Robotics: Science and Systems 2024
Delft, Netherlands, July 15-July 19, 2024

An abstract theory of sensor eventification

Yulin Zhang
Amazon Robotics*
North Reading, MA, USA
Email: zhangyl@amazon.com
* This work was done prior to joining Amazon.

Dylan A. Shell
Dept. of Computer Science & Engineering
Texas A&M University
College Station, TX, USA.
Email: dshell@tamu.edu

***Abstract*—Unlike traditional cameras, event cameras measure changes in light intensity and report differences. This paper examines the conditions necessary for other traditional sensors to admit eventified versions that provide adequate information despite outputting only changes. The requirements depend upon the regularity of the signal space, which we show may depend on several factors including structure arising from the interplay of the robot and its environment, the input–output computation needed to achieve its task, as well as the specific mode of access (synchronous, asynchronous, polled, triggered). Further, there are additional properties of stability (or non-oscillatory behavior) that can be desirable for a system to possess and that we show are also closely related to the preceding notions. This paper contributes theory and algorithms (plus a hardness result) that addresses these considerations while developing several elementary robot examples along the way.**

I. Introduction

Advances in sensing technologies have the potential to disrupt the field of robotics. Twenty-five years ago, the shift from sonar to LiDAR sensors triggered a significant change as robots became, rather abruptly, much more capable. Since information enters a robot through its sensors, a change to its sensor suite often has ramifications downstream — sometimes quite far downstream. Accordingly, it is useful to have tools to help us understand the impact of sensor modifications.

The last decade has seen steadily-growing interest in *event cameras*, a novel type of camera that operates on a separate principle from traditional devices [17], [9]. These cameras afford various new opportunities, and a growing body of work has begun exploring these possibilities [24], [41], [29], [33]. At a high level, event cameras report changes in intensity rather than an absolute measurement. These devices perform per-pixel differencing instead of operating with entire frames — the very concept of a 'frame', while crucial for devices employing a shutter (whether rolling or global), is absent from event cameras. Because this different principle of operation eases some hardware design constraints, current technology allows for consumer-grade devices that, when compared to frame-based cameras, can be much more efficient in transmitting image data and operate at considerably higher temporal resolution with greater dynamic range [9]. Not only are these performance traits attractive for reducing motion blur, but event cameras report information that is important for robots in key applications: their output naturally focuses on changes to the scene, picking up dynamic elements within the perceived environment (e.g., [20] and [42]).

Whether event cameras will form an impetus for new sets of innovative applications, or will drive some radical departure from existing methods, or even initiate a thorough re-examination of the field's underpinnings—all remains to be seen. This paper is not about event cameras *per se*. Instead, it asks the question:

> For any sensor, say, of type $X \in \{\text{compasses, IMUs, LiDARs, } \ldots\}$, is there a useful "event X" version?

The transformation from the raw sensor into an event version, a process which we dub *eventification*,[1] involves disentangling, conceptually, several different facets. We introduce theory by which one can formulate the preceding question in a meaningful way. The present paper is an abstract treatment of the essential properties that make event cameras interesting, expressed with reasonable rigor, and in adequate detail to lay open some connections that were not immediately apparent.

The long-term goal of this theory is to try to change the way our field interacts with the areas of sensor design and with signal processing researchers. Today, most roboticists are consumers: we see what is out there, we buy something from a catalogue, we bolt it to a robot, and then integrate it with software. Robot use-cases (i.e., task performance) should play a greater role in informing what sensors ought to exist, what should be designed, and how manufacturers might target roboticists.

A. Related work

Our work was inspired by the recent paper of Zardini, Spivak, Censi, and Frazzoli [38], wherein the authors provide a compositional architecture with which they express a model of a UAV system. That robot system has, as one specific sensor, an event camera. Their model leads one to ponder whether the 'eventfulness' of the camera might be obtained by some abstract transformation of a traditional camera — if so, what would such a transformation look like? Hence the present paper, which retains some of the spirit of their work. That same spirit is also apparent in the important, early paper of Tabuada, Pappas, and Lima [35] which provides an expressive mathematical framework through which aspects of robotic systems' behavior can be represented and examined.[2] Their work employs equivalence based on bisimulation; the notion of output simulation we employ is similar (but known to be

[1] A term inspired by [25].
[2] A recent ICRA workshop [43] attests to expanding interest in such topics.

distinct, cf. [28]). The concept of stutter bisimulation —where sequences may have repeated subsequences— was introduced and studied in the early model checking literature [1], [5], though we are not aware of applications to robotics. The present paper can be understood as generalizing output simulation so that, among other things, it may also treat a form of stutter.

The question of whether some sensors provide a system with a sufficiency of information has roots in the classic notion of observability [13], [11]. More recently, and more directly in the robotics community, the subject has been related to concepts such as perceptual limits [7], information spaces [14] (originally of von Neumann and Morgenstern), and lattices of sensors [15], [40]. Erdmann's work [8] reverses the question, asking not what information some given sensor provides, but what a (virtual) sensor ought to provide. His action-based sensors become, then, a computational abstraction for understanding the discriminating power needed to choose productive actions. The idea of the discriminating power and a (virtual) sensor wrapping some computation permeates this paper's treatment as well. Whether some transformation undermines the ability to extract sufficient information, especially as a model for non-idealized sensors, appears in [31] — a paper which we shall refer to again, later. An important class of transformations are ones that seek to compress or reduce information. These fit under the umbrella of minimalism, an idea with a long history in robotics [3], [21], but with adherents of a more recent generation having a greater focus on algorithmic [26] and optimization-based tools [27], [39].

Neuromorphic engineering, the field that pioneered event cameras, is concerned with a class of devices much broader than just cameras [18]. In recent years, along with advancements in spiking-neuron and neural computing [4], [22], [23], event-driven tactile sensors [36], synthetic cochlea [19], chemical concentration and gas detection sensors [34], [37] have been developed. We feel the robotics community could be better at informing sensor designers about what devices would be germane for robot use.

B. Paper Organization with a Preview of Contributions

The next section deals with preliminaries and begins by introducing, with some basic notation, definitions that have mainly been established elsewhere. Section III introduces the core notion of substitutable behavior (Definition 5) on which this work is based; it takes a new and general form, subsuming and unifying two previous concepts, while affording much greater expressive power. In Section IV this power is put to use. We give a basic structure, which we call an observation variator (Definition 9), that is capable of reporting differences in the signal space, leading to the formation of a derivative. We pose a form of optimization question, asking how to find a smallest variator, and then establish that minimization is NP-hard (Theorem 18). As we then show, modes of data acquisition affect the sensor's power, so Section V turns to this in depth, moving beyond synchronous data flow. The key result (Theorem 27) is that polling and event-triggered acquisition modes are equivalent to one another. Section VI considers the fully asynchronous data acquisition mode; doing so requires the variator to have additional structure (Definition 28, a monoidal variator). The problem, when expressed directly, appears complicated; we construct a conceptually simpler version, and show that they are actually equivalent (Theorem 35). The penultimate section motivates and examines some simple notions of stable behavior, which ensure the sensor will not chatter. But fortunately chatter-free behavior can be obtained, essentially for free, in problems of interest (Theorem 39). Section VIII offers a brief summary of the paper.

Overall, the work explicates the concept of eventification, and then identifies and explores some further connections. With an eye toward an axiomatic theory of sensing, some care has been exercised to be economical: additional structure is introduced just when actually demanded; for instance, only in Section VI do any algebraic properties make an appearance.

II. Background: Filtering Problems

To be analogous to event cameras, event sensors must couple raw sensor devices (i.e., physical components and electronics for energy transduction) with some computation (e.g., signal differencing). Thus, our treatment will consider them to be units that are abstract *sensori-computational devices* (borrowing this term of Donald [6]). These units implement a kind of abstract sensor (here, a term inspired by Erdmann [8]). We will use procrustean filters, a basic framework for treating (potentially stateful) stream processing units, to model such sensori-computational devices:

Definition 1 ([31])**.** A *sensori-computational device* is a 6-tuple (V, V_0, Y, τ, C, c) in which V is a non-empty finite set of states, V_0 is the non-empty set of initial states, Y is the set of observations, $\tau : V \times V \to \wp(Y)$ is the transition function, C is the set of outputs, and $c : V \to \wp(C) \setminus \{\varnothing\}$ is the output function. (We write $\wp(A)$ to denote the powerset of set A.)

A sensori-computational device translates between streams of discrete symbols. These objects are transition systems for processing streams, with finite memory (represented as the set of states) used to track changes in sequences as they're being processed incrementally. Acting as transducers, they receive a stream of observations as input, revealed one symbol at a time, and generate one output per input symbol. In our setting, the observations will come from a raw sensor or after some simple post-processing; outputs, represented abstractly as colors, encode either actions (for a policy) or state estimates (for a filter).

The sets of states, initial states, and observations for F are denoted $V(F)$, $V_0(F)$ and $Y(F)$, respectively. All the sensori-computational devices throughout this paper (i.e., units modeled, in the terminology of [31], via some filter F) will just be presented as a graph, with states as its vertices and transitions as directed edges bearing sets of observations. For simplicity, for all such devices we shall assume that $Y(F)$ is finite. The values of the output function will be visualized as a set of colors at each vertex, hence the naming of C and $c(\cdot)$.

Given a particular sensori-computational device $F = (V, V_0, Y, \tau, C, c)$, an observation sequence (or a string) $s = y_1 y_2 \ldots y_n \in Y^*$, and states $v, w \in V$, we say that w is *reached by* s (or s *reaches* w) when traced from v, if there exists a sequence of states $w_0, w_1, \ldots, w_n$ in F, such that $w_0 = v$, $w_n = w$, and $\forall i \in \{1, 2, \ldots, n\}, y_i \in \tau(w_{i-1}, w_i)$. (Note that Y^* denotes the Kleene star of Y.) We let the set of all states reached by s from a state v in F be denoted by $\mathscr{V}_F(v, s)$ and denote all states reached by s from any initial state of the filter with $\mathscr{V}_F(s)$, i.e., $\mathscr{V}_F(s) = \bigcup_{v_0 \in V_0} \mathscr{V}_F(v_0, s)$. If $\mathscr{V}_F(v, s) = \varnothing$, then we say that string s *crashes* in F starting from v.

We also denote the set of all strings reaching w from some initial state in F by $\mathscr{S}_w^F = \{s \in Y^* | w \in \mathscr{V}_F(s)\}$. The set of all strings that do not crash in F is called the *interaction language* (or, briefly, just *language*) of F, and is written as $\mathscr{L}(F) = \{s \in Y^* | \mathscr{V}_F(s) \neq \varnothing\}$. We also use $\mathscr{C}(F, s)$ to denote the set of outputs for all states reached in F by s, i.e., $\mathscr{C}(F, s) = \bigcup_{v \in \mathscr{V}_F(s)} c(v)$. When s crashes, the vacuous union gives $\mathscr{C}(F, s) = \varnothing$. Definition 1 ensures that any $\mathscr{L}(F)$ contains at least ε, the empty string; we have $\mathscr{C}(F, \varepsilon) = \bigcup_{v_0 \in V_0} c(v_0)$.

We focus on sensori-computational devices with deterministic behavior:

Definition 2 (deterministic)**.** An sensori-computational device $F = (V, \{v_0\}, Y, \tau, C, c)$ is *deterministic* or *state-determined*, if for every $v_1, v_2, v_3 \in V$ with $v_2 \neq v_3$, $\tau(v_1, v_2) \cap \tau(v_1, v_3) = \varnothing$. Otherwise, we say that it is *non-deterministic*.

Algorithm 2 in [30] can turn any non-deterministic sensori-computational device into one with an identical language but which is deterministic. Hence, without loss of generality, in what follows all sensori-computational devices will be assumed to be deterministic.

Overwhelmingly we shall give simple examples, but one easily gains expressive power by constructing complex sensori-computational devices by composing more elementary ones:

Definition 3 (direct product)**.** Given $F = (V, V_0, Y, \tau, C, c)$ and $F' = (V', V_0', Y', \tau', C', c')$, then their *direct product* is the 6-tuple $F \times F' = (V \times V', V_0 \times V_0', Y \times Y', \tau_{F \times F'}, C \times C', c_{F \times F'})$ with

$$\tau_{F \times F'}: \quad (V \times V') \times (V \times V') \to \wp(Y \times Y'),$$
$$((v_i, v_j'), (v_k, v_m')) \mapsto \tau(v_i, v_k) \times \tau'(v_j', v_m');$$

$$c_{F \times F'}: \quad (V \times V') \to \wp(C \times C') \setminus \{\varnothing\},$$
$$(v_i, v_j') \mapsto c(v_i) \times c'(v_j').$$

(Note: To save notational bloat, we shall only present pairwise products, trusting the reader will be comfortable with the obvious extension to any finite collection.)

Remark 1. If $s_1 s_2 \ldots s_n \in \mathscr{L}(F \times F')$ then each $s_i = (y_i, y_i')$ has $y_i \in Y(F)$ and $y_i' \in Y(F')$, and further $y_1 y_2 \ldots y_n \in \mathscr{L}(F)$, and $y_1' y_2' \ldots y_n' \in \mathscr{L}(F')$. The converse, however, needn't hold: e.g., for some $y_1 y_2 \ldots y_n \in \mathscr{L}(F)$ there may exist no $s_1 s_2 \ldots s_n \in \mathscr{L}(F \times F')$ with $s_i = (y_i, y_i')$.

The standard way to compare sensori-computational devices is in terms of input-output substituability, that is, whether one can serve as an functional replacement for another. The following expresses this idea.

Definition 4 (output simulation [26])**.** Let F and F' be two sensori-computational devices, then F' *output simulates* F if $\mathscr{L}(F) \subseteq \mathscr{L}(F')$ and $\forall s \in \mathscr{L}(F) : \mathscr{C}(F', s) \subseteq \mathscr{C}(F, s)$.

If F' output simulates F, the intuition is that then any stream of observations that F can process can also be processed effectively by F'; the outputs that F' yield will be consistent with those F could produce. In terms of functionality, F' may serve as an alternative for F.

When considering F' and F, often F would be treated as providing a specification (with $\mathscr{L}(F)$ circumscribing aspects of the world that may arise, and $\mathscr{C}(F, \cdot)$ characterizing suitable outputs); an output simulating F' realizes behavior that is acceptable under this specification. This is because such a sensori-computational device F' is able to handle all strings from F and yields some suitable outputs for each string. Note that the output may be the result of some sort of estimation (like a combinatorial filter [14]), or the output may be a representation of an action to be executed, and so encode a policy (e.g., [26]).

III. Generalized output simulation

For this paper, the point of departure is a more general notion of output simulation. We consider a case where one may specify some relation that modifies the strings of one sensori-computational device, so that the second device must process strings through (or in the image of) the relation.

Definition 5 (output simulation modulo a relation)**.** Given two sensori-computational devices F and F', and binary relation $\mathrm{R} \subseteq A \times B$ we say that F' *output simulates* F *modulo* R, denoted by $F' \sim F \pmod{\mathrm{R}}$, if $\forall s \in \mathscr{L}(F)$:

1) $\exists t \in \mathscr{L}(F')$ such that $s \mathrm{R} t$;
2) $\forall t \in B$ such that $s \mathrm{R} t$, $t \in \mathscr{L}(F')$ and $\mathscr{C}(F, s) \supseteq \mathscr{C}(F', t)$. (Notice that, as $t \in \mathscr{L}(F')$, $\mathscr{C}(F', t) \neq \varnothing$.)

Some sensori-computational device F is *output simulatable modulo relation* R if there exists some F' which output simulates F modulo R. More concisely, in such cases we may say that F is R*-simulatable*. When F' output simulates F modulo R, intuitively, the streams of observations F can process can also be effectively processed by F' after they've been pushed through binary relation R. Because R may be 1-to-1, 1-to-many, many-to-1, or many-to-many, this generalization gives the ability to treat several phenomena of interest.

Remark 2. As most relations we will use are binary relations, we'll suppress the 'binary' qualifier in that case. Also, when some relation is a (partial or total) function and it is clearer to

express it as a map, we will write it using standard notation for functions.

Definition 5 is the fundamental notion of behavioral substitutability that underlies our treatment in this paper. It is a non-trivial generalization of two prior concepts. One interesting and, as it turns out, particularly useful degree of flexibility is that the relation R can associate strings of differing lengths.

First, the preceding definition generalizes the earlier one:

Remark 3. When $\mathrm{R} = \mathbf{id}$, the identity relation, then Definition 5 recovers Definition 4 (the standard definition of output simulation, the subject of extensive prior study).

The specific requirement that F' handle at least the inputs that F does, explicit in Definition 4, becomes:

Property 6. For relation $\mathrm{R} \subseteq A \times B$, a necessary condition for any F to be R-simulatable is that $\mathrm{R} \cap (\mathscr{L}(F) \times B)$ be *left-total* in the sense that for every $s \in \mathscr{L}(F)$, there exist some t with $s\,\mathrm{R}\,t$. (Were it otherwise for some $s \in \mathscr{L}(F)$, then that string s suffices to violate condition 1 in Definition 5.)

And second, for the other generalization:

Remark 4. Definition 5 subsumes the ideas of *sensor maps* [31] (also called label maps). These are functions, $h : Y \to X$, taking individual observation symbols to another set. (One may model sensor non-ideality by applying such functions; for instance, observations $y \in Y$ and $y' \in Y$ can be conflated when $h(y) = h(y')$.) Specifically, sensor maps only give relations R restricted so that any $s\,\mathrm{R}\,t$ must have $|s| = |t|$, that is, the strings will have equal length.

Given a sensori-computational device F and general relation R, the question is whether any sensori-computational device F' exists to output simulate F modulo R. For the particular case that R is **id**, F always output simulates itself. This fact means that the prior work focusing on minimizing filters, such as [28], [32], [26], can be reinterpreted as optimizing size subject to output simulation modulo **id**.

Returning to more general relations R, the existence of a suitable F' is the central question in prior work on the destructiveness of label maps [31], [10]. General relations make the picture more complex, however, and these will be our focus as well as complex relations composed from more basic ones.

When R is many-to-1 it models compression or conflation. Output simulation of F modulo such a relation shows that F's behavior is unaffected by reduction of observation fidelity, i.e., it is compression that is functionally lossless. Relations R that are 1-to-many model noise via non-determinism. Output simulation modulo such relations show that operation is preserved under the injected uncertainty. And many-to-many relations treat both aspects simultaneously.

Going forward, we will use the open semi-colon symbol to denote relation composition, i.e., $\mathrm{U} \fatsemi \mathrm{V}$ is $\{(u,v) \mid \exists r \text{ s.t.} (u,r) \in \mathrm{U} \text{ and } (r,v) \in \mathrm{V}\}$. Beware that when both relations are functions (cf. Remark 2), the notation unfortunately reverses the convention for function composition, so $g \fatsemi f = f(g(\cdot)) = f \circ g$. (This will arise in, for example, Problem 2.)

Property 7. Given sensori-computational devices F and G, and left-total relations U and V on $\mathscr{L}(F) \times \mathscr{L}(G)$, with $\mathrm{U} \supseteq \mathrm{V}$, then $G \sim F \pmod{\mathrm{U}} \implies G \sim F \pmod{\mathrm{V}}$.

Hence, sub-relations formed by dropping certain elements do not cause a violation in output simulation if left-totalness is preserved. Before composing chains of relations, we examine further the connection raised in Remark 4.

Remark 5. Unlike sensor maps, the property of output simulating modulo some relation is not monotone under composition. For sensor maps, there is a notion of irreversible destructiveness: composition of a destructive map with any others is permanent, always resulting in a destructive map. That theory can talk meaningfully of a feasibility boundary in the lattice (e.g., title of [10]). For relations, composing additional relations can 'rescue' the situation. For instance, consider the device F_{rgb} in Figure 1. For relation $\mathrm{U} = \{(a,p),(a,q),(b,q),(b,t)\}$ there can be no G that output simulates F modulo U because q must either be green or blue, but can't be both. Formally $\{\text{green}\} = \mathscr{C}(F,a) \supseteq \mathscr{C}(G,q)$ since $a\,\mathrm{U}\,q$, and $\{\text{blue}\} = \mathscr{C}(F,b) \supseteq \mathscr{C}(G,q)$ since $b\,\mathrm{U}\,q$, and $\mathscr{C}(G,q) \neq \varnothing$. But with $\mathrm{V} = \{(p,a),(p,a'),(t,b),(t,b')\}$, which is not left-total (cf. Property 6), crucially, then it is easy to give some G' so that $G' \sim F_{\mathrm{rgb}} \pmod{\mathrm{U} \fatsemi \mathrm{V}}$. One can simply take F_{rgb} and add a' and b' to the edge sets with a and b, respectively.

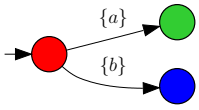


Fig. 1: A small sensori-computational device F_{rgb}, with $Y(F_{\mathrm{rgb}}) = \{a,b\}$, and $C = \{\text{red}, \text{green}, \text{blue}\}$.

Nevertheless, one may form a chain of relations:

Theorem 8. Given two relations R_1 and R_2, and sensori-computational device F, if there exists a sensori-computational device F_1 with $F_1 \sim F \pmod{\mathrm{R}_1}$ and there exists a sensori-computational device F_2 with $F_2 \sim F_1 \pmod{\mathrm{R}_2}$, then $F_2 \sim F \pmod{\mathrm{R}_1 \fatsemi \mathrm{R}_2}$.

Since Sections IV and V will consider particular relations that model properties specifically related to event sensors, this theorem can be useful when one is interested in devices under the composition of those relations.

IV. Structured Observations: Observation Differencing

The most obvious fact about event cameras is that the phenomena they are susceptible to (photons) impinge on some hardware apparatus (the silicon retina) in a way which produces signals (intensity) for which differencing is a meaningful operation. We talk about 'events' as changes in those signals because we can define and identify differences (e.g.,

Fig. 2: An iRobot Create drives down a corridor its wall sensor w generating output values as it proceeds.

in brightness). Thus far, our formalized signal readings are only understood to involve elements drawn from $Y(F)$, just a set. The idea in this section is to contemplate structure in the raw signal space that permits some sort of differencing. Accordingly, the pair definitions that follow next.

Definition 9 (observation variator)**.** An *observation variator*, or just variator, for a set of observations Y is a set D and a ternary relation $\mathrm{S}_D \subseteq Y \times D \times Y$.

Often the two will be paired: (D, S_D). Reducing cumbersomeness, the S_D will be dropped sometimes, but understood to be associated with D and $Y(F)$ for some sensori-computational device F. Anticipating some cases later, when $(y,d,y') \in \mathrm{S}_D$ we may also write it as a function: $y' = \mathrm{S}_D(y,d)$. But beware of the fact that it may be multi-valued, and it may be partial.

On occasion we will call D the set of *differences*, terminology which aids in interpretation but should be thought of abstractly (as nothing ordinal or numerical has been assumed about either the sets $Y(F)$ or D).

Definition 10 (delta relation)**.** For a sensori-computational device F with variator (D, S_D), the associated *delta relation* is $\nabla\big|_F^D \subseteq \mathscr{L}(F) \times (\{\varepsilon\} \cup (Y(F) \cdot D^*))$ defined as follows:

0) $\varepsilon \nabla\big|_F^D \varepsilon$, and
1) $y_0 \nabla\big|_F^D y_0$, for all $y_0 \in Y(F) \cap \mathscr{L}(F)$, and
2) $y_0 y_1 \ldots y_m \nabla\big|_F^D y_0 d_1 d_2 \ldots d_m$, where $(y_{k-1}, d_k, y_k) \in \mathrm{S}_D$.

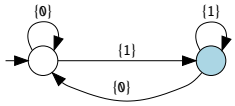
Intuitively, the interpretation is that S_D tells us that d_k represents a shift taking place to get to y_k from symbol y_{k-1}. (With mnemonic 'difference' for d_k.)

Leading immediately to the following question:

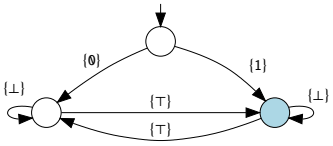
Question 1. For any sensori-computational device F with variator (D, S_D), is it $\nabla\big|_F^D$-simulatable?

Given F with variator (D, S_D), we call a device F' that output simulates F modulo $\nabla\big|_F^D$ a *derivative* of F. In such cases we will say F has a derivative under the observation variator D.

Example 1 (iRobot Create wall sensor)**.** In Figure 2 an iRobot Create moves through an environment. As it does this, the infrared wall sensor on its port side generates a series of readings. These readings (obtained via Sensor Packet ID: #8, with '0 = no wall, 1 = wall seen' [12, pg. 22]) have binary values. The Create's underlying hardware realizes some basic computation on the raw sensor to produce these values by thresholding luminance, either as a voltage comparison via



(a) A sensori-computational device F_{wall}, with $Y(F_{\text{wall}}) = \{0,1\}$; within the vertices, white encodes $\{0\}$, and azure $\{1\}$.



(b) A sensori-computational device F'_{wall} with observation variator is $D_2 = \{\bot, \top\}$, so $F'_{\text{wall}} \sim F_{\text{wall}} \left(\mathrm{mod}\, \nabla\big|_{F_{\text{wall}}}^{D_2}\right)$.

Fig. 3: Two sensori-computational devices describing the scenario in Example 1: (a) A model of the rather trivial transduction of the Create's wall sensor, and (b) its derivative under the observation variator D_2.

analogue circuitry or after digital encoding. To cross the hardware/software interface, the detector's binary signal is passed through a transducer, namely sensori-computational device of the form shown in Figure 3a.

A suitable observation variator is $D_2 = \{\bot, \top\}$, and ternary relation written in the form of a table as $(\text{row}, \text{cell-entry}, \text{column}) \in \mathrm{S}_{D_2} \subseteq \{0,1\} \times D_2 \times \{0,1\}$ as:

S_{D_2}	0	1
0	$\bot$	$\top$
1	$\top$	$\bot$

Using this variator, there is a sensori-computational device that output simulates F_{wall} modulo the delta relation $\nabla\big|_{F_{\text{wall}}}^{D_2}$. Figure 3b shows its derivative F'_{wall}. A direct interpretation for how D_2 encodes the variation in the bump signal is that $\top$ indicates a flip in the signal; while $\bot$ makes no change. Also, the first item in the sequence, some element from $Y(F_{\text{wall}})$, describes the offset from wall at time of initialization. □

The preceding example, though simple, illustrates why we have started from the very outset by considering stateful devices. This may have seemed somewhat peculiar because we are treating sensors and these are seldom conceived of as especially stateful. An event sensor requires *some* memory, and so state is a first-class part of the model. (As already touched upon, some authors have applied the moniker 'virtual' to sensors that involve some computational processing.)

Especially when exploring aspects of the delta relation's definition, most of our examples will involve very simple input–output mappings. It should be clear that they could quickly become rather more complex. For instance if, in Figure 2, the robot must tell apart odd and even doors, then a suitable adaption of the sensor is easy to imagine: the 2 states in Figure 3a become 4, and the outputs involve three colors, *etc*.

Example 2 (Lane sensor)**.** A self-driving car, shown in Figure 4, moves on a highway with three lanes. It is equipped with on-board LiDAR sensors to detect the vehicle's current lane. Supposing these are indexed from its right to left as 0, 1 and 2, then this lane sensor produces one of these three outputs. To construct a sensor that reports a change in the current

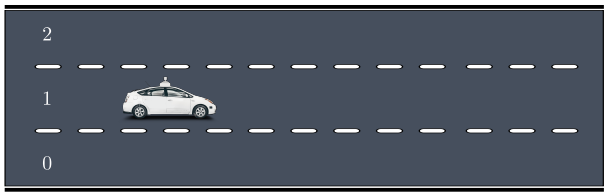


Fig. 4: A self-driving car, with on-board sensors to detect whether the vehicle changes to the left or right, or stays at the current lane.

lane, consider the observation variator is $(D_{3\text{-lane}}, \mathrm{S}_{D_{3\text{-lane}}})$ with $D_{3\text{-lane}} = \{\,\textsc{left}\,,\,\textsc{null}\,,\,\textsc{right}\,\}$ and, $\mathrm{S}_{D_{3\text{-lane}}}(i,d) = \min(\max(i+v(d)),0),2)$, where $v(\,\textsc{left}\,) = +1$, $v(\,\textsc{null}\,) = 0$, and $v(\,\textsc{right}\,) = -1$.

This observation variator will be able to transform any sequence of lane occupations into unique lane-change signals in a 3-lane road. □

Example 3 (Minispot with a compass)**.** Consider a Minispot, the Boston Dynamics quadruped robot in Figure 5. Assume that it is equipped with motion primitives that, when activated, execute a gait cycle allowing it to move forward a step, move backward a step, or turn in place $\pm 45°$, without losing its footing. Starting facing North, after each motion primitive terminates, the Minispot's heading will be one of 8 directions (the 4 cardinal plus 4 intercardinal ones).

Suppose the raw compass produces measurements $x \in \{\uparrow, \nearrow, \rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \nwarrow\}$, then consider the observation variator (D_3, S_{D_3}) with $D_3 = \{-, \emptyset, +\}$, where $(x, \emptyset, x) \in \mathrm{S}_{D_3}$, and $(x, +, y) \in \mathrm{S}_{D_3}$ if the angle from x to y is $45°$, and $(x, -, y) \in \mathrm{S}_{D_3}$ if the angle from x to y is $-45°$.

Given the four motion primitives, any sequence of those actions produces a sequence of compass measurements that the output variator D_3 can model. The constraint implied by such sequences means that a model of the raw compass will have a derivative under variator D_3. □

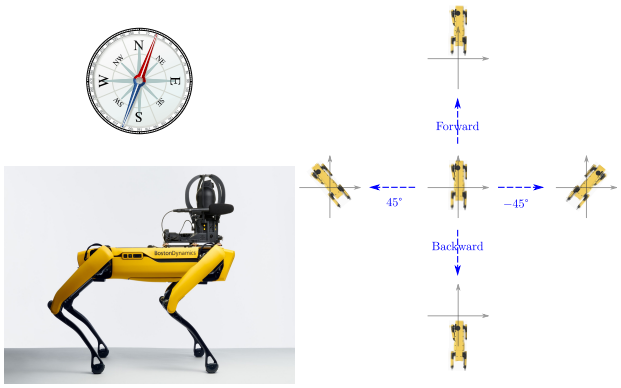


Fig. 5: A Boston Dynamics Minispot quadraped is equipped with a compass to give its heading. To simplify the control challenge, the Minispot is programmed with motion primitives to step forward and backward, and to turn in place by $\pm 45°$ (illustrated on the right).

The previous two examples show that constraints can be imposed on the signal space—in the first case owing to the range of lanes possible (i.e., a saturation that arises); or via structure inherited from the control system (i.e., only some transitions are achievable). For both situations, a simple 3 element set is sufficient only because the sequences of changes the sensor might encounter has been limited.

Example 4 (Minispot with a compass, revisited)**.** Suppose the Minispot of Example 3 has been enhanced by supplementing its motion library with a primitive that allows it to turn in place by $\pm 90°$. Now, after each motion primitive terminates, the compass signal can include changes for which (D_3, S_{D_3}) is inadequate. When Definition 10 is followed to define $\nabla\big|_F^{D_3}$, those sequences involving $90°$ changes fail to find any $d_k \in D_3$, and the relation is not left-total. Hence, via Property 6, there can be no derivative.

Naturally, a more sophisticated observation variator does allow a derivative. Supposing we encode $\{\uparrow, \nearrow, \ldots \nwarrow\}$ instead with headings as integers $\{0, 45, \ldots, 315\}$, then we might consider the variator $(\mathbb{Z}, +)$. By the plus we refer to the function $+ : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$, the usual addition on integers. To meet the requirements of Definition 9 strictly, we ought to take the restriction to the subset of triples (in the relation) where the first and third slots only have elements within $\{0, 45, \ldots, 315\}$. Then there are devices that will output simulate modulo $\nabla\big|_F^{\mathbb{Z}}$. □

The preceding illustrates how, for a sufficient observation variator, (D, S_D), some aspect of the signal's variability is expressed in the cardinality of D. Thus, seeking a small (or the smallest possible) D will be instructive; employing the whole kitchen sink, as was done with the integers above, fails to pinpoint the necessary information. Also, having stated that some variators are sufficient and touched upon Property 6, the following remark is in order.

Remark 6. Despite Property 6, one does not require that S_D be left-total for Question 1 to have an affirmative answer. For instance, some pairs of y and y' may never appear sequentially in strings in $\mathscr{L}(F)$, and so S_D needn't have any triples with y and y' together. (Though, obviously, the left-totalness of $\nabla\big|_F^D$ is required.)

Proposition 11. A sufficient condition for an affirmative answer to Question 1 is that $y' = \mathrm{S}_D(y,d)$ be a single-valued partial function whose $\nabla\big|_F^D$ is left-total. More explicitly, the requirement on $\mathrm{S}_D(y,d)$ is

$$(y,d,y') \in \mathrm{S}_D \text{ and } (y,d,y'') \in \mathrm{S}_D \implies y' = y''.$$

Placing stronger constraints solely on the variator, we obtain another sufficient condition:

Proposition 12. For (D, S_D), if for every $y, y' \in Y(F)$, there exists a unique $d \in D$ so $y' = \mathrm{S}_D(y,d)$ then Question 1's answer is affirmative.

The two previous propositions, while straightforward 'closed-form' sufficient conditions, make demands which sel-

dom hold for realistic sensors. For instance, S_D may be multi-valued in order to model noise. The complete answer for any S_D, deferred until Section IV-B, does appear in Lemma 17.

But first, a camera as an example is, of course, long overdue.

Example 5 (single-pixel camera)**.** A single-channel, 8-bit, single-pixel camera is a device that returns reading in the range $Y = \{0,\dots,255\}$ at any point in time. We might model such a camera via a sensori-computational device F_{cam} that does no state-based computation: have it use 256 vertices $V(F_{\text{cam}}) = \{v_0,\dots,v_{255}\}$, and outputs $C_{\text{cam}} = \{o_0,\dots,c_{255}\}$, so that $c(v_i) = \{o_i\}$, and transitions which consider only the last value $\tau_{\text{cam}}(v_i, v_j) = \{j\}$.

For the observation variator, we again can use standard integers $(\mathbb{Z},+)$. Now we restrict to the subset of triples where the first and third slots only have elements within $\{0,\dots,255\}$; the second slot then clearly only has elements $\{-255,\dots,255\}$.

The device F_{cam} with variator $(\mathbb{Z},+)$ (or the restriction described) is $\nabla\big|_F^D$-simulatable because a derivative F'_{cam} can be constructed for it. □

The integers used to model observation changes for the pixel intensities differ from those with the compass bearings: that is, the elements that remain after $+\colon \mathbb{Z}\times\mathbb{Z}\to\mathbb{Z}$ is restricted in Example 5 and Example 4 have different structure. We shall revisit this.

Rather more obviously, a single-pixel camera has only limited applicability. Next, we consider how to scale up.

A. Modeling complex sensors

We start with a definition that allows aggregation of output variators.

Definition 13 (direct product variator)**.** Given (D_1, S_{D_1}) as an output variator for F_1, and (D_2, S_{D_2}) for F_2, then the set $D_{1\times2} = D_1 \times D_2$ and $\mathrm{S}_{D_{1\times2}}$ defined via

$$\big((y_1,y_2),(d_1,d_2),(y'_1,y'_2)\big) \in \mathrm{S}_{D_{1\times2}} \Longleftrightarrow (y_1,d_1,y'_1)\in\mathrm{S}_{D_1} \wedge (y_2,d_2,y'_2)\in\mathrm{S}_{D_2} \quad (1)$$

is an observation variator for $F_1\times F_2$, and is termed the *direct product variator*.

Example 6 (iRobot Create bump sensors)**.** Recalling the iRobot Create of Example 1, these robots have left and right bump sensors, both of which provide binary values (obtained via Sensor Packet ID: #7 (bits 0 and 1, right and left, respectively, encoding '`0` = no bump, `1` = bump' [12, pg. 22]). As these are binary streams, just like the wall sensor, they can each be transformed with variator $(D_2 = \{\bot,\top\}, \mathrm{S}_{D_2})$ that tracks bit flips. And to track both, one constructs $\{\bot,\top\}\times\{\bot,\top\}$, and the ternary relation $\mathrm{S}_{D_{2\times2}}$. In this way, a $d_i = (\bot,\bot)$ would indicate that neither bump sensor's state has changed since previously. □

Example 7 (1080p camera)**.** A conventional 1080p camera has 3 channels at a resolution of 1920 × 1080. Using the F_{cam} of Example 5, apply the direct product (of Definition 3) to form an aggregate device, and the direct product (of Definition 13) to the variator $(\mathbb{Z},+)$. The triple relation is large (with 4.0×10^{11} elements). □

In order for this camera, assembled from the single-pixel ones, to not be unwieldy we must show how output simulation also aggregates. We do this in Proposition 15, but need the following definition first.

Definition 14 (length compatible relations)**.** A pair of relations, R_1 and R_2, each on sets of sequences $\mathrm{R}_1 \subseteq A\times B$ (with $A\subseteq{\Sigma_a}^*$, $B\subseteq{\Sigma_b}^*$), $\mathrm{R}_2\subseteq C\times E$ (with $C\subseteq{\Sigma_c}^*$, $E\subseteq{\Sigma_e}^*$) are *length compatible* if for every $a\in A$ and $c\in C$ with the same length (i.e., $|a|=|c|$), there exists some b and e of identical length ($|b|=|e|$) with $a\,\mathrm{R}_1\,b$ and $c\,\mathrm{R}_2\,e$.

In other words, when we consider the image of any sequence under R_1, along with the image of any sequence of identical length under R_2, both contain *some* pair of common-lengthed sequences.

We now express the property of interest.

Proposition 15 (product output simulation)**.** Given sensori-computational devices F_1 and F_2 which are output simulatable modulo length compatible relations $\mathrm{R}_1\subseteq A\times B$ and $\mathrm{R}_2\subseteq C\times E$, respectively, then device $F_1\times F_2$ is output simulatable modulo relation $\mathrm{R}_{1\times2}$ defined via:

$$\begin{array}{c}\big((a_1,c_1)\dots(a_n,c_n)\big)\,\mathrm{R}_{1\times2}\,\big((b_1,e_1)\dots(b_m,e_m)\big)\\ \Updownarrow\\ (a_1\dots a_n)\,\mathrm{R}_1\,(b_1,\dots,b_m)\ \wedge\ (c_1\dots c_n)\,\mathrm{R}_2\,(e_1,\dots,e_m)\,.\end{array} \quad (2)$$

Corollary 16. Given sensori-computational devices F_1 and F_2, with variators (D_1, S_{D_1}) and (D_2, S_{D_2}), a sufficient condition for direct product $F_1\times F_2$, with direct product variator $(D_{1\times2}, \mathrm{S}_{D_{1\times2}})$, to possess a derivative is that F_1 and F_2 do.

The proof of Corollary 16 and the foundation it builds upon, Proposition 15, construct the output simulating device via a direct product.

To summarize, we may compose sensori-computational devices to give a more complex aggregate device, and also compose output variators to give a composite variator. Question 1 for the aggregate device, can be answered in the affirmative by consider the same question for the individual devices.

B. Answering Question 1

In most of the examples we have considered, the argument for the existence of an output simulating sensori-computational device has been on the basis of the fact that the variator allows complete recovery of the original stream. Indeed, being based on the same reasoning, conditions like those in Propositions 11 and 12 are rather blunt. Remark 6 has already mentioned that some specific two-observation subsequences might never appear in any string in $\mathscr{L}(F)$, thus a set D smaller than one might naïvely expect may suffice. Even beyond this, and perhaps more crucially in practice, full reconstruction of the $y_0y_1\dots y_m$ from $y_0d_1d_2\dots d_m$ may be unnecessary as whole

subsets of $\mathscr{L}(F)$ may produce the same or a compatible output, some element of $\mathscr{C}(F, y_0 y_1 \dots y_m)$.

Hence, to answer an instance of Question 1 when some input streams are allowed to be collapsed by $\nabla\big|_F^D$, one must examine the behavior of the specific sensori-computational device at hand. But to provide an answer for a given (D, S_D), a computational procedure cannot enumerate the possibly infinite domain of $\nabla\big|_F^D$. We give an algorithm for answering the question; the procedure is constructive in that it produces a derivative of F if and only if one exists.

Question 1 (Constructive version). Given F with variator (D, S_D), find an F' such that $F' \sim F \ \left(\mathrm{mod}\, \nabla\big|_F^D\right)$ if one exists, or indicate otherwise.

In this case, we shall additionally assume that the set D is finite.

The procedure is given in Algorithm 1. It operates as follows: it processes input F to form an F'' (in lines 1–16) by copying and splitting vertices so each incoming edge has a single label. This F'' is processed to compute the differences between two consecutive labels (lines 18–30) and the results are placed on edges of F'. If it fails to convert any pair of consecutive labels, then it fails to compute the change of some string in F and reports 'No Solution'. In the above step, after computing the changes, a single string can arrive at multiple vertices in F'. We further check whether those strings, which will be distinct strings in $\mathscr{L}(F)$ but share the same image under $\nabla\big|_F^D$, have some common output or not. If that check (in lines 31–40) finds no common output, then it fails to satisfy condition 2 of Definition 5 and, hence, we report 'No Solution'. Otherwise the procedure merges those reached states to determinize the graph. As a consequence, Algorithm 1 will return a deterministic sensori-computational device that output simulates the input modulo the given delta relation. Its correctness appears as the next lemma.

Lemma 17. Algorithm 1 gives a sensori-computational device that output simulates F modulo $\nabla\big|_F^D$ if and only if there exists such a solution for Question 1.

Proposition 11 (Revisited). The proof of Proposition 11 stopped short of giving an actual sensori-computational device. One may simply employ Algorithm 1 to give an explicit construction.

C. Hardness of minimization

Earlier discussion has already anticipated the fact that an interesting question is: What is the minimal cardinality set D that is possible for a given F?

Decision Problem: Observation Variator Minimization (OVM)
Input: A sensori-computational device F, and $n \in \mathbb{N}$.
Output: TRUE if there exists a variator (D, S_D) and a sensori-computational device F', such that F' output simulates F modulo $\nabla\big|_F^D$ and $|D| \leq n$. FALSE otherwise.

Theorem 18. OVM is NP-hard.

Algorithm 1: Delta Transform: $\mathrm{DELTA}_D(F) \rightarrowtail F'$

Input : A device F, output variator (D, S_D)
Output: A determinized device F' that output simulates F modulo $\nabla\big|_F^D$; 'No Solution' otherwise

1 Create a sensori-computational device F'' as a copy of F with each state v'' in F'' corresponding to state v in F
2 **for** $v'' \in V(F'')$ **do** // Make vertex copies
3 $L \coloneqq Set(v''.IncomingLabels())$
4 **if** $v'' = v_0''$ **then** $L \coloneqq L \cup \{\varepsilon\}$;
5 **for** $\ell \in L$ **do**
6 Create a new state v_ℓ'' and set $c(v_\ell'') = c(v'')$
7 **for** $\ell \in L$ **do**
8 **for** *every outgoing edge* $v'' \xrightarrow{y} w''$, $w'' \neq v''$ **do**
9 Add an edge $v_\ell'' \xrightarrow{y} w''$ in F''
10 **for** *every* ℓ*-labeled incoming edge* $w'' \xrightarrow{\ell} v''$ **do**
11 **if** $w'' \neq v''$ **then**
12 Add an edge $w'' \xrightarrow{\ell} v_\ell''$ in F''
13 **else** // Self loop
14 Add edges $v_k'' \xrightarrow{\ell} v_\ell''$ in F'', for $k \in L$
15 Remove v'' from F''
16 Rename v_ε'' to v_0''
17 Create F' as a copy of F'' and $q \coloneqq Queue([v_0'])$
18 **while** $q \neq \varnothing$ **do** // Apply variator
19 $v' \coloneqq q.\mathrm{pop}()$
20 Find the corresponding state v'' in F'' and $\{\ell\} \coloneqq v''.IncomingLabels()$ // A singleton
21 **for** $z \in v'.OutgoingLabels()$ **do**
22 Initialize set $U_z \coloneqq \varnothing$
23 **if** $v' = v_0'$ **then**
24 $U_z \coloneqq \{z\}$
25 **else**
26 $U_z \coloneqq \{d : D \mid (\ell, d, z) \in \mathrm{S}_D\}$
27 **if** $U_z = \varnothing$ **then**
28 **return** 'No Solution'
29 Replace z in F' with label set U_z
30 Mark v' as visited, and add children of v' to q
31 Reinitialize $q \coloneqq Queue([v_0'])$
32 **while** $q \neq \varnothing$ **do** // State determinization
33 $v' \coloneqq q.pop()$
34 **for** $d \in v'.OutgoingLabels()$ **do**
35 $W' \coloneqq \{w' : V(F') \mid v' \xrightarrow{d} w'\}$, $X \coloneqq \cap_{w' \in W'} c(w')$
36 **if** $X = \varnothing$ **then**
37 **return** 'No Solution'
38 **if** $|W'| > 1$ **then** Merge all states in W' as m', set the $c(m') \coloneqq X$, and add m' to q ;
39 **else**
40 Add states in W' to q if not visited
41 **return** F'

D. Aside: Trimming initial prefixes and absolute symbols

Before moving on, a brief digression allays a potential niggling concern and serves as our first use of relation composition. Some readers might find it disconcerting that $\nabla\big|_F^D$'s definition includes the initial 'absolute' symbol y_0 for each sequence — this might be seen as a model for hybrid devices rather than sensors that report pure changes. One possible resolution is to introduce the new relation:

Definition 19 (Shave relation)**.** For a sensori-computational device F, the associated *shave relation* for $k \in \mathbb{N}$ is $\mathrm{S}_k \subseteq$

$\mathscr{L}(F) \times Y(F)^*$ defined as follows:

0) $\varepsilon \, \mathrm{S}_k \, \varepsilon$, and

1) $y_0 y_1 \ldots y_m \, \mathrm{S}_k \, y_k y_{k+1} \ldots y_m$.

Intuitively, S_k trims away prefixes of length k. Then we can compose relations to give $\nabla\big|_F^D \fatsemi \mathrm{S}_1$, which removes the initial 'absolute' symbol y_0. Hence by analogy to Question 1, one might ask whether sensori-computational device F with variator (D, S_D) is $(\nabla\big|_F^D \fatsemi \mathrm{S}_1)$-simulatable? This can be answered most directly by modifying Algorithm 1, inserting a step between lines 30 and 31 which replaces the labels of edges departing v_0' with ε labels and performs an ε-closure thereafter. Then, state determinization (lines 31–40) will succeed if and only if the answer to the question is affirmative.

V. Data acquisition semantics: Observation sub-sequences and super-sequences

It is worth closely scrutinizing the phrase 'event-based', a technical term used in multiple different ways. When people speak of event-based sensors or event sensors, they typically refer to a device capable of reporting changes. But in computing more generally, and in the sub-field of 'event processing' specifically, the phrase is used when a system is driven by elementary stimulus from without. Adopting that standpoint as an organizational principle in structuring software, one obtains event-based (or event-driven) software architectures. Such architectures will oftentimes impose synchronous operation. This contrasts with systems that must 'poll' to obtain information, and polling systems will usually do some sort of comparison or differencing between successive checks to detect a change. To de-conflate these slightly intertwined ideas, we must distinguish data acquisition from difference calculation. (Concretely: Definitions 9 & 10 dealt with the latter; Definitions 20 & 22 will deal with the former.)

The previous section determines if the set D of differences retains adequate information to preserve input–output behavior. All processing considered thus far preserves sequence lengths precisely, which may encode structured information. But such tight synchronous operation may be infeasible or an inconvenient mode of data acquisition for certain devices, and is inconsistent with, for example, event cameras. To emphasize this fact, recall Example 4. It showed, albeit only indirectly, how the lockstep flow of information from the world to the device affects whether D is adequate. In that example, the robot was allowed to turn 90° in a single step, whereas previously (in Example 3) it could do at most 45°. In practice, a 90° change in raw compass readings could happen for the robot in Example 3 if there was skew, or timing differences, or other non-idealities so that two actions occurred between sensor updates. One must be able to treat such occurrences, partly because they arise in practice, and partly because the implicit structure arising from synchronization is an artefact of the discrete-time model and is not something one wishes the event sensors to exploit. (The information baked in to time, especially as it is given privileged status, is often quite subtle, cf. [16].)

Thus, we next consider two other, practical ways in which sensors might generate the symbols that they send downstream. The first is that they may be *change-triggered* in that the sensori-computational device produces an output symbol only when a change has occurred. The second is for the element consuming the sensori-computational device's output to *poll* at some high frequency. Definitions that suffice to express each of these two modes, Definitions 20 and 22, are developed next.

(Note on notation: In what appears next, we consider sequences which may be from an observation set Y, or from a D of differences, or a combination, *etc.*; we use Σ as an arbitrary set to help emphasize this fact.)

Definition 20 (shrink)**.** Given $L \subseteq \Sigma^*$ and $\mathcal{N} \subseteq \Sigma$, then the $\mathcal{N}$-*shrink* is the single-valued, total function $\pi_{\mathcal{N}}$ defined recursively as follows:

$$\begin{aligned} \pi_{\mathcal{N}} : L &\to (\Sigma \setminus \mathcal{N})^*, \\ \varepsilon &\mapsto \varepsilon, \\ s_1 \ldots s_m &\mapsto s_1 \ldots s_m \text{ if } \forall j \in \{1, \ldots, m\}, s_j \notin \mathcal{N}, \\ s_1 \ldots s_i \ldots s_m &\mapsto \pi_{\mathcal{N}}(s_1 \ldots s_{i-1} s_{i+1} \ldots s_m) \text{ when } s_i \in \mathcal{N}. \end{aligned}$$

The intuition is that the $\mathcal{N}$-shrink drops all elements in $\mathcal{N}$ from sequences. The mnemonic for $\mathcal{N}$ is 'neutral' and the idea is that we will apply the $\mathcal{N}$-shrink relation in order to model change-triggered sensors; we can do this by choosing, for the set $\mathcal{N}$, symbols that reflect no change in signal. This assumes some subset of D will represent this no-change condition. Shortly, Section VI will address choices for D that guarantee such a subset is present.

Question 2. For sensori-computational device F and a set $\mathcal{N} \subseteq Y(F)$, is F $\pi_{\mathcal{N}}$-simulatable?

Question 2 (Constructive). For device F and $\mathcal{N} \subseteq Y(F)$, give some sensori-computational device G such that $G \sim F \pmod{\pi_{\mathcal{N}}}$ if any exists, or indicate otherwise.

Example 8 (Wall sensor, revisited)**.** In Example 1 the iRobot Create's traditional wall sensor produces a stream of `0`'s and `1`'s depending on the intensity of the infrared reflection it obtains. We discussed how, under output variator $D_2 = \{\bot, \top\}$, a derivative sensori-computational device exists that produces a stream of $\bot$s and $\top$s, the former occurring when there is no change in the presence/absence of a wall, and latter when there is. When one examines this derivative device under the $\{\bot\}$-shrink relation, we are considering whether the desired output can be obtained merely on a sequence of $\top$s. If the output depends on a count of the number of $\top$s, like the even- and odd-numbered doorways, then this is possible. If it depends on a count of the number of $\bot$s, or the interleaving of $\top$s and $\bot$s then it can not.

If some derivative, F' say, is $\pi_{\{\bot\}}$-simulatable, then it can operate effectively even if it is notified only when the wall-presence condition changes. It is in this sense that such F's are change-triggered. □

A constructive procedure for addressing Question 2 appears

Algorithm 2: Shrink Transform: $\textsc{Shrink}_{\mathcal{N}}(F) \rightarrowtail G$

Input : A sensori-computational device F, a set $\mathcal{N}$
Output: A deterministic sensori-computational device G if G output simulates F modulo $\pi_{\mathcal{N}}$; otherwise, return 'No Solution'

1 Make G, a copy of F
2 **for** $e' \in G.edges()$ **do** // Label replacement
3 **for** $\ell \in e'.labels()$ **do**
4 **if** $\ell \in \mathcal{N}$ **then**
5 Replace ℓ with ε on edge e'
6 Merge ε-closure$(V_0(G))$ as a single state v'_0 in G
7 $q \coloneqq Queue([v'_0])$
8 **while** $q \neq \varnothing$ **do** // State determinization
9 $v' \coloneqq q.pop()$
10 **for** $\ell \in v'.OutgoingLabels()$ **do**
11 $W \coloneqq \{w : V(G) \mid v' \xrightarrow{\ell} w\}$
12 $W' \coloneqq \cup_{w \in W} \varepsilon$-closure$(w)$
13 $X \coloneqq \cap_{w' \in W'} c(w')$
14 **if** $X = \varnothing$ **then**
15 **return** 'No Solution'
16 **if** $|W'| > 1$ **then**
17 Create a new state w'' inheriting all outgoing edges of W' in G, add a new edge $v' \xrightarrow{\ell} w''$, remove ℓ from v' to W
18 $c(w'') \coloneqq X$, add w'' to q
19 **else if** W' *is not visited* **then**
20 Add the single $w' \in W'$ to q
21 **return** G

in Algorithm 2. It operates as follows: first, it changes all the transitions bearing labels in $\mathcal{N}$ to ε-transitions between lines 1–5. It then shrinks those ε-transitions by determinizing the structure between lines 6–20. By doing so, it captures all the sequences possible after the shrink relation. The following lemma shows correctness:

Lemma 21. Algorithm 2 gives a sensori-computational device that output simulates F modulo $\pi_{\mathcal{N}}$ if and only if there exists a solution for Question 2.

The essence of $\mathcal{N}$-shrink is that, via relation $\pi_{\mathcal{N}}$, it associates to a string all those strings we obtain by winnowing away symbols within $\mathcal{N}$. A second, related definition expands the set of strings by adding elements of $\mathcal{N}$. (Thinking, again, of changes within $\mathcal{N}$ as 'neutral'.)

Definition 22 (pump)**.** Given $L \subseteq \Sigma^*$ and $\mathcal{N} \subseteq \Sigma$, then the $\mathcal{N}$-*pump* is the relation $\mathrm{P}_{\mathcal{N}} \subseteq L \times \Sigma^*$ defined as follows: $\forall (s_1 \dots s_m) \in L$,

0) $\varepsilon \, \mathrm{P}_{\mathcal{N}} \, \varepsilon$,
1) $(s_1 \dots s_m) \, \mathrm{P}_{\mathcal{N}} \, (s_1 \dots s_m)$,
2) $\forall b \in \mathcal{N}, k \in \{1, \dots, \ell\}$, $(s_1 \dots s_m) \mathrm{P}_{\mathcal{N}} (t_1 \dots t_\ell) \implies (s_1 \dots s_m) \mathrm{P}_{\mathcal{N}} (t_1 \dots t_k \, b \, t_{k+1} \dots t_\ell)$.

The intuition is that, to any string, the $\mathcal{N}$-pump associates all those strings with extra elements from $\mathcal{N}$ sandwiched between any two symbols, or at the very end. Notice that it does not place elements of $\mathcal{N}$ at the very beginning of the string.

Question 3. For sensori-computational device F and a set $\mathcal{N} \subseteq Y(F)$, is F $\mathrm{P}_{\mathcal{N}}$-simulatable?

Question 3 (Constructive). For device F and $\mathcal{N} \subseteq Y(F)$, give a device G such that $G \sim F \pmod{\mathrm{P}_{\mathcal{N}}}$ if any exists, or indicate otherwise.

Example 9 (The 45° Minispot, again)**.** Reconsider Example 3, with a derivative compass device for the observation variator $D_3 = \{-, \emptyset, +\}$. Whenever some downstream consumer of the change-in-bearing information queries, an element of D_3 is produced. If it polls fast enough, we expect that it would contain a large number of Ø values. Doubling the rate would (roughly) double the quantity of Ø values. At high frequencies, there would be long sequences of Øs and those computations on the input stream that are invariant to the rate of sampling would be $\mathrm{P}_{\{\emptyset\}}$-simulatable. □

Algorithm 3: Pump Transform: $\textsc{Pump}_{\mathcal{N}}(F) \rightarrowtail G$

Input : A sensori-computational device F, a set $\mathcal{N}$
Output: A deterministic sensori-computational device G if G output simulates F modulo $\mathrm{P}_{\mathcal{N}}$; otherwise, return 'No Solution'

1 Make G, a copy of F
2 Add a vertex v_{new} and set $c(v_{\text{new}}) = c(v_0)$. Add edges from v_{new} pointing to the destination of edges that depart $v_0 \in V_0(G)$
3 Update G's initial vertex: $V_0(G) \coloneqq \{v_{\text{new}}\}$.
4 **for** $v \in V(G) \setminus \{v_{new}\}$ **do** // Add self loops
5 Add a self loop at v bearing labels $\mathcal{N}$
6 $q \coloneqq Queue([v_{\text{new}}])$
7 **while** $q \neq \varnothing$ **do** // State determinization
8 $v \coloneqq q.pop()$
9 **for** $\ell \in v.OutgoingLabels()$ **do**
10 $W \coloneqq \{w : V(G) \mid v \xrightarrow{\ell} w\}$, $X \coloneqq \cap_{w \in W} c(w)$
11 **if** $X = \varnothing$ **then**
12 **return** 'No Solution'
13 **if** $|W| > 1$ **then**
14 Create a new state w' inheriting all outgoing edges of W in G, add a new edge $v \xrightarrow{\ell} w$, remove ℓ from v to W
15 $c(w') \coloneqq X$, add w' to q
16 **else if** W *is not visited* **then**
17 Add the single $w \in W$ to q
18 **return** G

A procedure for answering Question 3 constructively appears in Algorithm 3. Its operation is as follows: first, it creates an initial state, reached (uniquely) by the ε string (line 3). Then, in lines 4–5, it adds self loops bearing labels to be pumped at all states (except the newly created one). Finally, it checks whether the resulting structure output simulates the input, and determizes the structure between line 6–17. By doing so, it creates a deterministic structure to pump the elements in $\mathcal{N}$ using self loops. The following lemma addresses correctness.

Lemma 23. Algorithm 3 gives a sensori-computational device that output simulates F modulo $\mathrm{P}_{\mathcal{N}}$ if and only if there exists a solution for Question 3.

A. Relationships between shrinking and pumping

As understanding the connection between these two relations (shrink and pump) supplies some insight, we start with some basic facts.

Property 24. Immediately these relationships follow:

1) $\pi_{\mathcal{N}} = \mathrm{P}_{\mathcal{N}} \fatsemi \pi_{\mathcal{N}}$. The preceding statement generalizes to $\pi_{\mathcal{N}} = \mathrm{X}_1 \fatsemi \cdots \fatsemi \mathrm{X}_n \fatsemi \pi_{\mathcal{N}}$, where by $\mathrm{X}_1 \fatsemi \cdots \fatsemi \mathrm{X}_n$ we denote any sequence of concatenations under '$\fatsemi$' of relations **id**, $\mathrm{P}_{\mathcal{N}}$, and $\pi_{\mathcal{N}}$.
2) If $\mathcal{N} \neq \varnothing$, then $\mathrm{P}_{\mathcal{N}} \subseteq \pi_{\mathcal{N}} \fatsemi \mathrm{P}_{\mathcal{N}}$. More precisely: if some string in $s_1 s_2 \cdots s_n$ contains an $s_i \in \mathcal{N}$, then $\mathrm{P}_{\mathcal{N}}$ is a strict sub-relation; otherwise the two relations are equal.
3) If $\mathcal{N} \neq \varnothing$, then $\pi_{\mathcal{N}} \subsetneq \pi_{\mathcal{N}} \fatsemi \mathrm{P}_{\mathcal{N}}$.

To give some brief interpretation: Property 24.1) leads one to conclude, with $\mathcal{N} \subseteq Y(F)$, that determining if device F is $(\mathrm{P}_{\mathcal{N}} \fatsemi \pi_{\mathcal{N}})$-simulatable, then, is identical to Question 2. This is very intuitive, as the $\mathcal{N}$-shrink has the 'last word' so to speak, and hence will drop all symbols from $\mathcal{N}$ — it does not care whether those symbols were in the input or generated via the $\mathcal{N}$-pump operation.

The generalization mentioned in Property 24.1) implies other specific facts, like that $\pi_{\mathcal{N}}$ is idempotent. Properties 24.2) and 24.3) suggest that $\mathrm{P}_{\mathcal{N}}$ behaves differently from $\pi_{\mathcal{N}}$. In fact, they share many common properties (e.g., $\mathrm{P}_{\mathcal{N}}$ is idempotent too). Actually, as will be established shortly, the question of output simulation modulo each of these relations is equivalent under conditions we shall be directly concerned with.

Lemma 25. With $\mathcal{N} \subseteq Y(F)$ for an F being $\pi_{\mathcal{N}}$-simulatable implies that F is $\mathrm{P}_{\mathcal{N}}$-simulatable.

Lemma 26. For F, with $\mathcal{N} \subseteq Y(F)$, and all $s_1 s_2 \ldots s_k \in \mathscr{L}(F)$ having $s_1 \in Y(F) \setminus \mathcal{N}$, then device F being $\mathrm{P}_{\mathcal{N}}$-simulatable implies that F is $\pi_{\mathcal{N}}$-simulatable.

Theorem 27 (equivalence of pumping and shrinking)**.** Given any sensori-computational device F and $\mathcal{N} \subseteq Y(F)$, such that all $s_1 s_2 s_3 \ldots s_k \in \mathscr{L}(F)$ have $s_1 \in Y(F) \setminus \mathcal{N}$, then

$$F \text{ is } \mathrm{P}_{\mathcal{N}}\text{-simulatable} \iff F \text{ is } \pi_{\mathcal{N}}\text{-simulatable}$$

The intuitive interpretation is clear. If elements of $\mathcal{N}$ can be pumped, you cannot conduct any computation that depends on their number. This is true even when F has strings in its language that include some elements of $\mathcal{N}$ because, when such elements are encountered, they could either be pumped additions or originally in the string—two cases which cannot be distinguished. The following remark does emphasize that some care is needed, however.

Remark 7. Both Lemmas 25 and 26 construct a new device. That this should be necessary for Lemma 25 is scarcely surprising: if $H \sim F \pmod{\pi_{\mathcal{N}}}$, an attempt at redeploying device H directly under pumping could fail immediately since $\mathscr{L}(H)$ need contain no strings with any element of $\mathcal{N}$, yet for $\mathrm{P}_{\mathcal{N}}$, the device must consume many strings full of $\mathcal{N}$ elements. In Lemma 26, the case for construction of a new sensori-computational device is more subtle. We show this as an example.

Example 10. For F with $\mathcal{N} \subseteq Y(F)$, beware that

$$G \sim F \pmod{\mathrm{P}_{\mathcal{N}}} \not\Longrightarrow G \sim F \pmod{\pi_{\mathcal{N}}}.$$

Figure 6 is an example of a simple sensori-computational device, we shall refer to it as F_{tiny}. Figure 7 gives two more devices, G_1 and G_2. All three have $Y(F_{\text{tiny}}) = Y(G_1) = Y(G_2) = \{a, b, n\}$. Both G_1 and G_2 output simulate F_{tiny} modulo $\mathrm{P}_{\{n\}}$, but G_2 fails to output simulate F_{tiny} modulo $\pi_{\{n\}}$. The string $anb \in \mathscr{L}(F_{\text{tiny}})$, but $\pi_{\{n\}}(anb) = ab \notin \mathscr{L}(G_2)$. □

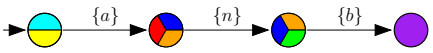
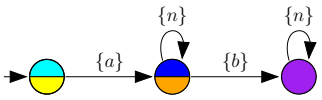
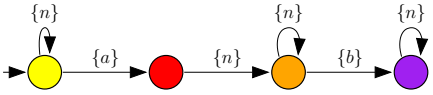


Fig. 6: An example of a simple sensori-computational device F_{tiny}, with $Y(F_{\text{tiny}}) = \{a, b, n\}$.



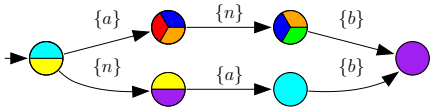
(a) A G_1 which output simulates F_{tiny} modulo $\mathrm{P}_{\{n\}}$, and modulo $\pi_{\{n\}}$ as well. Device G_1 can be obtained from Algorithm 3.



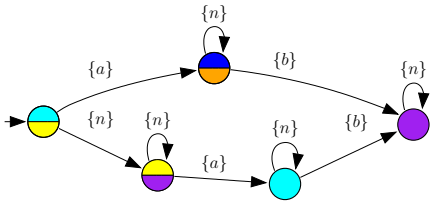
(b) A device G_2 that output simulates F_{tiny} modulo $\mathrm{P}_{\{n\}}$, but which fails modulo $\pi_{\{n\}}$.

Fig. 7: Two devices, related to F_{tiny}, the one in Figure 6, help illustrate how Theorem 27 is a statement about the existence of some device. A device that will output simulate modulo $\mathrm{P}_{\{n\}}$ need not modulo $\pi_{\{n\}}$; however, the devices Algorithm 3 produces will.

Remark 8. The condition on the first element of the sequences in Lemma 26 and Theorem 27 is necessary. The sensori-computational device F_{small} shown in Figure 8a is obtained by adding a string 'nab' to F_{tiny}. Figure 8b gives a device G_1' that output simulates F_{small} modulo $\mathrm{P}_{\{n\}}$. However, no device exists that can output simulate modulo $\pi_{\{n\}}$ because $\pi_{\{n\}}(na) = \pi_{\{n\}}(a) = a$, and $\{\text{cyan}\} \cap \{\text{red}, \text{blue}, \text{orange}\} = \varnothing$. This caveat is neither a particular concern nor limitation for us, as sensori-computational devices that are derivatives (Definition 10) have sequences where the first element is distinct. For these, the first element gives the offset or initial value, whereas the remainder has the role of tracking the dynamic variations. It is pumping or shrinking of these variations that is important.



(a) A new sensori-computational device F_{small} with an additional string '*nab*' being added to F_{tiny}.



(b) A device G'_1 that output simulates F_{small} modulo $\mathrm{P}_{\{n\}}$.

Fig. 8: After adding a new string, the sensori-computational device in Figure 6 is only output simulatable under $\mathrm{P}_{\{n\}}$, not under $\pi_{\{n\}}$.

VI. Data acquisition semantics reprise: Monoidal variators

The previous two sections do not break the atomicity of the symbols in the original signal space. For instance, the polling acquisition mode (modeled via the $\mathcal{N}$-pump relation) adds neutral elements to the stream; it does not consider what happens if a change is occurring continuously so that the query arrives amid a change. If we desire to query the sensor with maximal flexibility, such as if one were to model a general asynchronous interaction, then some extra structure is required. To move in this direction, we will need the output variator to possess some additional properties.

Definition 28 (monoidal variator)**.** A *monoidal variator* for an observation set Y, is a monoid $(D, \oplus, 1_D)$ and a right action[3] of D on Y, $\bullet : Y \times D \to Y$.

Being concise, we will write $(D, \bullet)$ for a monoidal variator, the notation showing an operation in the second slot that helps to indicate that it is an action and hence D has additional algebraic structure. (This is consistent with the previous notation when we would include the ternary relation within the pair.)

Some of the earlier examples had output variators that were monoidal or could be extended to be, while not so for others.

Example 11 (Lane sensor, again)**.** Building on Example 2, the observation variator was given as $D_{\text{3-lane}} = \{\,\text{LEFT}\,,\,\text{NULL}\,,\,\text{RIGHT}\,\}$. Given that there are 3 lanes, it means that one might wish to combine, say, two RIGHT actions, one after the other. As there are only three elements, two RIGHT actions might map to a RIGHT (as that seems less wrong than LEFT or NULL). Following this might give the following 'operator':

[3]Recall that $\bullet$ is a total function with two requirements— identity: $y \bullet 1_D = y$; compatibility: $(y \bullet d_1) \bullet d_2 = y \bullet (d_1 \oplus d_2)$, for all y in Y, and all d_1, d_2 in D.

$\oplus_1$	LEFT	NULL	RIGHT
LEFT	LEFT	LEFT	NULL
NULL	LEFT	NULL	RIGHT
RIGHT	NULL	RIGHT	RIGHT

But $\oplus_1$ fails to be a monoid operator as since the associativity rule does not hold: $(\,\text{LEFT} \oplus_1 \text{LEFT}\,) \oplus_1 \text{RIGHT} \neq \text{LEFT} \oplus_1 (\,\text{LEFT} \oplus_1 \text{RIGHT}\,)$.

Here is an alternative which does yield a valid operator, although it is still hard to give it a consistent interpretation:

$\oplus_2$	LEFT	NULL	RIGHT
LEFT	RIGHT	LEFT	NULL
NULL	LEFT	NULL	RIGHT
RIGHT	NULL	RIGHT	LEFT

But now the action causes difficulty. While NULL must map 0, 1 and 2, each to themselves, the form of $\oplus_2$ requires that the action treat $\text{RIGHT} \oplus_2 \text{RIGHT}$ identically with LEFT. This fails to describe lanes 0, 1 and 2, in a consistent fashion. The lanes do not seem to admit any monoidal variator. □

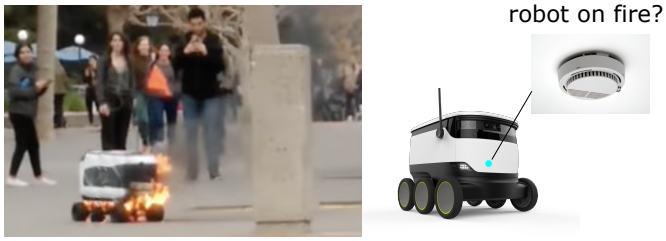


Fig. 9: A delivery robot equipped with a smoke detector in order to determine whether it has caught on fire.

Example 12 (Robot on fire)**.** Consider a sensor indicating that the robot has encountered some irrecoverable failure. For instance, the delivery robot shown in Figure 9 is equipped with a sensor to detect some irreversible condition. Once the sensor is triggered, it retains this status permanently.

Representing the status of the robot by 0 for 'normal' and 1 for 'abnormal', we may then use a monoid variator $D_2 = \{☺, ☹\}$, with 1_{D_2} is ☺, and the monoid operator $\oplus$ and the right action $\bullet$ in table form as:

$\oplus$	☺	☹
☺	☺	☹
☹	☹	☹

and

$\bullet$	☺	☹
0	0	1
1	1	1

□

Example 13 (Wall sensor, re-revisited)**.** The D_2 and table for S_{D_2} in Example 1 shows that it has the potential to be monoidal, and indeed an appropriate right action can be defined. □

The case in Example 13 also admits an inverse, leading on to the following.

Proposition 29. A sufficient condition for an affirmative answer to Question 1 when $(D, \bullet)$ is a monoidal variator is that D be, additionally, a group (i.e., possesses inverses), and there be a $y_0 \in Y(F)$ for which $y_0 \bullet D = Y(F)$.

Example 14 (90° Minispot, again)**.** In Example 4, we took as the output variator the integers and addition. After discussing restricting the transformation, what remained was, $\mathbb{Z}_8$, the cyclic group of order eight. □

Suppose a down-stream consumer of some device generates its queries in an asynchronous fashion. If that device measures changes, then it reports the change since the last query. When the consumer queries at a high frequency, the change sequence contains many elements, presumably with only moderate changes. Otherwise, the sequence is sparse and the change between symbols would be more considerable. To model this asynchronous data acquisition mode wherein the events reported are causally triggered by the down-stream element, we give the definition that follows. It expresses the idea that the sequences of changes should agree on the accumulated change, regardless of when the queries occur.

Definition 30 (monoid disaggregator)**.** Given the monoid $(D,\oplus,1_D)$ and observation set Y, the associated *monoid disaggregator* is a relation, $\partial\oplus\big|_Y \subseteq (\{\varepsilon\}\cup(Y\cdot D^*))\times(\{\varepsilon\}\cup(Y\cdot D^*))$ defined as:

0) $\varepsilon\,\partial\oplus\big|_Y\,\varepsilon$, and
1) $y_0\,\partial\oplus\big|_Y\,y_0$ for all $y_0\in Y$, and
2) $y_0d_1d_2\ldots d_m\,\partial\oplus\big|_Y\,y_0d_1'd_2'\ldots d_n'$ if $d_1\oplus d_2\oplus\cdots\oplus d_m = d_1'\oplus d_2'\oplus\cdots\oplus d_n'$.

Based on the above relation, and adhering to the established pattern, we have the natural question, posed in two forms:

Question 4. For any device F with monoidal variator $(D,\bullet)$ on $Y(F)$, is it $\left(\nabla\big|_F^D \fatsemi \partial\oplus\big|_{Y(F)}\right)$-simulatable?

Question 4 (Constructive). Given F with monoidal variator $(D,\bullet)$, find a sensori-computational device F' such that $F'\sim F\ \left(\mathrm{mod}\,\nabla\big|_F^D \fatsemi \partial\oplus\big|_{Y(F)}\right)$, or indicate none exist.

Remark 9. In the reflection at the beginning of Section V, on Example 4 and its relation to Example 3, attention was directed to the significance of missing an element from the symbol stream (there a skipped 45° gave the appearance of a 90° turn). With a monoidal variator one can talk meaningfully of a single symbol from the variator expressing accumulated changes over time: the monoidal variator will take you from any configuration to any other, regardless of how many elements in the sequence of 45° turns have occurred.

Question 4 is challenging because it is rather cumbrous. The source of this is Definition 30: it expresses the idea that asynchronous queries might happen at any time in a very direct and unwieldy way. So, instead, we will consider the 'accumulated changes' intuition just described in the previous remark. This gives a new relation (actually a function) and, following the pattern employed for the pump and shrinking cases, we will then form a connection between the two.

Definition 31 (monoid integrator)**.** Given the monoid $(D,\oplus,1_D)$ and observation set Y, the associated *monoid integrator* is a function $\int\oplus\big|_Y$ is given by:

$$\begin{aligned}\int\oplus\big|_Y\colon \{\varepsilon\}\cup(Y\cdot D^*) &\to \{\varepsilon\}\cup Y\cup(Y\cdot D)\\ \varepsilon &\mapsto \varepsilon\\ y_0 &\mapsto y_0\\ y_0d_1d_2\ldots d_m &\mapsto y_0(d_1\oplus d_2\oplus\cdots\oplus d_m).\end{aligned}$$

Notice that $\int\oplus\big|_Y$ is a rather different relation from the previous ones. All the relations express an alteration under which we wish the device to be invariant. Or more precisely: in which we seek to determine whether the requisite information processing *can* be invariant. The relations prior to Definition 31 all describe transformations which we may envision being produced and processed directly—it is easy to think of the robot operating in the world and tracing strings in the relation's image. Not so for the monoid integrator: it serves mostly as an abstract definition. All the strings are short: the robot gets at most two symbols. Nevertheless, the usual questions still apply:

Question 5. For any device F with monoidal variator $(D,\bullet)$, of $Y(F)$, is it $\left(\nabla\big|_F^D \fatsemi \int\oplus\big|_{Y(F)}\right)$-simulatable?

Question 5 (Constructive). Given F with monoidal variator $(D,\bullet)$, find a sensori-computational device F' such that $F'\sim F\ \left(\mathrm{mod}\,\nabla\big|_F^D \fatsemi \int\oplus\big|_{Y(F)}\right)$, or indicate none exist.

Algorithm 4: Monoid Integrator: $\textsc{Int}_D(F)\rightarrowtail G$

Input : A sensori-computational device F, a monoid variator with monoid $(D,\oplus,1_D)$
Output: A deterministic sensori-computational device G if G output simulates F modulo $\nabla\big|_F^D \fatsemi \int\oplus\big|_Y$; otherwise, return 'No Solution'

$F' \coloneqq \textsc{Delta}_D(F)$
if F' *is 'No Solution'* **then**
  return 'No Solution'
Initialize G with an initial state v_0
for *edge* $v_0' \xrightarrow{y} w'$ *in* F' **do**
  Create a state w in G and add edge $v_0 \xrightarrow{y} w$
  Associate w with w', $c(w) \coloneqq c(w')$
  $q' \coloneqq \mathit{Queue}([(1_D, w')])$
  while $q' \neq$ **do**
    $(d_{\mathrm{acc}}, v') \coloneqq q'.pop()$
    for $d \in v'.\mathit{OutgoingLabels}()$ **do**
      Let w' be the state such that $v' \xrightarrow{d} w'$
      if *there is no state* v_d *in* G **then**
        Create a state v_d in G, $c(v_d) \coloneqq c(w')$
        Add edge $w \xrightarrow{d} v_d$
      else
        $X \coloneqq c(v_d) \cap c(w')$
        if $X = \varnothing$ **then**
          return 'No Solution'
        else
          $c(v_d) \coloneqq X$
      if $(d_{\mathrm{acc}} \oplus d, w')$ *is not visited* **then**
        Add $(d_{\mathrm{acc}} \oplus d, w')$ to q'
return G

To answer these questions, a constructive procedure is given

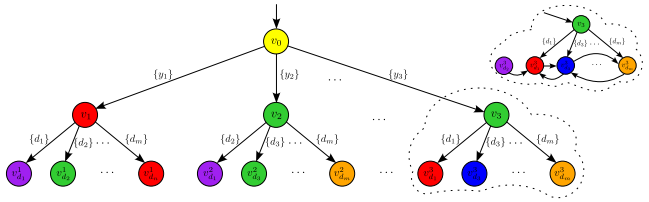


Fig. 10: A depiction of the tree with three layers employed in Algorithm 4 for the monoid integrator. Top right: the inset shows how Algorithm 5 modifies sub-trees so they gain extra edges (and potentially extra vertices) to make composite leaves as self-contained blocks with transitions to encode $(D,\bullet)$.

in Algorithm 4: it builds a three-layer tree, where the first layer is the initial state, the second layer consists of the states reached by a single label in $Y(F)$, the third layer consists of the states reached by strings yd where $y \in Y(F)$ and $d \in D$. The first layer is produced through lines 1–4, and the second layer through lines 6–7. The third layer is constructed via a depth-first search on the derivative F' as shown between lines 8–23 by keeping track of the accumulated change d_{acc} and creating the new state reached by yd_{acc} accordingly. The correctness of the algorithm follows next.

Lemma 32. Algorithm 4 gives a sensori-computational device that output simulates F modulo $\nabla\big|_F^D \fatsemi \int\oplus\big|_{Y(F)}$ if and only if there exists a solution for Question 5.

We remark on the fact that Algorithm 4 is unlike the previous algorithms. The preceding ones act as a sort of mutator: they begin by making a copy of their input graph, and local modifications are made before, finally, being determinized to catch inappropriate outputs (via the $c(\cdot)$ function) and to ensure that the resulting sensori-computational device is deterministic. This means that structure of their input F influences the structure of their result. But Algorithm 4 produces an sensori-computational device *de novo*, and its structure is essentially fixed: it is a 3-layer tree regardless of the specifics of F. The F serves really to define the input–output behavior. Figure 10 offers a visualization of the structure of F: the three-layer tree produces outputs for ε, the y_i elements, and strings of the form $y_j d_k$.

Now we return to consideration of Question 4.

Lemma 33. Given a sensori-computational device F with monoidal variator $(D,\bullet)$, F being $\left(\nabla\big|_F^D \fatsemi \partial\oplus\big|_F\right)$-simulatable implies that F is $\left(\nabla\big|_F^D \fatsemi \int\oplus\big|_F\right)$-simulatable.

Lemma 34. Given a sensori-computational device F with monoidal variator $(D,\bullet)$, F being $\left(\nabla\big|_F^D \fatsemi \int\oplus\big|_F\right)$-simulatable implies that F is $\left(\nabla\big|_F^D \fatsemi \partial\oplus\big|_F\right)$-simulatable.

Theorem 35. For any sensori-computational device F with monoidal variator $(D,\bullet)$,

$$F \text{ is } \left(\nabla\big|_F^D \fatsemi \partial\oplus\big|_F\right)\text{-simulatable} \iff F \text{ is } \left(\nabla\big|_F^D \fatsemi \int\oplus\big|_F\right)\text{-simulatable.}$$

Algorithm 5: Disaggregator: $\textsc{FillLeaves}_D(F') \rightarrowtail G$

Input : A monoid integrator device F' from Algorithm 4, and monoid variator D
Output: A deterministic device G that accepts additional elements in D

1 Initialize an empty graph M // Graph for D
2 **for** $d \in D$ **do**
3 | Create a state w_d in M
4 **for** $d_1, d_2 \in D$ **do**
5 | $d \coloneqq d_1 \oplus d_2$
6 | Form an edge $w_{d_1} \xrightarrow{d} w_{d_2}$ in M
7 Create graph G with a single vertex v_0, with output $c(v_0) \coloneqq c(v'_0)$
8 Let v'_0 be the initial state of F'
9 **for** *every outgoing edge* $v'_0 \xrightarrow{y} v'_y$ *in* F' **do**
10 | Add a vertex v_y to G
11 | Set $c(v_y) \coloneqq c(v'_y)$, and connect $v_0 \xrightarrow{y} v_y$
12 | Create M^y as a copy of M, adding M^y to G
13 | **for** *every outgoing edge* $v'_y \xrightarrow{d} v'_d$ *in* F' **do**
14 | | Form an edge $v_y \xrightarrow{d} w_d^y$
15 | | $c(w_d^y) \coloneqq c(v'_d)$
16 | Assign an arbitrary color to the remaining vertices in M^y
17 **return** G

The proof of Lemma 34 (available in the Supplement) uses Algorithm 5 to construct a sensori-computational device with a very specific form: two layers, as a tree, reaching 'composite-leaves' formed by connected blocks. The inset to Figure 10 illustrates this. (Note that Algorithm 5 itself takes as input Algorithm 4's output.) Since, for any F that is $\left(\nabla\big|_F^D \fatsemi \partial\oplus\big|_{Y(F)}\right)$-simulatable, there is a F' possessing this structure, we term it the *universal* monoid integrator. That same integrator will also output simulate F under $\nabla\big|_F^D \fatsemi \int\oplus\big|_{Y(F)}$. (Heed: the universality refers to its structural form, the particular outputs at each vertex will depend on the F and $(D,\bullet)$ used.) This structure is will lead to Theorem 39.

VII. Chatter-free behavior

In this brief section, we introduce two additional concepts that resemble some aspects of the relations in Section V, and which also connect with the topics just discussed in Section VI. We start by presenting a new, detailed example which will motivate a pair of additional definitions.

Example 15 (Driving drone)**.** The Syma X9 Flying Quadcopter Car, shown in Figure 11, is a robot marketed as a 'driving drone'. It is capable of switching between driving and flying modes, the idea being that it can make use of either mode of locomotion and determine what best suits the demands of its task. Imagine deploying such a robot in the

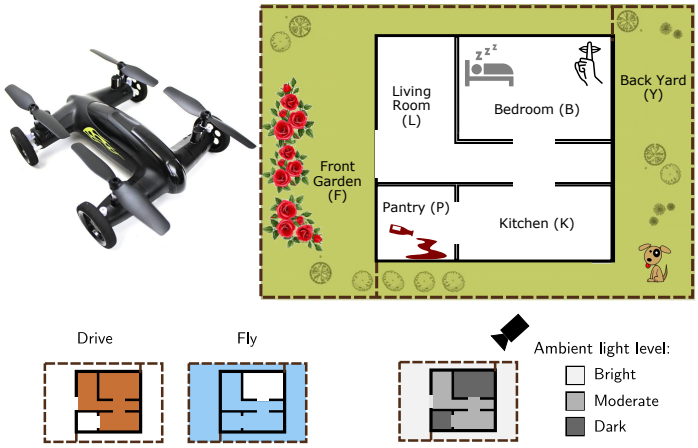


Fig. 11: A driving drone monitors a home environment. The robot is capable of both flight and wheeled locomotion and is equipped with a single-pixel camera. As an occupied residence, the space imposes complex constraints on how the vehicle may move. It must fly to avoid grass outdoors (F and Y) and liquids in the pantry (P); in the bedroom (B), it should drive to minimize noise. The robot is initially located in either the front garden (F) or the living room (L). To determine its state, the robot uses its single-pixel camera, which is capable of discerning just three different ambient light levels (Bright (*B*), Moderate (*M*), Dark (*D*)). The insets show: (left) the motion constraints and (right) the various light levels.

scenario shown as the map on right-hand side of the figure. The robot, starting in either the front garden (F) or the living room (L), will move about the home and garden. Its size and construction, along with task constraints, mean that it must adjust its mode of locomotion depending on where it is. The robot has a single-pixel camera with which it determines different levels of ambient light. (The figure's caption provides specific details, and further explanation.)

It uses a sensori-computational device that processes the light readings as input, and outputs the appropriate mode (driving or flying). Figure 12a gives such a device; it is essentially a state diagram encoding the problem constraints, topological structure, and raw sensor readings. As Figure 12a is essentially a transcription of the problem, it serves as a type of specification for acceptable input–output functionality.

An observation variator $D_\ell = \{+,-,=\}$ uses $+$ to capture the brightness increase (from dark to moderate, from moderate to bright), $-$ for brightness decrease (from bright to moderate, from moderate to dark), $=$ for brightness equivalence. A derivative device obtained by applying Algorithm 1 to Figure 12a with this variator is shown in Figure 12b. □

Figure 12c presents a device that, though different from the straightforward derivative in Figure 12b, also implements the functionality in Figure 12a under delta relation associated with D_ℓ. That is to say, it also output simulates modulo $\nabla\big|_F^{D_\ell}$ and is, thus, also a derivative. Figure 12c, being smaller that either 12a or 12b, might be desirable for practical purposes. But now consider that the '=' element of the variator is produced when there is no change in light levels. If events are triggered when the robot moves from one room (or region) to the next, then there may not be too many of them. On the other hand, if the robot is using these readings to localize, that is, to actually determine that it may have transitioned from one room or region to the next, then many such elements will likely be generated.

Particularly when there are cycles on elements such as the '=' symbol, as these are 'neutral' changes in the signal, this may induce oscillatory behavior in the device as it fluctuates between states, flip-flopping rapidly. One may ask, thus, whether there are sensori-computational devices that can avoid this issue.

Definition 36 (vertex stable)**.** For an $F = (V, V_0, Y, \tau, C, c)$ and a set $\mathcal{N} \subseteq Y(F)$, we say F is *vertex stable with respect to* $\mathcal{N}$ when, for all $s_1 s_2 \ldots s_{n-1} s_n \in \mathscr{L}(F)$ with $s_n \in \mathcal{N}$, $\mathscr{V}_F(s_1 s_2 \ldots s_{n-1}) = \mathscr{V}_F(s_1 s_2 \ldots s_n)$.

Intuitively: having handled an input stream of symbols, processing an additional element from $\mathcal{N}$ does not cause a vertex stable device to move to a new vertex.

Remark 10. The concept of vertex stability is related to, but distinct from, the concept of the $\mathcal{N}$-pump (from Definition 22). For instance, a vertex stable sensori-computational device may not always be ready for an element from $\mathcal{N}$. On the other hand, simulating modulo the $\mathcal{N}$-pump relation need not imply the device is vertex stable; Figure 13 provides such an example.

Figure 12d is another derivative for the driving drone scenario (and is smaller even than Figure 12c). It has not only a multi-state cycle on the '=' symbol, but this time the oscillatory behavior produces fluctuations in the output stream, with values going: blue, brown, blue, brown, ... Given that these describe actions for the robot to take-off, and then land, then take-off, ... this is highly undesirable behavior. One naturally asks whether a device exists which avoids this issue:

Definition 37 (output stable)**.** For an $F = (V, V_0, Y, \tau, C, c)$ and a set $\mathcal{N} \subseteq Y(F)$, we say F is *output stable with respect to* $\mathcal{N}$ when, for all $s_1 s_2 \ldots s_{n-1} s_n \in \mathscr{L}(F)$ with $s_n \in \mathcal{N}$, $\mathscr{C}(F, s_1 s_2 \ldots s_{n-1}) = \mathscr{C}(F, s_1 s_2 \ldots s_n)$ and $|\mathscr{C}(F, s_1 s_2 \ldots s_n)| = 1$.

This concept is related to the notion of chatter in switched systems, and work that schedules events of a switched system in order to be non-chattering [2].

Further we note, both Figures 12c and 12d differ from Figure 12b in that they provide a single output per state. One is always free to make a singleton prescription:

Property 38. For any device $F = (V, V_0, Y, \tau, C, c)$, suppose one constructs $F_{\text{sing}} = (V, V_0, Y, \tau, C, c_{\text{sing}})$, where c_{sing} is a version of c restricted to singleton choices, viz., picked so that for all $v \in V$, $c_{\text{sing}}(v) \subseteq c(v)$ and $|c_{\text{sing}}(v)| = 1$. Then $F_{\text{sing}} \sim F \pmod{\mathbf{id}}$.

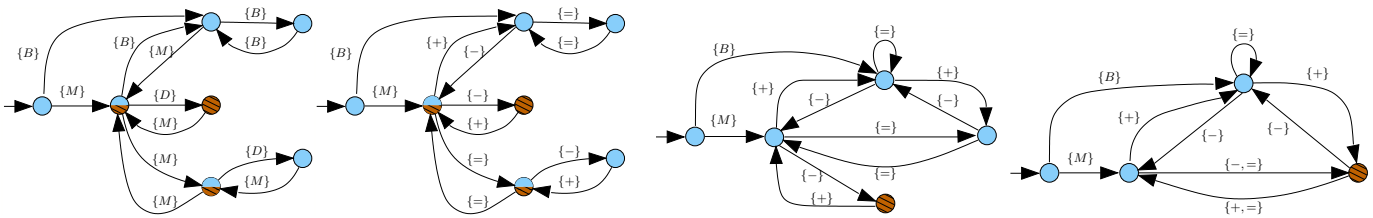


Fig. 12: Sensori-computational devices to choose appropriate locomotion modes for the driving drone in Figure 11, with blue=fly, brown=drive; (a) works in the original signal space. The other three are derivatives that operate in the space of changes as expressed via variator D_ℓ. Applying $\textsc{Delta}_{D_\ell}(\text{a})$ gives (b). Both (c) and (d) choose a single output for each vertex; (c) is output stable, while (d) is not.

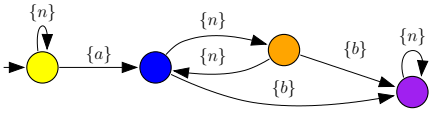


Fig. 13: A device that output simulates F_{tiny} modulo $\mathrm{P}_{\{n\}}$. (Recall that device F_{tiny} is the one in Figure 6.) It is neither vertex stable nor output stable with respect to $\{n\}$. The device can be made output stable, without becoming vertex stable, by altering the orange to become blue. Thereupon, vertex stability is possible if the two blue vertices are merged.

Hence, if one seeks an output stable sensori-computational device, then finding a vertex stable one will suffice (because, formally, Theorem 8 can be applied at last step to change to a singleton version).

The universal monoid integrator (from Algorithm 5) has blocks that encode the transitions arising from the action of the monoid. Being a monoid action, the 1_D element is neutral, which manifests in a simple fact: vertices in the third layer, after $\textsc{FillLeaves}_D(\cdot)$, have self loops labeled with 1_D. This specific structure leads to the following observation.

Theorem 39. For any device F with monoidal variator $(D,\bullet)$, $1_D \notin Y(F)$, which is $\left(\nabla\big|_F^D \fatsemi \int \oplus\big|_F\right)$-simulatable, there exists a single F' such that:

1) $F' \sim F \ \left(\mathrm{mod}\, \nabla\big|_F^D \fatsemi \partial\oplus\big|_{Y(F)}\right)$,
2) $F' \sim F \ \left(\mathrm{mod}\, \nabla\big|_F^D \fatsemi \mathrm{P}_{\{1_D\}}\right)$,
3) $F' \sim F \ \left(\mathrm{mod}\, \nabla\big|_F^D \fatsemi \pi_{\{1_D\}}\right)$,
4) F' is vertex stable with respect to $\{1_D\}$, and
5) F' is output stable with respect to $\{1_D\}$.

VIII. Summary and Outlook

This paper's focus has been less on sensors as used by people currently but rather on whether some hypothetical event sensor might be useful were it produced. So: what then is an event sensor, exactly? The preceding treatment has shown that there are several distinct facets. At the very core is the need to have some signal space on which differences can be meaningfully computed. This requires some basic statefulness, even if it is very shallow (like Example 5, the single-pixel camera). We formalize this idea in the concept of an observation variator. Also important is the model of event propagation. In this paper four separate cases have been identified and distinguished, namely: tightly coupled synchronous, event-triggered, polling, and asynchronous cases. At least in our framework, some of these choices depend on variators having certain properties with which to encode or express aspects of signal differences. Our model expresses these cases through relations. The notion of output simulation modulo those relations leads to decision questions, for which we were able to provide algorithms that give solutions if they exist. There remain other properties of interest and practical importance (such as vertex and output stability) which one might like to impose as constraints on the sensori-computational devices one seeks. We can meet these constraints when the variator possesses the algebraic structure of a monoid, as our final theorem is constructive.

More work remains to be done, but a start has been made on the question of whether information conflated in the process of forming an event sensor—the process of eventification—harms input–output behavior. We especially believe that this paper's extension of the notion of output simulation, and the algorithms we describe, ought to serve as a useful foundation for future work. One important limitation of the theory developed in this paper is that, as it depends on sequences of symbols, discrete time appears from the very outset. To directly model truly analog devices (as distinct from eventified digital ones) a theory dealing with continuous time may be required. Since the vast majority of robots process streams of digitized data, this may be a question of what one decides to treat as the atomic elements that generate observations. But one might also imagine a hybrid theory, connecting a continuous time approach with the model presented in this work.

Acknowledgements

The authors would like to thank the anonymous reviewers who identified several opportunities for improvements, e.g., Section IV-D arose directly in response to an insightful question. This work was partly supported by the NSF through award IIS-2034097, and partly through the support of Office of Naval Research Award #N00014-22-1-2476.

REFERENCES

[1] Michael C. Browne, Edmund M. Clarke, and Orna Grümberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59(1–2):115–131, 1988.

[2] Timothy M. Caldwell and Todd D. Murphey. Projection-Based Iterative Mode Scheduling for Switched Systems. *Nonlinear Analysis: Hybrid Systems*, 21:59–83, August 2016.

[3] Jonathan H. Connell. *Minimalist Mobile Robotics: A Colony Style Architecture for an Artificial Creature*. Academic Press, San Diego, CA, 1990.

[4] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[5] Rocco De Nicola and Frits W. Vaandrager. Three logics for branching bisimulation. *Journal of the ACM*, 42(2):458–487, March 1995.

[6] Bruce R. Donald. On Information Invariants in Robotics. *Artificial Intelligence — Special Volume on Computational Research on Interaction and Agency, Part 1*, 72(1–2):217–304, January 1995.

[7] Bruce R. Donald and James Jennings. Perceptual limits, perceptual equivalence classes, and a robot's sensori-computational capabilities (extended abstract). In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and System*, pages 1397–1405, Munich, Germany, September 1991.

[8] Michael A. Erdmann. Understanding action and sensing by designing action-based sensors. *International Journal of Robotics Research*, 14(5):483–509, October 1995.

[9] Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, Jorg Conradt, Kostas Daniilidis, and et al. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(1):154–180, January 2022.

[10] Shervin Ghasemlou and Jason M O'Kane. Accelerating the construction of boundaries of feasibility in three classes of robot design problems. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and System*, Macau, China, 2019.

[11] Robert Hermann and Arthur Krener. Nonlinear Controllability and Observability. *IEEE Transactions on Automatic Control*, 22(5):728–740, October 1977.

[12] iRobot Corp. iRobot® Create® 2 Open Interface (OI). Last Updated April 2, 2015.

[13] Rudolf Emil Kalman. Mathematical description of linear dynamical systems. *Journal of the Society for Industrial and Applied Mathematics, Series A: Control*, 1(2):152–192, 1963.

[14] Steven M. LaValle. Sensing and filtering: A fresh perspective based on preimages and information spaces. *Foundations and Trends in Robotics*, 1(4):253–372, February 2012.

[15] Steven M. LaValle. Sensor lattices: Structures for comparing information feedback. In *IEEE International Workshop on Robot Motion and Control (RoMoCo)*, pages 239–246, July 2019.

[16] Steven M. LaValle and Magnus B. Egerstedt. On time: Clocks, chronometers, and open-loop control. In *IEEE Conference on Decision and Control*, pages 1916–1922, December 2007.

[17] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128×128 120dB 15μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, 43(2):566–576, February 2008.

[18] Shih-Chii Liu and Tobi Delbruck. Neuromorphic sensory systems. *Current Opinion in Neurobiology*, 20(3):288–295, 2010.

[19] Shih-Chii Liu, André van Schaik, Bradley A. Minch, and Tobi Delbruck. Asynchronous binaural spatial audition sensor with $2 \times 64 \times 4$ channel output. *IEEE Transactions on Biomedical Circuits and Systems*, 8(4):453–464, 2014.

[20] Ana I. Maqueda, Antonio Loquercio, Guillermo Gallego, Narciso García, and Davide Scaramuzza. Event-based vision meets deep learning on steering prediction for self-driving cars. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5419–5427, Salt Lake City, UT, USA, June 2018.

[21] Matthew T. Mason. Kicking the sensing habit. *AI Magazine*, 14(1):58–58, Spring 1993.

[22] Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[23] Dharmendra S. Modha, Filipp Akopyan, Alexander Andreopoulos, Rathinakumar Appuswamy, John V. Arthur, Andrew S. Cassidy, Pallab Datta, Michael V. DeBole, Steven K. Esser, Carlos Ortega Otero, Jun Sawada, Brian Taba, Arnon Amir, Deepika Bablani, Peter J. Carlson, Myron D. Flickner, Rajamohan Gandhasri, Guillaume J. Garreau, Megumi Ito, Jennifer L. Klamo, Jeffrey A. Kusnitz, Nathaniel J. McClatchey, Jeffrey L. McKinstry, Yutaka Nakamura, Tapan K. Nayak, William P. Risk, Kai Schleupen, Ben Shaw, Jay Sivagnaname, Daniel F. Smith, Ignacio Terrizzano, and Takanori Ueda. Neural inference at the frontier of energy, space, and time. *Science*, 382(6668):329–335, 2023.

[24] Elias Mueggler, Henri Rebecq, Guillermo Gallego, Tobi Delbruck, and Davide Scaramuzza. The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM. *International Journal of Robotics Research*, 36(2):142–149, 2017.

[25] Gennaro Notomista, Sebastian F. Ruf, and Magnus B. Egerstedt. Persistification of robotic tasks using control barrier functions. *IEEE Robotics and Automation Letters*, 3(2):758–763, April 2018.

[26] Jason M. O'Kane and Dylan A. Shell. Concise planning and filtering: hardness and algorithms. *IEEE Transactions on Automation Science and Engineering*, 14(4):1666–1681, October 2017.

[27] Vincent Pacelli and Anirudha Majumdar. Task-driven estimation and control via information bottlenecks. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2061–2067, May 2019.

[28] Hazhar Rahmani and Jason M O'Kane. On the relationship between bisimulation and combinatorial filter reduction. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 7314–7321, Brisbane, Australia, 2018.

[29] Henri Rebecq, René Ranftl, Vladlen Koltun, and Davide Scaramuzza. Events-to-video: Bringing modern computer vision to event cameras. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3857–3866, Long Beach, CA, USA, June 2019.

[30] Fatemeh Z. Saberifar, Shervin Ghasemlou, Dylan A. Shell, and Jason M. O'Kane. Toward a language-theoretic foundation for planning and filtering. *International Journal of Robotics Research*, 38(2-3):236–259, March 2019.

[31] Fatemeh Zahra Saberifar, Shervin Ghasemlou, Jason M O'Kane, and Dylan A Shell. Set-labelled filters and sensor transformations. In *Robotics: Science and Systems*, Ann Arbor, Michigan, 2016.

[32] Fatemeh Zahra Saberifar, Ali Mohades, Mohammadreza Razzazi, and Jason M O'Kane. Combinatorial filter reduction: Special cases, approximation, and fixed-parameter tractability. *Journal of Computer and System Sciences*, 85:74–92, 2017.

[33] Nitin J Sanket, Chahat Deep Singh, Chethan Parameshwara, Cornelia Fermüller, Guido de Croon, and Yiannis Aloimonos. EVPropNet: Detecting Drones By Finding Propellers For Mid-Air Landing And Following. In *Robotics: Science and Systems*, Virtual, July 2021.

[34] Tanmoy Sarkar, Katharina Lieberth, Aristea Pavlou, Thomas Frank, Volker Mailaender, Iain McCulloch, Paul WM Blom, Fabrizio Torricelli, and Paschalis Gkoupidenis. An organic artificial spiking neuron for in situ neuromorphic sensing and biointerfacing. *Nature Electronics*, 5(11):774–783, 2022.

[35] Paulo Tabuada, George J Pappas, and Pedro Lima. Compositional abstractions of hybrid control systems. *Discrete Event Dynamic Systems*, 14(2):203–238, 2004.

[36] Tasbolat Taunyazov, Weicong Sng, Brian Lim, Hian Hian See, Jethro Kuan, Abdul Fatir Ansari, Benjamin Tee, and Harold Soh. Event-Driven Visual-Tactile Sensing and Learning for Robots. In *Robotics: Science and Systems*, Corvalis, Oregon, USA, July 2020.

[37] Tong Wang, Xiao-Xue Wang, Juan Wen, Zhe-Yuan Shao, He-Ming Huang, and Xin Guo. A bio-inspired neuromorphic sensory system. *Advanced Intelligent Systems*, 4(7):2200047, 2022.

[38] Gioele Zardini, David I. Spivak, Andrea Censi, and Emilio Frazzoli. A compositional sheaf-theoretic framework for event-based systems. *Electronic Proceedings in Theoretical Computer Science*, 333:139–153, February 2021.

[39] Yulin Zhang, Hazhar Rahmani, Dylan A. Shell, and Jason M. O'Kane. Accelerating combinatorial filter reduction through constraints. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 9703–9709, May 2021.

[40] Yulin Zhang and Dylan A. Shell. Lattices of sensors reconsidered when less information is preferred. In *[43]*, May 2021.

[41] Alex Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. EV-FlowNet: Self-Supervised Optical Flow Estimation for Event-based Cameras. In *Robotics: Science and Systems*, Pittsburgh, PA, USA, June 2018.

[42] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. Unsupervised event-based learning of optical flow, depth, and egomotion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 989–997, Long Beach, CA, USA, June 2019.

[43] IEEE ICRA 2021 Workshop — Compositional Robotics: Mathematics and Tools, May 2021. https://idsc.ethz.ch/research-frazzoli/workshops/compositional-robotics.html.