COMPUTER VISION AND IMAGE UNDERSTANDING
Vol. 62, No. 2, September, pp. 232–244, 1995

Navigational Functionalities*

EHUD RIVLIN AND AZRIEL ROSENFELD

Computer Vision Laboratory, Center for Automation Research, University of Maryland, College Park, Maryland 20742-3275

Received August 1, 1994; accepted May 12, 1995

A navigating agent can relate in three basic ways to an object: Avoidance (e.g., if the object is a threat or an obstacle), interception (e.g., if the object is prey or food), and reference (e.g., if the object can be used as a landmark). We illustrate these classes of object functionalities for the case in which the agent is a simple corridor-cleaning robot that uses the walls (and wall–floor junctions) as references in following the corridor, treats independently moving objects as threats and large stationary objects as obstacles, and treats small stationary objects as "prey" (trash to be swept up). © 1995 Academic Press, Inc.

1. INTRODUCTION

Objects can be categorized in terms of functionality—i.e., usefulness to a given agent in carrying out specific purposes or tasks. There has been considerable recent interest in function-based object recognition [16, 20–26]. Much of this work has dealt with cases in which the object functions as a "machine," i.e., a device for transferring force—for example, a chair, which can support a sitter (by transferring the sitter's weight to the floor); a cup, which can contain a liquid; and a knife, which can be used to cut into surfaces. However, there are many other important classes of functionalities; for example, if the agent is animate, some of the objects in its environment can function as food ("prey") or as threats ("predators"; note that this is a "negative" type of functionality).

In this paper we consider some basic classes of functionalities that objects can have for a navigating agent. Such an agent can relate in three basic ways to an object: Avoidance (e.g., if the object is a threat or an obstacle), interception (e.g., if the object is prey or food), and reference (e.g., if the object can be used as a landmark). We illustrate these classes of object functionalities for the case in which the agent is a simple corridor-cleaning robot that uses the walls (and wall–floor junctions) as references in following the corridor, treats independently moving objects as threats and large stationary objects as obstacles, and treats small stationary objects as "prey" (trash to be swept up). The robot is called "Sisyphus," because its work is never done (it moves endlessly back and forth along the corridor).

Sisyphus' behaviors, and the object functionalities that it must be able to recognize in order to carry out these behaviors, are listed in Section 2. In Section 3 we describe Sisyphus' visual modules, and in Section 4 we show how Sisyphus uses these modules to perform the necessary recognition tasks. Section 5 presents experimental results, and Section 6 discusses some basic control structure issues that arise in designing agents of this type. Finally, Section 7 discusses how this work could be extended to handle functionalities that involve dynamic aspects of navigation.

Issues of functionality also arise in connection with the domain in which navigation takes place. For example, the parts of a generic building can be described in terms of their functions—e.g., a hallway is a channel for motion beween rooms and must be wide enough for two people to pass [9]. A possible way to represent such generic models is by using frames, as in [27]. These high-level aspects of navigational functionality are beyond the scope of this paper.

2. SISYPHUS

Sisyphus is an agent equipped with visual sensors whose purpose is to clean corridor floors. The control aspects of this problem will be discussed in Section 6, where Sisyphus is modeled using a Discrete Event Dynamic System (DEDS) formalism [8, 13], and properties of its behavior are proved. In this and the next three sections we discuss primarily the recognition tasks performed by Sisyphus. In particular, in Section 3 we describe the visual modules used by Sisyphus to perform these recognition tasks. In Section 4 we demonstrate the feasibility of the recognition tasks in the context of Sisyphus' behaviors and environment. We define Sisyphus' environmnent to be a single corridor, thus avoiding problems related to planning a path through a building. Such path planning could be based in part on

* The support of the Advanced Research Projects Agency (ARPA Order 8459) and the U.S. Army Topographic Engineering Center under Contract DACA76-92-C-0009 is gratefully acknowledged, as is the help of Sandy German in preparing this paper.

1077-3142/95 $12.00
Copyright © 1995 by Academic Press, Inc.
All rights of reproduction in any form reserved.

recognition of specific locations; the achievability of this recognition task is also demonstrated.

In this corridor environment, Sisyphus must be able to carry out the following behaviors:

- Moving along the corridor
- Turning around when an end of the corridor is reached
- Avoiding obstacles (i.e., large objects)[1]
- Cleaning up litter (i.e., sufficiently small objects)
- Stopping (temporarily) if an independently moving object is present (such an object might be a threat).

Evidently, this means that Sisyphus must be able to recognize

- The corridor's walls and its ends
- Large objects (and free space)
- Small objects
- Independently moving objects
- Specific locations or landmarks.[2]

The definitions of "large" and "small" depend on Sisyphus' size, as well as on its capabilities of locomotion and manipulation. In addition, object "size" must be defined relative to the corridor coordinate system; for example, we are concerned with how far a large object extends above the floor or out from a wall at a height below the top of Sisyphus, but we may not care about other aspects of its size.

We do not attempt to cover the motion planning aspects of Sisyphus' tasks (avoiding loops, etc.), and we do not discuss the kind of cleaning equipment that Sisyphus has. We assume that keeping to the middle of the corridor is good enough for solving the coverage problem (the cleaning equipment includes extensors, etc.). Evidently, when Sisyphus perceives an obstacle, some planning has to be done to avoid the obstacle and clean in its vicinity.

In Section 3 we describe a set of visual modules that Sisyphus can use in these simple recognition tasks, in Section 4 we show how the modules are used to perform those tasks, and in Section 5 we present experimental results.

3. VISUAL MODULES

A set of visual modules that can be used by Sisyphus to carry out its recognition tasks will now be described. The tasks could be carried out using data obtained from senses other than vision, but here we are interested in achieving recognition, using vision only.

[1] A possible additional behavior might be pushing aside intermediate-size objects, but note that they may not be movable.

[2] This task is based on the localization algorithm described in [3, 15]. We limit ourselves to the localization aspect of the problem; path planning and positioning are not treated here.

3.1. *Computing Normal Flow*

A series of normal flow fields $[u^n(x, y, t), v^n(x, y, t)]$ will be used as visual input for many of Sisyphus' recognition tasks. If $f(x, y, t)$ is the image intensity function as it changes through time, and (u, v) is the image velocity as a projection of the 3-D motion, then

$$f_x u + f_y v + f_t = 0$$

or

$$(f_x, f_y) \cdot (u, v) = -f_t,$$

where the subscripts denote partial differentiation and "$\cdot$" denotes the inner product of vectors.

From this equation it follows that we can compute the projection of the optic flow (u, v) on the image gradient direction (f_x, f_y) at every image point. This normal flow f_n is given by

$$f_n = \frac{-f_t}{\sqrt{f_x^2 + f_y^2}}.$$

We assume that the normal flow is computed in real time. We also assume that $f_x, f_y \neq 0$ at a large number of pixels, e.g., that some amount of texture is present in the scene.

We further assume that Sisyphus is equipped with inertial sensors that provide information about its rotation. (Even when Sisyphus attempts to move in a straight line, it may undergo slight rotations because of unevenness of the floor.) Since the rotation is known, its effect on the flow can be subtracted using a process known as derotation. Indeed, if f_n is the value of the normal flow at (x, y) and the rotation of the sensor is (A, B, C), then the rotational flow at (x, y) is given by

$$u_r = Axy - B(1 + x^2) + Cy$$
$$v_r = A(1 + y^2) - Bxy - Cx.$$

If we let $(u_t, v_t) = (u, v) - (u_r, v_r)$, then the normal flow $f_n = (u_t, v_t) \cdot (f_x, f_y)$ is due to translation only. We may thus assume that Sisyphus is translating.

3.2. *Computing Time to Collision*

Let Sisyphus be moving along its optical axis (i.e., in the direction in which it is looking) with velocity V (see Fig.

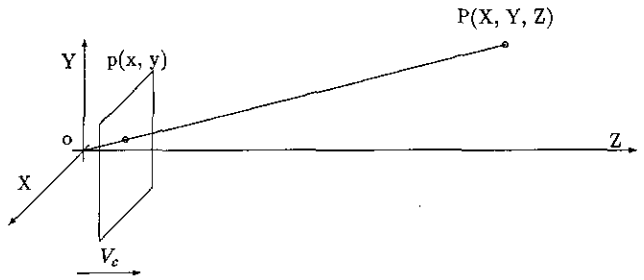

FIG. 1. The camera moves along the corridor with velocity V.

1). Let the image point $p(x, y)$ be the projection of scene point $P(X, Y, Z)$. After time dt, $P'(X, Y, Z - V_c dt)$ projects to $p'(x', y')$. Using perspective projection and assuming unit focal length, we have

$$x = \frac{X}{Z}$$

$$y = \frac{Y}{Z}$$

$$x' = \frac{X}{Z - Vdt}$$

$$y' = \frac{Y}{Z - Vdt}.$$

Thus we can obtain the velocity of image point $p(x, y)$ as

$$u = \lim_{dt \to 0} \frac{x' - x}{dt} = \frac{V}{Z} x$$

$$v = \lim_{dt \to 0} \frac{y' - y}{dt} = \frac{V}{Z} y.$$

Let the unit image gradient vector at $p(x, y)$ be (n_x, n_y). Then by the equation in Section 3.1 we have

$$f_n = un_x + vn_y$$

$$= \frac{V}{Z} xn_x + \frac{V}{Z} yn_y$$

$$= \frac{V}{Z} (xn_x + yn_y)$$

so that

$$\frac{V}{Z} = \frac{f_n}{xn_x + yn_y}.$$

Thus, the quantity Z/V is computable everywhere on the image (wherever features allowing the estimation of normal flow exist). This quantity represents the time to collision for the given scene point $p(x, y, z)$; it is a scaled version of the distance to that point. When Z/V is small, we have a potential hazard.

3.2.1. *Detecting Anomalies on the Floor.* Let the corridor geometry be as shown in Fig. 2. We assume that the floor is horizontal, the optical axis of the camera lens is parallel to the floor, and the distance of the center of the camera (nodal point) above the floor is known. As a result, the plane of the floor has an equation which is known in the coordinate system of the camera. Let this equation be $Z = pX + qY + c$. Expressing this equation in image coordinates (x, y) yields $1 - px - qy = c/Z$ or $Z = c/(1 - px - qy)$. Assuming that Sisyphus moves along its optical axis with velocity V, at a point (x, y) which is the image of a feature on the floor, we have

$$u = \frac{V}{Z} x = \frac{xV(1 - px - qy)}{c};$$

$$v = \frac{V}{Z} y = \frac{yV(1 - px - qy)}{c}.$$

Suppose that at an image point which corresponds to a point on the floor we measure a normal flow value f_n along some direction (n_x, n_y). As we saw, we should then have

$$f_n = (n_x, n_y) \cdot (u, v).$$

If this condition is not satisfied, we have detected an anomaly. The distance to an anomaly can be computed as described before, and the height of the anomaly above the floor can be computed from its angular extent, as in Fig. 3.

In principle Sisyphus might also be concerned about protrusions from the walls (see for example Fig. 7 in Section 5: the sink and the fire extinguisher). Keeping to the center solves part of the problem. A full solution should handle these protrusions in a similar way. Knowing the equations of the walls we have expectations. Anomalies, deviations from what is expected, should be detected and trigger the appropriate procedure for recognizing a potential obstacle.

3.3. *Estimating Object Dimensions*

When Sisyphus sees an anomaly on the floor it can use its own height and the visual angle to infer the distance to the anomaly, as in Fig. 3. Given the distance, the visual angles subtended by the object provide estimates of its height and width.

3.4. *Steering*

In an ordinary corridor, the walls intersect the floor along two parallel lines which can be detected using line detectors. There may be additional lines on the walls or

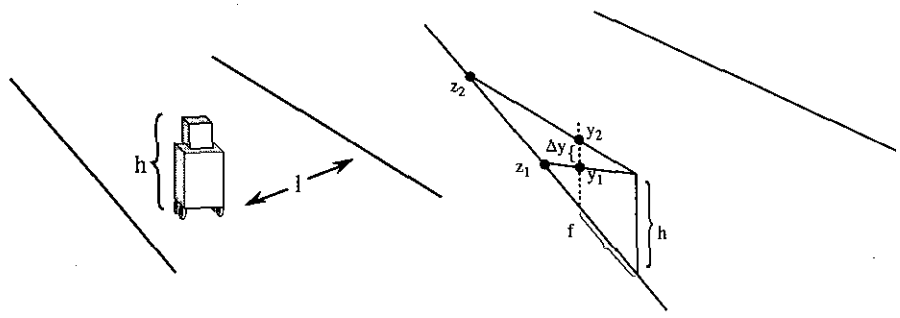

FIG. 2. Corridor geometry.

the floor, but those at which the walls meet the floor can be identified from knowledge of the floor plane and the distance between the walls.

Let Sisyphus have height h and be distance l from a wall (see Fig. 2). In the image, points at distance z_i on one of the parallel lines will have (x_i, y_i) values given by

$$x_i = l\frac{z_i - f}{z_i}$$

$$y_i = h\frac{z_i - f}{z_i}$$

(see Fig. 2), where f is the distance of Sisyphus from the origin. The angle α between the line (the base of the wall) and the x axis in the image is then given by

$$\arctan\left(\frac{\Delta x}{\Delta y}\right) = \arctan\left[\frac{hf\left(\frac{1}{z_1} - \frac{1}{z_2}\right)}{lf\left(\frac{1}{z_1} - \frac{1}{z_2}\right)}\right] = \arctan\left(\frac{h}{l}\right)$$

(see Fig. 2). To steer Sisyphus so it does not hit a wall, l must be prevented from becoming too small. As l gets smaller, the angle α (or β, for the other wall) between the wall and the x axis gets bigger. When Sisyphus is closer to the left wall we have $\alpha > \beta$ (see Fig. 4). When it is moving along the corridor axis, α and β are equal. Note that this method of obstacle avoidance (avoiding the walls) works even if the environment is not textured; indeed, the lines where the walls meet the floor can be detected more reliably in an untextured environment (see Fig. 7 in Section 5).

4. RECOGNITION TASKS

Sisyphus must perform the following recognition tasks to implement its behaviors:

1. Recognize the main cleaning area, i.e., the walls, floor, and ends of the corridor
2. Recognize independently moving objects
3. Recognize object size categories
4. Recognize specific places (localization).

We should point out that these behaviors also involve motor tasks whose analysis is beyond the scope of this paper. We consider here only the recognition tasks.

4.1. *Recognizing the Corridor Walls and Ends*

These tasks can be accomplished using the following three visual modules:

(a) Computing time to collision
(b) Detecting anomalies
(c) Steering.

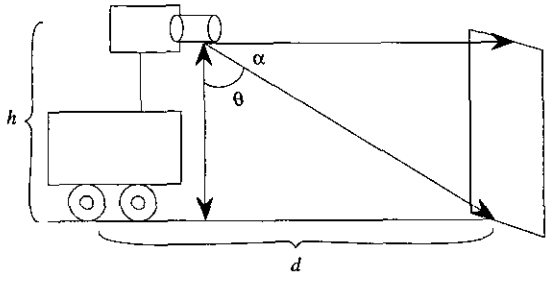

FIG. 3. Estimation of object height.

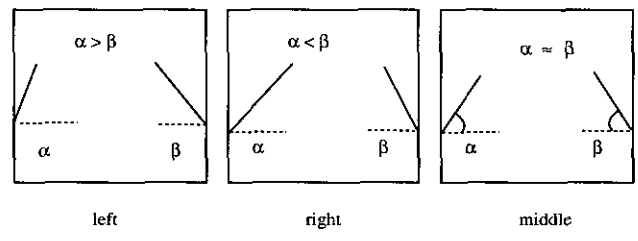

FIG. 4. Steering.

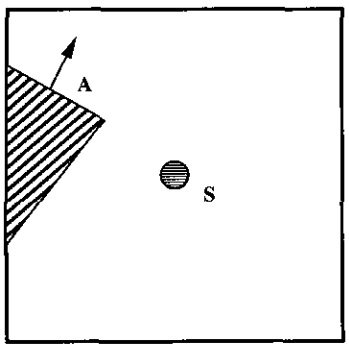

FIG. 5. Normal flow inconsistent with the FOE must arise from independent motion.

Using (a) we can identify the contiguous part of the scene that is the farthest away (maximum time to collision). This can ordinarily be used to define Sisyphus' direction of motion. Using (b) we find anomalies, if any. The motion can continue as long as no unexpected anomalies are detected. Finally, (c) allows Sisyphus to center itself along the main axis of the corridor. Clearly, there exists a great amount of redundancy in these modules, and this contributes to the robustness of recognition. For example, consider a situation where it is hard to detect the place where the walls and the floor meet, because they are strongly textured. In this case we should be able to get a reliable output from the time to collision module.

Recognizing the end of the corridor is done similarly. If we can detect no free space along the main axis of the corridor, we must be at a dead end. We can use the steering module to determine that we are on the main axis; then we use the other modules to check for "no free space."

4.2. *Recognizing Independently Moving Objects*

This task can be accomplished using the following three visual modules:

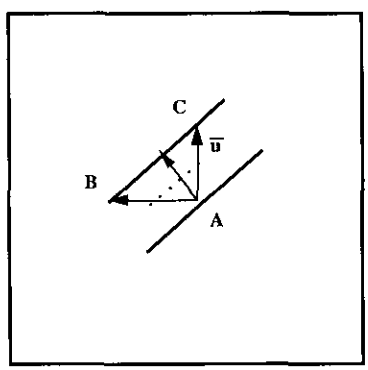

FIG. 6. If **u** is the normal flow at point A, then point A could move anywhere between B and C and the normal flow at the new position is constrained if there is no acceleration.

(a) Computing normal flow
(b) Computing time to collision
(c) Detecting anomalies.

Since Sisyphus is translating (or its flow field can be derotated; see Section 3.1), its flow field is characterized by a Focus of Expansion (FOE). Each normal flow measurement constrains the FOE to lie in a half plane; by intersecting these half-planes, the FOE can be localized as

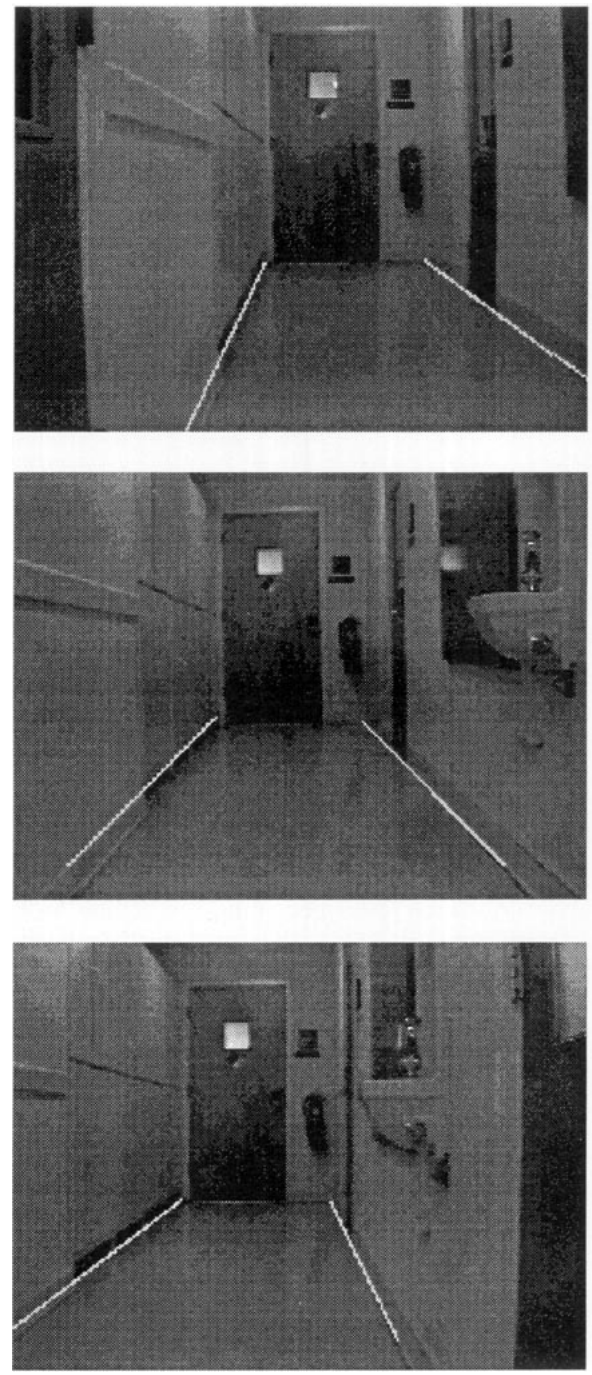

FIG. 7. Steering in a real corridor.

lying in a relatively small area [5, 6]. If the normal flow at a given point A is inconsistent with the location of the FOE (as in Fig. 5), point A cannot be a static environmental point, but must belong to an independently moving object. By moving the camera, Sisyphus can move the FOE to various image positions and thus completely detect anything moving independently. It can be shown that moving the FOE to the four corners of the image is sufficient to detect all independent motions [1].

The normal flow measurement module can also detect the existence of image regions having high accelerations; such accelerations are often characteristic of animate motion. If at time t we measure the normal flow at point A as in Fig. 6, the point should move to somewhere along segment BC, because the actual motion cannot be arbitrarily large. Thus, a measurement of normal flow at time $t + dt$ on BC that is not consistent signals a potential area of high acceleration.

If we know Sisyphus' velocity, detection of any normal flow value higher than a threshold value should indicate an independently moving object. If Sisyphus is stationary, detecting independently moving objects is trivial. Here we are paying the price of slower performance to achieve robust detection of independent motion (and hence greater safety).

Triggering a (temporary) stop when independent motion or high acceleration is detected should be time to collision dependent. There is no sense in stopping Sisyphus when something is moving slowly very far away. The time to collision module can check this. An example of the detection of independent motion in a corridor containing a moving human being will be shown in Section 5 (Fig. 10).

4.3. *Recognizing Object Size Categories*

For this task we use two visual modules:

(a) Computing time to collision
(b) Detecting anomalies.

Estimating the distance to, and hence the dimensions of, an anomaly was discussed in Section 3.3. For any (static) object that can be distinguished from its background, the time to collision module can be used to estimate its distance (since Sisyphus' velocity is known), and from this information its dimensions can be estimated.

This recognition task triggers the PICK and OBS behaviors as described in Section 6. For Sisyphus a small object is to be picked up and a large one is to be avoided.

4.4. *Recognizing a Specific Location*

For this task we use the localization algorithm described in [3, 15]. Basically two "reference views" of the specific location are needed. Localization can then be achieved by interpolation between these views; see Section 5.2.

5. EXPERIMENTS

We describe some experimental results obtained with these recognition tasks. All the experiments were done using a CCD camera in a corridor at our laboratory.

5.1. *Recognizing the Main Cleaning Area*

We implemented the steering visual module needed for the task. The module detects the corridor sides (where the walls meet the floor) using a version of the Hough transform which takes advantage of knowledge of the environment to limit the range of the search. The results obtained by this module in different steering situations are presented in Fig. 7.

5.2. *Recognizing a Specific Location*

A set of images was taken in a corridor. We used the place recognition algorithm described in [3, 15] to do localization. Because of the deep structure of the corridor, perspective distortions are noticeable. Nevertheless, the alignment results yield an accurate match in large portions of the

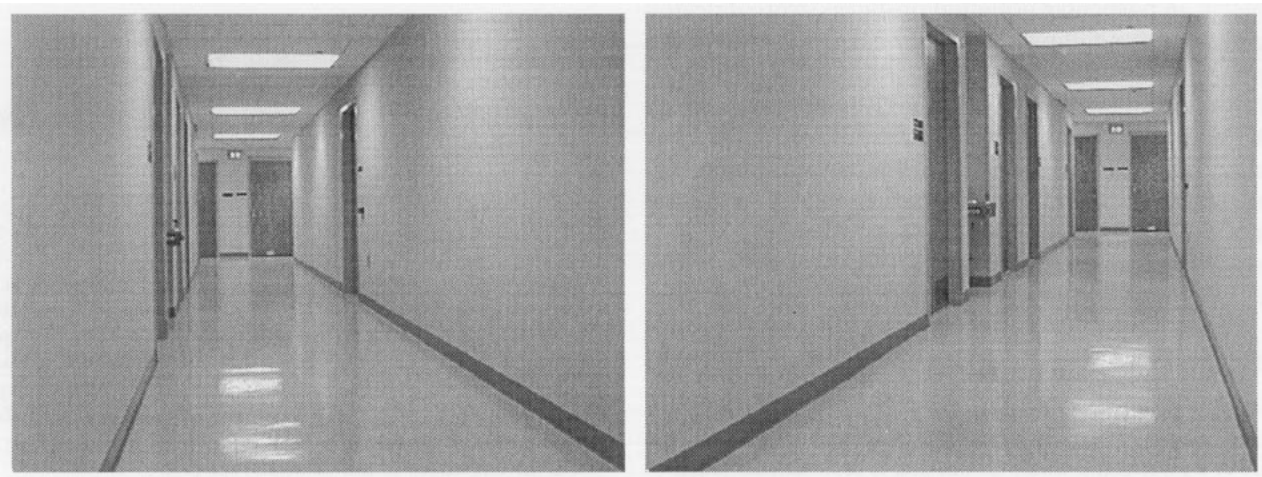

FIG. 8. Two reference views of a corridor.

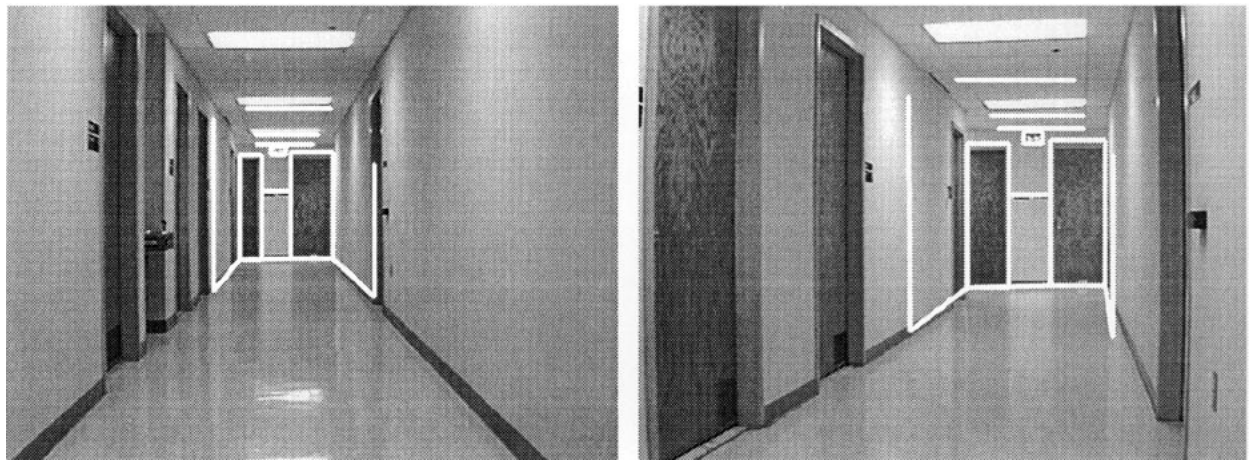

FIG. 9. Matching the corridor model with two images of the corridor. The right image was obtained by a relatively large motion forward (about half of the corridor length) and to the right.

image. Figure 8 shows two reference views of the corridor. Figure 9 (left) shows an overlay of a linear combination of these reference views of an image of the corridor. It can be seen that the parts that are relatively distant align perfectly. Figure 9 (right) shows the matching of the corridor model with an image obtained by a relatively large motion (about half of the corridor length). Because of perspective distortions the relatively near features no longer align (e.g., the near door edges). The relatively far edges, however, still match.

5.3. *Recognizing Independently Moving Objects*

Using the input from the normal flow module we implemented the independently moving object detection algorithm described in Section 4.2. Results obtained for an independent motion in the corridor (while the camera is translating forward) are shown in Fig. 10. The independently moving features are displayed as black dots.[3]

5.4. *Recognizing Object Size Categories*

Objects, in particular potential obstacles, were detected using the time to collision visual module. The results obtained by this module for forward translation toward a chair are presented in Fig. 11. If we can detect an object we can compute its size, as explained in Section 4.3. For example, knowing the height of the camera (and its focal length), the size of the waste basket in Fig. 12 was computed.

An interesting issue involves the dimensionality of the object; it can be inferred using qualitative shape information. We shall show now how to use rough metric properties to infer the dimensionality of an object. Our goal is to find out whether the object we are looking at is a 1-D, 2-D, or 3-D object. This categorization of the world into "sticks, plates, and blobs" can be useful to Sisyphus in various ways.[4]

To determine the dimensionality of the object we compute its thinness ratio T, defined [7] for an object of area A and perimeter P by[5]

$$T = \frac{4\pi A}{P^2}.$$

The isoperimetric inequality [2] states that, for a simple closed curve C of length L in the plane, the area A enclosed by C satisfies

$$L^2 - 4\pi A \geq 0$$

with equality holding if C is a circle. Hence the maximum thinness value is 1. It can be shown [11] that the thinness ratio of a regular n-gon is

$$T = \frac{\pi}{n} \cot \frac{\pi}{n}$$

which increases monotonically to 1 as the number of sides increases.

[3] This recognition task too can benefit by using senses other than vision. The use of a pyroelectric sensor for this task is demonstrated in [4].

[4] In [12], a recognition system is described that is based on a set of generalized blob models including sticks, plates, and blobs which, in three-space, are modeled as straight lines, circular disks, and spheres, respectively. Using geometric and relational constrains the system tries to achieve a consistent labeling of the scene. Our intention is completely different. We want to determine only the dimensionality of an object—i.e., is it a stick, a plate, or a blob.

[5] On the limitations of this definition see [17].

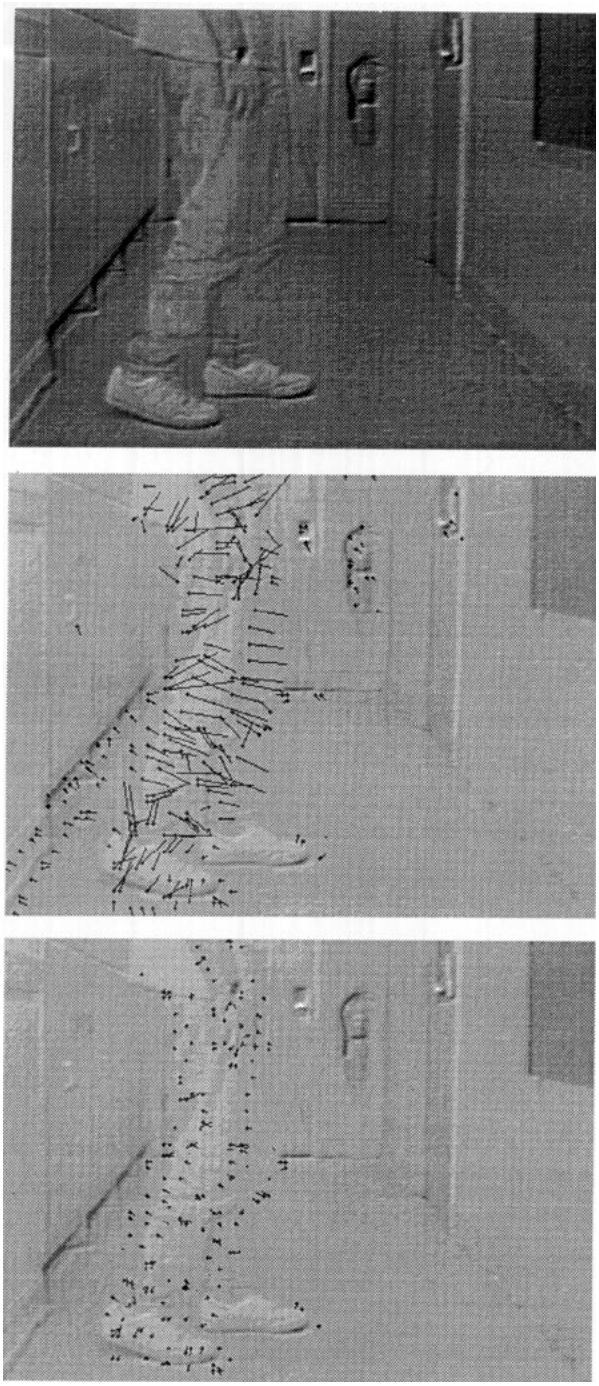

FIG. 10. Detecting independently moving objects. The arrows represent normal flows. Normal flow measurements that were inconsistent with the FOE were eliminated. Thresholding the normal flow values gave the same results. Black dots indicates places where independent motion was discovered.

We use the fact that line-like objects (sticks) will have a thinness ratio close to zero to define a 1-D object (stick) as an object with a lower than threshold thinness value, which does not change when we change our aspect (view angle). The distinction of a 2-D object from a 3-D one is less obvious. We define a 2-D object to be an object having a 1-D thinness property from some aspects. To find if an object is a plate we check its thinness and then move to check it again (a motion in a plane normal to the plate is the best).

A 3-D object is one with a higher than threshold thinness value that does not change significantly when we move around it. To remove the effects of a nonsmooth boundary we check the way in which the thinness value changes when we change resolution. We expect 1-D objects to keep their thinness values until they eventually disappear; 2-D objects will have lower thinness values as the resolution goes down, and 3-D objects will become circular and will tend to increase their thinness.

We compute the above qualitative shape information for a segmented object by performing the following steps: compute the object's thinness ratio, change resolution and compute the thinness ratio for every level in the pyramid; then move (exploration) and repeat the above computation. If the thinness ratio is lower than a threshold for all the steps above, the object is a 1-D object. If the thinness ratio is lower than a threshold for some of the steps above, and the change in the resolution was accompanied by a decrease in the thinness, it is a 2-D object. If the thinness ratio is higher than a threshold for some of the steps above, and the change in the resolution was accompanied by an increase in the thinness, it is a 3-D object.

Figure 13 shows the thinness ratio for a 1-D object at different resolutions. The ratio does not change with the resolution. Figure 14 shows the different thinness values obtained when looking at a plate from two different aspects. The way the thinness values change as we change resolution for a 3-D object is shown in Fig. 15. The algorithm we describe might fail. For example when we look at a 2-D plate from an aspect normal to it, a small motion will not reveal any significant changes in the thinness, and the thinness value might be high ($\pi/4$ for a square). A table which consists of a plate on four sticks will get a 2-D/1-D definition, while a "solid" table (or a box on the floor) will get a 3-D definition. However, the algorithm is expected to be robust when combined with other processes that extract shape from motion.

6. CONTROL STRUCTURE ISSUES

6.1. *Modules*

The architecture of Sisyphus directly reflects the relationship between behaviors, recognition tasks (corresponding to the different classes of object functionalities), and visual modules. A hierarchical view of this architecture is shown in Fig. 16.

Existing system architectures for visual recognition are modular; part of every system is devoted to general-purpose recovery tasks, which have been considered in the

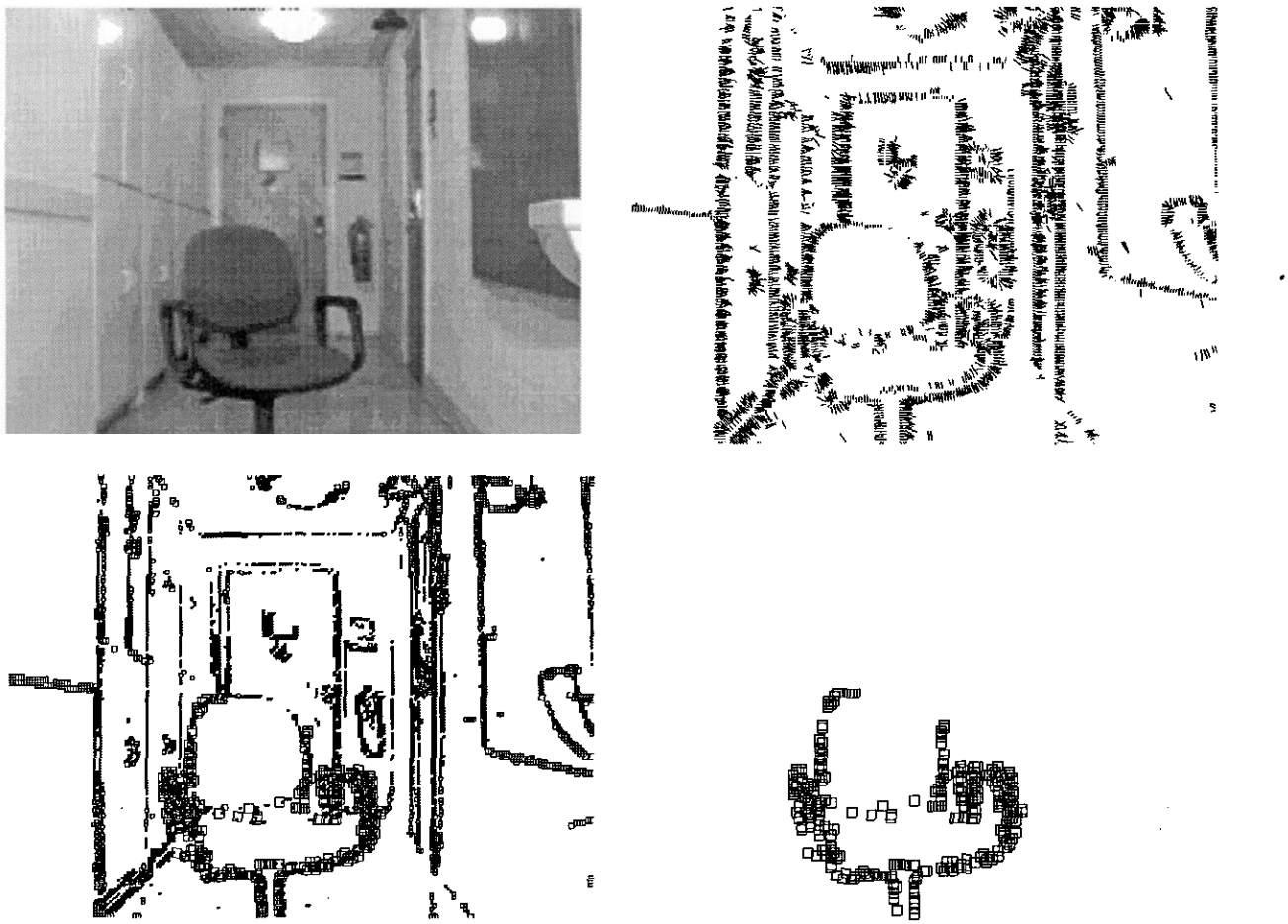

FIG. 11. Computing time to collision. Normal flows for the forward translation are presented in the second picture. The time to collision results are indicated using squares, where the size of the square is proportional to the time to collision value. The results of thresholding the time to collision are presented on the lower left.

literature in a modular fashion. Thus, most object recognition systems have been based on a modular architecture. As a result, the systems were often too general and not well suited for specific recognition tasks. When we study task-dependent recognition in terms of an agent operating in an environment, it is more appropriate to use an open labyrinthic design.

In a labyrinthic design [19], the recognition system can be viewed as a collection of processes or modules which perform particular recognition subtasks. These modules

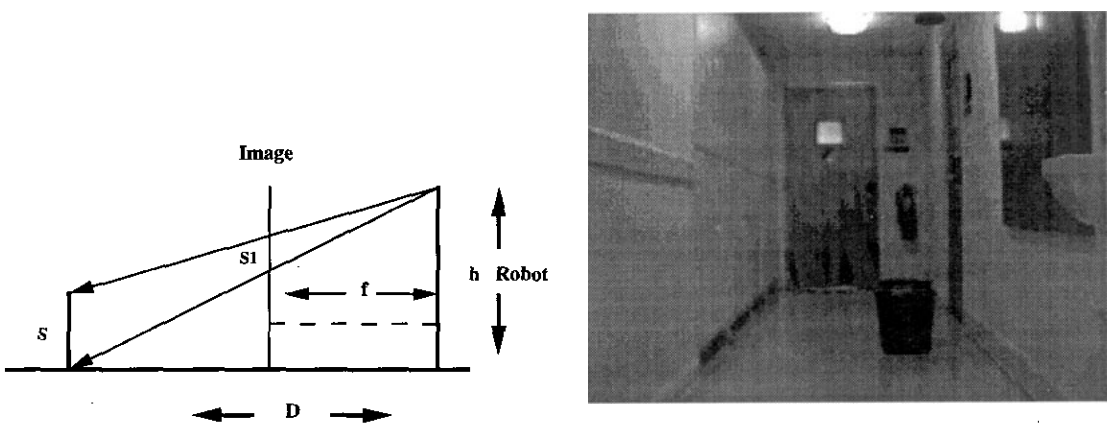

FIG. 12. Estimation of obstacle height. The distance is computed using the visual angle (330 cm for the wastebasket's base line). Knowing the focal length f (730 pixels), and $S1$ (82 pixels), we can compute S ($S1 : S = f : D$). For the image shown we obtained S = 37 cm (the real size of the wastebasket is 39 cm).

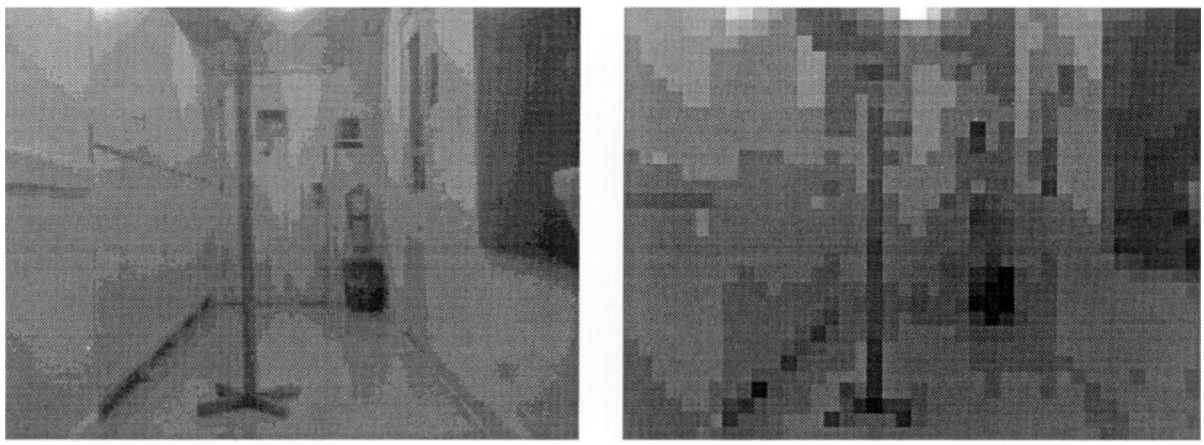

FIG. 13. A 1-D object: low thinness value (0.136) that does not change when we go up in the resolution pyramid (1:16, 0.141).

are connected dynamically to sensory modules or to other kinds of inputs. A configuration of modules that suits one agent will not necessarily suit another, but a labyrinthic design can suit every agent. The modules can work in parallel and can be simple or complex. We do not impose any restrictions on the order in which perceptual processes operate. Low-level processes can be guided by high-level considerations and vice versa.

We differentiate between low- and high-level modules; the former get their inputs directly from the sensory data, and the latter get some of their inputs from the low-level modules, as well as (possibly) directly from the sensory data. In this sense the low- and high-level modules are just simple and complex modules, respectively. The modules are controlled in a mixed top-down/bottom-up manner.

6.2. *States and Transitions*

We can use a state transition system [10] to describe agents as Discrete Event Dynamic Systems (DEDS). In this section we give such a description of Sisyphus, and we use the formalism to prove some safety properties about Sisyphus. In our description of Sisyphus, for every behavior (corresponding to a class of functionalities), we indicate the appropriate recognition tasks that are needed. These recognition tasks can be executed in parallel.

We assume that Sisyphus has three basic behaviors:

- M&C ("move and clean")
- S&W ("stop and wait," when an independently moving object is within some distance)
- PASS ("bypass an obstacle").

These behaviors are based on the following recognition tasks:

MCA—detect the main cleaning area
DE—detect a dead end
OBJD—detect potential obstacles and their dimensions
IMO—detect independently moving objects.

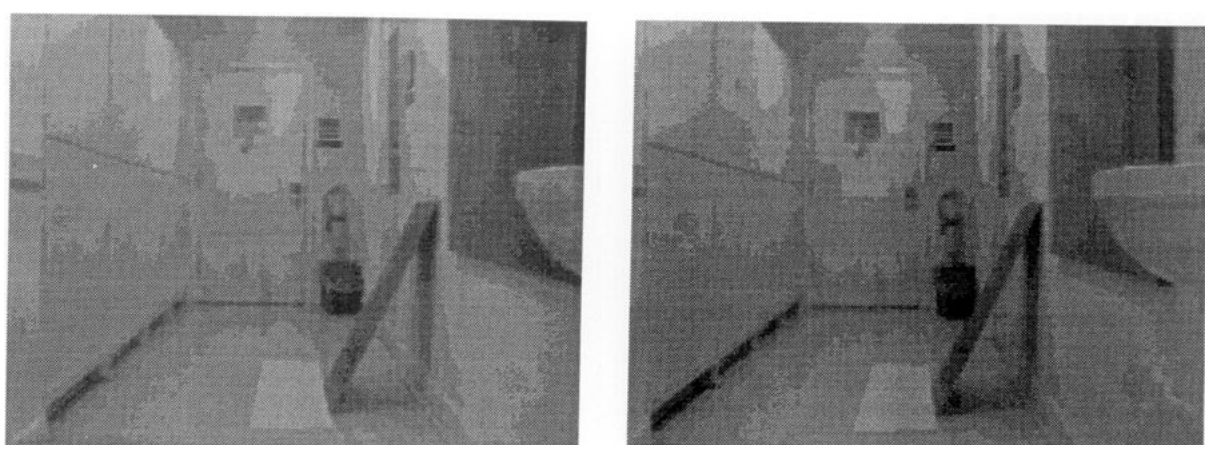

FIG. 14. A 2-D object: moving from the center (left picture) to the left (right picture), the thinness ratio changes from 0.20 to 0.16. The change in resolution was accompanied by decreasing thinness values until the object disappeared.

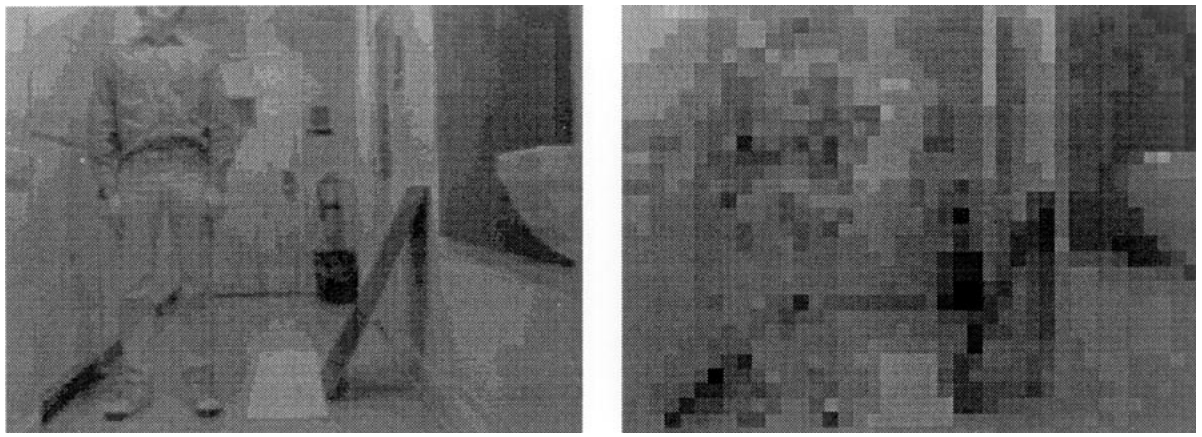

FIG. 15. A 3-D object: thinness values do not change when a motion takes place. A change in the resolution increased the thinness values from 0.21 (left image) to 0.48 (right image, 1:16).

We also need to specify the recognition tasks involved in each behavior and those that trigger switches from behavior to behavior. A behavior transition diagram is depicted in Fig. 17. Each node in the diagram represents a distinct behavior. To accomplish a behavior we need to carry out recognition tasks which are not shown in the diagram. The arcs represent the recognition tasks that trigger changes in behavior. These tasks run in parallel with whatever tasks are taking place during the behaviors. Since two or more events may happen simultaneously, a priority scheme is needed to arbitrate; larger numbers indicate higher priorities.

The default behavior of the system is M&C. Under M&C recognition of the main cleaning area (MCA) is carried out. The following recognition tasks are carried out in parallel with the M&C behavior: independently moving object detection (IMO), object detection (OBJD), and dead-end detection (DE). The highest priority is given to IMO, which runs in parallel with all other activities and triggers the S&W behavior. Similarly OBJD triggers the PASS behavior, and DE triggers the S&W behavior.[6]

We model Sisyphus using the state transition formalism. In this formalism, every event is composed of two parts: enabling conditions and actions. We specify the enabling conditions of an event e by using a state formula,[7] denoted $enabled_A(e)$, and a sequential program, denoted $action_A(e)$; $enabled_A(e)$ specifies the enabling condition of e, and $action_A(e)$ must always terminate when executed in any state satisfying $enabled_A(e)$. We assume, without loss of generality, that $action_A(e)$ is deterministic. We assume three visual modules are running. The first one detects independently moving objects. The second and the third detect obstacles, the main cleaning area, and independently moving objects, using different algorithms.[8] Sisyphus is moving in the M&C (move and clean) behavior; when it detects an independently moving object it switches to S&W (stop and wait). When an obstacle is detected Sisyphus goes to the PASS behavior (bypassing the obstacle). Sisyphus and the three visual modules are modeled as different processes. Processes VM1, 2, 3 are environment dependent. Their

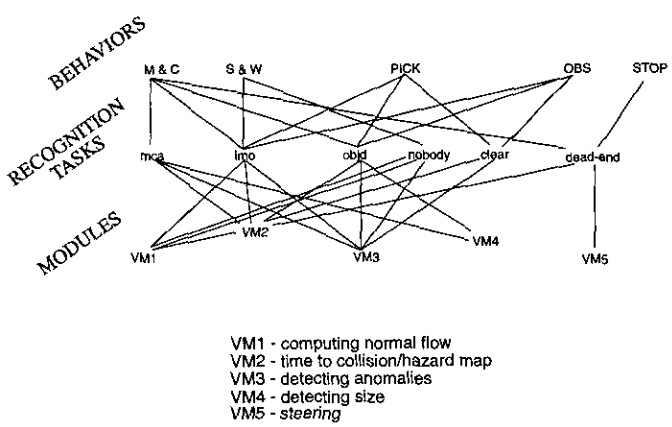

FIG. 16. The architecture of Sisyphus: a hierarchical view.

[6] In CSP notation, the transition diagram of Fig. 17 can be expressed as follows:

$$M\&C = [mca :: MCA \| imo :: IMO \| objd :: OBJD \| dead\text{-}end :: DE]$$

$$PICK = [imo :: IMO \| clear :: CLEAR]$$

$$OBS = [imo :: IMO \| clear :: CLEAR]$$

$$S\&W = [nobody :: NOBODY]$$

[7] A state formula is a formula in $Variables_A$ that evaluates to *true* or *false* at each state $s \in States_A$. When we say that a state satisfies a state formula we mean that the state formula evaluates to *true* at that state. A state formula can have parameters, that is, variables which are not state variables.

[8] The first can use a threshold on velocities, the second can use time to collision, and the third can detect anomalies.

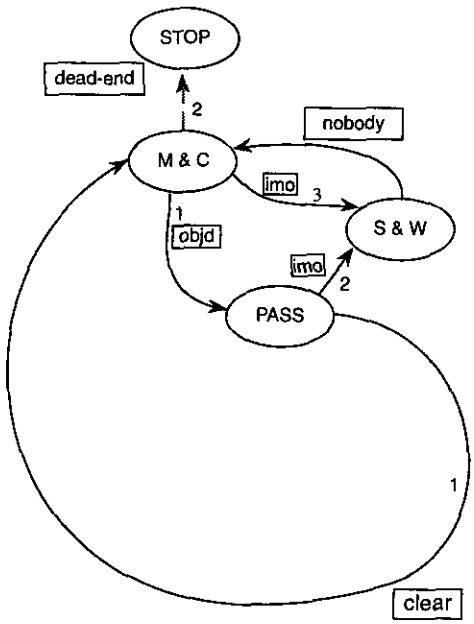

FIG. 17. Switching between behaviors: Sisyphus' behavior-transition diagram. Each node represents a distinct behavior; the arcs are recognition tasks.

states are changed "internally." Process S model the state of Sisyphus. S changes states by VM1, 2, 3, and this is why it will be constantly enabled.

VM1 state variables, initial conditions, and events:

$S_1 = \{moving, nobody\}$ Initially nobody.
vm1$_a$:
enabled $\equiv S_1 = moving$
action $\equiv S_1 \leftarrow nobody$; S_b
vm1$_b$:
enabled $\equiv S_1 = nobody$
action $\equiv S_1 \leftarrow moving$; S_a; S_e

VMi, i = 2,3 state variables, initial conditions, and events:

$S_i = \{moving, nobody, obst, clear\}$ Initially nobody.
vmi$_a$:
enabled $\equiv S_i = moving$
action $\equiv S_i \leftarrow nobody$; S_b
vmi$_b$:
enabled $\equiv S_i = nobody \vee S_i = obst \vee S_i = clear$
action $\equiv S_i \leftarrow moving$; S_a; S_e
vmi$_c$:
enabled $\equiv S_i = nobody$
action $\equiv S_i \leftarrow obst$; S_c
vmi$_d$:
enabled $\equiv S_i = obst$
action $\equiv S_i \leftarrow clear$; S_d

S state variables, initial conditions, and events:

$S = \{m\&c, s\&w, pass\}$ Initially m&c, an integer count initially 0;
S$_a$:
enabled $\equiv$ *true*
action $\equiv$ if $S = m\&c$ then $S \leftarrow s\&w$; *count*++
S$_b$:
enabled $\equiv$ *true*
action $\equiv$ count $-$; if $S = s\&w$ and count=0 then $S \leftarrow m\&c$
S$_c$:
enabled $\equiv$ *true*
action $\equiv$ if $S = m\&c$ then $S \leftarrow pass$
S$_d$:
enabled $\equiv$ *true*
action $\equiv$ if $S = pass$ then $S \leftarrow m\&c$
S$_e$:
enabled $\equiv$ *true*
action $\equiv$ if $S = pass$ then $S \leftarrow s\&w$; count++;

To illustrate the usefulness of this notation, we use it to prove the following safety property of Sisyphus: Whenever VM2 identifies an independently moving object, Sisyphus will enter an S&W state and will stay there as long as VM2 detects IMO. Denoting the VM2 state by S_2 and Sisyphus' state by S, we want to prove *Invariant*(A_0) where $A_0 \equiv$ *Invariant*$(S_2 = moving \Rightarrow S = S\&W)$. To prove this invariant we will use the rule stating that for a given system A that satisfies $inv(Q)$, it satisfies $inv(P)$ if $Q \Rightarrow P$ holds. We show that *Invariant*(A_1), *Invariant*(A_2), and because A_1, $A_2 \Rightarrow A_0$, *Invariant*(A_0) holds.

We define A_1 to be $A_1 \equiv (count \langle\rangle 0 \Rightarrow S = S\&W)$. We define A_2 to be $A_2 \equiv (S_2 = moving \Rightarrow count \langle\rangle 0)$. To prove *Invariant*$(A_1)$, *Invariant*$(A_2)$ we will use the following invariance rule:

$inv(P)$ is satisfied by system A if the following hold:

1. $Initial_A \Rightarrow P$
2. For every event e of A: $P \Rightarrow wp(P, e)$[9]

We start with A_2. Initially it holds (vacuously). We need to show that for every event e of the system $A_2 \Rightarrow wp(A_2, e)$. The only event that concerns us is **vm2$_b$**, but there the action part changes the count. A_1 holds initially (again vacuously). The only events that can change *count* are S_a, S_b, S_e. Both S_a and S_e set S to $S\&W$; S_b is conditioned on *count* = 0. So $inv(A_0)$ is proved. The ease of the proof is a result of the system's simplicity. Introducing counter or weighting schemas, adding states, or changing the level of atomicity would complicate the proof.

7. CONCLUDING REMARKS

Sisyphus is a simple illustration of a navigating agent for which objects can function as references (the corridor

[9] The notation *wp* denotes the weakest precondition. P is the weakest precondition of Q wrt S iff P is the largest set of states where the execution of S terminates with Q holding. The definition follows [18].

walls and ends), as things to be avoided (independently moving objects and obstacles), or as things to be "intercepted" (litter). (Sisyphus does not actually attempt to avoid independently moving objects, since this would require real-time motion control capabilities; instead, it simply "freezes" when such an object is detected. This behavior will result in avoidance, if we assume that the moving object will itself try to avoid Sisyphus.) A more detailed description of Sisyphus can be found in [14].

The functional classification of objects as things to be intercepted, avoided, or used as references appears to be adequate for an agent that navigates effortlessly in a simple environment and does not dynamically interact with objects. If we consider the dynamical aspects of navigation, or if the environment is more complex, these three classes of functionalities no longer constitute an adequate list. For example, in the case of a "real" navigating vehicle in a corridor we would need to be concerned with the vehicle's drive mechanism and how it interacts with the floor (which might have slippery spots or irregularities). Vehicles that have more versatile capabilities of locomotion can interact with the surfaces that they move on in a variety of ways (e.g., climbing), and they may also use other objects as aids to locomotion (e.g., a ladder, a bridge, . . .). We have also used a highly simplified classification of objects, primarily on the basis of their sizes and "dimensions"; we have not considered mechanical properties such as rigidity or movability, which would be very relevant to their roles as potential obstacles (e.g., consider a cloud of dust, a clump of tall grass, . . .). In general, objects can be classified, from a navigational standpoint, in terms of how they impede (or facilitate) the motion of an agent that has given locomotive capabilities. We are formulating a general theoretical framework for navigation tasks which will be described in a future paper.

REFERENCES

1. J. Y. Aloimonos, Purposive and qualitative active vision, in *Proceedings, DARPA Image Understanding Workshop, 1990,* pp. 816–828.
2. C. Bandle, *Isoperimetric Inequalities and Applications,* Pitman, London, 1990.
3. R. Basri and E. Rivlin, Localization using combinations of model views, in *Proceedings, International Conference on Computer Vision, 1993,* pp. 226–230.
4. R. A. Brooks and A. M. Flynn, Fast, cheap and out of control, A. I. Memo 1182, MIT, Cambridge, MA, 1989.
5. Z. Durić and Y. Aloimonos, Passive navigation: An active and purposive solution, Technical Report CAR-TR-482, Computer Vision Laboratory, Center for Automation Research, University of Maryland, College Park, 1991.
6. Z. Durić, A. Rosenfeld, and L. S. Davis. Egomotion analysis based on the Frenet-Serret motion model, in *Proceedings, International Conference on Computer Vision, 1993,* pp. 703–712.
7. R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis,* Wiley, New York, 1973.
8. Y. Ho, Dynamics of discrete event systems, *Proc. IEEE* **77,** 1989, 3–7.
9. D. J. Kriegman, T. O. Binford, and T. Sumanaweera, Generic models for robot navigation, in *Proceedings, DARPA Image Understanding Workshop, 1988,* pp. 453–460.
10. S. S. Lam and A. U. Shankar, A relational notation for state transition systems, *IEEE Trans. Software Eng., 1990.*
11. D. S. Mitrinovic, J. E. Pecaric, and V. Volenec, *Recent Advances in Geometric Inequalities,* Kluwer, Dordrecht, The Netherlands, 1988.
12. P. G. Mulgaonkar, L. G. Shapiro, and R. M. Haralick, Matching "sticks, plates and blobs" objects using geometric and relational constraints, *Image Vision Comput.* **2,** 1984, 85–98.
13. P. J. Ramadge and W. M. Wonham, The control of discrete event systems, *Proc. IEEE* **77,** 1989, 81–98.
14. E. Rivlin, Y. Aloimonos, and A. Rosenfeld, Purposive recognition: A framework, Technical Report CAR-TR-597, Computer Vision Laboratory, Center for Automation Research, University of Maryland, College Park, December 1991.
15. E. Rivlin and R. Basri, Localization and positioning using combinations of model views, *A. I. Memo 1376, MIT,* Cambridge, MA, 1992; Technical Report CAR-TR-631, Computer Vision Laboratory, Center for Automation Research, University of Maryland, College Park, July 1992.
16. E. Rivlin, A. Rosenfeld, and D. Perlis, Recognition of object functionality in goal-directed robotics, in *Proceedings, AAAI Workshop on Reasoning About Function, 1993.*
17. A. Rosenfeld and A. C. Kak, *Digital Picture Processing,* Academic Press, New York, 1982.
18. A. U. Shankar and S. S. Lam, Construction of network protocols by stepwise refinement, in *Stepwise Refinement of Distributed Systems* (J. W. de Bakker, W. P. de Roever and G. Rozenberg, Eds.), Springer-Verlag, Berlin, 1990.
19. A. Sloman, On designing a visual system, *J. Exp. Theoret. Artif. Intell.* **1,** 1989, 289–337.
20. L. Stark and K. Bowyer, Achieving generalized object recognition through reasoning about association of function to structure, in *Proceedings AAAI Workshop on Qualitative Vision, 1990,* pp. 137–141, 1990.
21. L. Stark and K. Bowyer, Indexing function-based categories for generic object recognition, in *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition, 1992,* pp. 795–797.
22. L. Stark and K. Bowyer, Indexing function-based categories for generic object recognition, *Comput. Vision Image Understanding,* to appear.
23. L. Stark, L. Hall, and K. Bowyer, An investigation of methods of combining functional evidence for 3-D object recognition, *Int. J. Pattern Recognit. Artif. Intell.,* to appear.
24. L. Stark, A. Hoover, D. Goldgof, and K. Bowyer, Function-based recognition from incomplete knowledge of shape, in *Proceedings IEEE Workshop on Qualitative Vision* (P. Kahn, Y. Aloimonos, and D. Weinshall, Eds.), *1993,* pp. 11–22.
25. M. Sutton, L. Stark, and K. Bowyer, Function-based generic recognition for multiple object categories, in *Three-Dimensional Object Recognition Systems* (A. Jain and P. Flynn, Eds.), Elsevier, Amsterdam, 1993.
26. L. Vaina and M. Jaulent, Object structure and action requirements: A compatibility model for functional recognition, *Intl. J. Intell. Systems,* **6,** 1991, 313–336.
27. E. L. Walker, Knowledge-based image understanding using incomplete and generic models, in *Proceedings, IEEE Conference on Computer Vision and Pattern Recognition,* 1993, pp. 699–700.