On Information Invariants in Robotics

Bruce Randall Donald*

TR 93-1341
April 1993

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*This paper describes research done in the Robotics and Vision Laboratory at Cornell University. Support for our robotics research is provided in part by the National Science Foundation under grants No. IRI-8802390, IRI-9000532, and by a Presidential Young Investigator award, and in part by the Air Force Office of Sponsored Research, the Mathematical Sciences Institute, Intel Corporation and AT&T Bell laboratories.

An abbreviated version of this report
has been submitted to *Artificial Intelligence*.

On Information Invariants in Robotics

Bruce Randall Donald*
Computer Science Department
Cornell University
Ithaca, New York

April 22, 1993

*This paper describes research done in the Robotics and Vision Laboratory at Cornell University. Support for our robotics research is provided in part by the National Science Foundation under grants No. IRI-8802390, IRI-9000532, and by a Presidential Young Investigator award, and in part by the Air Force Office of Sponsored Research, the Mathematical Sciences Institute, Intel Corporation, and AT&T Bell laboratories.

On Information Invariants in Robotics

Bruce Randall Donald

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.

Contents

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 [Erd89], although rigorous examples of information invariants can be found in the theoretical literature from as far back as 1978 (see, for example, [BK, Koz]).

Part I of this paper develops the basic concepts and tools behind information invariants in plain language, and builds on a number of examples. In part II, we provide a fairly detailed analysis; therein 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 [Don1] for a detailed review) has been to determine what information is required to solve a task, and to direct a robot's actions to acquire that information to solve it. Key questions concern:

1. What information is needed by a particular robot to accomplish a particular task?
2. How may the robot acquire such information?
3. What properties of the world have a great effect on the fragility of a robot plan/program?
4. What are the capabilities of a given robot (in a given environment or class of environments)?

These questions can be difficult, because 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. How much "information" is encoded in the assumptions, and how much "information" must the control

[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 [Erd89].

system or planner compute? Second, successful manipulation strategies often exploit properties of the (external) physical world (eg, compliance) to reduce uncertainty and hence gain information or 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. In other words, since during execution there may really very little "computation" in the sense of "algorithm," traditional techniques from computer science have been difficult to apply in obtaining meaningful upper and lower bounds on the true task complexity. We hope that a theory of information invariants can be used to measure the sensitivity of plans to particular assumptions about the world, and to in fact minimize those assumptions where possible.

We would like to develop a notion of information invariants that could be used to characterize 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. In computational geometry, for example, a rather successful measure has been developed for characterizing input sizes and upper and lower time bounds for geometric algorithms. Unfortunately, this measure seems less relevant in robotics, although it is a useful tool. Its apparent diminished relevance in embedded systems reflects the increasing belief of many roboticists that we may not reasonably assume that the robot, on booting, reads a geometric model of the world from a disk and proceeds to plan. We would also like to consider paradigms where the robot investigates the world and builds data structures that in some sense represent the external environment.

If a measure of intrinsic information invariants can be found, then it leads rather naturally to a measure of hardness or difficulty. If these notions are truly intrinsic, then these invariants could serve as "lower bounds" in robotics, in the same way that lower bounds have been developed in computer science.

In our quest for an intrinsic measure of the information requirements of a task, we are inspired by Erdmann's monograph on sensor design [Erd91]. Also, we note that many interesting lower bounds (in the complexity-theoretic sense) have been obtained for motion planning questions (see, eg, [Reif, HSS, Nat, CR]; see, eg, [Don2, Can, Bri] for upper bounds). Finally, Rosenschein has developed a theory of synthetic automata which explore the world and build data-structures that are "faithful" to it [Ros]. His theory is set in a logical framework where sensors are logical predicates. Perhaps our theory could be viewed as a geometric attack on a similar problem. This work was inspired by the theoretical attack on perceptual equivalence begun by [DJ] and by the experimental studies of [JR]. Horswill [Hors] has developed a semantics for sensory systems that models and quantifies the kinds of assumptions a sensori-computational program makes about its environment. In addition to the work discussed here in sec. 1, for a detailed bibliographic essay on previous research on the geometric theory of planning under uncertainty, see, eg., [Don1] or [Don3].

These goals 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 provides some practical and theoretical motivations for our approach. In part II 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 sec. 6 below. Section 9 attempts to discuss the semantics of sensor systems rather precisely; as such this section is a bit more 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 sec. 9 are then used to derive algebraic algorithms for reducing one sensor to another.

A preliminary version of this paper was reported in [Don4, Don5].

Part I – State, Communication, and Side-Effects

2 Examples

2.1 A Following Task

2.1.1 A Method of Inquiry

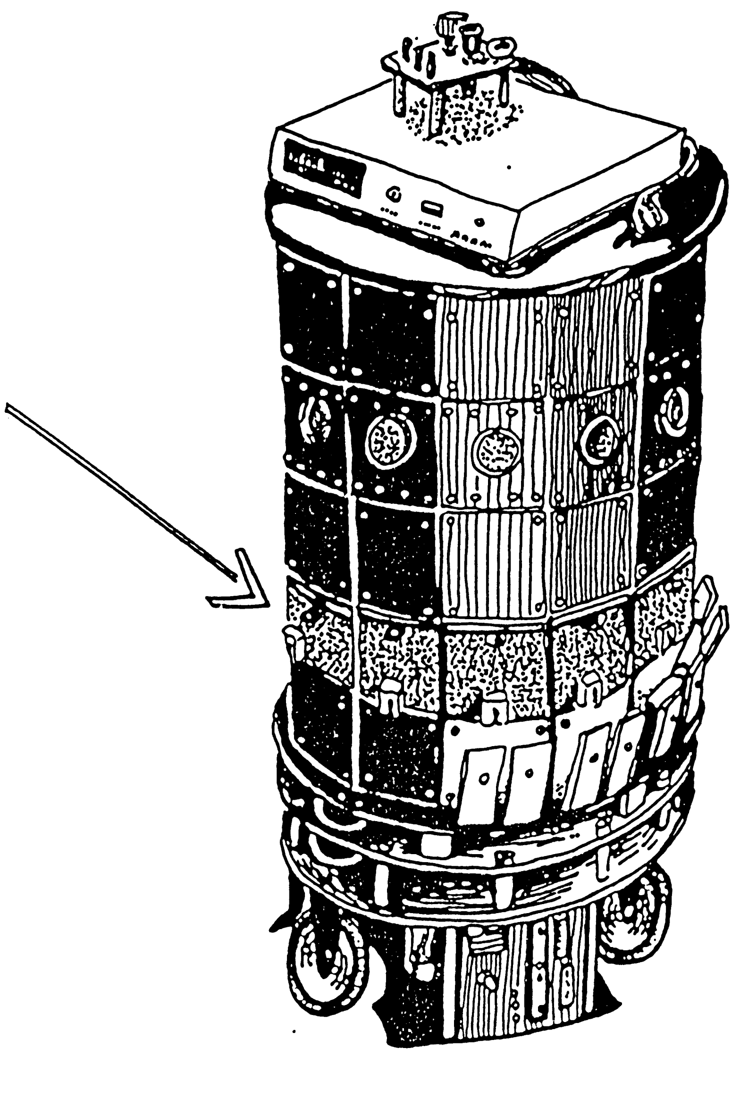
To introduce our ideas we consider a task involving two autonomous mobile robots. One robot must follow the other. Now, many issues related to information invariants can be investigated in the setting of a single agent. We wish, however, to relate our discussion to the results of Blum and Kozen (in sec. 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. 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 very 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 infra-red 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 [RD]. The robots we

Figure 1



Arthur Angelici Feb '92

Figure 1: The mobile robot TOMMY. LILY is very similar. Note (arrow) the IR Modems mounted on the enclosure.

have in mind are the Cornell mobile robots [RD], but the details of their construction are not important. The robots can move about by controlling motors attached to wheels. The robots are autonomous and equipped with a ring of 12 simple Polaroid ultrasonic sonar sensors. Each robot has 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.[2] It is possible to write such a control loop using only sonar readings and position/force control alone.[3]

We now augment the robots described in [RD] as follows. (This description characterizes the robots in our lab). We equip each robot with 12 infrared modems/sensors, arrayed in a ring about the robot body. Each modem consists of an emitter-detector pair. When transmitting or receiving, each modem essentially functions like the remote control for home appliances (eg, TV's).[4] In experiments, we have found that the communication bandwidth we can expect is roughly 2400 baud-feet. That is, at a distance of 1 foot between LILY and TOMMY, we can 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 (eg, `'Tommy`), the angle θ_i, and the

[2] These are the names of our robots.

[3] Many robots perform following tasks; for an example of such a following program, see, for example, [RD].

[4] 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.

speed v. That is, he transmits the following triple:[5] $\langle\ id,\ \theta_i,\ v\ \rangle$.

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 IR's), the heading, and the name of the other robot.[6] 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 the environment. The screen (see fig. 2) is in local coordinates for each robot.[7] 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*[8] [LoP] 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 very common way of synthesizing a a feedback control loop. See

[5] 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 (eg, reflections off white walls).

[6] This data is noisy, but since an adequate servo loop for following can be constructed using sonars alone [RD], the IR's 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.

[7] In the language of [DJ], 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 [RD]. The details are not particularly important to this discussion.

[8] See [Lat] for a good introduction.

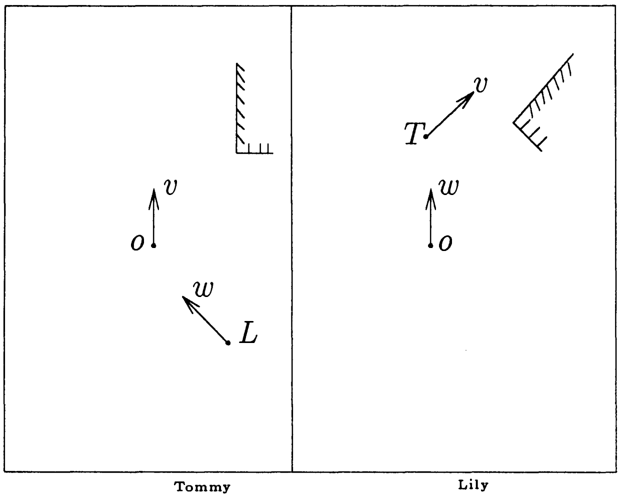


Figure 2: The "radar screens" of TOMMY and LILY. TOMMY (T) is approaching a wall (on his right) at speed v, while LILY (L) follows at speed w.

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 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

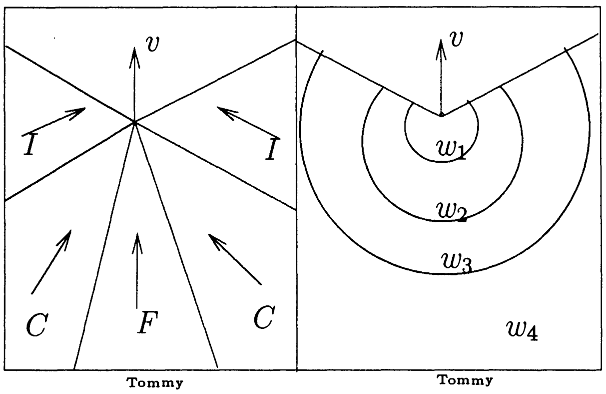


Figure 3: The statespace "radar screen" of TOMMY is partitioned to indicate the control for LILY. (For the task of following, we could partition LILY's screen instead, but this is clearer for exposition). On the left is LILY's direction control; and the regions are F (follow), C (correct), and I (intercept). 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.

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 & 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

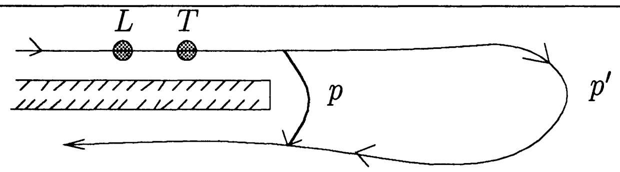


Figure 4: Following around a wall. The shorter path p is quicker by Δt than p', but it cannot be executed without more communication or state.

same path as p, but slowed-down so it takes the same time as[9] p'. 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*[10] 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 $\langle$ `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.

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 [RD]), we need merely replace that. However, the storage for the plan (or path) p could be significant, depending on the detail.

[9] So, p^* is the time-rescaled trajectory from p [DX1].

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

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 [Erd86] 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, and let us call this amount s.

There is a trade-off between speed (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.

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

$$c = |p|\Delta t = |D|\Delta t = s\Delta t. \tag{1}$$

Equation (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 E.1). Moreover, the methods we describe below typically yield results using "order" notation (big-oh $O(\cdot)$ or big-theta $\Theta(\cdot)$) instead of strict equality.

One example of provable information invariants is given in the *kinodynamic* literature [CDRX, DX1, DX2]. This work is concerned with provable planning algorithms for robots with dynamics. We give some details in appendix E.1. Here we note that Xavier, in [Xa, DX3] 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 (eg, the "cost" of a control strategy). One might view our work (and also [BK], below) as investigating information invariants in the absence of a performance measure. See appendix E.1 for more details.

2.2 The Power of the Compass

In 1978, Blum and Kozen wrote a ground-breaking paper on maze-searching automata [BK,Koz]. This section (2.2) is devoted to a discussion of their paper, *On The Power of the Compass* [BK], and we interpret their results in the context of autonomous mobile robots and information invariants. The reader is urged to consult the clear and readable paper [BK] for more details.

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

Question 2.2 [**DJ2**] *"Let us consider a rational reconstruction of mobile robot programming. There is a task we wish the mobot 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 nodes are control and sensing actions, computation, and state retention. A particular kind of state consists of geometric constructions; in short, we imagine the mobot 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 [LS] 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 rather precise answers to these questions in the setting of theoretical, situated automata. This section (2.2) didactically adopts the rhetorical "we" to compactly interpret their results. While these results are theoretical, we believe they provide insight into the 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).[11]

We say two (or more) automata *search a maze together* as follows. When two automata land on the same square, each transmits its internal state to the other.

Finally, 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 DFA's). 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. I believe that quanti-

[11] 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 [Fi] 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, eg., [HU; Ch. 5]). The proof that a two-counter DFA can simulate a Turing machine was first given by Minsky in 1961 [Min] but shorter proofs are now given in many textbooks, for example, see [HU; Thm. 7.9].

fying 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, eg, [Don3]) or the mechanics of the task (see, eg,. [Mas]) 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.[12] Next they prove the following:

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 very 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.[13] This is a very precise answer to part of our question 2.2.

However, the points (1-3) also demonstrate interesting information invariants. (1) = (2) demonstrates the equivalence (in the sense of information)

[12]See [BK] for references.

[13]Here is the idea. First, [BK] 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, [BK] show how all the free space can then be systematicically 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 $y_{\min} \leq n$, and the algorithm consumes $O(\log n)$ bits of storage. For details, see [BK].

of beacons and communication, and hence suggests that in a sense, side-effecting the environment is equivalent to collaborating with an autonomous co-agent. The equivalence of (1) and (2) to (3) suggests an equivalence between communication, state, and side-effecting the environment. Hence we may credit [BK] with a excellent example of information invariance.

2.2.1 The Power of Randomization

Erdmann's PhD thesis is an investigation of the power of randomization in robotic strategies [Erd89]. 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.[14] 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 sec. 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 some.

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 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

[14] 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).

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 sec. 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,[15] [BK] 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 node, there are some number $g \geq 0$ of edges leading to "free" other nodes, 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"* [BK]. Recall point (1) above (marked ($*$)). Blum and Kozen show, that in contrast, to (1), no two automata together can search all finite planar cubic[16] graphs. They then prove no three automata suffice. Later, Kozen showed that four automata do not suffice [Koz]. 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 [BS].

Hence, [BK,Koz] provide a very precise answer to the question, "What information does a compass provide?" We close by mentioning that in the

[15] 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 [Erd91].

[16] All nodes have degree $g = 3$.

flavor of sec. 2.2.1, there is a large literature on randomized search algorithms for graphs. As in sec. 2.2.1, randomization can improve the capability and performance of the search automata.

3 Discussion

We have described the basic tools and concepts behind information invariants. We illustrated by example how such invariants can be analyzed and derived. We motivated our work by making an analogy between information invariants and trade-offs. This brings one naturally to consider kinodynamic situations, in which performance measures, planning complexity, and robustness (in the sense of resistance to control uncertainty) are traded-off. It is worth remarking that Erdmann's invariants are also of this kind [Erd89].

Without a performance (cost) measure, it is more difficult to develop information invariants. In this regard our work is allied with Erdmann's proposal [Erd91] or the work of [DJ]. 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 mobots, and (c) dropable beacons. With regard to (a), we note that, of course, memory chips are cheap, but in the mobot 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 [Nat] 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.

Using very simple models, we explored applications that foregrounded use of these resources.

We then exploited automata-theoretic results to explore invariants that trade-off (internal) state, communication, and (external) state. While part I of this paper 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, the invariants (a)-(c) prove sufficiently powerful to quantify the information a compass supplies.

The compass invariant illustrates exactly 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.

Part II – Sensors and Computation

4 Measuring Information

We continue the discussion of information invariants begun above. In part I, we discussed 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?* and *(c) How can the robot change (side-effect) the environment in order to record state or sensory information to perform a task?* In part II, we introduce some additional concepts in order to extend our discussion to include two additional important measures of the information complexity of a robotic task: *(d) How much information is provided by sensors?*, and *(e) How much computation is required by the robot?* In particular, we describe 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 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), (b), (c), and (e) among collaborating autonomous agents.

5 Sensors

Intuitively, we can imagine a *sensor system* being implemented as a tree of computation, in which the nodes are control and sensing actions, computation, and state retention [DJ]. Given two sensor systems E and H, we would like to be able to quantify the information the sensors provide. In this endeavor, our work is allied with [DJ].[17] 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 that they deliver "equivalent" information, that is, whether $E \cong H$. More generally, we would like to compute when the one system delivers more information, i.e., $E < H$, and when the systems deliver *incomparable* kinds of information (that is, $E < H$ and $E > H$). Most ambitiously, we would like to be able to write an "equation" like

$$E \cong H + \boxed{\cdot} \tag{2}$$

and we could rigorously specify what box $\boxed{\cdot}$ 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 this paper we discuss some methods for achieving these goals. To illustrate our techniques, describe two sensors, the *radial* sensor [Erd91], 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 rather precisely.

Definition 5.1 *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 (2) represents "adding" something to H. Informally, this "something" is what we would like to call a *resource* (below, in sec. 5.2.1).

[17] We employ the notions of *concrete* and *virtual* sensors, following [DJ].

The precise semantics of $+$ are given in section 9. $\cong$ is an equivalence relation, but to see this rigorously requires reading section 9. However, we will try to give an idea of our model that can be understood without all the formalism.

In section 9 we formalize our model of sensor systems by viewing them 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 correspond to "data paths" through which information passes. Different embeddings of these graphs correspond to different spatial allocation of the "resources." Our idea involves asking: *What information is added (or lost) in a sensor system when we change its embedding?* and *What information is preserved under all embeddings?* In section 9 we also define the operator $+$ precisely as a way to "combine" sensor systems. The operation is like taking the union of two graphs. In order to determine what classes of embeddings preserve information, we examine *geometric codesignation constraints* (the term comes from [Cha]).

5.1 The Radial Sensor

We begin with a didactic example. In [Erd91] 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 (abstract) 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. Suppose the robot can command its direction of motion v_θ with respect to a global coordinate system. The *radial sensor* returns the value θ_g which is the direction of the goal from the current configuration. The robot, in turn, need only command v_{θ_g} to reduce its distance to the goal.[18] Let θ be a direction in absolute coordinates (that is, with respect to some fixed reference frame not on the robot). In this example, the robot knows its true orientation (which

[18] In the language of [DJ], the perceptual equivalence classes for this sensor are the rays emanating at the goal.

we will call h), and it can command the motion v_θ (which is in absolute orientation coordinates), and the robot will move in direction θ, regardless of h. This example easily generalizes to the case where there is uncertainty in the robot's control system (that is, the "aim" of v_θ) see [LMT, Erd91]. It is plausible (and indeed, Erdmann proves) that this sensor is necessary and sufficient to write a feedback loop that provably attains the goal.

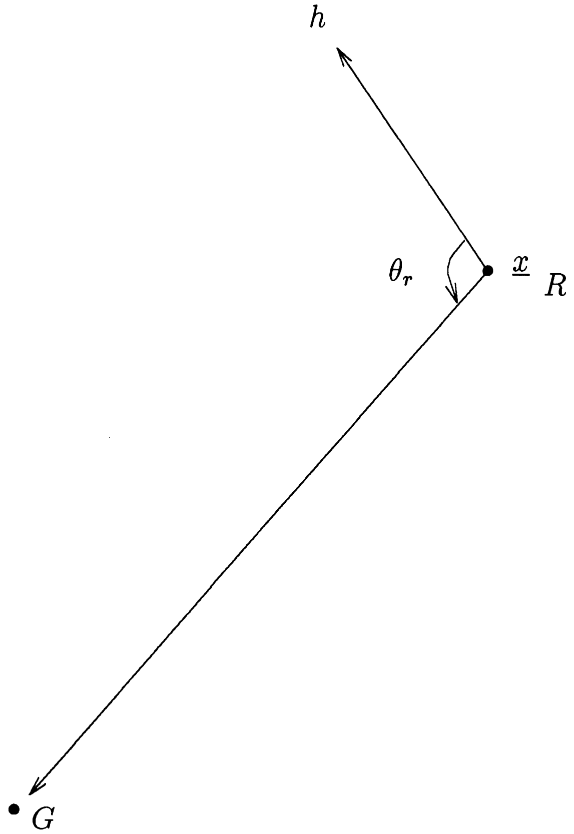


Figure 5: The Modified Radial Sensor E, showing heading h and relative goal direction θ_r.

We wish to adapt this example for the mobile robot domain, and therefore we make the following modification. See fig. 5. As before, the robot is at some configuration $\mathbf{x} \in \mathbb{R}^2$, and at some heading $h \in S^1$. Both these state variables are unknown to the robot. However, the robot can only command *relative* motions (relative to the local coordinate system specified by $(\mathbf{x}, h)$). Thus, instead of v_θ, it would command $v_{\Delta\theta}$, and the robot would move in (global) direction $h + \Delta\theta$. The radial sensor, in this case, need only return the angle θ_r which is the angle between h and the ray between $\mathbf{x}$ and the goal (see fig. 5).

It is interesting to note that in this modified model, the robot task (achieving the goal G, above) is still possible; in fact, we have not altered the robot capabilities required by Erdmann so far as this task is concerned. Hence, we still obtain essentially the same sensor. Thus, the radial sensor returns information that encodes the *relative* direction of the goal (relative to the *heading* h of the robot). In the special case where the robot always knows its heading, or, if the robot's heading is always fixed (say, due North, so that h is always identically zero), then even 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.

To summarize: the radial sensor we will consider 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 $(\mathbf{x}, h)$ of the robot beyond this. We will not describe possible physical implementations of the radial sensor, but see [Erd91] for a discussion.

5.2 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.[19] 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.

[19] I am grateful to Jeff Koechling for pointing out to me how lighthouses work. I understand that radio beacons for instrument-based aircraft navigation work the same way (however, they "rotate" at several megahertz).

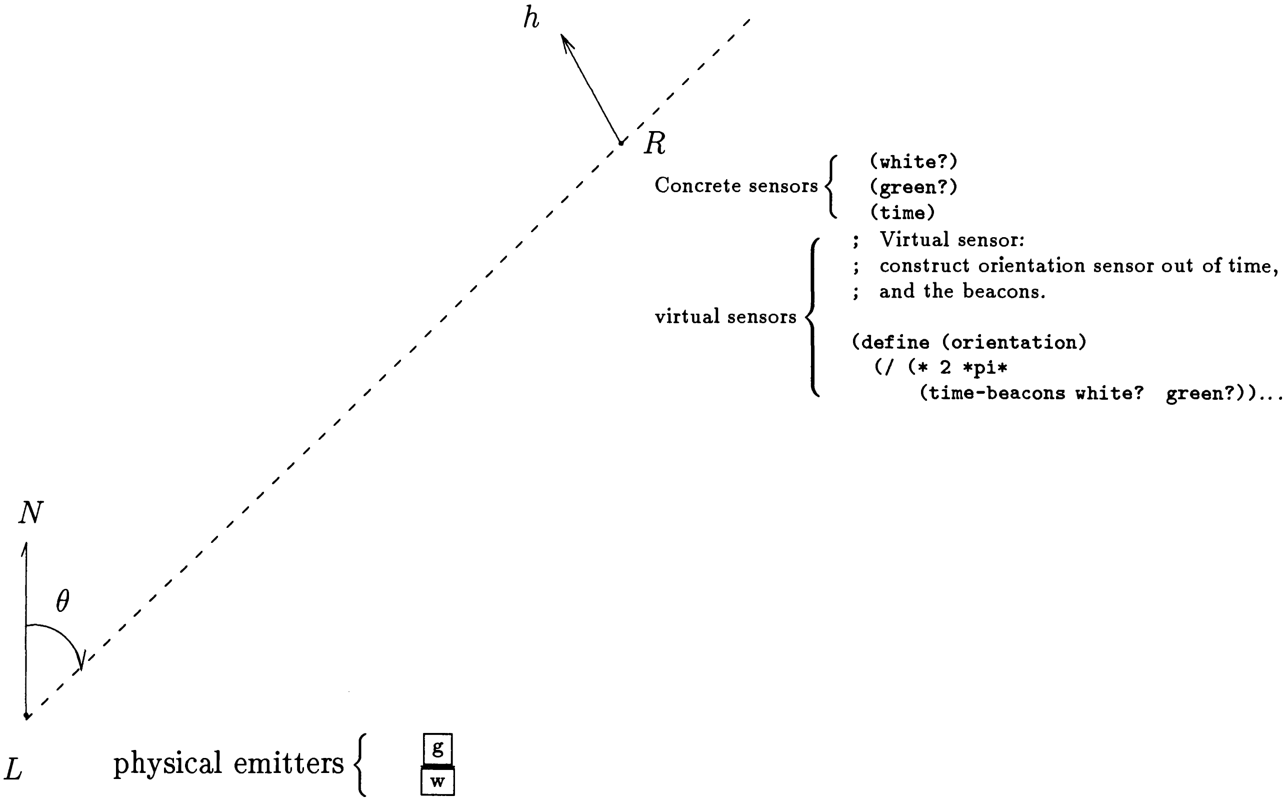


Figure 6: The "beacon" sensor H, which is based on the same principle employed by lighthouses.

5.2.1 Resources

Now, to analyze the information invariants, we must be very careful about the implementation of the sensor system, and, in particular, we must be careful to *count* how resources (a) – (e) (sec. 4) 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 (rotates) about L. The green light can only be seen by points on that ray. Second, there is an omnidirectional white light $\boxed{\text{w}}$ 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 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 [RD]. The concrete sensors above are interfaced to Scheme via library functions (as in [RD]). 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 computed as $\theta = 2\pi t/t_w$. (Assuming the ship is moving slowly, relative to t_w).

We can implement this as a *virtual sensor* `(orientation)` shown below (see [DJ]). 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 very 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:

Virtual Sensors. Given the resources above, we can implement the following virtual sensors "on" R:[20]

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

[20]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).

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

;  time between beacons
;  event  is type UNIT → BOOL.
(define (time-beacons event1 event2)
  (sleep event1)
  (let ((T0 (time)))
     (sleep event2)
     (- (time) T0)))

:  utility in scheme48 [RD].
;  sleep until thunk returns #t.
(define (sleep thunk) ....)
```

Resources R does not have. Real ships use lighthouse sensors as follows. The 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 sec. 4. Our robot R, however, is not a ship, and it has none of these resources. We call R *unenhanced*, following part I.

6 Reduction of Sensors

6.1 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. It should be clear that these sensors deliver incomparable information. By this, we mean that we cannot transform the information delivered by H into the information spec of E (and vice-versa), without consuming more resources. Hence, the sensors are *incomparable* (see sec. 4), in the sense that neither really delivers strictly more information than the other. In order to compare the sensor power, we introduce a mechanism called *reduction.*

6.2 Sensor Reduction

The analytic goal of sensor reduction is to be able to write "equations" like (2). The operational goal is to build one sensor out of another, and to measure the "power" 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 sec. 5.1, we assume that R is located at $\mathbf{x} \in \mathbb{R}^2$ and has heading $h \in S^1$. However, R cannot sense these state variables and it does not know its configuration $(\mathbf{x}, h)$. 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.

6.2.1 A Reduction by Adding a Compass

We sketch a way to construct sensor E from H. This way is very easy since it involves adding a very powerful tool, namely a *compass* to H. Here is the reduction (see fig. 7):

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

(2) We add a concrete sensor h_R called a *generalized compass*[21] to R. The compass senses the heading h. In the language of [LMT], the compass senses the projection of the perfect position sensor $p^* \in \mathbb{R}^2 \times S^1$ onto S^1.

(3) R computes θ using the function `(orientation)` above, and then computes $\theta_r = \pi - h - \theta$. (See fig. 7).

In this reduction all the changes are made to R and L remains the same. Thus we can effectively place (1-3) above in the box in eq. (2). The reduction also adds a small amount of computation (but only a constant amount).

[21] Since this orientation sensor could in principle be implemented using odometry or dead-reckoning, we call it a *generalized* compass.

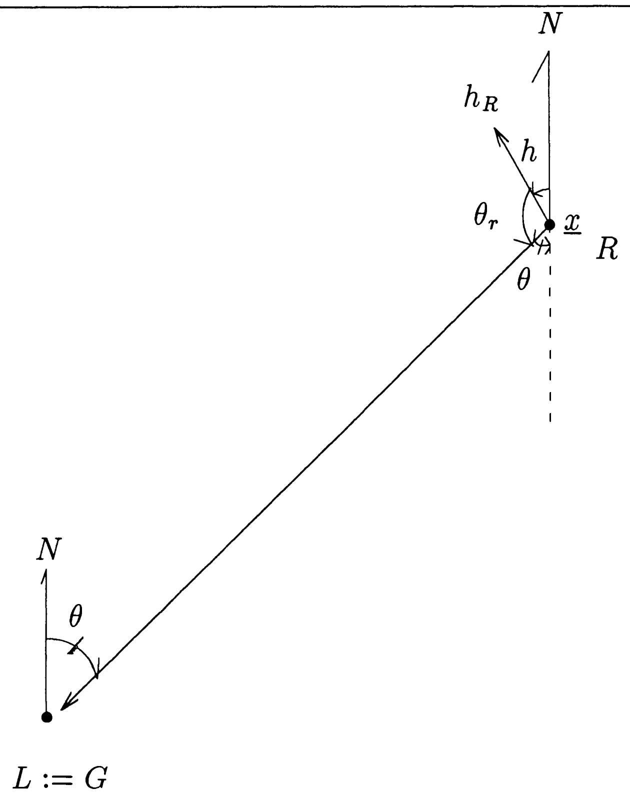


Figure 7: Reduction using a generalized compass h_R.

6.2.2 Reduction using Permutation and Communication

This next reduction 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 sec. 4. We consider adding communication primitives of the form $\text{COMM}(L \to R,$ *info*$)$, which indicates that L sends message *info* to R. Like permutation,

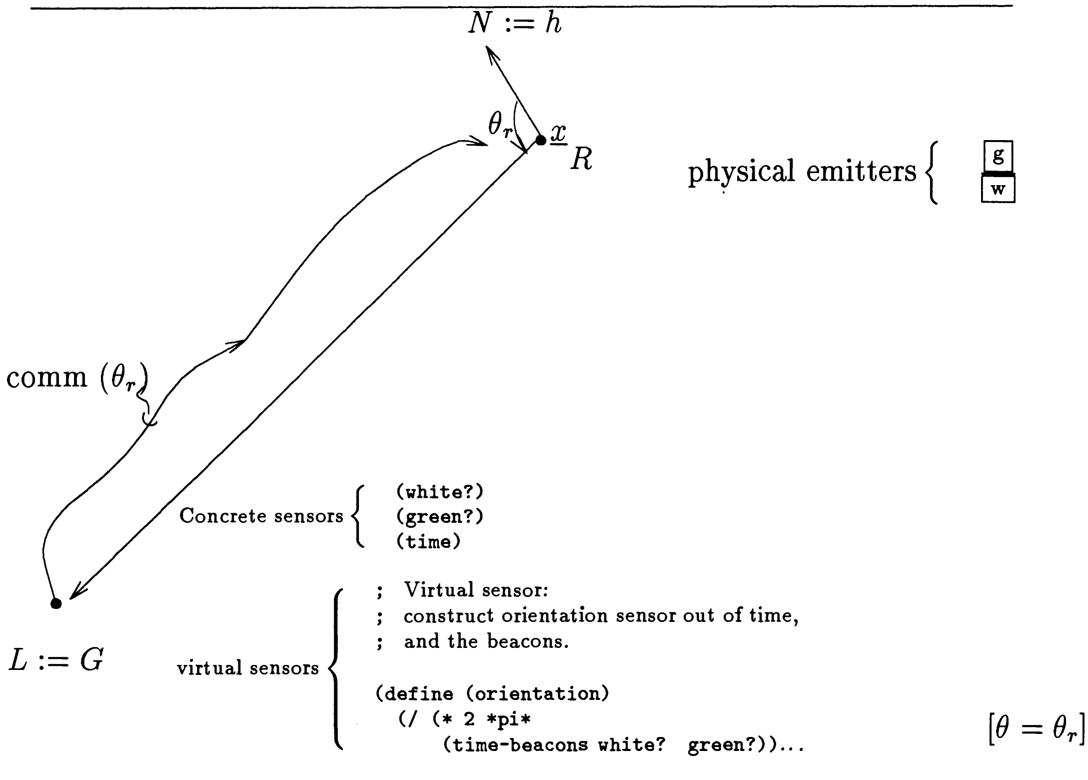


Figure 8: Reduction using Permutation and Communication.

communication only makes sense in a spatially distributed sensor system. In effect, 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., $L \to R$), and so we will henceforth abbreviate it by COMM(*info*).

Given these concepts, we can sketch another reduction. See fig. 8.

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

(ii) We move the physical emitters $\frac{\boxed{g}}{\boxed{w}}$ 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.

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

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

See fig. 8. Given (i-iv), 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($\cdot$) primitive described above:

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

Note that the permutation steps (ii - iv) 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.

6.3 Installation Notes

Installation notes describe how physical resources should be lined up during permutation. They often mask a prescription for calibration, as we shall see. In essence, by carefully lining up (calibrating) two resources, we add information to the system.[22] Hence, we may put off the task of dealing with sensor uncertainty from the execution phase to the installation or layout phase. We illustrate this as follows.

(1), (i) 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. (To see this, displace L from G and calculate).

(H) When installing the physical emitters on L, we must make sure that "North" for the emitters line up perfectly with true North. Compare (ii), below.

(2) When installing the generalized compass, we must make sure that it lines up perfectly with the heading of the robot.

[22] This section (6.3) devolves to a suggestion of Mike Erdmann [Erd], for which we are very grateful.

(ii) 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.

Hence, by eliminating uncertainty in installation, we perform a kind of calibration. This eliminates that uncertainty for the lifetime of the calibration.

6.3.1 Calibration Complexity

It is difficult to precisely measure the information gained in calibration. However, we note the following. First, the calibration in (H), (2), and (ii) each add an equivalent amount of information to the system: each installation requires calibration of two 1 DOF systems, each of which has configuration space S^1. Hence we say that (H), (2), and (ii) are *equivalent installation calibrations*.

Now let us consider (1) and (i) above. This installation requires a careful calibration of two 2DOF systems. To calibrate H at $L = G$ clearly adds information. Consider the following. We have so far considered the radial sensor E at a fixed goal G in the plane. Let us make this dependence explicit by writing E_G; hence we obtain a family of sensors $\{ E_G \}$ parameterized by $G \in \mathbb{R}^2$.

Similarly, let us denote by H_G the sensor system H installed with $L = G$. Now, our goal is to approximate one *particular* E_{G_0} using some H_G. Clearly, we could consider the case $G \neq G_0$; however in specifying E_{G_0} we specify G_0, and so this information is given. That is, it is no more work to locate H at G_0 than to locate E at G_0, and the latter is unavoidable; it is the only way to implement E_{G_0}. Hence, we are 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_0—there is a sense in which the problem of approximating E cannot be specified without calibrating to some G_0. This argument is similar to saying that certain algorithms must at least read all their input. In this case, we say that the calibrations (1) and (i) 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 6.1 *Consider two sensor systems S and Q. When S and Q require equivalent installation calibration, and when the calibration required to install Q is necessary to specify S, we say that S* dominates *Q* in calibration complexity.

In sec. 6.2.1 we described a reduction using a generalized compass that yields a new sensor system from H. In sec. 6.2.2 we described a reduction using permutation and communication, obtaining a different new sensor system from H. From the preceding discussion (sec. 6.3), 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), sec. 4) required for the task. Quantifying external state is tricky, since the time at which the resource is allocated (eg., the time of calibration) may be much earlier than the time of the task execution. We developed the rather detailed theory of calibration complexity (above), precisely to deal with this problem. Finally, it is worth noting the peculiar role of *time* in this analysis (in that calibration and execution may be very distant in time). We found it surprising that time would appear so crucial not only here, but also in the virtual sensor (`orientation`).

6.4 Comments on Power

The reduction in sec. 6.2.1 requires adding an orientation sensor (which may be implemented using a compass or odometry). The reduction in sec. 6.2.2 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 (ii-iv) above. Recall the orientation sensor h_R for R, described in sec. 6.2.1. Thus, in the language of eq. (2), we have sketched how

$$\begin{aligned} E_G &\cong H_G + h_R \\ E_G &\cong H^*_G + \textsc{comm}(\theta_r). \end{aligned} \tag{3}$$

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 sec. 7 and a formal definition in section 9. In section 9, we also describe the semantics of our sensor model in detail.

6.4.1 Output Communication

There is more to be said about this construction. The term $\text{COMM}(\theta_r)$ in (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 ($L \to R$) of this information within the permuted sensor system H^*_G. More strongly, it is clear that we can make the permuted sensor system H^*_G satisfy the information spec of E_G if we merely permit it one internal re-routing operation on the output information. In this case, we say the sensor is restricted to *output communication*.[23] We give a more formal, graph-theoretic model of this restriction in section 9.7.2.

7 A Hierarchy of Sensors

The examples above illustrate a general principle. It is analogous to the notion of *reduction* in the theory of computation. (The analogy is somewhat tenuous, but we hope not too misleading). Consider two sensor systems S and Q. Recall the definitions of *simulation* (def. 5.1) and *calibration complexity* from sec. 6.3.1.

Definition 7.1 *We write $S < Q$ when*

1. *Q simulates S ($S \cong Q$), and*
2. *S dominates Q in calibration complexity.*

Calibration exploits external state. Def. 7.1 allows us to order systems on how much information this external state (from calibration) yields. Now, let Q^* denote a permutation of sensor system Q, as described in sec. 6.2.2. (For a formal definition, see def. 9.6.)

Definition 7.2 *We write $S <^* Q$ if there exists some permutation Q^* of sensor system Q such that $S < Q^*$.*

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

Recall the meaning of COMM(*info*) from (3) and sec. 6.2.2. Finally,

Definition 7.3 *Consider two sensor systems S and Q, and let b be the output of sensor S. We say Q is* efficiently reducible *to S if*

$$S <^* Q + \text{COMM}(b).$$

In this case we write $S <_s Q$.

Note that in def. 7.3, we restrict the sensor system on the r.h.s. to output communication (sec. 6.4.1).

We have presented our examples before the definitions to make our paper easier to read. However, we recap a couple of crisp results on internal communication and permutation of resources.

Claim 7.4 *(a) $E_G < H_G + h_R$ and*
(b) $E_G < H_G^* + \text{COMM}(\theta_r)$.

Proof: Recall the discussion from sec. 6.3.1 on calibration complexity. To obtain (a), we use the reduction that employs a compass (sec. 6.2.1). The proof of (b) obtains by the reduction using permutation and communication (sec. 6.2.2). □

Now, recall eq. (3). The relation $E_G \cong H_G + h_R$, which derives from the compass reduction in sec. 6.2.1, does *not* imply efficient reducibility, since adding a new concrete sensor h_R is too powerful for our definitions. However, by the reduction in sec. 6.2.2:

Proposition 7.5 *The lighthouse sensor system H is efficiently reducible to Erdmann's radial sensor E, that is $E <_s H$.*

Proof: Recall from eq. (3) that $E_G \cong H_G^* + \text{COMM}(\theta_r)$, and that θ_r is the output of E_G. From this, and claim 7.4, we conclude that $E <_s H$. □

8 Information Invariants

The relation $<_s$ defines a hierarchy of sensors. Compare the perceptual lattice of [DJ], who propose a geometric program for the analysis and synthesis of sensors based on their perceptual equivalence classes. The relation $<_s$ orders sensor systems on the complexity of their information invariants.[24]

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

$$H_G + h_R = H_G^* + \text{COMM}(\theta_r). \tag{4}$$

Let us recall the restriction to output communication described in sec. 6.4.1 and def. 7.3. Recalling that h_R denotes the generalized compass, at first glance, we would appear to obtain the following information invariant: *a generalized 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 (4) is valid. However, a careful examination will reveal 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 9.8, we address the generality of eq. (4) in somewhat more detail. There

[24]It is possible to develop a geometric account of information invariance by pursuing the direction of [DJ]. For more on this connection, see appendix C. The account we give in section 9 is also geometric but with a different flavor. Appendix C deals with the geometry of lattices, where an element of the lattice represents (essentially) a knowledge state. In section 9 we examine different embeddings of sensor systems. "Permutations" or "automorphisms" of the function space of embeddings that preserve the sensor functionality are viewed as a kind of information-preserving transformation, and, hence, a model of information invariance.

[25]In the language of [DJ], communication and permutation permit us to map between the perceptual equivalence classes (PEC's) of E (the rays described in sec. 5.1) and the PEC's of H.

we model the colocation of resources as geometric *codesignation constraints.* This colocation can be modeled as a quotient map, and in section 9.8 we discuss its relationship to information invariance.

9 On The Semantics of Situated Sensor Systems

In this section, we formalize our model of sensor systems. We give more formal definitions of the reductions using permutation, and by "combining" sensor systems and "adding" resources. Section 9 attempts to discuss the semantics of sensor systems rather precisely; as such this section is a bit more 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 sec. 9 are then used to derive algebraic algorithms for reducing one sensor to another.

9.1 Situated Sensor Systems

Definition 9.1 *A* labelled graph *$\mathcal{G}$ is a directed graph (V, E) with vertices V and edges E, together with a labelling function that assigns a label to each vertex and edge. Where there is no ambiguity, we denote the labelling function by ℓ.*

Definition 9.2 *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.

By convention, we restrict the *components* to be the resources described in sec. 5.2.1; however, our definitions could, we feel, be generalized to other resources. For technical reasons, we define a one new resource called "output." Each sensor system must have exactly one vertex with this label. The output vertex of the sensor system is where the output of the sensor is measured. In the examples introduced in sec. 5 (the radial sensor E, lighthouse sensor system H, and the permuted lighthouse sensor H^*), we locate the output vertex on the "ship" at R.

Connections are like data-paths in that they carry information; a connection's label represents the information that will be sent along that path. One common connection is specified using the $\textsc{comm}(\mathit{info})$ primitive introduced in sec. 6.2.2. For example, recall the the permuted sensor system H^* introduced in sec. 6.4. Next, recall eq. (3):

$$\begin{array}{rcl} E_G & \cong & H_G + h_R \\ E_G & \cong & H^*_G + \textsc{comm}(\theta_r). \end{array} \tag{3}$$

Consider the sensor system specified by the bottom r.h.s. of eq. (3):

$$H^*_G + \textsc{comm}(\theta_r) \tag{$*$}$$

In the graph representation of $(*)$, the edge from the virtual `(orientation)` sensor at G to the output 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 the configuration of the component. Some components have configurations that vary with the state of the system (for example, in the lighthouse sensor system, all components mounted on the ship vary with the ship's configuration). Others are installed at fixed configurations. For example, the *emitters* $\frac{\boxed{\text{g}}}{\boxed{\text{w}}}$ in the lighthouse example, are installed at a specific position (L) and orientation (the white light flashes when the green light points North). So, for example, 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 $v \in V$). Now, we give

Definition 9.3 *A* situated *(or* embedded) sensor system $\mathcal{S}$ *is a sensor system* $\mathcal{S} = (V, E)$*, together with an embedding* $\phi : V \to C$ *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 essentially an embedded graph.

Convention 9.4 *The term* embedding *in def. 9.3 generally implies an injective map. We do not require our maps to be injective, and in fact, in order to colocate vertices, it it necessary for them to be non-injective. Because it is convenient, we will retain the term "embedding" in any case, to denote the map* $V \to C$ *that specifies the geometry of the components.*

In def. 9.3, the embedding function 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 embedding. We denote the *domain* of a (partial) embedding $\phi : V \to C$ by $\phi^{-1}C$. We denote its *image* by $\operatorname{im}\phi$.

Example 9.5 *H_G is a situated sensor system (H, ψ). H_G^* is a different embedding 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 embedding with the same domain. More formally:

Definition 9.6 *Let $\mathbb{S} = (\mathcal{S}, \phi)$ be a situated sensor system. A* permutation *$\mathbb{S}^*$* of *$\mathbb{S}$ is a situated sensor system $(\mathcal{S}, \phi^*)$ such that the domain $\phi^{-1}C$ of ϕ and the domain ${\phi^*}^{-1}C$ of ϕ^* are the same.*

Convention 9.7 *Consider the definition 5.1 of* simulation. *Suppose we write*

$$(\mathcal{S}, \phi) \cong (\mathcal{U}, \psi) \tag{5}$$

for two situated sensor systems. Suppose further 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. (5) is taken to mean that $(\mathcal{U}, \psi)$ simulates $(\mathcal{S}, \phi)$ for any *configuration of v and u.*

Note that the definition of permutation (def. 9.6) also makes sense for partial embeddings. However, see section 9.9.2 for a discussion of the semantics of permutation for unsituated sensor systems.

9.2 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 embedding 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 un-situated system $\mathcal{S}$. This is the notation we use in sec. 6.3.1

and after. Of course, for this notation to capture all the information above about v, we must specify the preimage[26] of G_0 under ϕ, but we did that in sec. 6.3.1 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 9.5. First, we formalize our discussion of $\mathcal{S}_{G_0}$, above:

Definition 9.8 *A* Pointed Embedding *of a sensor system* $\mathcal{S} = (V, E)$ *is a pair* (ϕ, G) *where* $\phi : V \to C$ *is a map of the vertices of* $\mathcal{S}$, *and* $G \in \operatorname{im}\phi$. *As usual, the map* ϕ *need not be injective, and it may be a partial function.*

Definition 9.9 *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 embedding (def. 9.8) of* $\mathcal{S}$. *We abbreviate* $(\mathcal{S}, \phi, G)$ *by* $\mathcal{S}_G$.

Hence, H_G in example 9.5 is a pointed sensor system. Next,

Definition 9.10 *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 9.5 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 $\mathcal{S}_G^*$.

9.3 Codesignation: Basic Concepts

If we view the configurations of components in a sensory system as "variables," then convention 9.7 gives a "default" for determining which variables are "free" and which are "bound." Here is another view:

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

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

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 [Cha]). Two vertices r and u *codesignate* under an embedding ϕ when $\phi(r) = \phi(u)$. More generally, r and u codesignate under *different* embeddings 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 sec. 9.2 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 embed 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:

> Embed $\mathcal{S}$ with vertex v at G_0. Now, vertex r of $\mathcal{S}$ will be somewhere, say, R; but we want to embed $\mathcal{U}$ 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 embeddings of $\mathcal{S}$ and $\mathcal{U}$. We call this a *codesignation constraint* after [Cha]. Here is how we may say this more precisely:

> Let $\mathcal{S}_{G_0}$ denote sensor system $\mathcal{S}$ embedded with vertex v at G_0 (as above). Embed the rest of $\mathcal{S}$ in any consistent manner, and denote this embedding by $\bar{\phi}$. Thus $\bar{\phi}$ is the *extension* of ϕ so that the restriction $\bar{\phi}_{|\{v\}}$ of $\bar{\phi}$ to $\{v\}$ is identical to ϕ. Now, let $R \in C$ be the configuration of r under $\bar{\phi}$, i.e., $R = \bar{\phi}(r)$. Denote by ψ the (partial) embedding 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 $\bar{\phi}$ of ϕ we choose, whereas G_0 does not.

In eqs. (2–4), we abbreviated this construction as follows:

$$\mathcal{S}_{G_0} + \mathcal{U}_R. \tag{6}$$

which is short for $(\mathcal{S}, \phi) + (\mathcal{U}, \psi)$ with ϕ and ψ defined as above. Note that (6) is not sufficient to specify the desired (partial) embedding unless we also note that the preimage (under the embedding ϕ) of G_0 contains vertex v of $\mathcal{S}$, and that

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

(*) represents a codesignation constraint; we will define such constraints formally below in sec. 9.5.1. We must also specify that G_0 is a constant and R is a free variable. The notation explained in (6) 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 below in defs. 9.11- 9.13 below.

9.4 Combining Sensor Systems

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

Definition 9.11 *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 (def. 9.2) by lifting the definition for graphs. We may define + on two embedded graphs whenever the embeddings are *compatible.* An embedding ϕ of $\mathcal{G}$ and an embedding ψ of $\mathcal{G}'$ are said to be *compatible* when the two embeddings agree on the intersection $V \cap V'$ (for total embeddings) or more generally, on $\phi^{-1}C \cap \psi^{-1}C$ (for partial functions). Given this definition 9.11 we have:

Claim 9.12 *The operator + defined in def. 9.11 is associative and commutative.*

Proof: Definitional. $\square$

9.5 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 embedding is a singleton, eg, $\phi(V) = \{ G \}$ and $\psi(U) = \{ R \}$. In these cases it suffices to abbreviate

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

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 embeddings.

Suppose $(\mathcal{S}, \phi)$ and $(\mathcal{U}, \psi)$ have compatible partial embeddings. 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 9 have this form: each was a pair $(v, u) \in V \times U$. A codesignation constraint is *compatible* with the embeddings 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 def. 9.15 below.

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

Definition 9.13 *Let $(\mathcal{S}, \phi)$ and $(\mathcal{U}, \psi)$ be two situated sensor systems with compatible partial embeddings. The* combined sensor system

$$(\mathcal{S}, \phi) + (\mathcal{U}, \psi) \tag{7}$$

is specified by (7), together with a set of codesignation constraints compatible with ϕ and ψ. We say the combination (7) is defined *when the partial embeddings ϕ and ψ are compatible.*

Definition 9.13 specializes to the particular cases such as eq. (3) we have considered, by appropriate choice of partial embeddings and codesignation constraints. To illustrate these choices, we give an example below, in section 9.6. The operator $+$ is associative and commutative (claim 9.12).

9.5.1 Codesignation Constraints

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

Definition 9.14 *Define the partial embedding $\phi + \psi$ as follows:*

$$\begin{array}{ccc} \phi + \psi : & \to & C \\ x & \mapsto & \begin{cases} \phi(x) \text{ if } x \in V; \\ \psi(x) \text{ if } x \in U. \end{cases} \end{array}$$

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

Definition 9.15 *A* codesignation constraint *is a pair $(x, y) \in (V \cup U)^2$.*

Definition 9.16 *We say a codesignation constraint (x, y) is* compatible with the partial embeddings 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 embedding $\overline{\phi + \psi}$ that extends $\phi + \psi$,

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

holds. A *noncodesignation* constraint requires *inequality* (instead of equality) in $(*)$.

9.6 Example: The Basic Idea

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

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

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 $\frac{\boxed{\text{g}}}{\boxed{\text{w}}}$ located at G and oriented Northward.

When H is situated at G as above to obtain H_G, the embedding is partial, leaving the position R of the ship, unspecified in H_G. h_R denotes the generalized 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 r.h.s. of eq. (3) to (6). As in (6), in eq. (3), once the preimages (under the embedding) of G and R are specified, the embedding of the combined sensor system becomes clear.

Now, H^*_G defines a new embedding of H (by "new" we mean different from H_G). The embedding depends on R but the equation (3) holds for any R. $\textsc{comm}(\theta_r)$ 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 $H^*_G + \textsc{comm}(\theta_r)$.

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 on the l.h.s. and both r.h.s. of eq. (3) indicates a codesignation constraint.

9.7 Example (continued): A More Formal Treatment

9.7.1 The Top of Equation (3)

We now rewrite eq. (3) using the general notation of section 9.5. 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 embedding of E. Let ϕ_G be a partial embedding of E that installs it at G, so that $E_G = (E, \phi_G)$.

Let ψ be a partial embedding of H. Let ψ_G be a partial embedding of H that installs the emitters $\frac{\boxed{\text{g}}}{\boxed{\text{w}}}$ 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 sec. 5.2.1. In particular, u_1 is the vertex of the virtual sensor (orientation).

Let u_o be the "output" vertex of H.

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

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

together with the codesignation constraints[27]

$$\begin{array}{c} \{ (u_1, u_i) \}_{1 < i \leq k} \\ \cup \left\{ \quad (v_1, v_2), (u_o, u_1), (u_1, w) \quad \right\}. \end{array} \qquad (8)$$

9.7.2 The Bottom of Equation (3)

Now, H^* denotes a different embedding of H. Call this embedding ψ^*. Let ψ^*_G denote the partial embedding that installs the concrete and virtual sensors at G. We will define codesignation constraints so that the emitters are installed on the ship.

Now, $\text{COMM}(\theta_r)$ defines the graph with vertices

$$(u_1, u_o)$$

and a single edge $e = (u_1, u_o)$ with $\ell(e) = \theta_r$. We observe that the restriction to output communication (sec. 6.4.1 and def. 7.3) is equivalent to the following:

> *The output vertex u_o must be the "head" vertex of the edge $e = (u_1, u_o)$.*

[27] 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.

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

$$(E, \phi_G) \cong (H, \psi_G^*) + \textsc{comm}(\theta_r) \tag{3−bot}$$

together with the codesignation constraints

$$\begin{array}{ll} & \{\,(u_1, u_i)\,\}_{1<i\leq k} \\ \cup & \left\{(v_1, v_2)\right\}. \end{array} \tag{9}$$

Hence the bottom codesignation constraints (9) for (3-bot) are different from the top codesignation constraints (8) for (3-top), in that in the bottom constraints, w does not appear (since it is associated with the generalized 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.

9.7.3 Calibration Complexity and Codesignation

The size of the set (8) or (9) (number of codesignation constraints) is one measure of *calibration complexity* (see sec. 6.3.1). However, this should be only part of the 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 $\boxed{\text{g}}$ in H. Break up the emitter $\boxed{\text{g}}$ 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 very easy to compute it (in fact, like "size" above, it can be computed in the same time it takes to read the input).

9.7.4 Task Specification: Freeing G

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. We now discuss this operation in terms of codesignation constraints.

Let r be the "central vertex" of the radial sensor E (this is the vertex located at G in fig. 5). To obtain the effect of freeing G, we rewrite eqs. (3-top) and (3-bot) as follows:

1. Remove the G subscripts: that is, replace ϕ_G by any embedding ϕ of E. Similarly, replace ψ_G by ψ and ψ_G^* by ψ^*.
2. Add the codesignation constraint (v_1, r) to (3-top).
3. Add the codesignation constraint (u_1, r) to (3-bot).

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.

9.7.5 Noncodesignation Constraints

To complete our model in this example, we must also introduce noncodesignation constraints so that $G \neq R$; this is necessary for our sensors to work. The noncodesignation constraint for both (3-top) and (3-bot) is

$$(u_1, v_1). \tag{10}$$

9.8 Generality and Codesignation

Consider a sensor system $\mathcal{S}$ with d vertices V, embedded via a map $\phi : V \to C$. The configuration space of this sensor system can be viewed as C^d, since any embedding ϕ can be represented as a point in[28] C^d. Consider a

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

codesignation constraint (u, v) for $u, v \in V$. This specifies a new embedding 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 sec. 8, we discussed how general eqs. (3) and (4) 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 each codesignation constraint specifies such an identification). Hence, eq. (4) 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.[29] 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 (eg., reductions and embeddings) on sensor systems.

It is interesting to note that the geometric structure of noncodesignation constraints is rather 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 embedding $\widetilde{\phi} = \pi(\phi)$:

$$\begin{array}{ccc} \phi & \in & C^d \\ & & \downarrow{\scriptstyle \pi} \\ \widetilde{\phi} = \pi(\phi) & \in & C/(u \sim v). \end{array} \tag{11}$$

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 embedding to

[29]Here is the construction: first, we obtain the 2-sphere S^2 as a quotient of $\mathbb{R}^3$ via the quotient map $\pi : \mathbb{R}^3 \to S^2$ given by $\pi(v) = \|v\|$. Next, we get $\mathbb{RP}^2$ from S^2 by identifying antipodal points v and $-v$ of S^2.

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 embedding

$$\phi' \in (C^2 - \Delta) \times C^{d-2}. \tag{12}$$

Combining (11) and (12) gives the general form for the configuration space of the sensor.

9.9 More General Codesignation Relations

9.9.1 The Semantics of Codesignation Constraints

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

$$\phi(u) = \psi(v) \tag{13}$$

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

$$\begin{tikzcd} U \arrow[dr, "\phi"] & \\ & C. \\ V \arrow[ur, "\psi"'] & \end{tikzcd} \tag{14}$$

Let us call this simple kind of codesignation constraints in (13), *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 (s.a.) predicate on $C \times C$. So far, in considering equality codesignation constraints, all the predicates we have used have been diagonals:[30]

$$\Phi(x,y) \text{ iff } x = y. \tag{15}$$

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

This choice (15) 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 $\frac{\boxed{\mathrm{g}}}{\boxed{\mathrm{w}}}$ to be colocated. 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 (15) 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 (def. 9.17) 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 secs. 9–10 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 (s.a.) 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 be an arbitrary variety Y in C^d; Y would be characterized by some polynomial inequalities, and embeddings $\phi \in Y$ 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 rather 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 9.17 *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 (15) is algebraically codesignated, all simple systems are algebraic codesignated. We consider only simple sensor systems in secs. 1–8. However, the algorithms in sec. 10 apply to all algebraically codesignated systems.

9.9.2 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 sec. 10.

The meaning of a permutation (see def. 9.6) is clear for a situated sensor system. Recall from section 9.8 that we can view an embedding ϕ and its permutation ϕ^* as elements of the configuration space[31] C^d. Now, suppose, for a moment, that for every embedding $\phi \in C^d$ it is possible to choose[32] a permutation ϕ^* satisfying def. 9.6. Imagine that for each $\phi_0 \in C^d$, we build a sequence of such choices, $\{\phi_0, \phi_1, \phi_2, \phi_3, \dots\} \subset C^d$, where $\phi_{i+1} = \phi_i^*$. This defines a map

$$\begin{array}{ccccccc} C^d & \rightarrow & C^d & \rightarrow & C^d & \rightarrow & \cdots \\ \phi_0 & \mapsto & \phi_1 & \mapsto & \phi_2 & \mapsto & \cdots \end{array} \tag{16}$$

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 embedding. Then by a permutation ϕ^* of ϕ we mean a different partial embedding with the same domain (the definition 9.6 still applies).

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

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

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

[31] We will ignore, for the moment, the necessity of quotienting C^d and removing diagonals.

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

$$\phi^{-1}C \;=\; \bigcap_{\bar{\phi}\in \mathrm{ex}\,\phi} \bar{\phi}^{-1}C \;=\; \phi^{*-1}C \;=\; \bigcap_{\bar{\phi^{*}}\in \mathrm{ex}\,\phi^{*}} \bar{\phi^{*}}^{-1}C, \tag{17}$$

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 embeddings. 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 9.18 *Let ϕ^* be a permutation of ϕ. Then* $\mathrm{ex}\,\phi$ *and* $\mathrm{ex}\,\phi^*$ *are disjoint, unless $\phi = \phi^*$.*

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

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

But ϕ^* is a permutation of ϕ, which implies that ϕ and ϕ^* have the same domain (def. 9.6). Since $\phi^{*-1}C = \phi^{-1}C$, therefore $\phi = \phi^*$. $\Box$

Let $\Sigma(\phi)$ denote all permutations of ϕ. Essentially, prop. 9.18 tells us that the map $\mathrm{ex} : \Sigma(\phi) \to \{\,\text{Regions in } C^d\,\}$ 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:

Claim 9.19 *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 $\phi^* = \psi_{|_{\phi^{-1}C}}$. $\Box$

Proposition 9.20 *Fix a partial embedding ϕ. The images of* $\mathrm{ex} : \Sigma(\phi) \to \{\,\text{Regions in } C^d\,\}$ *cover C^d, that is,*

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

Proof: Immediate from claim 9.19. □

We can summarize this as follows: we have viewed permutation as a self-map of $\Sigma(\phi)$. It is equivalent to view permutation as a self-map among 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 9.21 *The map*

$$\begin{array}{rccl} p_\phi : & C^d & \to & \Sigma(\phi) \\ & \psi & \mapsto & \phi^* \quad \text{s.t. } \psi \in \operatorname{ex} \phi^* \end{array} \tag{18}$$

is well-defined.

Proof: Observe that $p_\phi(\psi) = \psi_{|_{\phi^{-1}C}}$ (see claim 9.19). The map p_ϕ is defined for every $\psi \in C^d$, by props. 9.19 and 9.20. That $p_\phi(\psi)$ is uniquely defined by (18), we see from prop. 9.18. □

Now, suppose the domain $\phi^{-1}C$ of ϕ contains k vertices, $1 \leq k \leq d$. 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 9.22 *The following properties hold:*[33]

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

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

$$\begin{array}{rccl} p_\phi : & C^d & \to & C^k \\ & (z_1, \ldots, z_k, \ldots, z_d) & \mapsto & (z_1, \ldots, z_k). \end{array} \tag{19}$$

3. *Let ϕ^* be a permutation of ϕ. Then* $\operatorname{ex} \phi^* \subset C^d$ *is a cylinder over $\phi^* \in C^k$, and* $\operatorname{ex} \phi^* = {p_\phi}^{-1} \phi^*$.

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

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 embedded 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 the additional properties:[34]

$$G \in \operatorname{im}\phi \quad = \bigcap_{\bar\phi \in \operatorname{ex}\phi} \operatorname{im}\bar\phi, \qquad G \in \operatorname{im}\phi^* \quad = \bigcap_{\bar\phi^* \in \operatorname{ex}\phi^*} \operatorname{im}\bar\phi^*. \tag{20}$$

Thus for (partially) embedded systems, we have a handle on permutation, and now we know more precisely what the difference between (eg.) H_G and H_G^* is, (see sec. 6.3.1) in terms of permutation. Permutation corresponds to choosing a different equivalence class of C^d. However, it remains to define an "abstract" version of permutation that is applicable to unsituated sensor systems. This will allow us to understand H^* as a permutation of H.

9.9.3 Permutation of Unsituated Sensor Systems

We now define an "abstract" version of permutation that is applicable to unsituated sensor systems (for example, H, from sec. 5). Recall eq. (3), and in particular (3-top) and (3-bot). Consider the codesignation constraints (8) for eq. (3-top) and (9) for eq. (3-bot). Each set of codesignation constraints corresponds to a different family of embeddings. Each family of embeddings, in turn, corresponds to a region in C^d. These regions are disjoint (assuming that the noncodesignation constraint (10) is in place).

Hence, "abstract" or "unsituated" permutation corresponds selecting a different family of embeddings (or, equivalently, selecting a different set of 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 situated permutations discussed above (sec. 9.9.2). Hence, abstract permutation can be viewed as a map

[34]For pointed sensor systems, the surjection-like properties (props. 9.19 and 9.20) only hold for the class of pointed embeddings (with the same base point).

$$C^d/(u \sim v) - \Delta \quad \longrightarrow \quad C^d/(u' \sim v') - \Delta', \tag{21}$$

where $(u \sim v)$ and $(u' \sim v')$ denote different codesignation constraints and Δ and Δ' different noncodesignation constraints. Abstract permutation specializes to situated permutation as follows: for a partial embedding ϕ (resp. ϕ^*) to be chosen consistently with (21), we must have that $\operatorname{ex}\phi$ (resp. $\operatorname{ex}\phi^*$) is contained in the domain (resp. range) of (21). Thus, in general, the quotient structure for the permutation must respect the quotient structure for codesignation.

9.10 The Semantics of Reductions

Recall the definition of *efficiently reducible* (def. 7.3). To explore this notion, we now turn to the question of whether or not the relation $<^*$ in def. 7.2 is transitive. If $<^*$ is transitive, then $<_s$ will be transitive (to see this, recall def. 7.2, and that the $+$ operator is associative and commutative (claim 9.12). To complete the argument, we also need a technical lemma, given by the "distributive" property[35] of prop. B.3).

Consider three sensor systems, $\mathcal{U}$, $\mathcal{V}$, and $\mathcal{W}$, and their permutations:[36]

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

(22)

If $<^*$ is transitive, then if $\mathcal{U} <^* \mathcal{V}$ and $\mathcal{V} <^* \mathcal{W}$, then $\mathcal{U} <^* \mathcal{W}$. We explore when this property holds. From defs. 7.2 and 7.3 we can see that dominance in calibration complexity (def. 6.1) is transitive, and so we will concentrate here on the less obvious aspects of transitivity.[37] To simplify the discussion we only deal with codesignation constraints, but the argument generalizes *mutatis mutandis* for noncodesignation constraints.

[35] See appendix B.

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

[37] See secs. 9.7.3, and A.1 for more on computational calibration complexity.

9.10.1 Weak Transitivity

First, let us observe that $<^*$ always obeys a property that is like transitivity, but "weaker." We now elaborate. Suppose $\mathcal{U} <^* \mathcal{V}$. Then (def. 7.2), there exists some permutation $\mathcal{V}^* = (\mathcal{V}, \beta^*)$ of $\mathcal{V}$ such that $\mathcal{U} < \mathcal{V}^*$ (see def. 7.1 for the definition of $<$). So, we have

$$(\mathcal{U}, \alpha) < (\mathcal{V}, \beta^*). \tag{23}$$

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

$$(\mathcal{V}, \beta^*) < (\mathcal{W}, \gamma^*). \tag{24}$$

From (23) and (24), and the definition of $<$ (defs. 6.1, 7.2) we have

$$(\mathcal{U}, \alpha) < (\mathcal{W}, \gamma^*), \tag{25}$$

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

9.10.2 Strong Transitivity for Simple Sensor Systems

Simple sensor systems (def. 9.17) 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 $<^*$ is transitive: then, if $\mathcal{U} <^* \mathcal{V}$ and $\mathcal{V} <^* \mathcal{W}$, then $\mathcal{U} <^* \mathcal{W}$. In other words:

> Suppose $\mathcal{U} <^* \mathcal{V}$ and $\mathcal{V} <^* \mathcal{W}$. Then there exist permutations $\mathcal{V}^* = (\mathcal{V}, \beta^*)$ of $\mathcal{V}$ and $\mathcal{W}^* = (\mathcal{W}, \gamma^*)$ of $\mathcal{W}$ such that
>
> $$(\mathcal{U}, \alpha) < (\mathcal{V}, \beta^*) \tag{23}$$
>
> and
>
> $$(\mathcal{V}, \beta) < (\mathcal{W}, \gamma^*). \tag{26}$$
>
> (Compare (26) with (24)). Then if $<^*$ is transitive, then there exists another permutation $\mathcal{W}^+ = (\mathcal{W}, \gamma^+)$ of $\mathcal{W}$, such that
>
> $$(\mathcal{U}, \alpha) < (\mathcal{W}, \gamma^+). \tag{27}$$

Strong transitivity is a much stricter condition than weak transitivity. It requires that we be able to "compose" the embeddings β^*, β, and γ^* to somehow construct the embedding 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 (def. 9.17).

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 embedding 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 secs. 9.9.2 and 9.9.3. We call a permutation $\mathcal{V}^* = (\mathcal{V}, \beta^*)$ of $\mathcal{V}$ *valid* if its embedding β^* is valid. In this case we also say that the sensor system $\mathcal{V}^*$ is *valid.*

Lemma 9.23 *The relation $<^*$ (def. 7.2) is transitive for valid simple sensor systems (def. 9.17).*

Proof: Assume there exist valid permutations α, β, β^*, and γ^* so that (23) and (26) hold as above. We construct an embedding γ^+ so that (27) holds.

The picture we have is as follows:

$$\begin{tikzcd} & U \arrow[d, shift right, "\beta"'] \arrow[d, shift left, "\beta^*"] & \\ V \arrow[r, "\alpha"] & C & \\ & W. \arrow[u, shift left, "\gamma^*"] \arrow[u, shift right, "\gamma^+"'] & \end{tikzcd} \tag{28}$$

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 embedding we must simulate all colocations, because that way we will be sure to reproduce all codesignation constraints accurately in the new embedding. Because (only) colocation affects sensor semantics for simple sensor systems (def. 9.17), this suffices to ensure that the new embedding preserves the sensor semantics. In short, colocation is evidence for codesignation.

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

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

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

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

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. $\square$

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 (sec. 10).

We do not believe that the relation $<^*$ holds for arbitrary algebraically codesignated sensors. This is because the algebraic constraints may be incompatible. It would be of interest to find a restricted class that is larger than equality codesignation, for which transitivity holds.

We close with this corollary, which follows from lemma 9.23, by observing that the + operator is commutative and associative, and by the "distributive" property of prop. B.3.

Corollary 9.24 *The relation $<_s$ (def. 7.3) is transitive for simple sensor systems.*

10 Computational Properties

In this section, we give a computational model of simulation (def. 5.1), and discuss an algorithm for deciding the relations $<^*$ and $<_s$. This section relies heavily on the results of the previous section (9). 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.

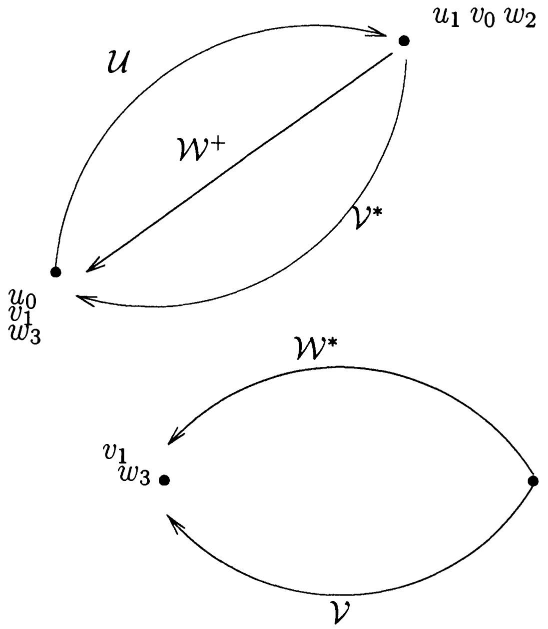


Figure 9: The situated sensor systems $\mathcal{U} = (U, \alpha)$, $\mathcal{V} = (V, \beta)$, $\mathcal{V}^* = (\mathcal{V}, \beta^*)$, $\mathcal{W}^* = (\mathcal{W}, \gamma^*)$, and $\mathcal{W}^+ = (\mathcal{W}, \gamma^+)$ for lemma 9.23. The vertices of $\mathcal{U}$, $\mathcal{V}$, and $\mathcal{W}$ are $U = \{u_0, u_1, \dots\}$, $V = \{v_0, v_1, \dots\}$ and $W = \{w_0, w_1, \dots\}$, (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 $\beta(v_0) = \gamma^*(w_2)$.

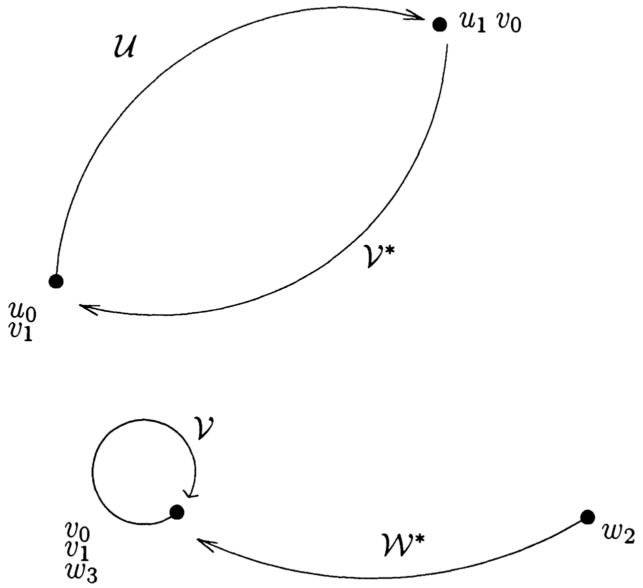


Figure 10: The case where $\beta^{-1}(\gamma^*(w_i))$ is not a singleton (in this case, it is $\{v_0, v_1\} \subset V$). In this example, $\beta(v_0) = \beta(v_1) = \gamma^*(w_3)$. Now, we note that v_0 and v_1 colocate under β but not under β^*. However, this difference cannot be semantic (i.e., it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

10.1 Algebraic Sensor Systems

The algorithms in this section (10) 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 embeddings of sensor systems, since these embeddings are functions. However, since our embeddings have a finite domain, each embedding 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 10.1 *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 embedding that is semi-algebraic (s.a.) and total. Let us define the *size* d of $\mathcal{U}$ to be the number of vertices in $\mathcal{U}$. Now,

Definition 10.2 *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 *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.

Example 10.3 *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 sec. 5.2.1). Other, equivalent simulations are also possible ("equivalent" means they compute the same value for θ). For example: let $(\mathbf{x}, h) \in \mathbb{R}^2 \times S^1$ 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 $\theta = \theta_0 + \tan^{-1}(\mathbf{x} - L)$. We note that this simulation function is not algebraic (because arctangent is not algebraic). See example 10.7, below.*

Now, if the configuration space C is algebraic, then so is the function space C^d. Hence, every embedding 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 $\mathrm{ex}\,\phi \subset C^d$ is also semi-algebraic. We then observe that the expression denoting "for all extensions (resp., there exists an extension) $\bar{\phi}$ of ϕ, $\mathbb{T}(\bar{\phi})$ holds" namely

$$\triangle \bar{\phi} \in \mathrm{ex}\,\phi : \quad \mathbb{T}(\bar{\phi})$$

is also semi-algebraic ($\triangle \in \{\forall, \exists\}$). To quantify over all extensions $\bar{\phi}$ of ϕ, we simply quantify over the configurations of the vertices *outside* the domain of ϕ. By sec. 9.9.2 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[38]

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

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:

$$\exists \phi^* \in \Sigma(\phi),\ \forall \overline{\phi^*} \in \mathrm{ex}\,\phi^* : \quad \mathbb{T}(\overline{\phi^*}). \tag{29}$$

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

Example 10.4 *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 $\psi : V \to C$ semi-algebraically, eg., 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 a s.a. representation of ψ by the name $\sigma(z_1, z_2, z_3)$:*

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

[38]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.

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

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

We can also encode the extensions and permutations of ϕ semi-algebraically. Specifically, we can encode any permutation $\underline{\phi^*}$ 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 (29) as

$$\exists z_1 \, \forall z_2 \, \forall z_3 : \quad \mathbb{T}(\sigma(z_1, z_2, z_3)). \tag{30}$$

If C has dimension r_c, then the formula (30) is a Tarski sentence in $3r_c$ variables.

We summarize:

Proposition 10.5 *If $\phi : V \to C$ is a semi-algebraic partial function, then the set $\operatorname{ex}\phi$ (ϕ's extensions) and the set $\Sigma(\phi)$ (ϕ's permutations) are also semi-algebraic.* □

To guess a *valid* permutation, (def. 9.10.2) we restrict the configurations to lie within the (algebraic) codesignation constraints, as described in secs. 9.9.2 and 9.9.3. (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 a s.a. set $D \subset \mathbb{R}^r$. The structure of the region D, especially in relation to the region $\operatorname{ex}\phi^*$, is discussed in secs. 9.9.2 and 9.9.3. We must restrict our choice of permutation to D. To guess a valid permutation, we modify (29) to be:

$$\exists \phi^* \in \Sigma(\phi), \; \forall \overline{\phi^*} \in D \cap \operatorname{ex}\phi^* : \quad \mathbb{T}(\overline{\phi^*}). \tag{31}$$

Definition 10.6 *We call a sensor system $\mathcal{U}$ <u>algebraic</u> if it is algebraically codesignated (def. 9.17), has an algebraic configuration space C, and a semi-algebraic algebraic simulation function $\Omega_{\mathcal{U}}$ (def 10.2).*

Example 10.7 *Recall example 10.3, above. The simulation function Ω_H in ex. 10.3 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 $<^*$.*

To construct an algebraic version of Ω_H, we use a simple trick from calculus (also used in kinematics; see, for example [DKM]). Let ϕ be a configuration of sensor system H (fig. 6). Define $\Omega'_H(\phi) = (\tan\frac{\theta}{2}, q)$, where $\theta = \Omega_H(\phi)$ (see ex. 10.3), 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 $u = \tan\frac{\theta}{2}$. 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 [DKM] for details. Essentially the graph of Ω'_H is a s.a. set in correspondence with graph of the non-algebraic map Ω_H. The correspondence is given by $\theta \mapsto u$.

10.2 Computing the Reduction $<^*$

Now, suppose we have two algebraic sensor systems $\mathcal{U}$ and $\mathcal{V}$. We wish to decide whether $\mathcal{U} <^* \mathcal{V}$. 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

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

(Here in sec. 10.2 the relation $<$ is used as in def. 7.1). That is, we wish to decide the following (assume that α and β are partial):

$$\left(\exists\beta^* \in \Sigma(\beta), \forall\bar{\alpha} \in \operatorname{ex}\alpha, \forall\overline{\beta^*} \in \operatorname{ex}\beta^*\right): \quad \Omega_{\mathcal{U}}(\bar{\alpha}) = \Omega_{\mathcal{V}}(\overline{\beta^*}). \tag{32}$$

Eq. (32) 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 $D_{\mathcal{U}}$ and $D_{\mathcal{V}}$ in C^d. So (32) becomes:

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

Using Grigoryev's algorithm (thm. A.1) we can decide (33) in the following time. (We use (41) 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 $D_{\mathcal{U}}$ and $D_{\mathcal{V}}$. In (33), the outer existential quantifier binds some number $k \leq d$ vertices of $\mathcal{V}$ that are in the domain of ϕ. The inner universal quantifer 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 (33) in time

$$(m\delta)^{O(r)^{4a-2}} = (n_\Omega + n_D)^{O(r_c d)^6}. \tag{34}$$

Definition 10.8 *Consider an algebraic sensor system $\mathcal{U}$, with d vertices. Recall we call d the* <u>size</u> *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 $\mathcal{U}$* <u>small</u> *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 def. 7.1) in a time that much faster than (34) (see appendix A.1 for how). Then we see the following

Lemma 10.9 *There is an algorithm for deciding the relation $<^*$ (def. 7.2) 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:*

$$(n_\Omega + n_D)^{r_c d^{O(1)}}. \tag{35}$$

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

Corollary 10.10 *For small*[39] *sensor systems (def. 10.8) of size d, there is an algorithm to decide the relation* $<^*$ *in time*

$$d^{r_c d^{O(1)}}. \tag{36}$$

□

Corollary 10.11 *For algebraic sensor systems, the relation* $<_s$ *(def. 7.3) can be decided in the same time bounds as in lemma 10.9 and cor. 10.10.*

Proof: Consider deciding $S <^* Q + \text{COMM}(\theta_r)$, as in def. 7.3. Recall the definition of *compatibility* for partial embeddings (sec. 9.4). We first observe that permutation (the * operation) and combination (the + operation) "commute" for compatible partial embeddings. This is formalized as the "distributive" property[40] shown in prop. B.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 (def. 9.11) of two algebraic sensor systems. Since $\text{COMM}(\theta_r)$ is a constant-sized sensor system (2 vertices, one edge) with only a constant number of codesignation constraints (1 constraint), we may guess how to combine it with a permutation Q^* of Q within the same time bounds given in lemma 10.9 and cor. 10.10. □

10.2.1 Simple Sensor Systems

Michael Erdmann [Erd] has suggested that the bound (35) in lemma 10.9 might be improved to

$$O(n_\Omega {n_D}^{r_c d}) \tag{37}$$

for *simple* sensor systems (def. 9.17). The argument is as follows. For simple sensor systems we consider n_D constraints in k dimensions, where $k = r_c d$. The resulting arrangement can be constructed in time $O({n_D}^k)$, and contains $O({n_D}^k)$ sign-invariant cells (with respect to the n_D codesignation constraints). Simplicity of the sensor system suggests that to determine equivalence, we may need to test only one representative point in each sign-invariant

[39]Recall all small systems are algebraic.

[40]See appendix B.

set. Simulation on a single point takes time $O(n_\Omega)$, and there are $O({n_D}^k)$ test points.

The chief obstacle to obtaining this improved bound is that, even for simple sensor systems, it appears we must test equivalence of the simulation functions on all points in each cell of the arrangement. (For example: consider two simulation functions that differ at exactly one point of a cell). This leads us to the seemingly necessary universal quantifier in eq. (33).

It also seems possible that (for simple sensor systems), our bound (35) might be improved by exploiting the linearity of the constraints. For example, one might employ the techniques of [LL], who give a quantifier elimination procedure for systems of linear constraints.

11 Conclusions

In this paper we suggested a theory of information invariance that includes sensors and computation. Our results generalize the work of [BK]; 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 sensory-computational devices. As such, perhaps our work could be called *On the generalized power of generalized compasses.*

However, we hope that information invariants can serve as a framework in which to measure the capabilities of robot systems, to quantify their power, and to reduce their fragility with respect to assumptions that are engineered into the control system or the environment. We believe that the equivalences that can be derived between communication, internal state, external state, computation, and sensors, can prove valuable in determining what information is required to solve a task, and how to direct a robot's actions to acquire that information to solve it. 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)?*

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.

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 communication, 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 equivalences that can be derived among systems containing the resources (a)-(e) (sec. 4). 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 heterogenous resource categories.

11.1 The Challenge of Mechanics

There is one resource we have not quantified, although we would like to. To add this resource to those listed in sec. 4, we write *(f) How much information is encoded or provided by the task mechanics?* The theme of exploiting task mechanics is important in previous work.[41] I would like to be more explicit about what "task mechanics" means, but it is difficult to find a precise definition. Despite this, its important in robotics is undeniable. I believe 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 of the system. We would like a way to develop information invariants that explicitly trade-off (f) with (a)-(e), 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. Nevertheless I feel it is valuable to give what I consider the most important open problem as a way of measuring what progress we have made towards a theory.

Fig. 11a 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

[41] For example, see the discussion of [Mas, EM, Erd91] in part I.

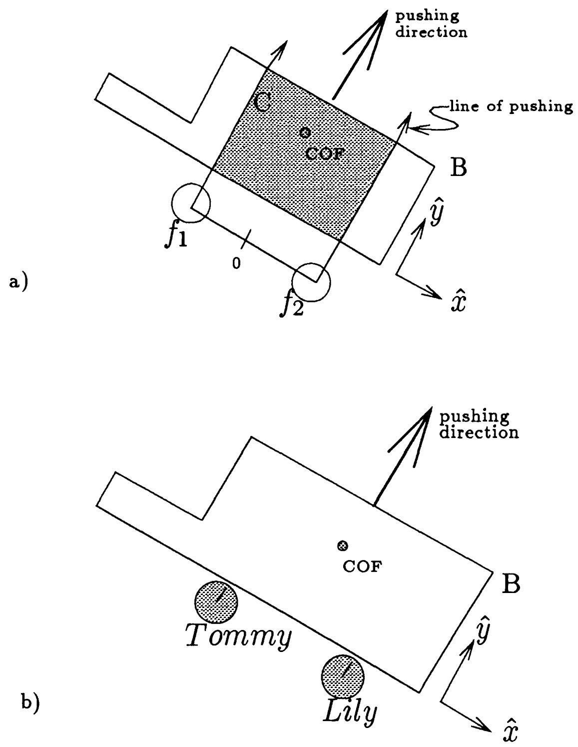


Figure 11: (a) the "two-finger" pushing task vs. (b) the two robot pushing task. The goal is to push the block B in a straight line.

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 some and the fingers can keep pushing, since we only need ensure the COF lies in some region C (see fig. 11a), instead of on a line. Second, if the COF moves outside C, then the fingers can move sideways to capture it again. For example, Jim Jennings implemented the following control loop on our PUMA:

> *($*$) Sense the reaction torque τ about the point* 0 *in fig. 11a. If $\tau = 0$, push forward in direction $\hat{y}$. If $\tau < 0$ move the fingers in $\hat{x}$; else move the fingers in $-\hat{x}$.*

From the mechanics perspective it might appear we are done. However, it is difficult to overstate how critically the control loop ($*$) above relies on global communication and control. Now, consider the analogous pushing task in fig. 11b. Each finger is replaced by an autonomous mobile robot with only local communication, configured as described in sec. 2.1 of part I. Each robot has a ring of one-bit contact ("bump") sensors. In addition, by examining the servo-loop in [RD], it is clear that we can compute a measure of applied force by observing the applied power, the position and velocity of the robot, and the contact sensors.

Now, we ask, how can the system in fig. 11b approximate the pushing strategy ($*$), 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 $\pm\hat{x}$. In our lab, Jim Jennings and Daniela Rus have performed experiments suggesting 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 sec. 2.1 of 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. Such investigations are a fruitful direction for future research.

We believe information invariants for these tasks can be analyzed using our methods. For example, it is clear 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. 11b. 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 conclusion, it is very difficult to analyze precisely the information encoded in the task mechanics—we tried to illustrate this point by alluding to two useful mechanical strategies, the COF pushing strategy and compliant align. However, we believe that information invariants in general, and communication invariants in particular, can help make explicit the information analysis and tradeoffs in sensory-computational systems that significantly exploit the task mechanics. We hope that these tools can be used to measure the information encoded in the task mechanics, and to quantify the resources required to perform robot tasks.

References

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

[BKR] **Ben-Or M., Kozen D., and Reif J.,** "The Complexity of Elementary Algebra and Geometry", J. Comp. and Sys. Sciences, Vol. 32, (1986), pp. 251-264.

[BS] **Blum, M. and W. Sakoda** *On the capability of finite automata in 2 and 3 dimesnional space,* Proc. 17^{th} Symp. Found. Computer Science, pp. 147-61 (1977).

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

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

[Cha] **Chapman, D.** *Planning for Conjunctive Goals,* Artificial Intelligence, **32** (3), pp. 333-378 (July) (1987).

[CDRX] **Canny, J., B. Donald, J. Reif, and P. Xavier** *On The Complexity of Kinodynamic Planning,* 29^{th} Symposium on the Foundations of Computer Science, White Plains, NY pp. 306-316. (1988).

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

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

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

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

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

[Don4] **B. R. Donald** *Information Invariants in Robotics: Part I – State, Communication, and Side-Effects,* Proc. IEEE International Conference on Robotics and Automation, Atlanta, May. (1993).

[Don5] **B. R. Donald** *Information Invariants in Robotics: Part II – Sensors and Computation,* Proc. IEEE International Conference on Robotics and Automation, Atlanta, May. (1993).

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

[DJ2] **Donald, B. R. and J. Jennings** *Constructive Recognizability for Task-Directed Robot Programming,* Proc. IEEE International Conference on Robotics and Automation, Nice, France (1992).

[DJ3] **Donald, B. R. and J. Jennings** *Sensor Interpretation and Task-Directed Planning Using Perceptual Equivalence Classes,* Proc. IEEE International Conference on Robotics and Automation, Sacramento, CA (1991).

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

[DX1] **Donald, B. R. and P. Xavier** *A Provably Good Approximation Algorithm for Optimal-Time Trajectory Planning,* Proc. IEEE International Conference on Robotics and Automation, Scottsdale, AZ (1989).

[DX2] **Donald, B. R. and P. Xavier** *Provably Good Approximation Algorithms for Optimal Kinodynamic Planning for Cartesian Robots and Open Chain Manipulators,* Proc. 6^{th} ACM Symposium on Computational Geometry, Berkeley, CA (1990).

[DX3] **Donald, B. R. and P. Xavier** *Time-Safety Trade-offs and a Bang-Bang Algorithm for Kinodynamic Planning,* Proc. IEEE International Conference on Robotics and Automation, Sacramento, CA (1991).

[Erd] **Erdmann, M.** *Personal Communication,* (1992).

[Erd86] **Erdmann, M.** *Using Backprojections for Fine Motion Planning with Uncertainty,* IJRR Vol. 5 no. 1 (1986).

[Erd89] **Erdmann, M.** *On Probabilistic Strategies for Robot Tasks,* Ph.D. thesis, MIT Department of EECS, MIT A.I. Lab, Cambridge MIT-AI-TR 1155 (1989).

[Erd91] **Erdmann, M.** *Towards Task-Level Planning: Action-Based Sensor Design,* Carnegie-Mellon School of Computer Science, Tech. report, CMU-CS-92-116, Feb. (1991).

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

[Fi] **Fischer, P. C.** *Turing Machines with Restricted Memory Access,* Information and Control, **9** (4), pp. 364-379 (1966).

[Gri] **Grigoryev D. Yu.,** "Complexity of Deciding Tarski Algebra" Jour. Symbolic Computation, **5**, (1), (1988), pp. 65–108.

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

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

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

[JR] **Jennings, J. and D. Rus** *Unpublished Draft Notes on Mobot Manipulation Experiments,* Cornell University (1992).

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

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

[LL] **Lassez, C., and J.-L. Lassez** *Quantifier Elimination for Conjunctions of Linear Constraints via a Convex Hull Algorithm,* in "Symbolic and Numerical Computation for Artificial Intelligence," ed. Donald, B. *et al.,* (Academic Press: London) (reference [DKM]). (1992).

[LoP] **Lozano–Pérez,** T. *Spatial Planning: A Configuration Space Approach,* IEEE Trans. on Computers (C–32):108–120 (1983).

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

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

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

[Min] **Minsky, M.** *Recursive Unsolvability of Post's Problem of 'Tag' and Other Topics in the Theory of Turing Machines,* ANNALS OF MATH, **74**, (3), pp. 437-455 (1961).

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

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

Reif J., "Complexity of the Mover's Problem and Generalizations," Proc. 20th IEEE Symp. FOCS, (1979). Also in "Planning, Geometry and Complexity of Robot Motion", ed. by J. Schwartz, J. Hopcroft and M. Sharir, Ablex publishing corp. New Jersey, (1987), Ch. 11, pp. 267-281.

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

[Tar] **Tarski A.,** "A Decision Method for Elementary Algebra and Geometry" Univ. of Calif. Press, Berkeley, (1948), second ed. 1951.

[Xa] **Xavier, P. G.** *Provably-Good Approximation Algorithms for Optimal Kinodynamic Robot Plans,* PhD. Thesis available as CU-CS-TR 92-1279 (April) from Comp. Sci. Dept., Cornell University (1992).

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, and Kevin Newman; 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, and Amy Briggs for providing invaluable comments and suggestions on drafts of this paper. Thanks to Loretta Pompilio for drawing the illustration in figure 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. 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 Coverse. 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 like to thank the anonymous referees for their comments and suggestions.

APPENDICES

A Algebraic Decision Procedures

The algorithms in section 10 are algebraic and use the theory of real closed fields;[42] for an introduction to algebraic decision procedures see, for example the classic paper of [BKR], or discussions in books such as [DKM, ch 1-4; CLS]. In sec. 10, we reduce our computational problem to deciding the truth of a Tarski formula [Tar]; the algebraic algorithms can then decide such a sentence. Tarski's language is also called the *language of semi-algebraic (s.a.) 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:

$$\begin{array}{rcl} \Big(\forall x \exists y \exists z \forall w\Big) & & \\ xy^2 - 16w^4 & \leq & 0 \\ & \vee & \\ \frac{3}{4}xw^2 + x^7 + 78w & < & 0 \\ & \wedge & \\ z^4 + 5w^3 + 4x^2y^2 - y + 7x & = & 0. \end{array} \tag{38}$$

More generally, we can think of a Tarski sentence as

$$\begin{array}{ll} \Big(\triangle_1 x_1 \; \triangle_2 x_2 \; \cdots \; \triangle_r x_r\Big) & \\ & s_1(x_1, \ldots, x_r) \; R_1 \; 0 \\ & \qquad C_1 \\ & s_2(x_1, \ldots, x_r) \; R_2 \; 0 \\ & \qquad C_2 \\ & \qquad \vdots \\ & \qquad C_m \\ & s_m(x_1, \ldots, x_r) \; R_m \; 0, \end{array} \tag{39}$$

[42] 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."

where each $\triangle_i$ is a *quantifier*, each R_j is a *real relation*, and $C_1, \ldots, C_m$ are *logical connectives*. A *quantifier* $\triangle_i$ 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 $\wedge$.[43] Each $s_1, \ldots, s_m$ is a polynomial in $\mathbb{R}[x_1, \ldots, x_r]$, and so (39) is a sentence in r variables. We call the set $Y \subset \mathbb{R}^r$ defined by (38) or (39) 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 (39). 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

$$\mathbb{T}(x_1, \ldots, x_r) \iff \begin{array}{c} s_1(x_1, \ldots, x_r)\ R_1\ 0 \\ C_1 \\ s_2(x_1, \ldots, x_r)\ R_2\ 0 \\ C_2 \\ \vdots \\ C_m \\ s_m(x_1, \ldots, x_r)\ R_m\ 0. \end{array} \tag{40}$$

$\mathbb{T}(\cdot)$ is called a *semi-algebraic predicate*. Hence, (39) can be written

$$\left(\triangle_1 x_1\ \triangle_2 x_2\ \cdots\ \triangle_r x_r\right)\ :\ \mathbb{T}(x_1, \ldots, x_r).$$

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

$$\triangle\mathbf{x} \in Y :\quad \Phi(\mathbf{x}).$$

Let δ be the a total degree bound for the polynomials $s_1, \ldots, s_m$ in (39). We call the number of polynomials m the *size* of the Tarski sentence (39) and of the s.a. predicate $\mathbb{T}(\cdot)$ in (40). Observe however, that to calculate a bound $O(\delta r m)$ on the number of terms in (39), we would employ the degree bound δ and the number of variables r as well.

[43] So $\leq$ and $\geq$ can be built out of these.

Now, it is rather remarkable that one can decide such sentences in complete generality: although Tarski's original algorithm [Tar] was non-elementary,[44] this bound has been improved by a chain of researchers since then. For example, [BKR] 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 sec. 10, we use this result:

Theorem A.1 (Grigoryev [Gri]) *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 $a \leq r$ is the number of quantifier alternations in the prenex form of the formula, can be decided in time*

$$(m\delta)^{O(r)^{4a-2}}. \tag{41}$$

Proof: See [Gri]. □

A.1 Application: Computational Calibration Complexity

Recall the discussion in sec. 9.7.3. We wish to develop an algorithm for deciding the relation $<^*$ between sensor systems. Comparing the calibration complexity (defs. 6.1, 7.1) of two sensor systems seems easier than the issues of embedding and simulation, because the calibration complexity does not change with the embedding, so long as the embedding 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 sec. A, we can develop tools to measure the complexity of algebraic relations such as those encountered in algebraically codesignated sensor systems (def. 9.17).

Now, to decide the relation $<^*$, we must be able to decide dominance in calibration complexity (see def. 6.1). We propose to measure calibration and installation complexity by the complexity of the codesignation constraints.

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

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 (41), for example. Eq. (41) can then be used as a measure of the sensor's calibration complexity. These bounds can then be compared (using big-Oh ($O(\cdot)$) notation) to determine which sensor dominates in terms of calibration complexity. The comparison can done in essentially the same time it takes to read the input, and the time required is very small compared to (34), the time for the algebraic simulation.

It is relatively easy to measure the complexity of the sensor specification, as we have described. However, to measure the "intrinsic" complexity of the algebraic varieties defined by the codesignation constraints is harder. One may then ask, what is the intrinsic degree of the variety, or how many connected components does it have. This issue can be subtle: for example, the formulæ $x^2 = 1$ and $x^{10000} = 1$ have very different degrees; yet they specify the same root in $\mathbb{R}$ (with different multiplicity). The issue of intrinsic algebraic complexity raises a number of complicated issues in algebraic geometry.

B Distributive Properties

Recall the definition of *compatibility* for partial embeddings (sec. 9.4). In this appendix, we prove some technical properties about the permutation of partial embeddings. These properties are algebraic, and we call them the "*distributive* properties."

Definition B.1 *Let ϕ and ψ be compatible partial embeddings. 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 embeddings, combination and permutation commute. That is: for two compatible partial embeddings ϕ and ψ, if ϕ^* and ψ^* are compatible permutations, then

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

In answer, we can now show the following:

Claim B.2 *Consider two compatible partial embeddings 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 $\phi + \psi$. $\square$

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 def. 9.13)?

In answer, we show the following:

Proposition B.3 *Consider two sensor systems $\mathcal{S}$ and $\mathcal{U}$ as above. Assume their embeddings 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 B.2. $\square$

C An 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 [DJ]. The resulting model is somewhat different in flavor from that of section 9.

Here is a alternate geometric model for an example of information invariance. Let $\mathcal{U}$ be an arrangement of perceptual equivalence classes, as in [DJ; 5.1]. A simple control strategy may be modeled as a subgraph of the RR-graph [DJ3] on $\mathcal{U}$. Now consider the lattice of perceptual equivalence classes formed by fixing the task environment and varying the sensing map, as in [DJ; 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 [45] $\mathcal{W}$, 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.

D 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.[46] Such a framework would be something like a "logical" framework.

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

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

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

$$C = \{\, \ell \mid \ell \text{ is a location.}\,\}. \tag{42}$$

C can be taken to have no structure whatsoever. All the definitions, constructions, and proofs of section 9 then go through *mutatis mutandis:* there is no geometry anywhere.

It is now worth asking, *what are the implications for section 10?* 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 $\Omega_{\mathcal{U}} : C^d \rightarrow R$ where C is as in (42), and R is an arbitrary set. At this point we lose the algebraic properties we exploited to derive the algorithms of section 10. Hence our algorithms do not obtain when we remove the geometric structure. In particular, we lose our main computational result, lemma 10.10. It seems plausible, however, that other deductive mechanisms might be used, instead, to obtain similar results in the abstract (non-geometric) case.

E Provable Information Invariants with Performance Measures

E.1 Kinodynamics and Trade-Offs

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 sec. 2.1. That these invariants eq. (1) are related to kinodynamics [CDRX,DX1,DX2] should come as no surprise, since the execution time for a control strategy is taken as "cost". In [Xa], Pat Xavier introduced a new algorithmic mechanism for measuring kinodynamic trade-offs (see [DX3] for a brief description). These techniques were used to quantify the trade-offs 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

$$\epsilon_T = f_r(\epsilon_S), \tag{43}$$

where f_r is a function conditioned on complexity r. Hence (43) 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 trade-offs are one source of information invariants, and one may even find provable, rigorous characterizations for information questions therein (eg., [DX3, Xa]). However, there is something a bit disatisfying about this line of attack. First, it makes controls, not sensing, the senior partner, much in the same way that in the [LMT] theory (see [Don1]), recognizability is a second-class citizen compared with reachability. In [LMT], this is a consequence of a bias towards sensorless manipulation [EM]; 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.