt to address some of the issues raised in this section.

Acknowledgments

This paper could never have been written without discussions and help from Jim Jennings, Mike Erdmann, Dexter Kozen, Jeff Koechling, Tomás Lozano-Pérez, Daniela Rus, Pat Xavier, and Jonathan Rees. I am very grateful to all of them for their generosity with their time and ideas. The robots and experimental devices described herein were built in our lab by Jim Jennings, Russell Brown, Jonathan Rees, Craig Becker, Mark Battisti, Kevin Newman, Dave Manzanares, and Greg Whelan; these ideas could never have come to light without their help and experiments. I would furthermore like to thank Mike Erdmann, Jim Jennings, Jonathan Rees, John Canny, Ronitt Rubinfeld, Seth Hutchinson, Sundar Narasimhan, and Amy Briggs for providing invaluable comments and suggestions on drafts of this paper. Thanks to Loretta Pompilio for drawing the illustration in Fig. 1. Debbie Lee Smith and Amy Briggs drew the rest of the figures for this paper and I am very grateful to them for their help. I am grateful to Jeff Koechling, Mike Erdmann, and Randy Brost for explaining to me how lighthouses, ADFs, and VORs work. This paper was improved by incorporating suggestions made at the *Workshop on Computational Theories of Interaction and Agency*, organized at the University of Chicago by Phil Agre and Tim Converse. I would like to thank Phil Agre, Stan Rosenschein, Yves Lesperance, Brian Smith, Ian Horswill, and all members of the workshop, for their comments and suggestions. I would also like to thank the anonymous referees for their comments and suggestions. I am grateful to Phil Agre who carefully edited this paper and made many invaluable suggestions on presentation.

Appendix A. Algebraic decision procedures

The algorithms in Section 9 are algebraic and use the theory of real closed fields;[56] for an introduction to algebraic decision procedures see, for example the classic paper of [2], or discussions in books such as [18, Chapters 1–4] and [10]. In Section 9, we reduce our computational problem to deciding the truth of a Tarski formula [48];

[56] Also called "Tarski's Language" or the "first-order language of algebra and geometry". One common mathematical term is "the first-order theory of real closed fields".

the algebraic algorithms can then decide such a sentence. Tarski's language is also called the *language of semi-algebraic sets.* Such sets are real semi-algebraic varieties defined by polynomial equalities and inequalities, where the polynomial coefficients are algebraic numbers. A Tarski formula is a logical sentence that quantifies existentially or universally over each of the real variables. A typical Tarski formula might be:

$$\begin{aligned}(\forall x \exists y \exists z \forall w) : \; & xy^2 - 16w^4 \leqslant 0 \\ & \vee \tfrac{3}{4} xw^2 + x^7 + 78w < 0 \\ & \wedge z^4 + 5w^3 + 4x^2y^2 - y + 7x = 0.\end{aligned} \tag{A.1}$$

More generally, we can think of a Tarski sentence as

$$\begin{aligned}(\Diamond_1 x_1 \; \Diamond_2 x_2 \; \cdots \; \Diamond_r x_r) : \; & s_1(x_1, \ldots, x_r) \; R_1 \; 0 \\ & C_1 \; s_2(x_1, \ldots, x_r) \; R_2 \; 0 \\ & \vdots \\ & C_{m-1} \; s_m(x_1, \ldots, x_r) \; R_m \; 0,\end{aligned} \tag{A.2}$$

where each $\Diamond_i$ is a *quantifier*, each R_j is a *real relation*, and $C_1, \ldots, C_m$ are *logical connectives.* A *quantifier* $\Diamond_i$ is either $\forall$ or $\exists$, and it quantifies over a real variable x_i. A *real relation* is a relation among real values, and is one of $<$, $>$, or $=$. A logical connective is one of $\vee$ or $\wedge$.[57] Each $s_1, \ldots, s_m$ is a polynomial in $\mathbb{R}[x_1, \ldots, x_r]$, and so (A.2) is a sentence in r variables. We call the set $Y \subset \mathbb{R}^r$ defined by (A.1) or (A.2) a *semi-algebraic set*, and, conversely, a set $Y \subset R^r$ is called *semi-algebraic* if it can be written in a form like (A.2). The set Y is called *algebraic* if the only real relation we require is equality ($=$). The boolean characteristic function $\mathbb{T}(\cdot)$ of a semi-algebraic set such as Y is defined as

$$\begin{aligned}\mathbb{T}(x_1, \ldots, x_r) \Longleftrightarrow \; & s_1(x_1, \ldots, x_r) \; R_1 \; 0 \\ & C_1 \; s_2(x_1, \ldots, x_r) \; R_2 \; 0 \\ & \vdots \\ & C_{m-1} \; s_m(x_1, \ldots, x_r) \; R_m \; 0.\end{aligned} \tag{A.3}$$

$\mathbb{T}(\cdot)$ is called a *semi-algebraic predicate.* Hence, (A.2) can be written

$$(\Diamond_1 x_1 \; \Diamond_2 x_2 \; \cdots \; \Diamond_r x_r) : \; \mathbb{T}(x_1, \ldots, x_r).$$

Let Φ be an s.a. predicate. Let $\boldsymbol{x}$ denote $(x_1, \ldots, x_r)$, and for a quantifier $\Diamond$, let $\Diamond \boldsymbol{x}$ denote $(\Diamond x_1, \ldots, \Diamond x_r)$. If $\mathbb{T}_Y$ is an s.a. predicate for the s.a. set Y, we will abbreviate the sentence $\exists \boldsymbol{x} : \; (\mathbb{T}_Y(\boldsymbol{x}) \wedge \Phi(\boldsymbol{x}))$ as follows:

$$\exists \boldsymbol{x} \in Y : \; \Phi(\boldsymbol{x}). \tag{A.4}$$

Given this convention (A.4), a little manipulation shows that as a consequence, the formula $\forall \boldsymbol{x} : \; (\mathbb{T}_Y(\boldsymbol{x}) \Rightarrow \Phi(\boldsymbol{x}))$ is therefore equivalent to

$$\forall \boldsymbol{x} \in Y : \; \Phi(\boldsymbol{x}).$$

[57] So $\leqslant$ and $\geqslant$ can be built out of these.

Let δ be the total degree bound for the polynomials $s_1, \ldots, s_m$ in (A.2). We call the number of polynomials m the *size* of the Tarski sentence (A.2) and of the s.a. predicate $\mathbb{T}(\cdot)$ in (A.3). Observe however, that to calculate a bound $O(\delta rm)$ on the number of terms in (A.2), we would employ the degree bound δ and the number of variables r as well.

Now, it is remarkable that one can decide such sentences in complete generality: although Tarski's original algorithm [48] was non-elementary,[58] this bound has been improved by a chain of researchers since then. For example, Ben-Or et al. [2] showed how to decide the first-order theory of real closed fields with a purely algebraic algorithm in time $2^{2^{O(m)}}$ and space $2^{O(m)}$. In Section 9, we use this result:

Theorem A.1 (Grigoryev [29]). *Sentences in the theory of real closed fields can be decided in time doubly-exponential only in the number of quantifier alternations. More specifically, the truth of a Tarski sentence for m polynomials of degree $< \delta$ in r variables, where $a \leqslant r$ is the number of quantifier alternations in the prenex form of the formula, can be decided in time*

$$(m\delta)^{O(r)^{4a+2}}. \tag{A.5}$$

Proof. See [29]. □

A.1. Application: computational calibration complexity

Recall the discussion in Section 8.7.4. We wish to develop an algorithm for deciding the relation $\leqslant^*$ between sensor systems. Comparing the calibration complexity (Definitions 5.1, 6.2) of two sensor systems seems easier than the issues of immersion and simulation, because the calibration complexity does not change with the immersion, so long as the immersion respects the codesignation constraints. The essential idea behind computing calibration complexity is to measure the complexity of the codesignation constraints that specify a sensor system. One measure, of course, is the *number* of codesignation constraints, but other measures, such as the degree and the quantification, are also important. Using the algebraic methods from Section A, we can develop tools to measure the complexity of algebraic relations such as those encountered in algebraically codesignated sensor systems (Definition 8.17).

Now, to decide the relation $\leqslant^*$, we must be able to decide dominance in calibration complexity (see Definition 5.1). We propose to measure calibration and installation complexity by the complexity of the codesignation constraints. In general, one may measure the complexity of the codesignation constraints by comparing the complexity of the semi-algebraic varieties that the algebraic codesignations specify. One way to do this is to count the number, degree, quantification, and dimension of the semi-algebraic codesignation constraints. This gives numbers for m, δ, a, and r for an algebraic complexity measure such as (A.5), for example. Eq. (A.5) can then be used as a measure of the sensor's calibration complexity. These bounds can then be compared

[58] Tarski developed this algorithm around 1920, but it was not published until later.

(using big-Oh ($O(\cdot)$) notation) to determine which sensor dominates in terms of calibration complexity. The comparison can be done in essentially the same time it takes to read the input, and the time required is very small compared to (35), the time for the algebraic simulation.

Some of the complexity in our theory results from a decision to proceed through an abstract definition of a sensor system, independent of the underlying configuration space, and then to map that system into a particular space. One may ask whether this approach, though it mirrors much of modern geometry, is essential to the results obtained. We believe that it would be possible to start with an *a priori* configuration space (see Eq. (38)), instead of constructing it as a quotient of set differences. This would eliminate some of the technical baggage required (codesignation, non-codesignation and so forth). However, it appears that this approach would leave unanswered the question of measuring the complexity of the underlying configuration space—and hence it would not yield a computational theory of calibration complexity.

A.2. Application: simulation functions

Recall the discussion of simulation functions for components, edges, and sensor systems in Section 8. We now discuss simulation functions and their encodings. It is important that simulation functions work on permuted sensor systems. Here is how this might be accomplished.

A.2.1. Vertex versus graph permutations

We now consider two orthogonal kinds 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 8.6. 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 suffices for all reductions in this paper, and the machinery in Sections 8.1 and 9 suffices to compute the reductions $\leqslant^*$ and $\leqslant_1$.

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 sensor system $\mathcal{U}$ 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 data-path. 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 $\mathcal{U}^*$ of $\mathcal{U}$. Suppose $\mathcal{U}^* = \sigma(U^*)$, that is, $\mathcal{U}^*$ is the vertex permutation U^* followed by the edge permutation σ above. $\mathcal{U}^*$ has the same edge connectivity as $\sigma(\mathcal{U})$. However, in $\mathcal{U}^*$, the vertices are immersed as in U^*.

To summarize: let $(\mathcal{U}, \phi)$ be a situated sensor system. A graph permutation of $\mathcal{U}$ is given by $\mathcal{U}^* = (\mathcal{U}, \phi^*)$ where $\phi^* = (\phi^*, \sigma)$, ϕ^* is a vertex permutation, and σ is an edge permutation.

So, 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. Although vertex permutation suffices for all the examples in this paper, graph permutation is useful (and required) in [17]. Graph permutation is also required for some of the applications discussed in Section 1.1 (particularly (3) *design* and (4) *universal reduction*—see Section A.4). Here, we will content ourselves with answering two questions: (i) if we permit graph permutation, does it change our complexity bounds? and (ii) does graph permutation give us a more powerful reduction?

We first turn to question (i). Fortunately, we can extend our computational results to graph permutation without difficulty. To do this, we model a graph permutation of a sensor system $\mathcal{U}$ as a vertex permutation of $\mathcal{U}$, followed by an edge permutation of $\mathcal{U}$. Using this scheme, we can compute all our reductions ($\leqslant^*$, $\leqslant_1$, etc.) within the same time bounds given in Lemma 9.9 and Corollary 9.10, permitting graph permutation in place of vertex permutation throughout. Our other lemmas also go through *mutatis mutandis*.

We now elaborate. An *adjacency matrix* for a sensor system with d vertices is a $d \times d$ binary matrix. An *adjacency matrix with bandwidth* has non-negative integer entries. An entry of b in row v, column u specifies a (directed) edge of bandwidth $\log \mathbb{K}(b)$ between[59] vertices v and u. Given an edge permutation σ, we can construct a new adjacency matrix, and the edge simulation functions (such as Ω_e in Section 8.1) can be constructed from the adjacency matrix. Now, we may view the edges (data-paths) in our sensor system as part of its configuration. Hence, in different configurations, the system may have different "wiring diagrams" (different edges). We now consider this such "configurations" and the resulting "configuration space". In particular, we wish to demonstrate their algebraicity.

Consider a sensor system $\mathcal{U}$ with d vertices V, and $O(d^2)$ edges E. When we permit graph permutation, a configuration of this system can be specified by a pair (ϕ, σ), where $\phi: V \to C$ is an immersion (Definition 8.4) of $\mathcal{U}$, and σ is an edge permutation. As we have discussed, ϕ lives in the configuration space C^d. What about σ? σ is a member of the permutation group on d^2 elements. σ can be modeled as a $d^2 \times d^2$ binary matrix called a *permutation matrix*. Every permutation matrix has a single 1 in each row and column, the other entries being zero. Let $\mathbb{Z}_2$ denote the field $\mathbb{Z}/2$. Then, the space

[59] This representation is not hard to extend to components with multiple inputs and outputs, using an $rd \times sd$ matrix.

of permutation matrices is $\mathbb{O}(\mathbb{Z}_2, d^2)$, the *orthogonal* group of $d^2 \times d^2$ binary matrices. Each element is an orthogonal matrix, with determinant ± 1.

Every "rewiring" of $\mathcal{U}$ using only existing edges is encoded by a permutation $\sigma \in \mathbb{O}(\mathbb{Z}_2, d^2)$. So, to model vertex permutation plus rewiring, we extend our usual sensor configuration space from C^d to $C^d \times \mathbb{O}(\mathbb{Z}_2, d^2)$. It is not hard to extend this model to add one extra wire (output communication), or several extra wires (for k-wire reductions (Section 8.10.3)). The space $\mathbb{O}(\mathbb{Z}_2, d^2)$ is algebraic, and the computation of edge simulation functions from adjacency matrices is s.a.

Now, how expensive it is to compute the reductions $\leqslant^*$ and $\leqslant_1$ using graph permutation? Perhaps surprisingly, even with this extended configuration space (which has dimension $d^4 + r_c d$ instead of $r_c d$), we still obtain the same complexity bounds given in Lemma 9.9 and Corollary 9.10 (so long as r and s are constants). This is because [60] (see Eqs. (35–37)) $n^{(d^4+r_c d)^{O(1)}}$ is still $n^{(r_c d)^{O(1)}}$.

We now address question (ii): does graph permutation give us a more powerful reduction? In answer we show the following:

Lemma A.2 (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 [61] $\mathbb{V}$, 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. [62]

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

[60] Another way to see this is as follows: even if we try each of the $(d^2)!$ edge permutations, this additional $(d^2)!$ factor is absorbed by the $O(1)$ in the second exponent.

[61] See Section C.1.

[62] 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.

Let us denote by $\mathrm{cl}(\mathcal{U})$ the linear-space clone transformation of $\mathcal{U}$ described in Lemma A.2. Now consider any k-wire reduction $\leqslant_k$ (Section 8.10.3). We see that:

Corollary A.3. *Let $k \in \mathbb{N}$. Suppose that for two sensor systems $\mathcal{U}$ and $\mathcal{V}$, we have $\mathcal{V} \leqslant_k \mathcal{U}$ (using graph permutation). Then $\mathcal{V} \leqslant_k \mathrm{cl}(\mathcal{U})$ (using only vertex permutation).*

Class 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 data-paths in the system, in addition to bandwidth, have a *type*, of the form SOURCE $\rightarrow$ DESTINATION, where both SOURCE and DESTINATION $\in \{L, R\}$. When permuting the edges of $\mathcal{U}$ to obtain $\mathcal{U}^*$, 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 $\mathcal{U}^*$, we may use an internal edge (of sufficient bandwidth) to connect any two components where SOURCE = DESTINATION. External edges (of sufficient bandwidth) can be used when SOURCE $\neq$ DESTINATION. Hence, in *class* edge permutation, we permute edges within a class. Class edge permutation leaves unchanged the complexity bounds and the lemmas of Section A.2.1.

In this example, maintaining exactly two physical locations can be done using simple codesignation constraints. More generally, we take SOURCE, DESTINATION $\in C$.

A.3. Application: parametric codesignation constraints

Recall Eq. (34), in which we formulated the sensor reduction problem as an s.a. decision procedure. We now discuss some technical details of this equation, using the notation and hypotheses of Section 9.2.

In order to allow for sufficient generality, we must permit $\mathcal{V}$'s codesignation constraints to depend on $\mathcal{U}$'s configuration $\overline{\alpha}$. That is, the s.a. set $D_{\mathcal{V}\mathcal{U}}(\overline{\alpha})$ is an s.a. function of $\overline{\alpha}$. Recall that $(\mathcal{U}, \overline{\alpha})$ denotes the sensor system $\mathcal{U}$ installed at configuration $\overline{\alpha}$. Now, given that sensor system $\mathcal{U}$ is at configuration $\overline{\alpha}$, we are interested in whether or not sensor system $\mathcal{V}$ can simulate $(\mathcal{U}, \overline{\alpha})$, but *only when* $\mathcal{V}$'s configuration $\overline{\beta^*}$ satisfies some constraint $D_{\mathcal{V}\mathcal{U}}(\overline{\alpha})$ that *depends* on $\overline{\alpha}$. That is we are interested in the question:

"Does $(\mathcal{V}, \overline{\beta^*})$ simulate $(\mathcal{U}, \overline{\alpha})$, given that $\overline{\beta^*}$ lies in $D_{\mathcal{V}\mathcal{U}}(\overline{\alpha})$?"[63]

For example, consider the reduction in Proposition 6.6. Here $\overline{\alpha}$ specifies (among other things) the ship's configuration (x, h) in the radial sensor E. We think of (x, h) as one "coordinate" of $\overline{\alpha}$. The parametric codesignation constraint $D_H(\overline{\alpha})$ is used to ensure that the corresponding ship in the lighthouse sensor H is also placed at (x, h). The question "Can H simulate E?" only makes sense given that (i) H and E are both installed at G and (ii) the ships in H and E are in the same configuration. Static codesignation constraints (that are invariant with $\overline{\alpha}$) ensure (i), whereas parametric codesignation

[63] In particular, we do not care what happens when $\overline{\beta^*} \notin D_{\mathcal{V}\mathcal{U}}(\overline{\alpha})$.

constraints (that vary with $\overline{\alpha}$) ensure (ii). This could be implemented as follows: let $\pi_{E,x}$ (resp. $\pi_{H,x}$) be the projection of E's (resp. H's) configuration that returns the ship's configuration. So, in particular, $\pi_{E,x}(\overline{\alpha}) = (x, h)$. These projections are clearly semi-algebraic functions. Then (this aspect of) the parametric codesignation constraint D_H could be implemented as

$$\beta \in D_H(\overline{\alpha}) \iff \left(\pi_{H,x}(\beta) = \pi_{E,x}(\overline{\alpha})\right). \tag{A.6}$$

The fact that we have an equality constraint (=) in Eq. (A.6) reflects the fact that E and H are *simple* sensor systems (Definition 8.17). In general (for arbitrary algebraic sensors systems), D_H could specify a more complicated s.a. relation between $\overline{\alpha}$ and β.

Formally, parametric codesignation constraints as D_H (A.6) and $D_{\mathcal{VU}}$ (see Eq. (34)) can be modeled as *parametric s.a. sets* (see [6]):

Definition A.4 (*Canny*). A *parametrically-defined semi-algebraic set* $D(\alpha)$ is defined as follows. $D(\alpha)$ is an s.a. set which is a function of some argument α. Hence there is an implicitly defined s.a. predicate $\mathbb{T}_D(z, \alpha)$ which is true iff $z \in D(\alpha)$. Now, let Y be an s.a. set with predicate $\mathbb{T}_Y$. So, when we write $D(\alpha) \subset Y$ we mean $\forall z\ \mathbb{T}_D(z, \alpha) \Rightarrow \mathbb{T}_Y(z)$, which gives us an s.a. predicate $\Phi_D(\alpha)$ which is true of those values of α such that $D(\alpha) \subset Y$.

A.4. Application: universal reductions

We can now use the tools from Sections A.2–A.3 to develop an algorithm for "universal reduction" (application 4 of Section 1.1). Universal reduction requires graph permutation (see Appendix A.2.1).

Let $\mathcal{U}$ and $\mathcal{V}$ be sensor systems. Suppose we are given a specification for $\mathcal{U}$, and a "bag of parts" for $\mathcal{V}$. The specification, as usual, is encoded as a simulation function $\Omega_{\mathcal{U}}$ as described in Section 8.1. We are also given a simulation function $\Omega_{\mathcal{V}}$ for $\mathcal{V}$. The bag of parts consists of boxes and wires. Each box is a sensori-computational component ("block box") that computes a function of (a) its spatial location or pose and (b) its inputs. The "wires" have different bandwidths, and they can hook the boxes together. Recall we are given a simulation function Ω_v for each component $\ell(v)$ and a simulation function Ω_e for each edge e (indeed, this is how the global simulation functions $\Omega_{\mathcal{U}}$ and $\Omega_{\mathcal{V}}$ are encoded; see Section 8.1). Then, our algorithms (above) decide, can we immerse the components of $\mathcal{V}$ so as to satisfy the specification of $\mathcal{U}$? The algorithms also give the immersion (that is, how the boxes should be placed in the world, and how they should be wired together). Hence, we can ask, can the specification of $\mathcal{U}$ be implemented using the bag of parts $\mathcal{V}$?

Now, suppose that in addition to the specification for $\mathcal{U}$, we are given an encoding of $\mathcal{U}$ as a bag of parts, and an immersion to implement that specification. Suppose further that $\mathcal{U} \leqslant_1 \mathcal{V}$. Since this reduction is relativized both to $\mathcal{U}$ and to $\mathcal{V}$, it measures the "power" of the components of $\mathcal{U}$ relative to the components in $\mathcal{V}$. By universally quantifying over the configuration of $\mathcal{U}$, we can ask, "can the components of $\mathcal{V}$ always do the job of the components of $\mathcal{U}$?"

More specifically: let α be a configuration of the sensori-computational system $\mathcal{U}$. Let $\mathcal{U}^\star = (\mathcal{U}, \alpha^\star)$ be a graph permutation of $(\mathcal{U}, \alpha)$ (Section A.2.1). Let $\Sigma^\star(\alpha)$ denote the set of all graph permutations of α, so, if $\mathcal{U}$ has d vertices, then $\Sigma^\star(\alpha) = \Sigma(\alpha) \times \mathbb{O}(\mathbb{Z}_2, d^2)$. Thus $\alpha^\star \in \Sigma^\star(\alpha)$, and $\alpha^\star$ encodes the spatial immersion of $\mathcal{U}$ as well as its wiring connectivity. By Sections 8.9.2 and A.2.1, $\Sigma^\star(\alpha)$ is s.a.

Similarly, let β be a configuration of $\mathcal{V}$. Hence, we can decide the Tarski sentence

$$\left(\forall \alpha^\star \in \Sigma^\star(\alpha),\ \exists \beta^\star \in D_{\mathcal{V}\mathcal{U}}(\alpha^\star) \cap \Sigma^\star(\beta)\right) :\ (\mathcal{U}, \alpha^\star) \leqslant_1 (\mathcal{V}, \beta^\star), \tag{A.7}$$

where $D_{\mathcal{V}\mathcal{U}}(\cdot)$ is a parametric s.a. codesignation constraint (Section A.3). When Eq. (A.7) holds, we say that $\mathcal{U}$ *universally reduces to* $\mathcal{V}$, (or that there is a *universal reduction* from $\mathcal{U}$ to $\mathcal{V}$). Hence, is possible to compute universal reductions algebraically. With the notation and hypotheses as above throughout Appendix A, the time complexity of deciding (A.7) is given by Eq. (A.5), which becomes

$$(m\delta)^{\mathrm{O}(r)^{4a-2}} = (n_\Omega + n_D)^{\mathrm{O}(r_c d + d^4)^{14}}. \tag{A.8}$$

Eq. (A.8) is still $(n_\Omega + n_D)^{(r_c d)^{\mathrm{O}(1)}}$. Hence we have that

Corollary A.5. *Universal reductions (Eq. (A.7)) can be computed in the same time bounds given in Eqs. (35)–(37).*

Appendix B. Relativized information complexity

Let us specialize Definition 6.4 to monotonic sensor systems:

Definition 6.4 (*Monotonic*). Consider two monotonic sensor systems S and Q, and let b be the output of sensor S. We say S is *efficiently reducible* to Q if

$$S \leqslant^* Q + \mathrm{COMM}(b). \tag{4}$$

In this case we write $S \leqslant_1 Q$.

For the sensors we have considered, their complexity could essentially be characterized using the size $\log \mathbb{K}(b)$ of the *output* b. We now generalize this definition slightly. Our motivation is as follows. There are sensor systems whose complexity cannot be well-characterized by the number of bits of output.[64] 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 [14], 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 $P(G \mid V)$ 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

[64] This discussion devolves to a suggestion of Sundar Narasimhan [42], for which we are very grateful.

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).[65]

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 6.1 (*First part*). 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 5.2. We define the *maximum bandwidth* $\mathrm{mb}(\mathcal{S})$ to be the greater of the internal bandwidth, external bandwidth, and the output size of $\mathcal{S}$.

The maximum bandwidth is an upper bound on the relative intrinsic output complexity (relativized to the information complexity of the components (Sections 8 and 12)). 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, Erdmann's radial sensor E (Section 4.1) is essentially a black box plus output communication.

More generally, we call a sensor system *monotonic* if its internal and external bandwidths are bounded above by its output size. So, black box sensors are trivially monotonic. All the sensor systems in this paper are monotonic. But some of the systems in our forthcoming work [17] are not.

In light of this discussion, we now give a generalized definition of the reduction $\leqslant_1$, using relativized information complexity.

First, let S be a monotonic sensor system with output b as in Definition 6.4. In this case, we define $\mathrm{COMM}(S)$ to be $\mathrm{COMM}(b)$.

[65] Now one may ask why prefer one model over another and there can be many answers. [42] advocates *Minimum Description Length*, or *MDL*. This theory attempts to minimize $L(M) + L(D \mid M)$ where $L(M)$ is the length of model and $L(D \mid M)$ is the length of the data given that the model is minimal.

More generally, for (possibly) nonmonotonic sensors, we will let COMM(S) be 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 6.1) of S as an upper bound on k. Finally, we generalize Definition 6.4 to nonmonotonic sensor systems as follows:

Definition 6.4 (*Generalized*). Consider two sensor systems S and Q. We say S is *efficiently reducible* to Q if

$$S \leqslant^* Q + \text{COMM}(S). \tag{B.1}$$

In this case we write $S \leqslant_1 Q$.

Appendix C. Distributive properties

In this appendix, we prove some technical properties about the permutation of partial immersions. These properties are algebraic, and we call them the "*distributive* properties". First, we consider "pure" permutation and combination (i.e., without output vertices, as in Definition 8.12). Then, in Sections C.1–C.2 we generalize to include permutation and combination of output vertices. Recall the definition of *compatibility* for partial immersions (Section 8.4).

Definition C.1. Let ϕ and ψ be compatible partial immersions. We say the permutations ϕ^* and ψ^* are *compatible permutations of ϕ and ψ*, if ϕ^* and ψ^* are also compatible.

We would like to show that for immersions, combination and permutation commute. That is: for two compatible partial immersions ϕ and ψ, if ϕ^* and ψ^* are compatible permutations, then

$$\phi^* + \psi^* = (\phi + \psi)^*?$$

In answer, we can now show the following:

Claim C.2. *Consider two compatible partial immersions ϕ and ψ, together with two compatible permutations ϕ^* and ψ^*. Then*

(1) $\phi^* + \psi^* \in \Sigma(\phi + \psi)$.

(2) *Let $\gamma^* \in \Sigma(\phi + \psi)$. Then there exists $\phi^* \in \Sigma(\phi)$, $\psi^* \in \Sigma(\psi)$, such that $\gamma^* = \phi^* + \psi^*$.*

Proof. (2) First, let γ^* be a permutation of $\phi + \psi$. Let $\phi^* = \gamma^*|_{\phi^{-1}C}$ and $\psi^* = \gamma^*|_{\psi^{-1}C}$. Then ϕ^* is a permutation of ϕ and ψ^* is a permutation of ψ, and $\phi^* + \psi^* = \gamma^*$.

(1) Conversely, suppose ϕ^* and ψ^* are compatible permutations of ϕ and ψ. Then we observe that since the domains of ϕ and ϕ^* (resp., ψ and ψ^*) are identical, therefore the domains of $\phi^* + \psi^*$ and $\phi + \psi$ are identical. Hence, $\phi^* + \psi^*$ is a permutation of $\phi + \psi$. □

Next, we ask, for sensor systems, do combination and permutation commute? That is: for two sensor systems $\mathcal{S}$ and $\mathcal{U}$, is it true that

$$\mathcal{S}^* + \mathcal{U}^* = (\mathcal{S} + \mathcal{U})^*$$

whenever $+$ is defined (see Definition 8.12)?

In answer, we show the following:

Proposition C.3. *Consider two sensor systems $\mathcal{S}$ and $\mathcal{U}$ as above. Assume their immersions are compatible, so that $\mathcal{S} + \mathcal{U}$ is defined. Then,*

(1) *Let $\mathcal{S}^*$ and $\mathcal{U}^*$ be compatible permutations of $\mathcal{S}$ and $\mathcal{U}$. Then $\mathcal{S}^* + \mathcal{U}^*$ is a permutation of $\mathcal{S} + \mathcal{U}$.*

(2) *Let $(\mathcal{S}+\mathcal{U})^*$ be a permutation of $\mathcal{S}+\mathcal{U}$. Then there exist compatible permutations $\mathcal{S}^*$ and $\mathcal{U}^*$ of $\mathcal{S}$ and $\mathcal{U}$ resp. such that $\mathcal{S}^* + \mathcal{U}^* = (\mathcal{S} + \mathcal{U})^*$.*

Proof. Let $\mathcal{S} = (\mathcal{S}, \phi)$, $\mathcal{U} = (\mathcal{U}, \psi)$, $\mathcal{S}^* = (\mathcal{S}, \phi^*)$ and $\mathcal{U}^* = (\mathcal{U}, \psi^*)$, and apply Claim C.2. □

C.1. Combination of output vertices

Recall the definition of *combination* in Section 8.5. There, we considered two sensor systems $\mathcal{S}$ and $\mathcal{U}$. Both have output vertices, say, v_0 and u_0 resp. When we combine the two sensor systems $\mathcal{S}$ and $\mathcal{U}$ to form $\mathcal{S} + \mathcal{U}$, we must specify the unique output vertex of the new, combined sensor system. We now show how to choose output vertices in a consistent manner so that the combination operation $+$ remains associative and commutative.

First, we view each sensor system as a *pointed graph*—a graph with one distinguished vertex called the *output vertex*.[66] We define $+$ on two pointed graphs in such a manner as to produce a new pointed graph. For example let (G_1, u_1) be a pointed graph with output vertex u_1. Let (G_2, u_2) be another pointed graph. Then

$$(G_1, u_1) + (G_2, u_2) = (G_1 + G_2, u_1 + u_2),$$

where $G_1 + G_2$ denotes combination (Definition 8.12). The output vertex $u_1 + u_2$ is defined as follows: let $\mathbb{V}$ be the universe of all possible vertices. So, for any graph G_i with vertices and edges (V_i, E_i), we have $V_i \subset \mathbb{V}$. We insist that $\mathbb{V}$ have a total-order $\succ$. Define $u_1 + u_2 = \min_{\succ}(u_1, u_2)$.

It is easy to see that under this definition, the operation $+$ on pointed graphs is both associative and commutative.

C.2. Output permutation

Recall Definition 8.6. There, we also permitted a permutation to change which vertex has the "output device" label. This kind of permutation is not required for the *monotonic*

[66] We must be careful not to confuse a *pointed graph* with a *pointed sensor system* (Definition 8.8).

sensor systems (Appendix B) considered in this paper, but it is needed for the general theory, and it is used explicitly in [17]. We formalize this notion here.

We define an operation called *output permutation* on pointed graphs (Section C.1). The effect of this operation is to choose a new distinguished vertex. For example, for a graph G with distinguished point u_0, we could choose a new distinguished vertex u_1. We represent this operation by

$$(G, u_0) \mapsto (G, u_1).$$

We call (G, u_1) an *output permutation* of (G, u_0).

Now, following Appendix A.2.1, let us call our existing notion of permutation (Definition 8.6) by the name *vertex* permutation (to distinguish it from *output* permutation). It is possible to compose output permutations and vertex permutations. We adopt

Convention C.4. We use the term *permutation* to include both output permutations and vertex permutations. Similarly, we will use the operator * for any permutation.

This convention is necessary to make combination and permutation commute in general.

C.3. Discussion

In Appendices B and C, we have made sure that combination (the $+$ operation) and permutation (the * operation) commute. So, for example, for any sensor system $\mathcal{S}$, 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 Definition 6.2, for the reduction $\leqslant_1$ (see generalized Definition 6.4) 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 $(Q + \text{COMM}(S))^*$ in Eq. (B.1) 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.

Appendix D. On alternate geometric models of information invariants

We have presented a geometric model of information invariants. I am grateful to John Canny and Jim Jennings for suggesting that I provide an "abstract" example of information invariants, using the language and concepts developed in [14]. The resulting model is somewhat different in flavor from that of Section 8.

Here is a alternate geometric model for an example of information invariance. Let $\mathcal{U}$ be an arrangement of perceptual equivalence classes, as in [14, 5.1]. A simple control strategy may be modeled as a subgraph of the RR-graph [16] on $\mathcal{U}$. Now consider the lattice of perceptual equivalence classes formed by fixing the task environment and varying the sensing map, as in [14, 5.2]. Let $\mathcal{U}$ and $\mathcal{V}$ be two arrangements of perceptual equivalence classes in the lattice. Then there is an information invariant for $\mathcal{U}$ and $\mathcal{V}$

when they have a common coarsening[67] $\mathcal{W}$, together with a control strategy on $\mathcal{W}$. Note that by construction, this control strategy agrees on the overlap of $\mathcal{U}$ and $\mathcal{V}$.

This example is simple; it remains to develop and exploit this geometric model for other kinds of information invariants.

Appendix E. A non-geometric formulation of information invariants

There are several places where we have exploited the geometric structure of robotics problems in constructing our framework. First, our sensors are geometrical (in that they measure geometric quantities). Second, the configuration of a sensor is geometrical, in that each component is physically placed and oriented in physical space.

It is of some interest to derive an "abstract" version of our framework in which geometry plays no role.[68] Such a framework would be something like a "logical" framework.

It is not hard to formulate our approach in a geometry-free manner. First one would say that the "value" or the "output" of a sensor is simply a value in some set. Next, one would replace the configuration space C of a component by any set of the form

$$C = \{z \mid z \text{ is a location}\}. \tag{E.1}$$

C can be taken to have no structure whatsoever. All the definitions, constructions, and proofs of Section 8 then go through *mutatis mutandis*: there is no geometry anywhere. In particular, our (formerly geometric) codesignation constraints now reduce to Chapman's (propositional) codesignation constraints [7].

It is now worth asking, *what are the implications for Section 9?* It is easy to extend the definition of a simulation function $\Omega_{\mathcal{U}}$ for a sensor system $\mathcal{U}$: one obtains a set map $\Omega_{\mathcal{U}} : C^d \to R$ where C is as in (E.1), and R is an arbitrary set. At this point we lose the algebraic properties we exploited to derive the algorithms of Section 9. Hence our algorithms do not obtain when we remove the geometric structure. In particular, we lose our main computational result, Lemma 9.10. It seems plausible, however, that other deductive mechanisms might be used, instead, to obtain similar results in the abstract (non-geometric) case.

Appendix F. Provable information invariants with performance measures

F.1. Kinodynamics and tradeoffs

It is possible to develop provable information invariants in the special case where we have performance measures. Consider once again the information invariants discussed above in Section 2.1. That these invariants (Eq. (1)) are related to kinodynamics [8, 19, 20] should come as no surprise, since the execution time for a control strategy is taken

[67] A *coarsening* of $\mathcal{U}$ and $\mathcal{V}$ is a partition $\mathcal{W}$ such that both $\mathcal{U}$ and $\mathcal{V}$ are finer than $\mathcal{W}$.

[68] I am grateful to Stan Rosenschein for encouraging me to develop this generalization.

as "cost". In [49], Pat Xavier introduced a new algorithmic mechanism for measuring kinodynamic tradeoffs (see [21] for a brief description). These techniques were used to quantify the tradeoffs between planning complexity, executor complexity, and "safety" (clearance). Essentially, Xavier considers how closely (ε_T) one can approximate an optimal-time trajectory and how much "safety" ε_S—in the sense of headway—is required to execute the approximate solution with an uncertain control system. Xavier obtained "equicomplexity" curves in the ε_T-ε_S plane. These curves may be interpreted as follows. For a fixed "complexity" r (which may be equivalently viewed as (i) the running time of the planner, (ii) the space requirements of the planner, or (iii) the discretization density of the phase space for the dynamical system representing the robot), Xavier's planner obtains a kinodynamic solution which satisfies a one-parameter family of approximations of the form

$$\varepsilon_T = f_r(\varepsilon_S), \tag{F.1}$$

where f_r is a function conditioned on complexity r. Hence (F.1) represents an information invariant as well, and, if we view the "following distance" d as being similar to the clearance parameter ε_S, such kinodynamic methods appear attractive. We believe that these methods could be used to prove information invariants like (1); while they require specific assumptions about the dynamics and geometry, they are quite general in principle. Pursuing such theorems is a fruitful line of future research.

Kinodynamic tradeoffs are one source of information invariants, and one may even find provable, rigorous characterizations for information questions therein (e.g., [21, 49]). However, there is something a bit dissatisfying about this line of attack. First, it makes controls, not sensing, the senior partner, much in the same way that in the theory of Lozano-Pérez et al. [38] (see [11]), recognizability is a second-class citizen compared with reachability. In [38], this is a consequence of a bias towards sensorless manipulation [26]; in kinodynamics, it is a consequence of model-based control. Second, kinodynamics relies on a measure of cost (in this case, time), and hence the results emphasize performance, not competence.

Glossary of symbols[69]

		Section/ Appendix	Page	Definition	Figure (equation)
$\mathbb{R}$	real numbers	2.1.2	222		
S^1	unit circle	2.1.2	222		
p, p'	trajectories	2.1.2	222		
$S \cong Q$	Q simulates S	4.1	233,244,251	4.1	(3),(6)
$+$	combination of sensor systems	8.10	253,244	8.10	(3)
E	the radial sensor	4.1	234		5
G	the goal configuration	4.1	234		5

[69] For some symbols, the first page reference points to the beginning of the (sub)section explaining or containing that symbol.

		Section/ Appendix	Page	Definition	Figure (equation)
R	ship	4.1	234		5
x, $\underline{x}$	ship's position	4.1	234		5
h	ship's heading	4.1	234		5
θ_r	angle between h and the goal direction	4.1	234		5
N	direction of North	4.2.1	235,236		6
H	the lighthouse (beacon) sensor	4.2.1	235,236		6
L	lighthouse	4.2.1	235,236		6
θ	R's bearing from L	4.2.1	235,236		6
$\boxed{g}$	rotating green light	4.2.1	235,236		6
$\boxed{w}$	flashing white light	4.2.1	235,236		6
`(white?)`	1-bit white light sensor	4.2.1	235,236		6
`(green?)`	1-bit green light sensor	4.2.1	235,236		6
`(time)`	clock	4.2.1	235,236		6
`(orientation)`	orientation sensor	4.2.1	235,236		6
h_R	generalized compass (installed on R)	5.2.1	239		7
p^*	sensed position	5.2.1	239		
COMM$(\cdot)$	communication primitive	5.2.1	239		
COMM$(L \rightarrow R, \mathit{info})$	communicate *info* from L to R	5.2.1	239		
COMM(θ_r)	datapath labeled θ_r	5.2.1, 8.7.2	239, 256,244		(3)
$S, Q, \ldots$	sensor systems	5.2.1	239		
b	output of a sensor S	5.4.1, 8.1	244, 247,245	5.2, 8.3, 6.4	(4)
$\mathbb{K}(b)$	number of values b can take on	5.4.1, 6	244, 245	5.2	
mb(S)	maximum bandwidth of S	6, B	245, 294ff	6.1	
COMM(b)	datapath with bandwidth $\log \mathbb{K}(b)$	8.7.2	256,245,268	6.4	(4),(4′)
COMM(S)	datapath with bandwidth mb(S)	B	294,296		(B.1)
E_G	radial sensor installed at G	5.3.1	242,251,244	8.8	(3)
H_G	lighthouse sensor installed at G	5.3.1	242,244,251	8.8	(3)
*	(vertex) permutation	5.3.1	242,250,244	8.6	(3)
H^*	permutation of H	3	244		(3)
H_G^*	permutation of H_G	3	244		(3)
$\leqslant$	simulation and domination	6.2	245,265	6.2	(24)–(28)
$\leqslant^*$, $\leqslant_0$	0-wire reduction	6.3	245,264,268	6.3	(4), (4′)
$\leqslant_1$	1-wire ("efficient") reduction	6.4	245	6.4	
$\leqslant_k$	k-wire reduction	8.10.3	267		
$\leqslant_\infty$	reduction using global communication	8.10.4	270ff	8.27	
$\leqslant_{\mathcal{P}}$	reduction using polynomial communication	8.10.4	270ff	8.29	
$\mathcal{G} = (V, E)$	a graph with vertices V and edges E	8.1	247	8.1	
d	number of vertices in V	9.2	274,274		(35)
$\mathcal{S}, \mathcal{U}, \mathcal{V}, \mathcal{W}, \ldots$	sensor systems	8.2	247,267,268	8.2	9, 10
$\phi, \psi, \ldots$	immersions	8.1	247,250	8.4	
ℓ	labelling function	8.1	247,250	8.4	
C	configuration space	8.1	247		
$(\mathcal{S}, \phi)$	situated sensor system	8.1	247,250	8.6	
ϕ^*	permutation of an immersion	8.1	247,250	8.6	
$\mathcal{S}^*$, $(\mathcal{S}, \phi^*)$	permutation of a sensor system	8.1	247,250	8.6	
(ϕ, G)	pointed immersion	8.7	251	8.7	
$\mathcal{S}_G$	pointed sensor system	8.7	251	8.7	
$\mathcal{S}_G^*$	pointed permutation	8.7	251	8.7	

		Section/ Appendix	Page	Definition	Figure (equation)
$\overline{\phi}$	extension of a partial immersion	8.3	252		
$\overline{\phi^*}$	extension of a permutation	9.1	271		
u_o, v_o	output vertex	8.5	253		
Δ, Δ_{ij}	diagonal	8.8, 10	259, 275		(14),(38)
Φ,$\mathbb{T}$...	s.a. predicate	9.1, A	271, 286,287		(A.3)
ex ϕ	extensions of ϕ	8.9.2	261,263		(20)
$\Sigma(\phi)$	(vertex) permutations of ϕ	8.9.2	261,263		(20)
$\Sigma^*(\phi)$	graph permutations of ϕ	A.4	293		(A.7)
im ϕ	image of ϕ	8.9.2	261,264		(22)
$\Omega_{\mathcal{U}}$	simulation function	9.1	271,247,274	9.2	(34)
$\diamondsuit$	quantifier	9.1	271,287		(A.3)
$D_{\mathcal{U}}, D_{\mathcal{V}}, D_{\mathcal{V}\mathcal{U}}, \ldots$	s.a. codesignation constraints	9.2	274,274		(34)
r_c	dimension of C	9.2	274,274		(35)
δ	degree bound	9.2	274,274		(35)
n_Ω	simulation complexity	9.2	274,274	9.8	(35)
n_D	s.a. codesignation complexity	9.2	274,274	9.8	(35)
σ	edge permutation	A.2.1	289		
$\mathcal{U}^*$, $(\mathcal{U}, \phi^*)$	graph permutation of $\mathcal{U}$, $(\mathcal{U}, \phi)$	A.2.1	289		
$\mathbb{O}(A, d)$	group of orthogonal matrices	A.2.1	289		
cl	clone function	A.2.1	289,291	A.2	

References

[1] M. Blum and D. Kozen On the power of the compass (or, why mazes are easier to search than graphs), in: *Proceedings 19th Symposium on Foundations of Computer Science*, Ann Arbor, MI (1978) 132–142.

[2] M. Ben-Or, D. Kozen and J. Reif, The complexity of elementary algebra and geometry, *J. Comp. Syst. Sci.* **32** (1986) 251–264.

[3] M. Blum and W. Sakoda On the capability of finite automata in 2 and 3 dimensional space, in: *Proceedings 17th Symposium on Foundations of Computer Science* (1977) 147–161.

[4] A. Briggs, An efficient algorithm for one-step compliant motion planning with uncertainty, *Algorithmica* **8** (2) (1992) 195–208.

[5] R.G. Brown, Forthcoming Ph.D. Thesis, Computer Science Department, Cornell University, Ithaca, NY, USA.

[6] J. Canny, On computability of fine motion plans, in: *IEEE International Conference on Robotics and Automation*, Scottsdale, AZ (1989).

[7] D. Chapman, Planning for conjunctive goals, *Artif. Intell.* **32** (3) (1987) 333–378.

[8] J. Canny, B. Donald, J. Reif and P. Xavier, On the complexity of kinodynamic planning, in: *29th Symposium on the Foundations of Computer Science*, White Plains, NY (1988) 306–316.

[9] J. Canny and J. Reif, New lower bound techniques for robot motion planning problems, *28th Annual IEEE Symposium on Foundations in Computer Science*, Los Angeles, CA (1987).

[10] D. Cox, J. Little and D. O'Shea, *Ideals, Varieties, and Algorithms*, Undergraduate Texts in Mathematics (Springer-Verlag, New York, 1991).

[11] B.R. Donald, Robot motion planning, *IEEE Trans. Rob. Autom.* **8** (2) (1992).

[12] B.R. Donald, The complexity of planar compliant motion planning with uncertainty, *Algorithmica* **5** (3) (1990) 353–382.

[13] B.R. Donald, *Error detection and recovery in robotics*, Lecture Notes in Computer Science **336** (Springer-Verlag, New York, 1989).

[14] B.R. Donald and J. Jennings, Constructive recognizability for task-directed robot programming, *J. Rob. Autonom. Syst.* **(9)** (1) (1992) 41–74.

[15] B.R. Donald and J. Jennings, Constructive recognizability for task-directed robot programming, in: *Proceedings IEEE International Conference on Robotics and Automation*, Nice, France (1992).

[16] B.R. Donald and J. Jennings, Sensor interpretation and task-directed planning using perceptual equivalence classes, in: *Proceedings IEEE International Conference on Robotics and Automation*, Sacramento, CA (1991).

[17] B.R. Donald, J. Jennings and D. Rus, Information invariants for distributed manipulation, in: R. Wilson and J.-C. Latombe, eds., *The First Workshop on the Algorithmic Foundations of Robotics* (A.K. Peters, Boston, MA, 1994). A revised version of: B.R. Donald, J. Jennings and D. Rus, Towards a theory of information invariants for cooperating autonomous mobile robots, in: *International Symposium on Robotics Research (ISSR)*, Hidden Valley, PA (1993).

[18] B.R. Donald, D. Kapur and J. Mundy, *Symbolic and Numerical Computation for Artificial Intelligence* (Academic Press, London, 1992).

[19] B.R. Donald and P. Xavier, A provably good approximation algorithm for optimal-time trajectory planning, in: *Proceedings IEEE International Conference on Robotics and Automation*, Scottsdale, AZ (1989).

[20] B.R. Donald and P. Xavier, Provably good approximation algorithms for optimal kinodynamic planning for cartesian robots and open chain manipulators, in: *Proceedings 6th ACM Symposium on Computational Geometry*, (Berkeley, CA 1990).

[21] B.R. Donald and P. Xavier, Time-safety trade-offs and a bang-bang algorithm for kinodynamic planning in: *Proceedings IEEE International Conference on Robotics and Automation*, Sacramento, CA (1991).

[22] M. Erdmann, Personal Communication (1992).

[23] M. Erdmann, Using backprojections for fine motion planning with uncertainty, *Int. J. Rob. Res.* **5** (1) (1986).

[24] M. Erdmann, On probabilistic strategies for robot tasks, Ph.D. Thesis, MIT-AI-TR 1155, Department of EECS, MIT AI Lab, Cambridge, MA (1989).

[25] M. Erdmann, Towards task-level planning: action-based sensor design, Tech. Report, CMU-CS-92-116, Carnegie-Mellon School of Computer Science, Pittsburgh, PA (1991).

[26] M. Erdmann and M. Mason, An exploration of sensorless manipulation, in: *IEEE International Conference on Robotics and Automation*, San Francisco, CA (1986).

[27] M. Fischer, N. Lynch and M. Merritt, Easy impossibility proofs for distributed concensus problems, *Distrib. Comput.* **1** (1986) 26–39.

[28] P.C. Fischer, Turing machines with restricted memory access, *Inf. Control* **9** (4) (1966) 364–379.

[29] D.Y. Grigoryev, Complexity of deciding Tarski algebra, *J. Symb. Comput.* **5** (1) (1988) 65–108.

[30] J.E. Hopcroft, J.T. Schwartz and M. Sharir, On the complexity of motion planning for multiple independent objects; *PSPACE*-hardness of the "Warehouseman's Problem", *Int. J. Rob. Res.* **3** (4) (1984) 76–88.

[31] J.E. Hopcroft and J. Ullman *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).

[32] I. Horswill, Analysis of adaptation and environment, *Artif. Intell.* **73** (1995) (to appear).

[33] J. Jennings and D. Rus, Active model acquisition for near-sensorless manipulation with mobile robots, in: *International Association of Science and Technology for Development (IASTED) International Conference on Robotics and Manufacturing*, Oxford, England (1993).

[34] J. Jennings, Sensor interpretation and task-directed planning for autonomous agents, Ph.D. Thesis, Computer Science Department, Cornell University, Ithaca, NY (1994)

[35] D. Kozen, Automata and planar graphs, fundamentals of computing theory, in: L. Budach, ed., *Proceedings Conference on Algebraic, Arithmetic, and Categorical Methods in Computation Theory* (Akademie Verlag, Berlin, 1979).

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

[37] T. Lozano-Pérez, Spatial planning: a configuration space approach, *IEEE Trans. Comput.* **32** (1983) 108–120.

[38] T. Lozano-Pérez, M.T. Mason and R.H. Taylor, Automatic synthesis of fine-motion strategies for robots, *Int. J. Rob. Res.* **3** (1) (1984).

[39] V.J. Lumelsky and A.A. Stepanov, Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape, *Algorithmica* **2** (1987) 403–430.

[40] M.T. Mason, Mechanics and planning of manipulator pushing operations, *Int. J. Rob. Res.* **5** (3) (1986).
[41] M. Minsky, Recursive unsolvability of Post's problem of 'Tag' and other topics in the theory of Turing machines, *Ann. of Math.* **74** (3) (1961) 437–455.
[42] S. Narasimhan, Personal Communication (1993).
[43] B.K. Natarajan, On planning assemblies, in: *Proceedings Fourth Annual Symposium on Computational Geometry*, Urbana, IL (1988) 299–308.
[44] J. Rees and B.R. Donald, Program mobile robots in scheme, in: *Proceedings IEEE International Conference on Robotics and Automation*, Nice, France (1992).
[45] J. Reif, Complexity of the mover's problem and generalizations, in: *Proceedings 20th Annual IEEE Symposium on Foundations of Computer Science* (1979); also in: J. Schwartz, J. Hopcroft and M. Sharir, eds., *Planning, Geometry and Complexity of Robot Motion* (Ablex, Norwood, NJ, 1987) Ch. 11, 267–281.
[46] S.J. Rosenschein, Synthesizing information-tracking automata from environment descriptions, Teleos Research, Tech. Report No. 2 (1989).
[47] D. Rus, Fine motion planning for dexterous manipulation, Ph.D. Thesis, Tech. Report CU-CS-TR 92-1323, Computer Science Department, Cornell University, Ithaca, NY (1992).
[48] A. Tarski, *A Decision Method for Elementary Algebra and Geometry* (University of California Press, Berkeley, CA, 2nd ed., 1951).
[49] P.G. Xavier, Provably-good approximation algorithms for optimal kinodynamic robot plans, Ph.D. Thesis, Tech. Report CU-CS-TR 92-1279, Computer Science Department, Cornell University, Ithaca, NY (1992)