Set-labelled filters and sensor transformations

Fatemeh Zahra Saberifar
Dept. of Math. and Comp. Sci.
Amirkabir University of Technology,
Tehran, Iran
fz.saberifar@aut.ac.ir

Shervin Ghasemlou, Jason M. O'Kane
Dept. of Comp. Sci. & Eng.
University of South Carolina
Columbia, South Carolina, USA
sherving@cse.sc.edu, jokane@cse.sc.edu

Dylan A. Shell
Dept. of Comp. Sci. & Eng.
Texas A&M University
College Station, Texas, USA
dshell@cs.tamu.edu

***Abstract*—For a given robot and a given task, this paper addresses questions about which modifications may be made to the robot's suite of sensors without impacting the robot's behavior in completing its task. Though this is an important design-time question, few principled methods exist for providing a definitive answer in general. Utilizing and extending the language of combinatorial filters, this paper aims to fill that lacuna by introducing theoretical tools for reasoning about sensors and representations of sensors. It introduces new representations for sensors and filters, exploring the relationship between those elements and the specific information needed to perform a task. It then shows how these tools can be used to algorithmically answer questions about changes to a robot's sensor suite. The paper substantially expands the expressiveness of combinatorial filters so that, where they were previously limited to quite simple sensors, our richer filters are able to reasonably model a much broader variety of real devices. We have implemented the proposed algorithms, and describe their application to an example instance involving a series of simplifications to the sensors of a specific, widely deployed mobile robot.**

I. Introduction

This paper lays theoretical groundwork for reasoning about sensors and their associated estimation processes, with the goal of strengthening the link between idealized models and practical —that is, imperfect, imprecise, and limited— realizations of those idealized models in hardware. This kind of groundwork is important because we believe that the lack of any such theory has hampered progress in robotics to date.

In this paper, we address the problem of determining whether certain perturbations in sensor specification or performance affect the behavior of a robotic system. (Figure 1 provides a visual overview.) We seek to understand which changes to a sensor suite might make a system inoperable, which changes result in graceful degradation of performance, and which changes might have little or no effect. Transformations that, in the picture, represent hardware realizations of the ideal range sensor may either preserve behavior or alter it. In that latter case, we say it is destructive. The key insight —an insight not unique to this paper, but one whose long history is briefly reviewed in Section II— is that whether a transformation is destructive or not depends on the information needed for the task being performed.

To motivate these questions more concretely, consider the pair of scenarios that follows. They emphasize the paper's focus on pragmatic concerns, in spite of its apparent theoretical flavor.

This material is based upon work supported by the National Science Foundation under Grants IIS-1527436, IIS-1526862, IIS-0953503, IIS-1453652.

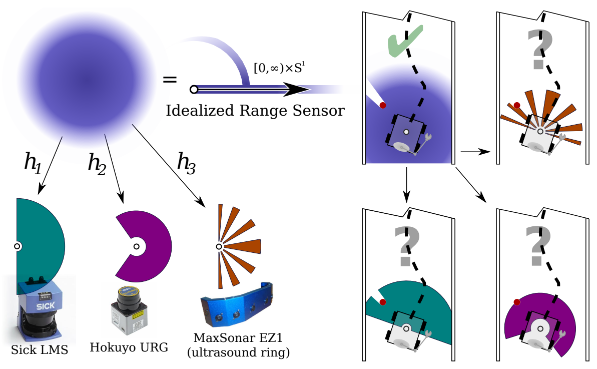


Fig. 1. Abstract sensors enable the creation and analysis of general algorithms that use pure, generic properties of certain types of sensing devices. This paper explores how to weaken sensor abstractions in order to faithfully model real hardware limitations. For a strong notion of behavioral equivalence, we examine conditions which ensure that a robot with a weaker sensor behaves as if it possessed an idealized one, algorithms for transforming sensor descriptions, as well as examining the hardness of deciding related questions. The theory and algorithms we introduce allow one to turn the question marks in the diagram into precise yes/no determinations.

Example 1. Your robot is stationed on a distant planet and, though fully operable initially, has recently encountered a problem. It appears that debris has become affixed to one of the sensors. Should operations be altered by taking more conservative paths around obstacles because the robot's position estimates now involve greater error than previously? Or has the mission been entirely compromised? Assuming that the debris cannot be dislodged, what tasks are still feasible?

Example 2. You lead an R&D team who have built and tested a successful prototype robot, which performs cosmetic services (e.g., manicures, pedicures, facials, hair-weaves, etc.) efficiently and safely. Then…disaster! You discover that the sensor provided to your factory in bulk (say S_1), differs from the device (S_0) supplied by the same manufacturer to the team who built and tested the prototype. A successful redesign of the robot might require answers to these kinds of questions: Can S_1 be used directly as a plug-and-play replacement for S_0? If not, can we adjust some software parameters to make it work? Which parameters and what should the adjustments be? If S_1 necessarily incurs a loss in performance, how can this be understood—perhaps only the hair-styling functionality is affected? Supposing we can procure S_0 at greater cost through another vendor, is this worth doing?

Underlying these scenarios is the problem of how to ascertain whether or not a particular sensor modification is destructive for a given task. We formalize this question, providing theoretical foundations as well as algorithms to address this problem, by posing and studying sensor transformations within the context of *combinatorial filters*. These filters provide a broad, abstract theoretical treatment of algorithmic processes that aggregate information. The word filter is, of course, most familiar as a term used to describe practical estimation components of robots and their controller software. The filters treated in this paper subsume those, representing a larger class.

Until now, most work on combinatorial filters has been hampered by being too simple to model concisely anything but the most limited devices. One important innovation in the present paper is the expansion of filters to include implicitly represented observation sets on transitions. Previously, when sets of observations were treated, they required duplication (usually of an edge in a graph structure), causing substantial blow-up of the model. This paper introduces a form which is much richer: the work permits transitions to describe a large (or even infinite) set of labels. As should become clear, this enables a vast advancement in terms of practicability.

The contributions of this paper are summarized below.

1) We introduce, in Section III, a generalized formulation of combinatorial filters, which model a broad class of sensing and estimation systems. By using set-labelled transitions, our new formulation is more amenable to analysis of realistic sensor systems than prior models for combinatorial filters, and, through judicious injections of nondeterminism, is suitable for a broader range of useful algorithmic manipulations.
2) We identify, in Section IV, useful special classes of these generalized combinatorial filters and classify the relationships between those classes. We present algorithms for converting between these classes without altering the filter's behavior, when such conversions are possible.
3) We show, in Section V, how to model sensor transformations in this framework, and describe an algorithm to determine whether a sensor transformation is destructive.
4) We prove, in Section VI, that the broader question of finding a non-destructive sensor transformation that is, in a certain sense, maximal, is NP-hard.
5) We explore, in Section VII, a detailed example of how these techniques might be applied.

Throughout, we show examples computed by a Python implementation of the paper's new algorithms.

II. Related work

A gap remains between the theory of discrete combinatorial filters and the probabilistic, typically recursive Bayes formulations, employed most often in practice on robots today. Both types have a long history. The probabilistic filters go back to Kalman [7], having found use in several important problems in mobile robotics, including estimation of robot pose and map information [1, 14]. This class of filters is well-known within the community, with a vast surge of interest catalyzed by the publication of the book by Thrun et al. [16].

The discrete filters we focus on in this paper have their roots in the minimalist manipulation work of Erdmann and Mason [3] and Goldberg [5]. They were formalized more generally by LaValle [9, 10], though this paper evolves those models in a new direction. These sorts of representations have been employed in the form of so-called combinatorial filters to successfully solve a wide a range of useful tasks; recent examples include target tracking [19], mobile robot navigation [11, 17], and manipulation [8].

One criticism that has been leveled at combinatorial filters is their expression complexity and comparative size even for very simple sensors. A series of techniques have also been developed to reduce or simplify the representation of information within such filters [12, 13, 15]. An important contribution of the present paper is a significant increase in the complexity of sensors that may be effectively treated by discrete filters. The formulation for filters below does not assume that the set of possible observations is finite. We describe results for filters with infinite (though finitely described) subsets of $\mathbb{R}$ as labels. This idea was inspired by Veanes et al. [18], who developed symbolic finite transducers that are concise and expressive for processing strings over large alphabets.

III. Standard filters and procrustean filters

A. Standard filters

The central object of study in this paper is a discrete transition system. We follow the precedent in the literature (e.g. *cf.* models of [9, 13, 19]) with the following definition, in which we have added the appellation 'standard' to distinguish from the more general filters introduced below.

Definition 1. A *standard filter* is a tuple $\langle Q, q_0, Y, \delta, C, c\rangle$, with:

1) a finite set, Q, of states,
2) one particular initial state $q_0 \in Q$,
3) a set of possible observations Y,
4) a transition map $\delta : Q \times Y \nrightarrow Q$, which is a partial function.
5) a set C, which we call an output space, and
6) an output function $c : Q \to C$.

We use $\mathcal{F}_{\text{std}}$ to denote the set of all standard filters.

The underlying idea is that a filter receives a sequence of observations and produces a sequence of outputs in response, transitioning from state to state according to the filter's transition map. For any given sequence of observations $y_1 y_2 \ldots y_n \in Y^\star$, one traces these on some standard filter F by, starting at q_0, transitioning from state to state by following $\delta(q_{i-1}, y_i) = q_i$, for $i = \{1, \ldots, n\}$. There is no requirement that $\delta(q, y)$ have a value for every $q \in Q$ and $y \in Y$ pair. When observation sequences are encountered with this kind of missing transition, the resulting state is undefined. For each sequence of states, $q_0, q_1 \ldots q_n$, we say that F outputs $c_0 c_1 \ldots c_n$, if $c_i = c(q_i)$, for $i = \{0, \ldots, n\}$. We occasionally use the term *color* to refer to a specific output. All standard filters have the property that for any $y_1 y_2 \ldots y_n$ at most a single output $c_0 c_1 \ldots c_n$ can be produced; for input strings of observations that always result in defined state transitions on the filter, exactly one output is produced.

Theoretically, standard filters represent minimal, non-trivial information processing constructs for operating on sequences of observations; it is difficult to pose a more fundamental abstraction than this basic type of filter. Practically they have the obvious advantage of a very straightforward implementation. Both of these aspects are important motivators for the present study.

B. Procrustean filters

Although perhaps not immediately obvious, it is useful to consider a generalization of the standard filter in which the observations that mark transitions and the outputs in each state are sets rather than single values:

Definition 2. A *procrustean filter*, or *p-filter* for short, is a tuple $\langle Q, Q_0, Y, \tau, C, c\rangle$, with:

1) a finite set, Q, of states,
2) a non-empty initial set of states $Q_0 \subseteq Q$,
3) a set of possible observations Y,
4) a transition function $\tau : Q \times Q \to 2^Y$,
5) a set C, which we call an output space, and
6) an output function $c : Q \to 2^C - \{\varnothing\}$.

We use $\mathcal{F}_\mathrm{p}$ to denote the set of all p-filters.

Given any p-filter $\mathrm{F} = \langle Q, Q_0, Y, \tau, C, c\rangle$, an observation sequence $y_1 \dots y_n \in Y^\star$, and an output sequence $c_0 c_1 \dots c_n \in C^+$, we say that $y_1 y_2 \dots y_n$ yields $c_0 c_1 \dots c_n$ under F, if there exists a sequence of states $q_0, q_1, \dots, q_n$:

1) $q_0 \in Q$, and
2) for each $i \in \{1, \dots, n\}, y_i \in \tau(q_{i-1}, q_i)$, and
3) for each $i \in \{0, 1, \dots, n\}, c_i \in c(q_i)$.

The set of output sequences yielded by $y_1 \dots y_n$ under F is denoted $[y_1 \dots y_n]_\mathrm{F}$. Note that the empty string ϵ yields single-element output sequences, so that $[\epsilon]_\mathrm{F} = \bigcup\limits_{q_0 \in Q_0} c(q_0) \neq \varnothing$.

In this definition, the standard filter has been extended to include non-determinism on three fronts: (i) there may be more than one initial state; (ii) transitions can occur if there is some element in in the labelled transition (τ); and (iii) the output can be any of the elements associated with the state (via c).

C. P-filter equivalence

Because we consider, in Section IV, algorithms intended to transform the representation of a p-filter without altering its behavior, we must introduce a notion of equivalence between p-filters.

Definition 3. Two p-filters F_1 and F_2 are *equivalent* if, for every observation $y_1 \dots y_n \in Y^\star$, we have $[y_1 \dots y_n]_{\mathrm{F}_1} = [y_1 \dots y_n]_{\mathrm{F}_2}$.

Informally speaking, two filters are equivalent to one another if the outputs they produce are identical given the same inputs from $Y^\star$. That is if F_1 and F_2 produce the same outputs, regardless the observations received, then the two filters are equivalent (even if their states, labels, *etc.* differ).

D. Label spaces and operations thereon

It is helpful to think of the transition in a p-filter from q_i to q_j as bearing the label $\ell = \tau(q_i, q_j)$, in which ℓ represents a set of observations. This model is particularly important for systems in which the observation space is large or infinite—including most nontrivial real sensor systems—in which it would be, at best, computationally intractable to list observations individually.

To represent such labels practically, we assume that each element in the image of τ is contained in a label space L, in which each label $\ell \in L$ is a set of observations. The label sets may be represented in a variety of ways. We only require that L be equipped with the following six operations.

1–3. UNION, which accepts two labels and computes a new label representing their union, along with INTERSECTION and DIFFERENCE, which operate *mutatis mutandis* for the intersection and set difference operations.
4. EMPTY, which accepts a label and returns TRUE if and only if the label represents the empty set.
5. CONTAINS, which accepts a label and an observation, and decides whether that observation is member of the set represented by that label.
6. REPRESENTATIVE, which accepts a non-empty label and returns an observation contained in the set represented by that label.

Any data structure capable of answering these queries is suitable for representing the labels in the algorithms in this paper. Some examples follow.

Example 3. Suppose $Y = \mathbb{R}$. Since each label should represent a set of real numbers, one option is to let each label represent a finite union of real intervals. The intervals may be bounded or unbounded. Each interval may also be open, closed, or half-closed. Figure 2 shows an example. To represent a label from this label space, we use a data structure with three parts:

1) A list of n real number *endpoints* $e_1, \dots, e_n \in \mathbb{R}$.
2) A list of $n+1$ boolean *interval flags* $f_1, \dots, f_{n+1}$. The interpretation is that, for each $1 < j < n$, the real numbers between e_j and e_{j+1} are included in the set if and only if f_j is TRUE. At the extremes, real numbers less than e_1 are in the set when f_1 is TRUE, and likewise numbers greater than e_n are in the set when f_n is TRUE.
3) A list of n boolean *endpoint flags* $p_1, \dots, p_n$, with the semantics that, for any $1 \leq j \leq n$, the real number e_j is in the label's observation set if and only if p_j is TRUE.

Note that any finite union of real intervals (including, for example, the empty set and the full real line, which have $n = 0$) can be expressed in this format.

The UNION, INTERSECTION, and DIFFERENCE operations can be implemented by performing a left-to-right sweep, adding endpoints and flags appropriately to the result label.

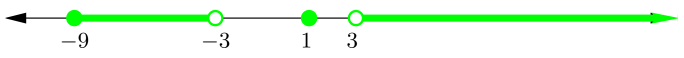


Fig. 2. An interval label for the set $[-9, -3) \cup \{1\} \cup (3, \infty)$. The label data structure has 4 endpoints $(-9, -3, 1, 3)$, 5 interval flags (FALSE, TRUE, FALSE, FALSE, TRUE), and 4 endpoint flags (TRUE, FALSE, TRUE, FALSE).

The EMPTY method requires a simple check for any endpoint flags or interval flags that are TRUE. The CONTAINS check can be implemented by a binary search for the correct interval, followed by a check against the relevant flag. REPRESENTATIVE should return an element, either an endpoint or in the interior of an interval (in the general case, perhaps the midpoint between two endpoints) for which the corresponding flag is TRUE.

Example 4. Labels that represent a finite number of elements —as is the case for many simple sensors such beam detectors or bump sensors— can be modeled by storing the elements explicitly in almost any container data structure, such as a balanced binary tree or a hash table.

Example 5. We expect that a common case will involve observations spaces that are composed, via Cartesian product, from simpler sets. That is, we may generally have $Y = Y_1 \times \cdots \times Y_m$, in which each Y_i is an observation space for which we have a suitable label space, including the requisite operations. In such a case, we can define a label space $\hat{L}$ over Y in which each label represents a union of Cartesian products of sub-labels, in the form $\bigcup_i \left(\ell_1^{(i)} \times \cdots \times \ell_m^{(i)}\right)$, where $i \in \{1, \ldots, m\}$. Under this representation, a UNION between labels becomes a mere concatenation of Cartesian product lists. The INTERSECTION operation requires pairwise intersections between each of the constituent Cartesian products of each of the two labels:

$$\left[\bigcup_i \left(\ell_1^{(i)} \times \cdots \times \ell_m^{(i)}\right)\right] \cap \left[\bigcup_j \left(m_1^{(j)} \times \cdots \times m_m^{(j)}\right)\right]$$
$$= \bigcup_i \bigcup_j \left((\ell_1^{(i)} \cap m_1^{(j)}) \times \cdots \times (\ell_n^{(i)} \cap m_m^{(j)})\right).$$

The DIFFERENCE operation is similar, but requires first a refinement —see below— of the labels along each dimension.

E. Label refinement

Several of the algorithms in Sections IV and V rely on a subroutine to compute of a *refinement* of a set of labels. Specifically, we need in several places an algorithm that accepts as input an unordered set of labels $\ell_1, \ldots, \ell_n$, and produces as output an unordered set of labels $\ell'_1, \ldots, \ell'_m$, such that $\bigcup_i \ell_i = \bigcup_j \ell'_j$ and, for each $\ell' \in \{\ell'_1, \ldots, \ell'_m\}$ and each $x_1, x_2 \in \ell'$, we have

$$\{\ell \in \{\ell_1, \ldots, \ell_n\} \mid x_1 \in \ell\} = \{\ell \in \{\ell_1, \ldots, \ell_n\} \mid x_2 \in \ell\}.$$

The intuition is, given a set of labels, to compute a partition of the observations spanned by those labels. This partition should be fine enough to separate the input labels from one another, in the sense that the set of corresponding input labels is constant across all observations in each output label. Such a partition is valuable because it enables us to 'drop down' from the level of sets to the level of individual observations, by selecting a REPRESENTATIVE from each of the output labels, without danger of missing any structure inherent to the input label set.

Algorithm 1: REFINELABELS($\ell_1, \ldots, \ell_n$)

```
$r \leftarrow \ell_1$
for $\ell \in \{\ell_2, \ldots, \ell_n\}$ do
    $r \leftarrow$ UNION($r, \{\ell\}$)
$R \leftarrow (r)$
for $\ell \in \{\ell_1, \ldots, \ell_n\}$ do
    $R' \leftarrow (\ )$
    for $r \in R$ do
        $R'$.append(INTERSECT($r, \ell$))
        $R'$.append(DIFFERENCE($r, \ell$))
    $R \leftarrow R'$
return $R$
```

Algorithm 1 shows how one can perform this operation in a general way, for any label space that supports the UNION, INTERSECTION, and DIFFERENCE operations. The algorithm starts with a single label representing the complete set of relevant observations, and then refines that partition using each of the input labels.

IV. NORMAL FORMS

This section introduces and analyzes several particular classes of p-filters. A defining feature of p-filters is their ability to represent nondeterminism, which occurs in the selection of an initial state from Q_0, in the transitions made in response to each observation (since the labels of out-edges at each state need not necessarily be disjoint), and in the selection of an output upon arrival at each new state. The various classes we consider all vary based on (i) how much of this generality is used in the presentation of the p-filter, and (ii) whether that nondeterminism impacts the behavior of the p-filter.

A. Single-outputting normal form

First, we recall the fact that Definition 2 allows each state to be labelled with a set of possible outputs, any of which may be selected each time the filter visits that state. What happens if each state has only a single output?

Definition 4. A p-filter $\mathrm{F} = \langle Q, Q_0, Y, \tau, C, c\rangle$ is *single-outputting* if $|c(q_k)| = 1$ for every reachable $q_k \in Q$. A state q_k is considered reachable if some sequence of observations in $Y^\star$ gives a sequence of states, starting from a $q_0 \in Q_0$, ending in q_k.

Having a singleton output set at each state, while a seemingly significant constraint, does not limit the expressivity of such filters. That is, every p-filter has an equivalent single-outputting presentation. For this reason, we occasionally refer this as *single-outputting normal form*, by analogy to the normal forms used in formal language theory or database design.

Algorithm 2 shows how to convert an arbitrary p-filter to an equivalent single-outputting filter. The algorithm makes duplicates of each state, one for each of its outputs, and those single outputs to each of these new states. If there had been a

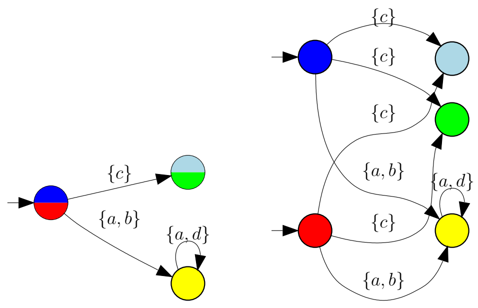


Fig. 3. [left] A p-filter that is not in single-outputting normal form. [right] The result of applying our implementation of Algorithm 2 to this filter. Outputs for each state are shown as colors.

transition from p to q on observation y, then in the new filter has y-transitions from all the states derived from p to all the states derived from q. Figure 3 illustrates this.

Algorithm 2: TOSINGLEOUTPUTTINGFORM(F)

```
Initialize $Q'$, $Q_0{}'$, $\tau'$, and $c'$ as empty
// Build states and output function:
for $q \in Q$ do
    for $c_i \in c(q)$ do
        Add $q_{c_i}$ to $Q'$, and let $c'(q_{c_i}) = c_i$
        if $q \in Q_0$ then
            Add $q_{c_i}$ to $Q_0'$
// Construct transition function:
for $(q, r, U) \in \tau$ do
    for $c_i \in c(q)$ do
        for $d_i \in c(r)$ do
            Let $\tau'(q_{c_i}, r_{d_i}) = U$
return $\langle Q', Q_0', Y, \tau', C, c' \rangle$
```

B. State-determined normal form

Another source of nondeterminism in Definition 2 arises from the transition model. Because the labels for each state need not be disjoint, the model allows multiple 'next' states to be indicated for the same observation. Definition 5 expresses this idea.

Definition 5. We call a p-filter $F = \langle Q, Q_0, Y, \tau, C, c \rangle$ *state-determined* if $|Q_0| = 1$, and for every triple of states $q_1, q_2, q_3 \in Q$ with $q_2 \neq q_3$, $\tau(q_1, q_2) \cap \tau(q_1, q_3) = \varnothing$.

In a state-determined filter, there are never any choices about which states the filter might be in; each observation string can be traced to at most one final state. As with single-outputting normal form, every p-filter is equivalent to a state-determined filter. We therefore adopt the terminology *state-determined normal form*.

Algorithm 3 shows how to convert an arbitrary p-filter into state-determined normal form. The idea is a forward search over sets of states, starting from the initial states. This requires the use of Algorithm 1 to ensure that the edges in the new filter are drawn correctly. See Figure 4.

C. Deterministic p-filters

Definitions 4 and 5 describe two distinct ways of presenting a filter, each of which places some restrictions on the kind of

Algorithm 3: TOSTATEDETERMINEDFORM(F)

```
Initialize $W$, $W_0$, $\tau'$, and $c'$ as empty
for $v_0 \in V_0$ do
    Add $v_0'$ to $W$ and $W_0$, and let $c'(v_0') = c(v_0)$
Initialize queue $Q \leftarrow V_0$
while $Q$ not empty do
    $s' \leftarrow Q$.pop
    // Refine each label and determine which states each
       refinement maps to:
    $L \leftarrow$ all outgoing edge labels of WHENCE($s'$)
    $L' \leftarrow$ REFINELABELS($L$) // cf. Algorithm 1
    $d_{\mathrm{Lab}}[.] = \varnothing$ // Empty the map
    for $l' \in L'$ do
        For every WHENCE($s'$) record which states you reach
        with REPRESENTATIVE($l'$) by adding them to $d_{\mathrm{Lab}}[l']$
    // Produce new states as needed:
    for $s \in d_{Lab}[l]$ for some $l$ do
        if $t \in W$, where $t$ corresponds with $s$ then
            Let $\tau'(s', t) = l$ // Add transition on $l$
        else
            Create new state $t$ corresponding to $s$
            Let $c'(t) = c(s)$ and add $t$ to $W$
            $Q$.push($t$) // Add to queue to be processed
            Let $\tau'(s', t) = l$ // Add transition on $l$
return $\langle W, W_0, Y, \tau', C, c' \rangle$
```

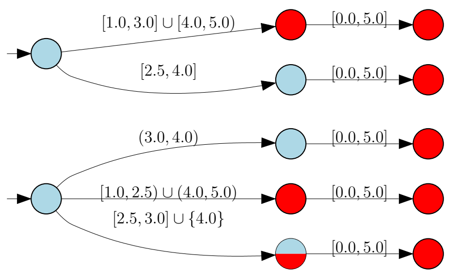


Fig. 4. [top] A p-filter that is not in state-determined normal form. [bottom] The result of applying our implementation of Algorithm 2 to this filter.

nondeterminism directly present in the filter. Note, however, that these two normal forms are, in a certain sense, duals of one another. Algorithm 2 may, in eliminating multi-output states, introduce some overlapping labels or multiple initial states; Algorithm 3 may, in eliminating overlapping labels and multiple initial states, introduce some multi-output states.

A certain class of p-filters, however, can be represented in a way that is simultaneously single-outputting and state-determined. We call these filters *deterministic*.

Definition 6. A p-filter $F = \langle Q, Q_0, Y, \tau, C, c \rangle$ is *deterministic* if every observation sequence in $Y^\star$ yields at most one output sequence in $C^\star$.

Note that, in this context, deterministic does not mean that each observation sequence determines a unique state, but only that each observation sequence, if it yields any output, yields a single, determined output. Figure 5 illustrates the difference. In that sense, the property of being deterministic is a property of the p-filter's *behavior*, rather than a property of its *representation*.

Deterministic p-filters are closely related to standard filters,

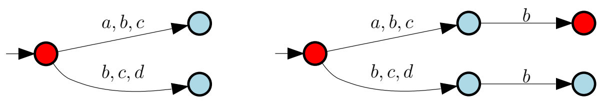


Fig. 5. [left] A p-filter that, in spite of some labels that are not disjoint, is a deterministic p-filter. [right] A p-filter that is not deterministic. Note that the observation sequence *bb* yields two distinct output sequences: red, blue, red; and red, blue, blue.

as is made clear in the following lemma.

Lemma 1. *For every deterministic p-filter* $\mathrm{F} = \langle Q, Q_0, Y, \tau, C, c\rangle$*, there exists a standard filter* $\mathrm{S} \in \mathcal{F}_{std}$ *where, for every* $y_1 \dots y_n \in Y^\star$, $[y_1 \dots y_n]_\mathrm{F} = \{c'_0 c'_1 \dots c'_n\}$*, the latter being the output of* S *on* $y_1 \dots y_n$.

Proof: One may construct a suitable standard filter $\mathrm{S} = \langle 2^Q - \{\varnothing\}, \{q_0\}, Y, \delta, C, c'\rangle$, defining δ by exploring each of the (finite) observation sequence prefixes that visit every Q in F, and labelling the transitions that are made. Tracing a prefix string on F might cause one to come to a choice point, where some observation y is both $\tau(q_i, q_j)$ and $\tau(q_i, q_k)$ and $q_j \neq q_k$: both choices should be taken in constructing δ, which is why the states in S are subsets of Q. All outputs along all choices must always produce the same output, o, otherwise F would not be a deterministic p-filter. Thus, c' maps to that o. ■

Thus, determining whether a given filter is deterministic is of direct interest in practice, since standard filters are those that are directly amenable to implementation. Fortunately, we can establish some relationships between the set of deterministic p-filters and the normal forms introduced above. The next three lemmas do this work, and Figure 6 illustrates the set relationships implied by these results.

Lemma 2. *Any p-filter that is both single-outputting and state-determined is deterministic.*

Proof: Following the procedure described in Lemma 1 with a single-outputting and state-determined filter never leads to any choices. Therefore, only singleton subsets of 2^Q are involved. Also, for all states, there is never any choice for the value that c' should provide either. As a result, any observation sequence can yield at most one output sequence. ■

Lemma 3. *All deterministic filters are single-outputting.*

Proof: Suppose $\mathrm{F} = \langle Q, Q_0, Y, \tau, C, c\rangle$ is a deterministic p-filter which is not single-outputting. There must be some sequence $y_1 y_2 \dots y_n \in Y^\star$ of observations causing F to arrive in some q_k where $|c(q_k)| \neq 1$. Since $\varnothing$ cannot be in the image of c, there must be at least two distinct elements of C. But then, a string ending in either element of $c(q_k)$ is in $[y_1 \dots y_n]_\mathrm{F}$, *reductio ad absurdum.* ■

Lemma 4. *A state-determined filter is deterministic if and only if it is single-outputting.*

Proof: For the forward direction, Lemma 2 suffices. The backward direction is implied by Lemma 3. ■

This (finally!) concludes the infrastructure necessary to evaluate sensor transformations.

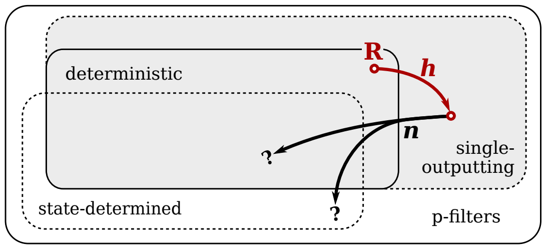


Fig. 6. A Venn diagram showing set inclusion properties for the filters and various representations studied herein. Deterministic filters, a strict subset of all p-filters, are essentially equivalent in behavior to the standard filters from the literature. Sensor maps take some filter (R in the picture) and transform it into another (as h does).

V. Sensor maps

A. Sensor maps

Having established some subclasses of p-filters and their relationship to standard filters, next we are interested in modifications produced by altering the fidelity of the observations provided as input to the filter.

We model degradations from the idealized model (and, indeed, other kinds of changes to the sensor's behavior) using sensor maps.

Definition 7. Given p-filter $\mathrm{F} = \langle Q, Q_0, Y, \tau, C, c\rangle$, a *sensor map from* Y *to* K, is a function $h : Y \to K$, where K is some other set. We say that h is concordant with F because its domain is the set of observations of the filter.

Though sensor maps are defined in terms of single observations, we can 'lift' a sensor map to apply to sets or to entire p-filters in the obvious way.

Definition 8. Given a sensor map $h : Y \to K$, its *extension to sets* (also denoted h, with the difference always clear from context) is a function $2^Y \to 2^K$, defined by $h(S) = \{h(s) | s \in S\}$. Likewise, the *extension to filters* of h, is a function $\mathcal{F}_\mathrm{p} \to \mathcal{F}_\mathrm{p}$, under which $\mathrm{F} = \langle Q, Q_0, Y, \tau, C, c\rangle$ maps to p-filter $h(\mathrm{F}) = \langle Q, Q_0, K, \tau', C, c\rangle$ in which $\tau'(q_1, q_2) = h(\tau(q_1, q_2))$.

The intuition is that a map h describes some alteration in the perceptual classes (*cf.* [2]) provided as input to the filter. If the outputs yielded under this transformation are ultimately the same, then we conclude that h, despite it potentially eliminating some information, must retain the "kernel" of information actually used in processing the input observations. Interestingly, it captures this notion of information abstractly, as none of the elements of Y need even be in K.

Note that the ability to compute $h(\mathrm{F})$ given h and F depends on the ability of our label space to efficiently compute the extension of h to sets, that is, to the labels of F.

Consider the following illustrative examples of how sensor maps work.

Example 6. Your robot is equipped with a camera, and triplets of red–green–blue values within an array comprise Y. Now imagine that rose-tinted lenses are placed over the camera.

Applied pixel-wise, $h_{\text{rose}} : \langle r, g, b\rangle \mapsto \langle r, 0, 0\rangle$. Certain scenes that produce distinct inputs, $y_1 \neq y_2$, may now be indistinguishable under the transformation, $h_{\text{rose}}(y_1) = h_{\text{rose}}(y_2)$, when, for example, two scenes differ only in elements of the spectrum filtered out by the lenses.

Example 7. Sensor maps need not only reduce the set. Suppose your sensor incurs cross-talk due to poor cable routing and cheap shielding. Where formerly a given circumstance would produce an observation y_i, this might be modeled with a sensor map $y_i \mapsto \{y_i, y_i', y_i''\}$. It may be that $y_i' \in Y$, or it might be some heretofore unseen class of signal. What we are interested in is whether this cross-talk is destructive or not. As is clear, the answer to this depends on whether some other y_j where $y_i' \in h(y_j)$ exists. Even existence of such a y_j is insufficient, as y_i and y_j might occur in every pre-image together.

Next, we formalize the notion of a destructive sensor map, which is based on a generalized notion of equivalence between p-filters.

Definition 9. Given two p-filters $\mathrm{F} = \langle Q, Q_0, Y, \tau, C, c\rangle$ and $\mathrm{G} = \langle R, R_0, Z, \upsilon, D, d\rangle$ and a sensor map $h : Y \to Z$ mapping from the observation space of F to the observation space of G, we say that F *is equivalent to* G *modulo* h, denoted

$$\mathrm{F} \geqq \mathrm{G} \mod h,$$

if for every observation sequence $y_1 \dots y_n \in Y^\star$,

$$[y_1 \dots y_n]_{\mathrm{F}} = [h(y_1) \dots h(y_n)]_{\mathrm{G}}.$$

Note that we eschew the traditional equivalence symbol '≡' for this relation because it is not symmetric: $\mathrm{F} \geqq \mathrm{G} \mod h \not\Rightarrow \mathrm{G} \geqq \mathrm{F} \mod h$. The intuition is that if G, given observations mutated by h, exhibits the same behavior that F exhibits when given those same observations, but unmutated, then any difference between F and G is merely in the change in manifestation of the observations induced by h; the underlying structure is the same. In contrast, if the two filters can generate different outputs under these conditions, then there must be some other explanation for those differences. This observation motivates the idea of a nondestructive sensor map.

Definition 10. Given a p-filter $\mathrm{F} = \langle Q, Q_0, Y, \tau, C, c\rangle$ and a concordant sensor map $h : Y \to K$, we say that h is *non-destructive* if $\mathrm{F} \geqq \mathrm{h(F)} \mod h$.

Informally, a nondestructive filter is one that preserves enough structure that the filter still works after applying it, as long as the labels are updated accordingly. A destructive filter is one that creates enough ambiguity (initially expressed in the resulting p-filter by states with overlapping out-edges) that the correct outputs can no longer be determined solely by the observations.

Example 8. Suppose $h : Y \to K$ is an injective map, so that if $h(y) = h(z)$, then $y = z$. Because this kind of map does not introduce the possibility of conflating any two observations, it is clear that h is non-destructive. In the particular case of interval labels (recall Example 3), this implies that any sensor map that is a strictly-increasing or strictly-decreasing—including, for example, affine maps—, is non-destructive. Contrapositivelty, we can also conclude that every destructive sensor map is non-injective.

B. Deciding destructiveness

We can now address the algorithmic problem posed by the examples in Section I.

Decision Problem: Sensor map destructive test (SMDT)
Input: A p-filter F and a concordant sensor map h
Output: TRUE if h is non-destructive on F, or FALSE otherwise.

Our algorithmic approach to solving this problem depends on whether the input filter F is deterministic.

If we have an instance (F, h) of SMDT in which F is deterministic —a case that should be quite common, since deterministic filters are those that are most directly implementable— then we can use Algorithm 3 along with Lemma 4 to determine whether h is destructive. The intuition is to compute $h(F)$, then convert that mapped filter to state-determined normal form and check whether the result is also in single-outputting normal form. See Algorithm 4.

Algorithm 4: SENSORMAPDESTRUCTIVETEST1(F, h)

```
G ← TOSTATEDETERMINEDFORM($h$(F))
return ISSINGLEOUTPUTTING(G)
```

The algorithm itself is strikingly simple, which we view as a positive feature. The groundwork from the earlier portions of this paper enable a complex question like SMDT to be resolved in a compact and elegant way.

To solve instances of (F, h) of SMDT in which F is not deterministic, the situation is somewhat more complex, because we must check explicitly whether $F \geqq h(F) \mod F$. Algorithm 5 shows how to perform this check. After converting to state-determined normal form, if necessary, the algorithm uses a forward search over pairs of states, one from each filter, that are reachable by some observation sequence. For each such pair, we verify that the output colors specified by each filter are the same. For full generality, we show the algorithm for arbitrary pairs of filters, not just for an F and its $h(F)$.

VI. HARDNESS OF SENSOR MINIMIZATION

The treatment of sensor maps in Section V raises the question of why it is of interest to consider a variety of sensor maps. Cannot one instead simply find the sensor map that is, in some sense, the 'most aggressive' nondestructive map for a given filter? In this section, we present a hardness result establishing that, unless $P = NP$, no efficient algorithm can find the nondestructive sensor map of minimal image size for a given filter, even approximately. Specifically, we consider the following decision problem.

Algorithm 5: EQUIVALENCEMODULOMAP($\mathrm{F}_1, \mathrm{F}_2, h$)

```
Convert $\mathrm{F}_1$, $\mathrm{F}_2$ to state determined form if needed.
Initialize queue $Q \leftarrow U_0 \times V_0$
while $Q$ is not empty do
    $(s_1, s_2) \leftarrow Q$.pop
    if $c(s_1) \neq d(s_2)$ then
        return False // Output sets are not equal
    $U_1 \leftarrow$ REFINELABELS(labels leaving $s_1$)
    $U_2 \leftarrow$ REFINELABELS(labels leaving $s_2$)
    $U_2' \leftarrow$ { pre-image of each element of $U_2$ under $h$}
    $L \leftarrow$ REPRESENTATIVES($U_1 \cup U_2'$)
    for $l \in L$ do
        $s_1' \leftarrow$ state that $\mathrm{F}_1$ transitions to on $l$
        $s_2' \leftarrow$ state that $\mathrm{F}_2$ transitions to on $h(l)$
        $Q$.push($(s_1', s_2')$) // To be processed
return True
```

Decision Problem: Sensor minimization (SM)

Input: A p-filter $\mathrm{F} = \langle Q, Q_0, Y, \tau, C, c\rangle$ and integer n.
Output: TRUE if there exists a set K and a sensor map $h : Y \to K$, nondestructive for F, with $|K| \leq n$. FALSE otherwise.

Theorem 5. SM *is NP-hard.*

Proof: Reduction from 3-coloring. A detailed proof, omitted here due to space limits, appears as a supplemental document. ☐

Note, *a fortiori*, that the proof of Theorem 5 does not depend any essential way on the specific number 3. In fact the chromatic number of the graph coloring instance and the image size of the smallest nondestructive sensor map for the corresponding filter are always equal. Combined with known results on the inapproximability of chromatic numbers [20], this leads directly to the following stronger result.

Corollary 6. *The optimization problem of finding, for a given filter, the nondestructive sensor map with the smallest image size, is NP-hard to approximate to within* $n^{1-\epsilon}$.

VII. CASE STUDY: MINIMIZING THE ROOMBA

The following simple scenario, of the sort that the authors have often assigned in introductory robotics courses, illustrates the utility of the machinery developed in this paper. We wish to have an iRobot Roomba vacuum cleaning robot follow walls (on its port side) while avoiding negative obstacles. Five range sensors on the robot provide sufficient information to carry out this basic task. We approach this problem by constructing a filter whose outputs are actions for the robot, and then we are able to analyze the effect of sensor maps on this filter with our implementation of the algorithms described in the earlier sections of the paper.

We begin by describing the set of observations for idealized versions of the robot's sensors. Each of $\{w, c_1, c_2, c_3, c_4\}$ is fundamentally a device that measures distance, so it is useful to model each output with a real number that represents the range reading; naturally, the product of these five sensors gives a label space with $\mathbb{R}^5$. Each state in the filter produces an output that is interpreted as velocity commands—linear as $\dot{x}$ and angular as $\dot{\theta}$. The filter is shown pictorially in Figure 8.

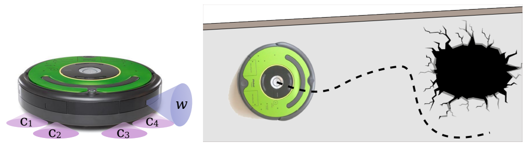
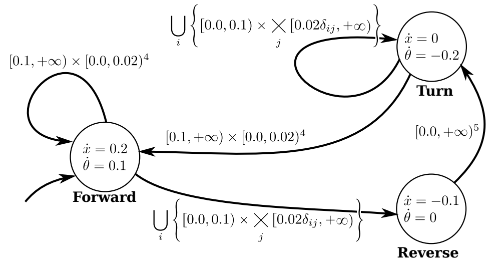


Fig. 7. An iRobot Roomba is equipped with a collection of simple sensors including four cliff sensors and a wall sensor, each uses IR to measure distance. As a simple example, we consider a filter which maps sensor readings into motor commands on a robot tasked with following a wall on its left, while avoiding negative obstacles.



$\bigcup_i \left\{[0.0, 0.1) \times \bigtimes_j [0.02\delta_{ij}, +\infty)\right\}$
$\dot{x} = 0$
$\dot{\theta} = -0.2$
Turn
$[0.1, +\infty) \times [0.0, 0.02)^4$
$[0.1, +\infty) \times [0.0, 0.02)^4$
$[0.0, +\infty)^5$
$\dot{x} = 0.2$
$\dot{\theta} = 0.1$
Forward
$\bigcup_i \left\{[0.0, 0.1) \times \bigtimes_j [0.02\delta_{ij}, +\infty)\right\}$
$\dot{x} = -0.1$
$\dot{\theta} = 0$
Reverse

Fig. 8. A visual representation of the filter (IDEAL) that solves navigation problem for the Roomba where edge labels are subsets of $\mathbb{R}^5$, and the values in each vertex are velocities that the robot executes for some small finite time. (The δ_{ij} is the Kronecker delta, $i, j \in \{1, 2, 3, 4\}$.)

From this a series of filters are constructed via transformations that coarsen the label space. Sensor map h (detailed below) applied to the IDEAL filter, gives a filter CREATE (SIGNALS), whose labels are based on the data that can be read from the physical sensors through the software interface (see [6]). Sensor map f (also below) transforms CREATE (SIGNALS) into CREATE (SYMBOLS) representing a second level of abstraction —in this case, a quantization based on thresholding— available through the robot's hardware interface. Map g further reduces the set of labels, while the final map we define, k, is destructive. Table I collects this information. The rows in the table also summarize the relationships visually. Starting from IDEAL one produces the others via composition of the sensor maps, for example, COMBINED SENSOR results from applying map the $g \circ f \circ h$. The conclusion that only the final map to a sensorless model is destructive, shows that for this filter, neither the specific distance measurements nor the individual identities of the sensors themselves are necessary. A robot designed exclusively for this task could, therefore, likely be designed to be simpler than a Create.

Name	**Observation Space**	**Notes**
IDEAL	$\mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R}$	
$\downarrow h =$ **clipToRange**$(\cdot)$		
CREATE (SIGNALS)	$[0, 1023] \times [0, 4095]^4$	Ranges from [6, pg.27].
$\downarrow f =$ **threshold**$_T(\cdot)$	($T = 10$ and 20 for w and c_i resp.)	
CREATE (SYMBOLS)	$\{0, 1\} \times \{0, 1\}^4$	*cf.* [6, pgs.22–23].
$\downarrow g =$ **min**$(\cdot)$		
COMBINED SENSOR	$\{0, 1\}$	
$\downarrow k = 0$	(Constant map)	
SENSORLESS	$\{0\}$	(Destructive)

TABLE I. A HIERARCHY OF FILTERS FOR THE IROBOT CREATE.

References

[1] G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localisation and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, 2001.

[2] B. R. Donald and J. Jennings. Sensor interpretation and task-directed planning using perceptual equivalence classes. In *Proc. IEEE International Conference on Robotics and Automation*, pages 190–197, Sacramento, CA, 1991.

[3] M. Erdmann and M. T. Mason. An Exploration of Sensorless Manipulation. *IEEE Transactions on Robotics and Automation*, 4(4):369–379, August 1988.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[5] K. Y. Goldberg. Orienting Polygonal Parts Without Sensors. *Algorithmica*, 10:201–225, 1993.

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

[7] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[8] S. Kristek and D. A. Shell. Orienting Deformable Polygonal Parts without Sensors. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robot Systems (IROS)*, 2012.

[9] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[10] S. M. LaValle. Sensing and Filtering: A Fresh Perspective Based on Preimages and Information Spaces. *Foundations and Trends in Robotics*, 1(4):253–372, 2010.

[11] R. Lopez-Padilla, R. Murrieta-Cid, and S. M. LaValle. Optimal gap navigation for a disc robot. In *Proc. Workshop on the Algorithmic Foundations of Robotics*, 2012.

[12] J. M. O'Kane. Decentralized Tracking of Indistinguishable Targets using Low-Resolution Sensors. In *Proc. International Conference on Robotics and Automation*, 2011.

[13] J. M. O'Kane and D. A. Shell. Automatic Reduction of Combinatorial Filters. In *Proc. IEEE International Conference on Robotics and Automation*, 2013.

[14] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In I.J. Cox and G.T. Wilfong, editors, *Autonomous Robot Vehicles*, pages 167–193. Springer, Berlin, Heidelberg, 1990.

[15] Y. Song and J. M. O'Kane. Comparison of Constrained Geometric Approximation Strategies for Planar Information States. In *Proc. International Conference on Robotics and Automation*, 2012.

[16] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA., 2005.

[17] B. Tovar, R. Murrieta-Cid, and S. M. LaValle. Distance-optimal navigation in an unknown environment without sensing distances. *IEEE Transactions on Robotics*, 23(3): 506–518, June 2007.

[18] M. Veanes, P. Hooimeijer, B. Livshits, D. Molnar, and N. Bjorner. Symbolic finite state transducers: algorithms and applications. In *Proc. of ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'12)*, pages 137–150, New York, NY, USA, 2012.

[19] J. Yu and S. M. LaValle. Shadow Information Spaces: Combinatorial Filters for Tracking Targets. *IEEE Transactions on Robotics*, 28(2):440–456, 2012.

[20] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3:103–128, August 2007.