Information Spaces for Mobile Robots

Benjamín Tovar Anna Yershova Jason M. O'Kane Steven M. LaValle
Dept. of Computer Science
University of Illinois
Urbana-Champaign, IL 61801. USA
[btovar,yershova,jokane,lavalle]@uiuc.edu

Abstract

Planning with sensing uncertainty is central to robotics. Often, sensor limitations prevent accurate state estimation of the robot. Two general approaches can be taken for solving robotics tasks given sensing uncertainty. The first approach is to estimate the state and to solve the given task using the estimation as the real state. However, estimation of the state may sometimes be harder than solving the original task. The other approach is to avoid estimation of the state, which can be studied by defining the information space, the space of all histories of actions and sensing observations of a robot system. Considering information spaces brings better understanding of the problems involving uncertainty, and also allows finding better solutions to such problems. In this paper we give a brief description of the information space framework, followed by its use in some robotic tasks.

1 Introduction

Often robots have to plan and execute tasks while being uncertain about their state and the environment in which they are acting. From a robotic perspective, the *state of a robot system*, or simply the *state*, represents the information that together with the control input, fully specifies the situation of the robot system. It refers to the position in space, velocities in joints or wheels, levels of energy consumption, etc. Classical approaches for robot planning assume a perfect knowledge of the robot state. Such perfect knowledge is virtually unattainable, given noisy readings from the available sensors and limitations on the number of sensors the robot can have. Therefore, some crucial information may simply be unavailable to the robot (for example, the information about the orientation is not available to the robot without a compass). Therefore, many further research efforts have been focused on the estimation of the state. If such estimations are reliable, they can be considered as the *true* robot state, forgetting that there is uncertainty in the state information. In control theory, for example, the concept of an *observer* is well understood [2], and if the observer converges sufficiently fast, the value of the state variables of the observer is taken as the value of the state variables of the system. In mobile robotics, *simultaneous localization and mapping* (SLAM) approaches have received considerable attention in recent years [3, 21, 25]. The goal of the SLAM approaches is to correctly estimate the current state of the robot. However, an interesting approach is to avoid the state estimation all together. In fact, the necessity for the knowledge of the robot state can be considered as an artifact of a planning algorithm. While knowing the state of the robot is *sufficient* to solve a task, it may not be *necessary*. In other words, a robot may not know its current state, and still be able to solve a specific task. It has been shown in numerous robotics works that robots can efficiently solve complicated tasks with no estimation of its current state. Much of the early work in this direction was in the context of object manipulation [9, 20]. Other work includes information invariants [5], sensor design [8], bug algorithms for navigation [19, 13], robot localization [6], POMDPs [16], and error detection and recovery [4].

All of these works present seemingly different approaches for solving given robotics tasks. In this paper we describe a framework based on *information spaces* that generalizes planning strategies for robotic systems with sensing uncertainty. For this we first formulate the general planning problem presented to the robot. This usually includes: the *state space*, i.e. the set of states where the robot can be; the *action space*, i.e. the set of actions that the robot can perform; the *observation space*, i.e. the set of observations that is available to the robot from sensors; *sensor mappings*, which produce an observation for each state of the robot; *state transition function*, which produces a state for each action; and the *goal*, which is expressed in terms of the

histories of actions and observations.
To achieve the goal defined by the planning problem above, a broader definition of *robot state* should be used. Such states, called the *information states*, and the space where they live, the *information space* allow to express planning problems where tasks with partial information naturally live. In this paper we explore some results for robot planning in the information space framework. There are many exciting open research problems with information spaces. It is our hope that this paper will stimulate further research in analyzing the information spaces for robotics systems and bring more efficient strategies for solving robotics tasks.

2 Preliminaries

In the following discussion, let X denote the *state space*, and let $U(x)$ the set of actions available to the robot from state $x \in X$. At each stage k, it is assumed that a nature action θ_k is chosen from a set $\Theta(x_k, u_k)$, given the current state of the robot $x_k \in X$, and the action executed $u_k \in U(x_k)$. The role of $\Theta(x_k, u_k)$ is to model events the robot cannot control. For example, it can model control execution inaccuracies, unpredictable changes in a dynamic environment, etc. Let f be the *state transition equation*, that produces a state, $f(x, u, \theta)$ for every $x \in X$, $u \in U$ and $\theta \in \Theta(x, u)$. Note that f is not known for every planning problem. For simplicity of presentation, we assume that time is discrete. The continuous time case is developed in [17]. For a more extensive description see [17, 23]. A robot may retrieve information regarding its state from three sources, *initial conditions*, *sensor observations* and *actions executed*:

- **Initial conditions.** The initial conditions refer to all the information the robot is given prior to the planning task. For example, the initial state $x_1 \in X$ may be given, or the initial state may lie in a given subset $X_1 \subset X$. Also, the initial state may follow a given probability distribution $P(x_1)$ over X. Note that the calibration of a robotic system is considered in the previous cases. If all the state variables are calibrated, then the initial state is known. On the contrary, if only some state variables are calibrated, this correspond to the case when a subset X_1 is given. The initial conditions will be denoted with η_0.

- **Sensor observations.** A sensor is a device that provides some measurement of the current state. Thus, a sensor observation provides measurements of the state during execution. Formally, let Y denote the *observation space*, and let h denote a *sensor mapping.* If given the state, the observation is completely determined, h takes the form $h : X \to Y$. Other important case is when nature interferes with the observation. In this case the mapping takes the form $y = h(x, \phi) \in Y$, with $\phi \in \Phi$, in which $\Phi(x)$ is the set of nature sensing actions defined for each $x \in X$. Finally, the observation may also depend on previous states, in which the mapping for the k observation takes the form $y_k = h(x_1, ..., x_k, \phi_k)$.

- **Actions executed.** An action executed may provide valuable information regarding the robot state. For example, in the absence of control errors, if the robot is commanded to move one meter east, it is known that the robot is now one meter further east than before.

2.1 The Information Space

The information available to the robot when the plan is at stage k should be determined either from the new observations, or the accumulation of previous information. It is assumed that the robot keeps a record of each of the observations made. Thus, the *observation history*, $\tilde{y} = (y_1, y_2, ..., y_k)$, is the ordered sequence of observations up to stage k. Similarly, the *action history*, $\tilde{u} = (u_1, u_2, ..., u_{k-1})$, is the record of the actions taken. It runs until $k-1$, because action u_{k-1} is applied in state x_{k-1}, to yield the current state x_k, where the observation y_k is made. The *information state* at stage k is defined as

$$\eta_k = (\eta_0, \tilde{u}_{k-1}, \tilde{y}_k),$$

that is, the initial condition together with the history. Alternatively, an information state can be expressed recursively as

$$\eta_k = (\eta_{k-1}, u_{k-1}, y_k),$$

since the difference between the previous and the current information state consists of the new observation made and the new action taken. The set of all possible information states is called the *information space*, $\mathcal{I}$. Usually we do not deal with $\mathcal{I}$ directly, given that the size of an information state grows linearly in time, and this becomes intractable very fast. Thus, we have to look for methods that collapse the information space. One simple method for collapsing the information space is based on the inferences that can be done given an information state. If the information state η_k is available, it is possible to compute the set $X_k(\eta_k)$ in which the actual x_k is known to lie. The set $X_k(\eta_k)$ is called a *derived information state.* To compute the derived information state, we have to infer over the observations and actions performed. For the observations, we can define

$$H(y) = \{x \,|\, y = h(x, \psi), \text{for } \psi \in \Psi(x)\}$$

that is, the set of all possible states the robot may be in given an observation. The set $H(y)$ is called the *preimage* of y. Similarly, if we let the actions available depend on the current state, the robot can determine a set of states V where it may be, by computing:

$$V(U_k) = \{x' \,|\, U_k = U(x') \text{ for } x' \in X\}$$

in which U_k are the actions available at stage k. The current state then lies in the set $H \cap V$. Note, however, that it can be assumed that the robot has some kind of sensor that detects which kind of actions are available. This reduces the computation of V and H into only the computation of H. Thus, only the case when U is fixed for all $x \in X$ is important.
From the state transition equation, it is possible to know which states may be reached if action u is applied at state x. Let F be this set, formally defined as

$$F(x, u) = \{x' \in X \,|\, \exists \theta \in \Theta(x, u) \\ \text{for which } x' = f(x, u, \theta)\}$$

If we further assume that X is countably infinite, the derived information state $X_k(\eta_k)$ can be computed using induction. Note that F and H eliminate the direct appearance of nature actions. The base case ($k = 1$) of the induction is

$$X_1 = \eta_0 \cap H(y_1).$$

This first step consists only of making the initial condition consistent with the first observation. Now assume inductively that $X_k(\eta_k) \subseteq X$ is available, and $X_{k+1}(\eta_{k+1})$ should be computed. First note that $\eta_{k+1} = (\eta_k, u_k, y_{k+1})$, and the new information is provided only by u_k and y_{k+1}. From eq. 1, the state is anywhere in $H(y_{k+1})$. On the other hand, if x_k was known, after applying u_k, the state lies somewhere in $F(x_k, u_k)$. Since x_k is unknown, but it is known that $x_k \in X_k(\eta_k)$, the new derived information state is

$$X_{k+1}(\eta_k, u_k, y_{k+1}) = \bigcup_{x_k \in X_k(\eta_k)} F(x_k, u_k) \cap H(y_{k+1}).$$

Given that the derived information state is always a subset of X, the *derived information space* denoted by $\mathcal{I}^\circ$, can be defined as $\mathcal{I}^\circ = 2^X$. Note that if X is finite, $\mathcal{I}^\circ$ is also finite, which makes it preferable if the number of stages is much larger than the size of X.

3 Examples of Information Spaces

In this section we present several examples in which the state is unknown, and the concept of information space comes naturally. We do not intent to give a full range of applications, rather, the examples are related to the previous work we developed. As we said in the introduction, we hope for an increased interest in information spaces, since they offer an exciting point of view from which robotic problems can be analyzed.

3.1 Visibility-based Pursuit-Evasion

In the pursuit-evasion problem, a robot, called the *pursuer*, has to move in such a way that it could find another robot, called the *evader*. In a complete antagonistic setting, the evader does not want to be found, and can move arbitrarily fast compared to the pursuer. Assume that the pursuer has a map of the environment, and it is perfectly localized with respect to this map. How the pursuer should plan its movements in order to find all of the evaders? The answer depends on which sensors are available to the pursuer. Since the pursuer does not know where the evaders are, we can provide the pursuer with an ideal sensor called the *evader locator*, which when used, will tell the location of the evaders to the pursuer. While this is a valid formulation of the pursuit-evasion problem, its solution is trivial, given that we provided the pursuer with perfect information of the state of the task. Thus, a more interesting formulation considers providing the robot with sensors that report robot only *local* information. For example, providing the pursuer a camera, can only tell if an evader is present in the current visible region, or not. Such setting of the pursuit-evasion problem was presented in [10], and we describe it here from the information space framework.
Formally, assume that the pursuer moves in a connected open set $R \subset \mathbb{R}^2$. The boundary of R, ∂R is assumed to be polygonal and simply-connected. The evader is modeled as a moving point in R. The evader position $e(t)$ at time t is determined by a continuous position function $e : [0, \infty) \mapsto R$. The pursuer is also modeled as a point, with position $p(t)$. The pursuer has an exact geometric representation of R, and it is perfectly localized with respect to R. The pursuer also has a *visibility sensor*, which returns the visibility region from its current position. For a point $q \in R$, the

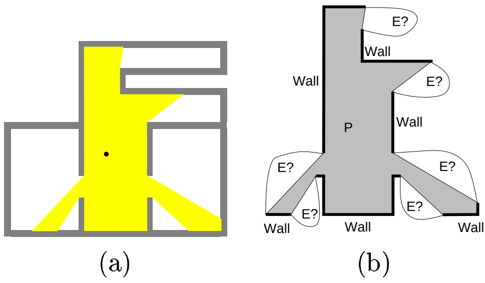


Figure 1: Each shadow region is a portion of the environment that may or may not contain the evader.

visibility region $V(q)$ includes all the points in R that can be joined with q through a line segment without intersecting δR. The task is to find a path $p : [0,1] \mapsto R$ for the pursuer such that the evader is guaranteed to be detected, regardless of its position function $e(t)$, which is unknown to the pursuer.

The state yields the position of the pursuer and evader, $x = (p, e)$, which results in the state space $X \subset \mathbb{R}^2 \times \mathbb{R}^2 = \mathbb{R}^4$. Since the position of the evader is unknown, the state is unknown. The observation space Y, is a collection of subsets of R. For each $q \in R$, the sensor yields a visibility polygon $V(p) \subset R$. Consider the information state at time t. For the initial condition, $p(0)$ is given and the evader may lie anywhere in R. The input history $\tilde{u}_t$, can be expressed as the position function of the pursuer. Thus, the information state is defined as:

$$\eta_t = ((p(0), P), p(t), \tilde{y}_t).$$

Since the pursuer position is always known, the interesting part is the subset of R in which the evader may lie. Thus, the derived information state can be expressed as $X_t(\eta_t) = (p(t), E(\eta_t))$, in which $E(\eta_t))$ is the subset of R that is known to contain the evader, given η_t.

The visibility region divides P in several *shadow regions.* That is, regions that are not visible to the robot (Figure 1). When an evader may be hidden in one of these regions, the region is said to be *contaminated*, otherwise it is said to be *cleared.* As the pursuer moves, the shadow regions appear, disappear, merge or split. Such events, called *visual events*, are produced by combinatorial changes in the visibility region. The visual events provide the only way to vary $E(\eta_t)$. For example, if a shadow region disappear, it means that the given region is now visible to the pursuer, and thus does not contain the evader. Also, if a contaminated region merges with a cleared one, the new region should be labeled contaminated, etc. The visual events induce a decomposition on P, called the *aspect graph*[14], or the *visibility-cell decomposition*[11]. In these decomposition, if the robot moves inside a cell there is not significant change in information. The robot receives about the same information from the sensors. Such movements are called *conservative* in the sense that they preserve the current robot's information. In contrast, when the robot crosses one of the cells' boundary edges, the structure of the visibility region suffers a drastic change, and the robot's information may be modified [7]. In these case there are two kinds of visual events. One kind is triggered when the robot crosses an environment's boundary generalized inflection ray, and the other when it crosses the complement of bitangent line segments of the boundary. An *inflection* is a change in the sign of the curvature of the environment's boundary. We use the term "generalized," as in [18], to include polygonal boundaries. Given a generalized inflection, an *inflection ray* is found by extending a ray from the inflection until it hits another point of the environment's boundary. A bitangent line segment is a segment completely contained in the environment representation, whose supporting line is tangent to two points of the boundary, and whose endpoints are these points of tangency. A common general position assumption is that no line is tangent to more than two points of the boundary (thus the term *bitangent*). For each bitangent, its *complement* is found by extending outward from each point of tangency until the environment's boundary is hit again (see Figure 3).

With these decomposition we can collapse the information space even further. It can be proved [11] that each of the cells produced are convex. Thus, if the pursuer is inside a cell, it can detect if the evader is also inside or not. Further, it can compute which cells are cleared, or become recontaminated when moving from one cell to the other. The state space is now discrete, since the exact position of the pursuer and evader is not relevant anymore, only whether or not they are inside a given cell. We can encode $E(\eta_k)$ as binary vector, with a label for each cell indicating if it is cleared or not. With this, the solution plan $p(t)$ can be found using a simple search in the derived information space [10].

3.2 Visibility-based tasks with Gap Navigation Trees

In the previous example, in principle, the planning strategy used the exact geometry of both visibility regions and the environment. However, we are interested in such information as an intermediate product, since it is only important if a certain region is cleared, or if it becomes recontaminated by merging with other regions. Thus, we are not interested in the exact description of the visibility regions, but in how they change. As presented in [12, 26, 27], we can further collapse the information state by designing a sensor that detects the combinatorial changes in the visibility region. Furthermore, we can eliminate the need of a map, and the robot can solve some visibility-based tasks in unknown environments.

A visibility region is composed of edges completely contained in the environment boundary, and of edges collinear with the position of the robot. The later are called *spurious edges*. When a spurious edge either appears, disappears, splits or merges with another, a combinatorial change in the visibility region occurs. From the robot perspective, the spurious edges are the discontinuities in depth information in the environment. Note that geometric information of the spurious edges is not relevant for the visual event detection. The events will be the same in spite of the exact length and angular position of the spurious edges. Their order is relevant, though, since we are interested in which discontinuity disappeared, or merged with another, for example. Although the precise distances to the walls may be unknown, the robot only needs a type of edge detector that can detect each of the discontinuities, and return their order relative to the robot's heading. Each of this discontinuities is referred to as a *gap*, and the sensor as a *gap sensor* [24]. As it was shown in [27, 12, 26], a robot using a gap sensor, with no other sensing ability is assumed, i.e., it has neither a compass nor a reliable odometer, can compute shortest paths information for unknown environments, localize itself and perform pursuit-evasion. The ideal gap sensor can be easily realized through a range sensor (i.e., laser or sonar) or using computer vision techniques.

Each gap *hides* a connected region of the environment that is occluded to the robot from its current position. A label of "L" or "R" is assigned to a gap to indicate the direction of the part of R that is hidden behind the gap. This corresponds to transitions of the gap sensor from "far to near" (left) or "near to far" (right), if the gaps are detected by a counterclockwise scan with respect to the robot's heading (see Figure 2.(a)).

When the robot moves in the environment, the gaps, as reported by the gap sensor, may change. It is assumed that the robot can track the gaps at all times and record any topological change. There are four possible ways in which gaps change:

Gap appearance. A gap, not detected before, is now tracked by the gap sensor. The gap is said to be visible.

Gap disappearance. A gap is no longer detected by the gap sensor. The given gap is not visible for the gap sensor.

Gaps merge. Several gaps merge into a single one.

Gap split. One gap splits into several gaps.

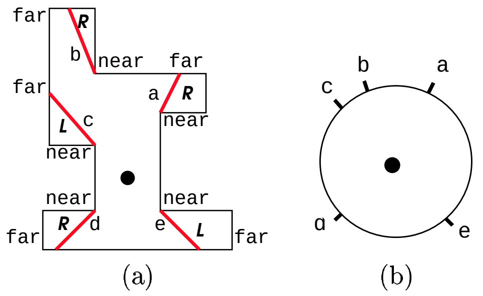


Figure 2: The robot's view of the environment. The position of the robot is shown with a black disk. (a) The environment and the respective labeling of the gaps detected. (b) Angular position of the gaps detected in the visibility region.

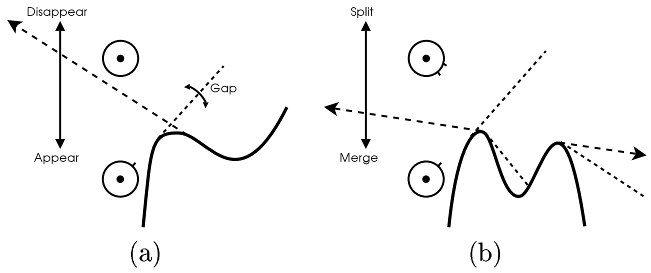


Figure 3: Inflections and bitangents of ∂F. (a) Appearance and disappearance of gaps occur when the robot crosses inflection rays. (b) Splits and merge occur by crossing bitangent complements.

If a gap appears, the region behind it was just visible to the robot, but now is "hidden" by the gap. Similarly, when a gap disappears, the region of the environment behind the gap is now visible to the robot. With bitangents, exactly two gaps may merge into one, and one gap splits exactly into two gaps. These four gap topological changes are called the *gap critical events*.

Appearances and disappearances of gaps are related to generalized inflections of ∂R. As illustrated in Figure 3 (a), appearances and disappearances of gaps occur when the robot crosses inflection rays. Merges and splits of gaps, are related to the bitangents of ∂R, and they occur when the robot crosses bitangent complements. (Figure 3 (b)). Note that R need not be a polygon, but may be any piecewise-analytic closed curve.

In this sensing model, the observation space Y is defined by the set of all of the ordered circular sequences of possible readings of gaps. Thus, $\{L, L, R\} \in Y$ correspond to a sensor reading where two "left" gaps and a "right" gap are detected. Note this sensor reading is indistinguishable from $\{R, L, L\}$ and $\{L, R, L\}$, since a compass is not available. Even more, with only gap readings, the exact position of the robot cannot be determined, and different neighborhoods of points will generate the same sensor reading across the whole environment. The input space is determined by the gap chasing movements, that is, the commands to the robot to move towards a gap.

3.2.1 Encoding information states

Remember that the robot can track the gaps all of the time and record any of their topological changes. Thus, it can detect that from the transition $\{L_1, R_1, R_2, L_2\}$ to $\{L_1, R_2, L_2\}$, the gap R_1 disappeared. The gap sensor only will report to the robot that a gap, detected before in this order, disappeared, for example. This identification of gaps is implicit at the sensor level, and it is possible if we assume coherency between the robot's motion and gap changes (i.e., small position changes of the robot will produce small angular position changes in the gaps). The gaps and their topological changes are encoded into a tree, hereafter referred to as T. The tree T is the *Gap Navigation Tree* of the environment. The root of T moves along with the robot. Each child of the root represents a gap that is currently visible, and they are maintained in the circular order of the gaps they represent. In T, we will use the terms gaps and nodes interchangeably, because except the root, each node encodes a gap.
As the robot moves, critical events are triggered. As events occur, T is updated as follows: if a gap disappears, the corresponding node is removed from T. If a gap appears, it is added as a child of the root of T in a location that preserves the circular ordering of gaps. If a gap splits, then the corresponding child of the root will be replaced with two children. If two gaps merge, the two corresponding children of the root become the children of a new node, d, and d becomes a child of the root.
The relation of T with an information state is immediate. In fact, T is nothing more than the sensor history of the current information state. For example, assume that at time t_1 the state of T is T_1. At t_1 the robot is commanded to chase the sequence of gaps α, which brings T into the state T_2. Comparing T_1 and T_2 we can readily obtain the sequence α followed (input history), with the respective changes as reported by the gap sensor (sensor history). Note, however, that we are assuming that α is the shortest sequence of gaps to take T_1 into T_2. In this sense, all sequences of gaps that take T from one state to another are *equivalent* to the shortest one, since at the end, they modify T in the same way.
There is a very close relation between the visibility graph and the Gap Navigation Tree. Once the GNT is known for an environment, it can be shown that the robot will follow optimal paths in distance, even though no distance information was ever measured[27]. Adding cleared and contaminated labels to the gaps, pursuit-evasion in the absence of a map can also be solved [12]. One interesting observation is that the GNT induces an equivalence relation in the set of environments with piecewise-analytic closed curves boundaries. For example, all the environments in Figure 4 have the same family of GNTs. This means that with the GNT framework, the robot cannot disambiguate one from another.

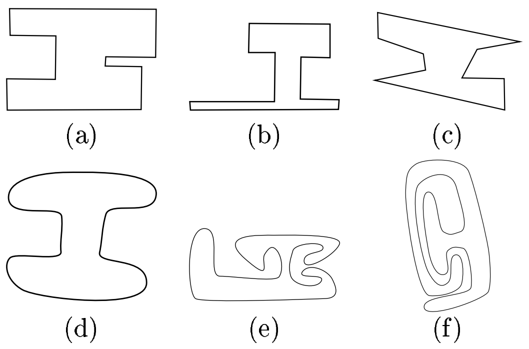


Figure 4: Environment equivalence. All the environments shown share the same family of GNTs. A robot could not disambiguate one from another using the sensing capabilities presented, yet it can navigate optimally in each of them.

3.3 Bitbots

In the previous examples, we used visibility information directly, either by computing it from a map, or by detecting visibility changes through the gap sensor. Now we present an example where the robot solves visibility-tasks without any visibility related sensor. In fact, the robot, called Bitbot [28], has only one sensor, a contact sensor. The contact sensor gives one bit of information indicating whether or not there is a contact with the boundary of the environment. The Bitbot can only choose among two types of movements in a polygonal environment. First, it can follow the walls in either direction. Second, when approaching a reflex vertex v

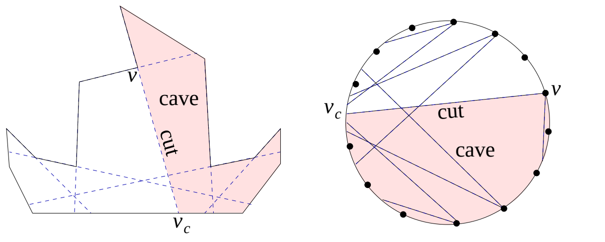


Figure 5: A polygon and all its cuts are shown on the left. For a cut $[v, v_c]$ its cave is shaded. The corresponding cut diagram is shown on the right.

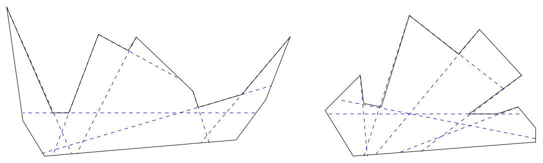


Figure 6: Some polygons having the cut diagram shown in Figure 5.

(Figure 5), it can choose to go straight off the reflex vertex along the continuation of the edge, and land on the opposite edge of the environment.

The state space is defined as $X \subset \mathbb{R}^2 \times E$, where each valid state $x = (q, e), q \in Q^e, e \in E$ represents the Bitbot position q with respect to the environment e it is in. The set E represents all possible environments the Bitbot may be in. Since neither e nor q are known to the robot, the state is unknown. The observation space is determined by the output of the contact sensor. We assume that the contact sensor indicates if the robot is currently in contact with a reflex vertex (i.e., a corner) of the environment, in contact with a non-reflex vertex, or in contact with a wall. Thus $Y = \{\textbf{reflex}, \textbf{nonReflex}, \textbf{wall}, \textbf{noContact}\}$. The action space, $U = \{$**goRight**, **goLeft**, **goRightOff**, **goLeftOff**$\}$ represents the actions to move right and left along the walls, or right and left along the walls followed by going off from the reflex vertices.

Given a reflex vertex v in the environment boundary, consider an edge incident to this vertex, with maximal extension inside the environment. When the robot decides to go straight off the reflex vertex, it follows exactly this segment, which is called a *cut*. For each reflex vertex there are two cuts, corresponding to the two incident edges at this vertex. An example of a polygon with the set of all of the cuts is shown on Figure 5. Consider an environment representation in which nodes representing the polygon vertices are arranged in a circle, respecting its circular order along the boundary. Each edge in the polygon has its counterpart along the circle too. For each cut in the polygon, a chord is added to the circle, from a node corresponding to the reflex vertex to the corresponding edge. This representation, called the *Cut diagram* of the polygon, contains the information related to inflections and bitangents of the environment boundary, as it is shown in [28]. Particularly, through the diagram we can *conservatively* determine if two given reflex vertices may be endpoints of a bitangent in the environment. This test is conservative, since two vertices may be said to form a bitangent when in fact they do not. This is because different polygons will share the same Cut diagram, and for some of them the bitangent does exist (Figure 6).

As it is, the Bitbot cannot construct the Cut diagram. With the Bitbot capabilities assumed until now, the robot cannot count the number of vertices in the environment, information that is needed to construct the Cut diagram. Thus, the Bitbot is provided with a *marker*, or *pebble*, that labels a single position in the environment boundary. With this, the robot can transverse the boundary exactly one time to count the number of vertices, and to go straight off each cut, encoding in which edge, and which order the chords should be added. Once the Cut diagram is built, the pebble is not needed.

The Cut diagram offers a discrete version of the state space. For example, if the reading from the sensor is **noContact**, the Bitbot knows that is somewhere along a certain cut, but not its exact position. This allows the use of the nondeterministic derived information states framework presented before. As it is presented in [28], it is possible to solve a version of the pursuit-evasion problem with a search in this collapsed state space, together with the bitangent information available in the diagram.

3.4 Almost-Sensorless Localization

Consider now a mobile robot equipped with a contact sensor, an environment map, and a compass. The reliable motions available to this sort of robot are severely limited. Lacking odometry or a sense of time, the robot can only choose a direction of motion and travels in that direction until it reaches the boundary of the environment. Suppose this robot is kidnapped and released at an unknown position. Can the robot localize itself? To formalize, let the environment be described by a simply-connected polygonal environment X. At each step the robot chooses an action (that is, a direction of motion) from $U = S^1$. The resulting state $x' = f(x, u)$

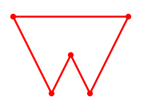
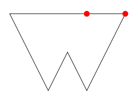
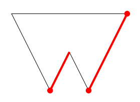
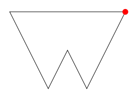
i	u_i	η_{i+1}	i	u_i	η_{i+1}
0			2		
1			3		

Figure 7: A localizing sequence for a simple non-convex polygon. The derived information state at each step is shaded.

is the first boundary point touched by moving from x in direction u.

We can define localization as a planning problem over nondeterministic derived information states. Figure 7 shows a rudimentary example environment along with a localization plan for it and the derived information states traversed along the way. The initial condition is total uncertainty, so that $\eta_0 = X$. The goal is to reach some singleton information state,

$$\eta_G = \{\eta \in \mathcal{I} \mid |\eta| = 1\},$$

or equivalently, $\eta_G = \{\{x\} \mid x \in X\}$. To complete the problem definition, we must describe the information transition function $f : \mathcal{I} \times U \to \mathcal{I}$. First consider two special cases:

- If η is a single point, then $f(\eta, u)$ can be computed by a ray shooting query [1] in X.
- If η is a segment along the boundary of X, then $f(\eta, u)$ can be computed geometrically. Sweep a normal line across the segment, tracking changes to the environment edge first intersected by the sweep line. At each change, a new segment is added to the resulting information state.

These two cases are illustrated in Figure 8. For arbitrary information transitions, observe that any reachable information state can be described by a finite union of points and open segments along the environment boundary. Therefore, for an arbitrary reachable information state action pair, the resulting information state is simply the union of partial results given by the two special cases described above.

To solve the localization task, we need a strategy that will reach one of the goal information states. Since there are no observations for this problem, we can describe the strategy as a sequence of actions. More generally, a policy over information space can be defined.

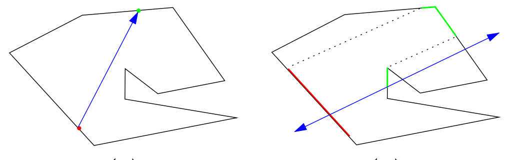


Figure 8: Computing the information transition function $F(\eta, u)$ for the special cases when (a) η is a single point and (b) η is a segment. All other reachable derived information states can be described by finite unions of these two special cases.

```
LOCALIZE($\eta$)
if $\eta$ contains a segment $s$ then
  return a direction parallel to $s$
else if $\eta$ contains at least two points $p$ and $q$ then
  if $q$ is visible from $p$ then
    return a direction parallel to $p - q$.
  else
    return a direction parallel to the gap hiding $p$ from $q$.
  end if
else
  terminate
end if
```

Figure 9: A motion strategy for localization with a simple robot, expressed as a policy over derived information space.

Figure 9 shows a localization policy that originally appeared in [22]. This policy will eliminate segments from the information state first, then iteratively merge pairs of the remaining points until the information state is a single point. A more complex example appears in Figure 10.

3.5 Probabilistic Information Spaces

The planning examples described above present nondeterministic state uncertainty. However, for some tasks, probability distribution over the state space and nature actions are available, and have been used in an information space context. One of such approaches is the well known *Kalman filter*. In the case of the Kalman filter, the transition function f, and the sensor mapping h are both linear functions, and nature actions, θ and ψ, can be modeled as Gaussians. Thus, the derived information states will follow a Gaussian distribution too. Each Gaussian is specified by an n-dimensional mean vector μ, and an $n \times n$ symmetric covariance matrix, Σ. Since the Kalman filter relies on linear models, f takes the well-known form

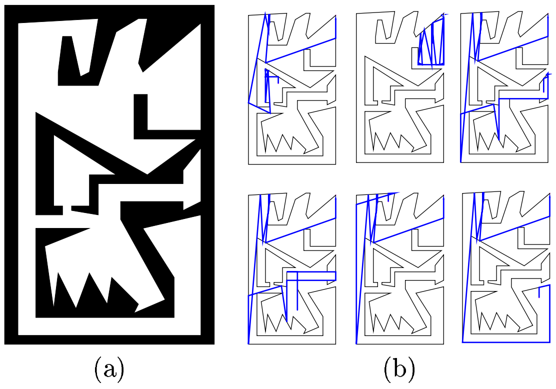


Figure 10: (a) An irregular environment for which the the policy in Figure 9 terminates in 30 steps. (b) Execution traces for 6 different starting positions. For each starting position, the final position is the upper right corner of the environment.

$$x_{k+1} = A_k x_k + B_k u_k + G_k \theta_k,$$

in which A_k, B_k, and G_k are real-valued matrices of appropriate dimensions. The subscript k is used because the Kalman filter works even if f is different in every stage. Similarly, the sensor mapping becomes

$$y_k = C_k x_k + H_k \psi_k.$$

Since an information state $P(x_k|\eta_k)$ is represented by its mean vector and its covariance matrix, the goal is to compute μ_k and Σ_k at the stage k. Such updating expressions can be found in textbooks on stochastic control (i.e., [15]).

If we assume that nature can be modelled probabilistically, and it follows a Markov model (its actions depend only on the current state, as opposed to actions or state histories), the derived information state becomes a conditional probability distribution. The set functions H and F become $P(x_k|y_k)$ and $P(x_{t+1}|x_k, u_k)$, respectively. To compute $P(x_k|y_k)$ Bayes rule is applied as:

$$P(x_k \cap y_k) = P(x_k|\ y_k)P(y_k) = P(y_k|\ x_k)P(x_k).$$

Bayes rule will require the knowledge of $P(x_k)$, which is replaced by a derived information state[1]. Since each information state is a probability distribution over X, it can be written as $P(x_k|\ \eta_k)$, if it is derived from η_k. As before, derived information states can be computed inductively [17]. In this case, the derived information space is the set $\mathcal{P}(X)$, of all probability distributions over X. Thus, the planning problem can be expressed again entirely in terms of the derived information space. A goal region can be specified as constraints on the probabilities. For example, for some particular $x \in X$, the goal might be to reach any derived information state for which $P(x|\eta_k) > 0.9$. Further, it is possible to embed $\mathcal{I}$ in $\mathbb{R}^n$ with each state $x \in X$ representing a vertex of a $(n-1)$-simplex. The coordinates of each vertex are expressed using probabilities $(p_1, p_1, \ldots, p_n)$ as barycentric coordinates. Here, p_i is the probability of being in state x_i. Since $p_1 + \cdots + p_n = 1$, the vertices of the simplex (i.e., $(1, 0, \ldots, 0)$, $(0, 1, \ldots, 0)$, $\cdots$, $(0, 0, \ldots, 1)$) correspond to the cases when the state is completely known. A planning problem of this kind is known as a *Partial Observable Markov Decision Process* (POMDP). Solving efficiently POMDPs is an active area in the research community [16, 29]. The problem is clearly very difficult, since the dimension of the space grows linearly with the number of states.

[1]In this context, derived information states have been also called *belief states*.

4 Conclusions

In this paper we have presented information spaces - a notion which combines all planning problems for robots with sensing uncertainty into one framework. Each information state represents the current knowledge of the robot about its progress after taking each action and sensor measurement. We have described several examples of information spaces for different problems, such as pursuit-evasion tasks for robots with different sensing capabilities and robot localization. These examples show that considering planning problems in terms of information spaces allows a better understanding of the structure of the problem. Moreover, the solutions for robotics tasks naturally lie in the spaces of information states, which allows finding better plans for the robots. Considering information spaces opens new opportunities for characterizing the robotics tasks. It is possible to characterize sensors based on their power. It is also possible to design robots with minimal sensor requirements for a given task, by comparing generated information spaces, as it was shown on the example of pursuit-evasion task, which was solved with robots with hierarchy of sensors. Information spaces also allow to characterize the essential information needed to solve the required tasks, allowing design of task specific sensors, as was shown in the presented example on localization. There are many opportunities to contribute the research on planning for mobile robots using information spaces. It is our hope that this work will stimulate the community to progress in solving challenging problems in robotics.

References

[1] B. Chazelle and L. G. Guibas. Visibility and intersection problems in plane geometry. *Disc. and Comp. Geom.*, 4:551–589, 1989.

[2] C.-T. Chen. *Linear System Theory and Design.* Holt, Rinehart, and Winston, New York, NY, 1999.

[3] H. Choset and K. Nagatani. Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization. *IEEE Int. Conf. Robot. & Autom.*, 17(2):125–137, April 2001.

[4] B. R. Donald. Planning multi-step error detection and recovery strategies. *Int. J. Robot. Res.*, 9(1):3–60, 1990.

[5] B. R. Donald. On information invariants in robotics. *Artif. Intell.*, 72:217–304, 1995.

[6] G. Dudek, K. Romanik, and S. Whitesides. Localizing a robot with minimum travel. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 437–446, 1995.

[7] F. Durand. *3D Visibility: Analytical study and applications.* PhD thesis, Université Grenoble I – Joseph Fourier Sciences et Géographe, July 1999.

[8] M. Erdmann. Understanding action and sensing by designing action-based sensors. *Int. J. Robot. Res.*, 14(5):483–509, 1995.

[9] K. Y. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10:201–225, 1993.

[10] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani. Visibility-based pursuit-evasion in a polygonal environment. In F. Dehne, A. Rau-Chaplin, J.-R. Sack, and R. Tamassia, editors, *WADS '97 Algorithms and Data Structures (Lecture Notes in Computer Science, 1272)*, pages 17–30. Springer-Verlag, Berlin, 1997.

[11] L. J. Guibas, R. Motwani, and P. Raghavan. The robot localization problem. In K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, editors, *Proc. 1st Workshop on Algorithmic Foundations of Robotics*, pages 269–282. A.K. Peters, Wellesley, MA, 1995.

[12] L. Guilamo, B. Tovar, and S. M. LaValle. Pursuit-evasion in an unknown environment using gap navigation graphs. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 2004.

[13] I. Kamon, E. Rivlin, and E. Rimon. Range-sensor based navigation in three dimensions. In *IEEE Int. Conf. Robot. & Autom.*, 1999.

[14] J.J. Koenderink and A.J. van Doorn. The singularities of the visual mapping. *Biological Cybernetics*, (24):51–59, 1976.

[15] H. Kwakernaak and R. Sivan. *Linear Optimal Control Systems.* Wiley, New York, NY, 1972.

[16] M. Littman L. Kaelbling and and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[17] S. M. LaValle. *Planning Algorithms.* Cambridge University Press (also available at http://msl.cs.uiuc.edu/planning/). To be published in 2006.

[18] S. M. LaValle and J. Hinrichsen. Visibility-based pursuit-evasion: The case of curved environments. *IEEE Transactions on Robotics and Automation*, 17(2):196–201, April 2001.

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

[20] M. T. Mason. *Mechanics of Robotic Manipulation.* MIT Press, Cambridge, MA, 2001.

[21] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *AAAI National Conference On Artificial Intelligence*, 2002.

[22] J. M. O'Kane and S. M. LaValle. Almost-sensorless localization. In *IEEE Int. Conf. Robot. & Autom.*, 2005.

[23] J. M. O'Kane, B. Tovar, P. Cheng, and S.M. LaValle. Algorithms for planning under uncertainty in prediction and sensing. *Chapter 18 in Autonomous Mobile Robots: Sensing, Control, Decision-Making, and Applications*, 2005. To appear.

[24] S. Rajko and S. M. LaValle. A pursuit-evasion bug algorithm. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 1954–1960, 2001.

[25] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial Intelligence Journal*, 2001.

[26] B. Tovar, L. Guilamo, and S. M. LaValle. Gap navigation trees: Minimal representation for visibility-based tasks. In *Proc. Workshop on the Algorithmic Foundations of Robotics*, 2004.

[27] B. Tovar, S. M. LaValle, and R. Murrieta. Optimal navigation and object finding without geometric maps or localization. In *Proc. IEEE International Conference on Robotics and Automation*, 2003.

[28] A. Yershova, B. Tovar, S.M. LaValle, and R. Ghrist. Bitbots, simple robots solving complex tasks. In *AAAI*, 2005.

[29] N. Zhang and W. Lin. A model approximation scheme for planning in partially observable stochastic domains. *Journal of Artificial Intelligence Research*, 7:199–230, 1997.