The Saphira Architecture for Autonomous Mobile Robots

Kurt Konolige
Karen Myers
Artificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025
{konolige,myers}@ai.sri.com

November 14, 1996

Abstract

Mobile robots, if they are to perform useful tasks and become accepted in open environments, must be *autonomous*: capable of acquiring information and performing tasks without programmatic intervention. Autonomy has many different aspects; here we concentrate on three central ones: the ability to attend to another agent, to take advice about the environment, and to carry out assigned tasks. All three involve complex sensing and planning operations on the part of the robot, including the use of visual tracking of humans, coordination of motor controls, and planning. We show how these capabilities are integrated in the Saphira architecture, using the concepts of coordination of behavior, coherence of modeling, and communication with other agents.

1 Autonomous Mobile Agents

What are the minimal capabilities for an autonomous mobile agent? Posed in this way, the question is obviously too broad; we would like to know more about the task and the environment: What is the agent supposed to do — will it just have a limited repertory of simple routines, or will it have to figure out how to perform complex assignments? Will there be special engineering present in the environment, or will the agent have to deal with an unmodified space? How will its performance be judged? Will it have to interact with people, and in what manner?

As has become clear from the mobile robot competitions at the last three NCAI conferences [32], the more restricted the environment and task (the less *open-ended*), the better mobile agents can perform. Designers are adept at noticing regularities, and taking advantage of them in architectural shortcuts. As a consequence, contest creators have become more subtle in how they define the rules, striving to reward mobile agents that exhibit more autonomy. To do so, they have had to grapple with refinements of the questions just posed. Although there may be no definitive answers, we can try to address these questions, like the contest creators, by articulating a scenario in which the autonomous agent must perform. To push our research, we have tried to make the environment and task as natural and open-ended as possible, given current limitations on the robot's abilities.

Fortunately, in designing a scenario we had outside help: in March 1994 we were approached by the producers of the science show "Scientific American Frontiers," who were interested in showcasing the future of robotics. After some discussion, we decided on a scenario in which our robot, Flakey, would be introduced to the office environment as a new employee, and then asked to perform a delivery task. To realize the scenario, Flakey would need at least the following capabilities:

Attending and Following. A supervisor would introduce Flakey to the office by leading it around and pointing out who inhabited each office. Flakey would have to locate and follow a human being. It would also have to know if a human was present by speech, e.g., going to an office door and inquiring if anyone was present.

Taking advice. Advice from the teacher would include map-making information such as office assignments and information about potential hazards ("There's a possible water leak in this corridor."). It would also include information about how to find people ("John usually knows where Karen is").

Tasking. Flakey would have to perform delivery tasks using its learned knowledge. The task was chosen to illustrate the different types of knowledge Flakey had: maps, information about office assignments, general knowledge of how to locate people. There was to be no engineering of the environment to help the robot; it would have to deal with offices, corridors, and the humans that inhabited them, without any special beacons, reflective tags, markers, and so on. Any machine-human communication would use normal human-human modalities: speech and gestures.

The scenario was made more difficult by three factors: there were only 6 weeks to prepare; the supervisor would have no knowledge of robotics (it was Alan Alda, the program host); and the scenario was to be completed in one day, so the robot hardware and software had to be very robust. We converged on this scenario because it was the most open-ended one we could think of that would be doable with current equipment and algorithms, and because it would hint at what future mobile agents would be like.

We believe that any mobile agent able to perform well in an open-ended scenario such as this one must incorporate some basic features in its architecture. Abstractly, we have labeled these the

three C's: *coordination*, *coherence*, and *communication*.

Coordination. A mobile agent must coordinate its activity. At the lowest level there are effector commands for moving wheels, camera heads, and so on. At the highest level there are goals to achieve: getting to a destination, keeping track of location. There is a complex mapping between these two, which changes depending on the local environment. How is the mapping to be specified? We have found, as have others [5, 10, 8, 2], that a layered abstraction approach makes the complexity manageable.

Coherence. A mobile agent must have a conception of its environment that is appropriate for its tasks. Our experience has been that the more open-ended the environment and the more complex the tasks, the more the agent will have to understand and represent its surroundings. In contrast to the reactivists: "the environment is the model" [4], we have found that appropriate, strong internal representations make the coordination problem easier, and are indispensable for natural communication. Our internal model, the Local Perceptual Space (LPS), uses connected layers of interpretation to support reactivity and deliberation.

Communication. A mobile agent will be of greater use if it can interact effectively with other agents. This includes the ability to understand task commands, as well as integrate advice about the environment or its behavior. Communication at this level is possible only if the agent and its respondent internalize similar concepts, for example, about the spatial directions "left" and "right." We have taken only a small step here, by starting to integrate natural language input and perceptual information. This is one of the most interesting and difficult research areas.

In the rest of this paper we describe out approach to autonomous systems. Most of the discussion is centered on a system architecture, called Saphira, that incorporates the results of over a decade of research in autonomous mobile robots. Our emphasis is on the integration of a variety of representations and reasoning techniques to achieve a complete mobile robotic system — one that can follow a person around, listen and respond to humans, learn a map of the environment, and navigate with dexterity and robustness.

2 The Robot Server

The Saphira architecture is designed to operate with a *robot server*, that is, a mobile robot platform that provides a set of robotic services in a standard format. This client/server paradigm abstracts Saphira from the particulars of any one robot, and enables us to port it readily to different robots, as long as they adhere to the server protocol.

The robot server is responsible for controlling the low-level operation of the motors based on commands from a client, and for operating the sensors and packaging the results to send back to the client. Figure 1 shows a typical server operating system. On a fast interrupt cycle, the server controls power to the motors and checks encoder readings. On a slower cycle, it manages the sonars and any other sensors or communications devices. On top of the lowlevel controls, the server implements a set of basic services.

- Movement control: forward/reverse velocity, angular heading. The server has *setpoints* for velocity and heading, and controls the motors on a fast servo cycle (10–50 ms) so that the robot maintains the setpoint values. The client sends commands to change the setpoints, thus controlling robot motion at a slower cycle (100 ms).

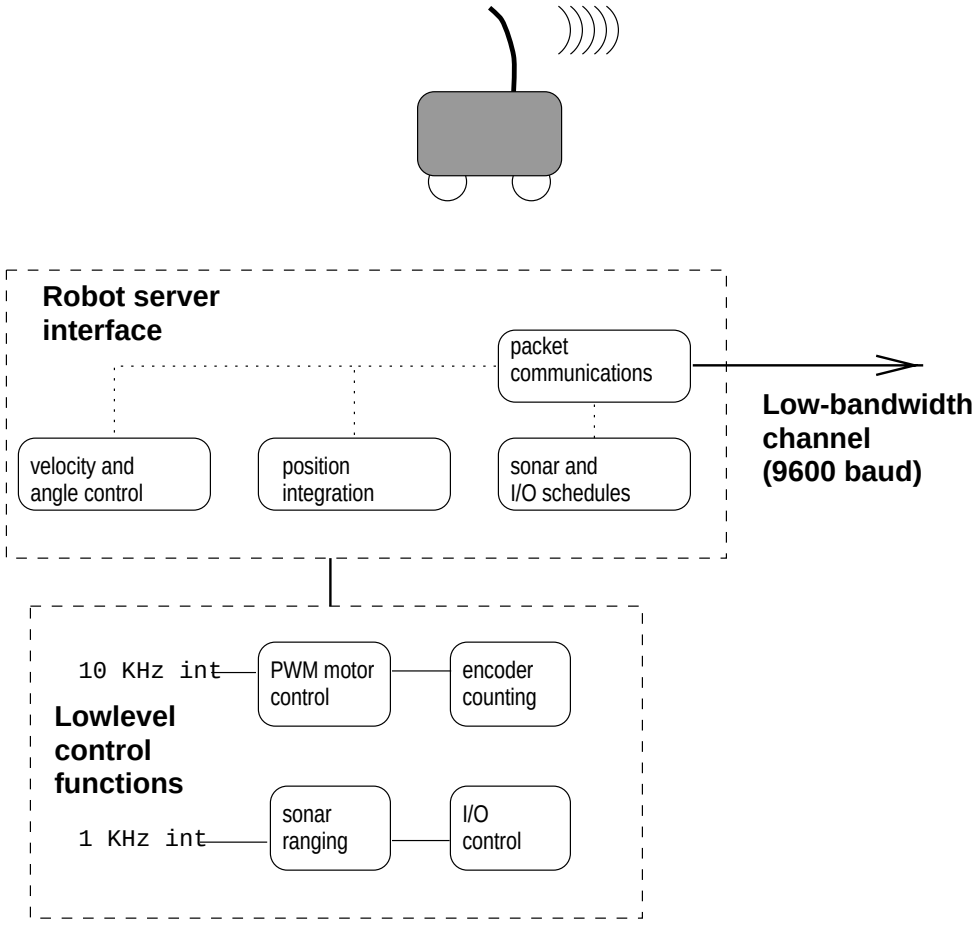


Figure 1: Server Operating System. Basic services include motor control, position integration, and sensor control. These services are derived from motor actuators and sensors on the robot.

- Position integration. The server updates the robot's internal dead-reckoned position using information from the encoders.

- Sonar and other sensors. The server manages the sensors, controlling the timing of the sonar and IR firing, and any other devices such as a camera pan/tilt head. The client can send commands to change the sensor schedules.

- Communication. The server sends an information packet to the client every 100 ms, containing information on the position, velocity, and sensor readings. It receives commands from the client to update its setpoint variables and sensor schedules.

By using setpoint control, the server can communicate with a client over a low-bandwidth channel with some latency, since the high-speed servo control of motors and sensors is done locally. For example, we have successfully controlled a robot by running Saphira on a remote host machine with a telephone connection to the site of the robot server.

With vision sensors, the amount of information processing required means that either all visual processing must be done on-board, or the bandwidth of the communication must be increased. We have used the former method, producing stereo range information for obstacle avoidance and people-tracking, and packaging the results into a small number of bytes for use by the Saphira client (see Section 3.2.3).

Most current mobile robot platforms can support the kind of server operations specified in the protocol. To date, Saphira has controlled four different robotics platforms.

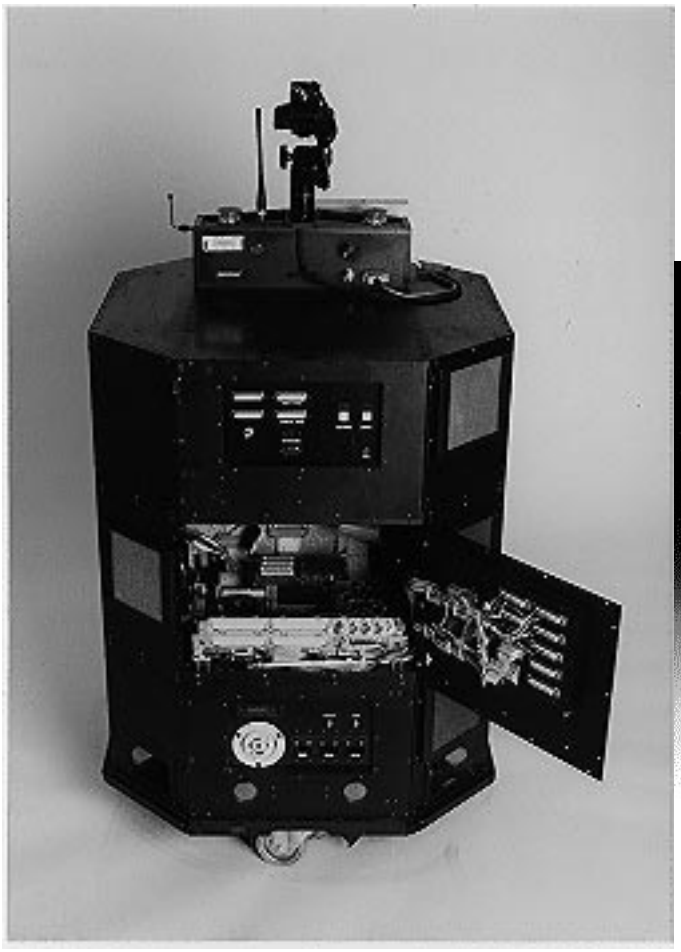
- Flakey (SRI International). This is a custom mobile robot platform first built in 1984 (Figure 2(a)). Its shape is an octagonal cylinder, approximately .5 meters in diameter and one meter high. Two driven wheels 7" in diameter are located on the sides, and passive casters balance it front-to-back. Flakey's sensors include a ring of 12 sonar sensors on the bottom, and a stereo camera pair mounted on a pan/tilt head. Flakey also has a speaker-independent continuous speech recognition system called CORONA, developed at SRI, and a standard text-to-speech program for speech output. The speech interface is integrated as part of the Saphira client, rather than the robot server.

- Erratic and Pioneer (SRI International and Real World Interface, Inc.). Erratic is a smaller version of Flakey with the same differential drive and sonar sensors, but without the vision capabilities (Figure 2(b)).[1] Pioneer is a commercial version of Erratic made by Real World Interface, Inc.

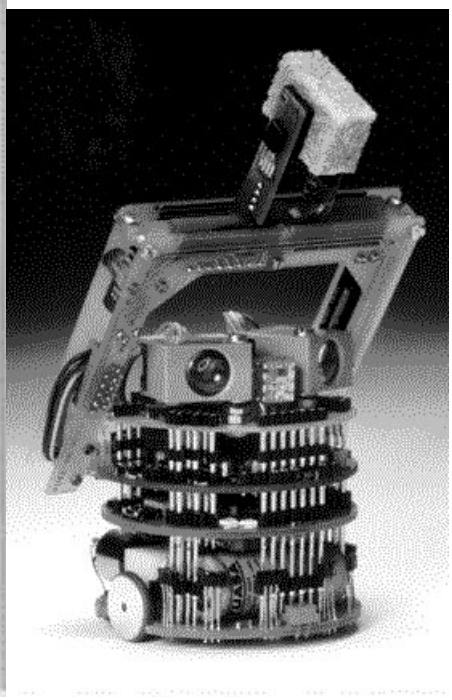
- Khepera (Swiss Federal Institute of Technology). Khepera is a tiny robot only 2 inches wide, with differential drive wheels and IR proximity sensors (Figure 2(c)). It also has add-on modules for vision and gripping.

- B14/B21 (Real World Interface, Inc.). These are commercially-made research mobile robots with a full range of sensing capabilities, including sonars, IR proximity, and vision systems. Instead of differential drive, they have a synchro-drive system in which three driven wheels turn in unison [13, p. 144].

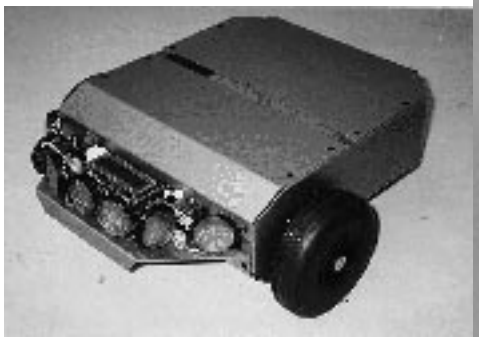
[1] As vision systems become smaller and use less power, there is a possibility of equipping small robots like Pioneer with interesting vision capabilities. Newton Labs of Washington has developed a small color vision sensor that extracts colored blobs from images in realtime, and we have used this system on Pioneer to find and approach soda cans. And SRI is developing a small realtime stereo vision system for Pioneer, similar to the one on Flakey.

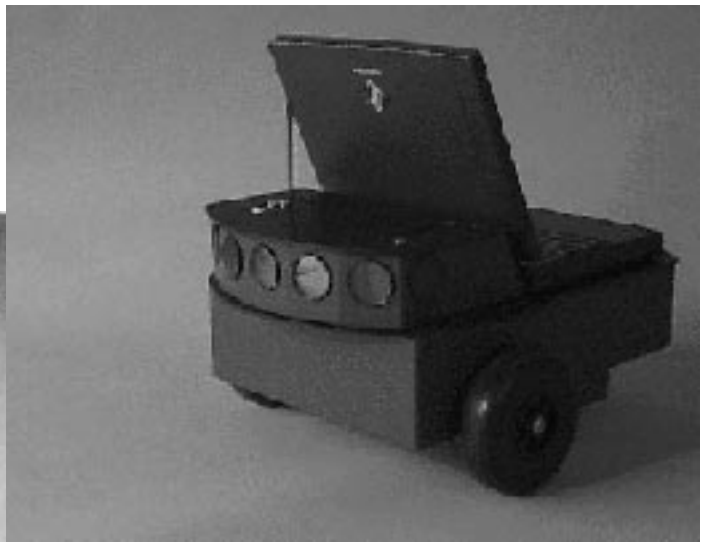


(a)



(c)





(b)

Figure 2: Saphira servers. (a) Flakey has a camera with stereo splitter on top, and a radio ethernet link. Sparcstation multiprocessor shown in the open compartment. Stereo sonars are the small discs around the bottom. (b) Erratic and Pioneer are two smaller versions of Flakey have the same motion and sonar capabilities, but no vision system. (c) Khepera is a tiny robot with differential drive, IR sensors, and an optional gripper.

The Saphira client, which controls the high-level operation of the robot, can exist either on the robot itself, or off-board on a host computer. The smaller robots typically have room only for the server, and the Saphira client runs on another off-board host computer. Pioneer is an interesting intermediate case: Saphira can run on a laptop on top of the robot, or on an off-board host connected by a radio modem.

On Flakey and the B14/B21 robots, all systems run on-board. Flakey has a 2-processor Sparcstation configuration, with one processor dedicated to the server and the vision system, the other to Saphira and a speech understanding system.[2]

Both Flakey and Erratic performed well in past AAAI robot competitions in 1992–1994 [7, 15, 16].

2.1 Stereo Vision System

While sonars and IR proximity sensors are useful for obstacle avoidance and low-resolution surface extraction, passive vision produces much more data and is potentially the best sensing modality for mobile robots. On the down side, visual images must be processed to produce useful information such as range or recognized objects, and such processing is expensive and unreliable. As processing costs go down, we can expect many more mobile robots to use visual sensing as their primary modality.

The vision system on Flakey consists of two cameras synchronized to capture stereo images. These images are input to a stereo algorithm using the *census method* [35], which results in full-frame dense stereo maps at a rate of about 2.5Hz. Figure 3 shows a stereo pair and the results of the stereo algorithm on the pair. The stereo results are coded so that closer objects are whiter.

The stereo results can be interpreted directly as range information, and we use them in several ways.

1. To identify surfaces that could be possible obstacles by matching their height against the ground plane.

2. To find and track person-like objects.

A short description of the tracking algorithm is given in Section 3.2.3.

3 The Saphira Architecture

The Saphira architecture [30, 29, 7] is an integrated sensing and control system for robotics applications. At the center is the LPS (see Figure 4), a geometric representation of space around the robot. Because different tasks demand different representations, the LPS is designed to accommodate various levels of interpretation of sensor information, as well as *a priori* information from sources such as maps. Currently, the major representational technologies are:

- A grid-based representation similar to Moravec and Elfes' occupancy grids [21], built from the fusion of sensor readings.

- More analytic representations of surface features such as linear surfaces, which interpret sensor data relative to models of the environment.

[2] At the time of the demonstration, Flakey had a single on-board processor, which ran the vision algorithms and basic motor control. The rest of the work was done by an off-board Sparcstation connected through a radio Ethernet.

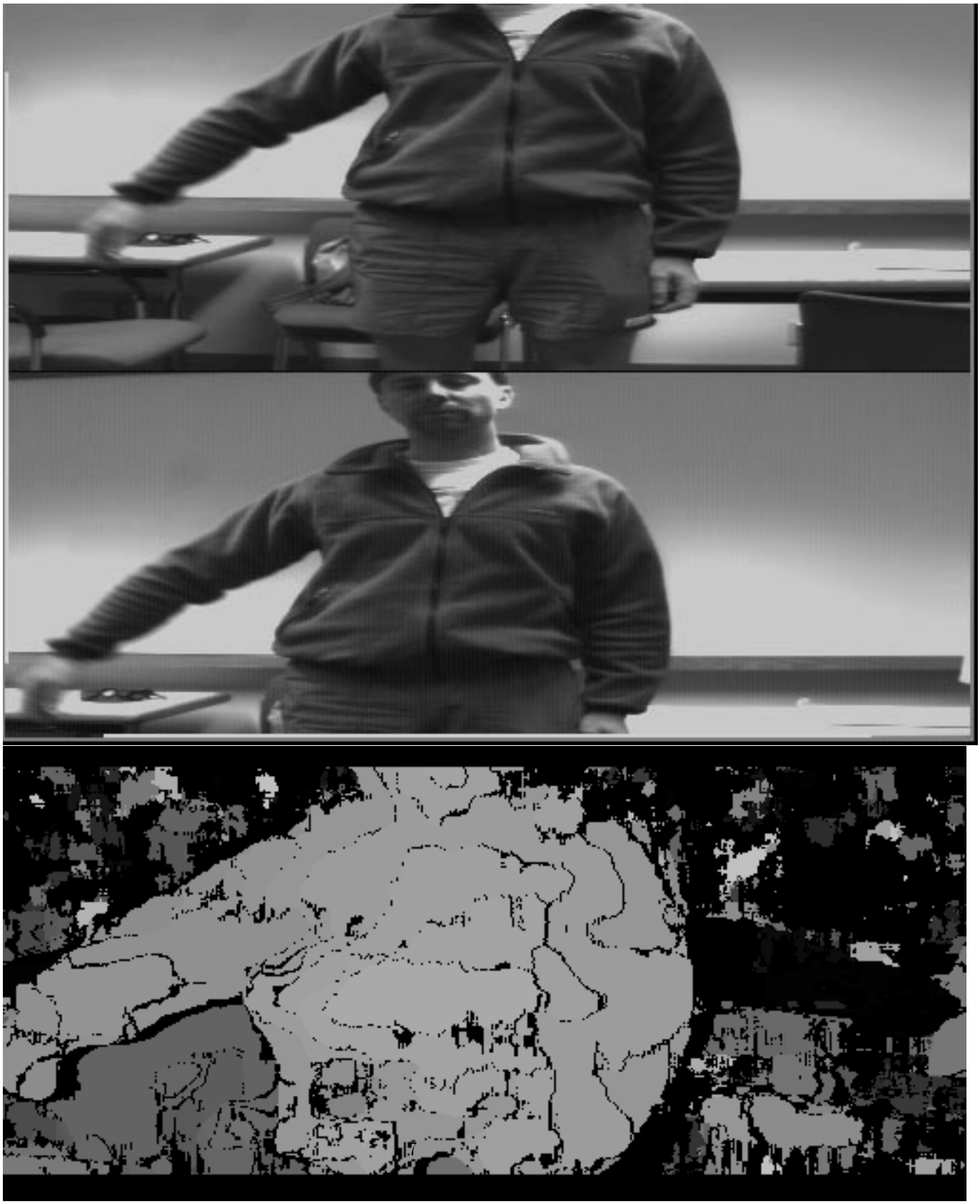


Figure 3: The top picture is a stereo pair from Flakey's cameras. The images are 640x240 pixels. The bottom pair shows the results of the census stereo algorithm: closer areas are coded white, farther areas black. Census algorithms are from John Woodfill of Interval Research Corporation.

- Semantic descriptions of the world, using structures such as corridors or doorways (*artifacts*). Artifacts are the product of bottom-up interpretation of sensor readings, or top-down refinement of map information.

The LPS gives the robot an awareness of its immediate environment, and is critical in the tasks of fusing sensor information, planning local movement, and integrating map information. The perceptual and control architecture make constant reference to the local perceptual space. The LPS gives the Saphira architecture its representational coherence. As we will show in Section 3.2, the interplay of sensor readings and interpretations that takes place here lets Saphira constantly coordinate its internal notions of the world with its sensory impressions. One can think of the internal artifacts as Saphira's *beliefs* about the world, and most actions are planned and executed with respect to these beliefs.

In Brooks' terms [3], the organization is partly vertical and partly horizontal. The vertical organization occurs in both perception (left side) and action (right side). Various perceptual routines are responsible for both adding sensor information to the LPS and processing it to produce surface information that can be used by object recognition and navigation routines. On the action side, the lowest level behaviors look mostly at occupancy information to do obstacle avoidance. The basic building blocks of behaviors are fuzzy rules, which give the robot the ability to react gracefully to the environment by grading the strength of the reaction (e.g., turn left) according to the strength of the stimulus (e.g., distance of an obstacle on the right).

More complex behaviors that perform goal-directed actions are used to guide the reactive behaviors, and utilize surface information and artifacts; they may also add artifacts to the LPS as control points for motion. At this level, fuzzy rules blend possibly conflicting aims into one smooth action sequence. Finally, at the task level complex behaviors are sequenced and their progress is monitored through events in the LPS. The horizontal organization comes about because behaviors can choose appropriate information from the LPS. Behaviors that are time-critical, such as obstacle avoidance, rely more on very simple processing of the sensors because it is available quickly. However, these routines may also make use of other information when it is available, e.g., prior information from the map about expected obstacles.

3.1 Behaviors

At the control level, the Saphira architecture is behavior-based: the control problem is decomposed into small units of control called *basic behaviors*, like obstacle avoidance or corridor following. One of the distinctive features of Saphira is that behaviors are written and combined using techniques based on fuzzy logic (see [30] and [28] for a more detailed presentation). Figure 5 shows the main components of behavior processing in Saphira.

Each behavior consists of an *update function* and a set of fuzzy rules. The purpose of the update function is to extract information from the LPS and turn it into a set of fuzzy variables appropriate for the behavior. For example, an obstacle-avoidance behavior might have the following variables, indicating where the robot's path is blocked:

front-left-blocked
front-right-blocked
side-left-blocked
side-right-blocked

Each fuzzy variable takes a value from the interval [0..1], indicating the degree to which its condition holds. Fuzzy variables are computed using a *transfer function* from the state of the LPS to the

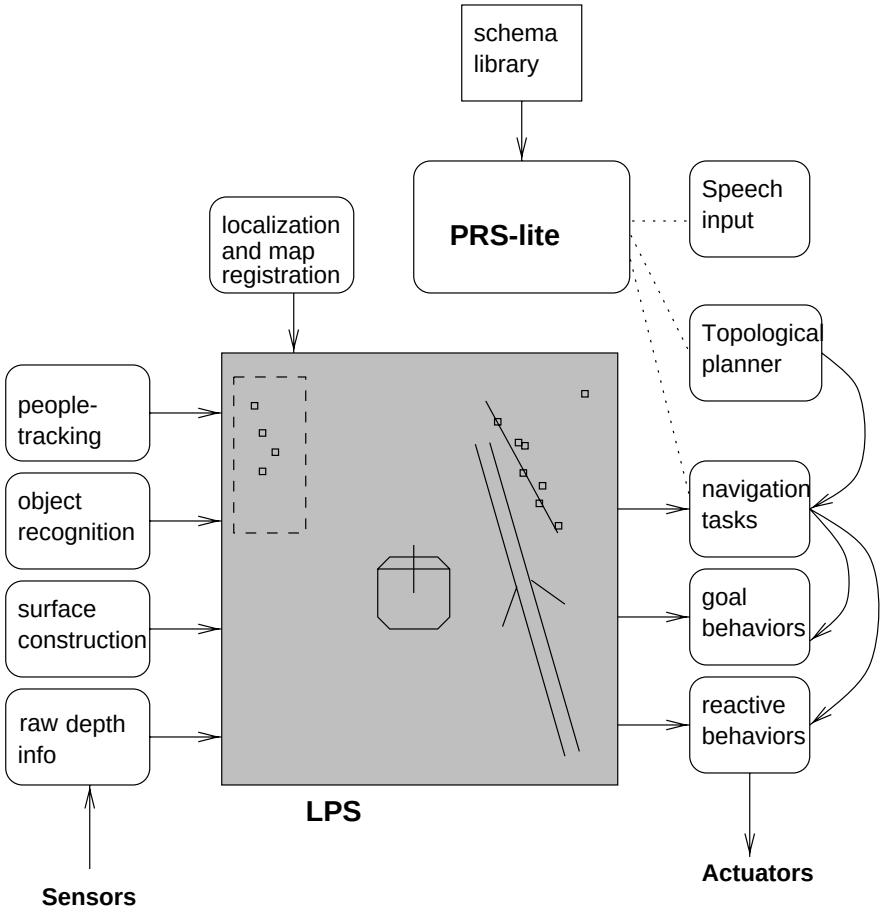


Figure 4: Saphira system architecture. Perceptual routines are on the left, action routines on the right. The vertical dimension gives an indication of the cognitive level of processing, with high-level behaviors and perceptual routines at the top. Control is coordinated by PRS-Lite, which instantiates routines for navigation, planning, execution monitoring, and perceptual coordination.

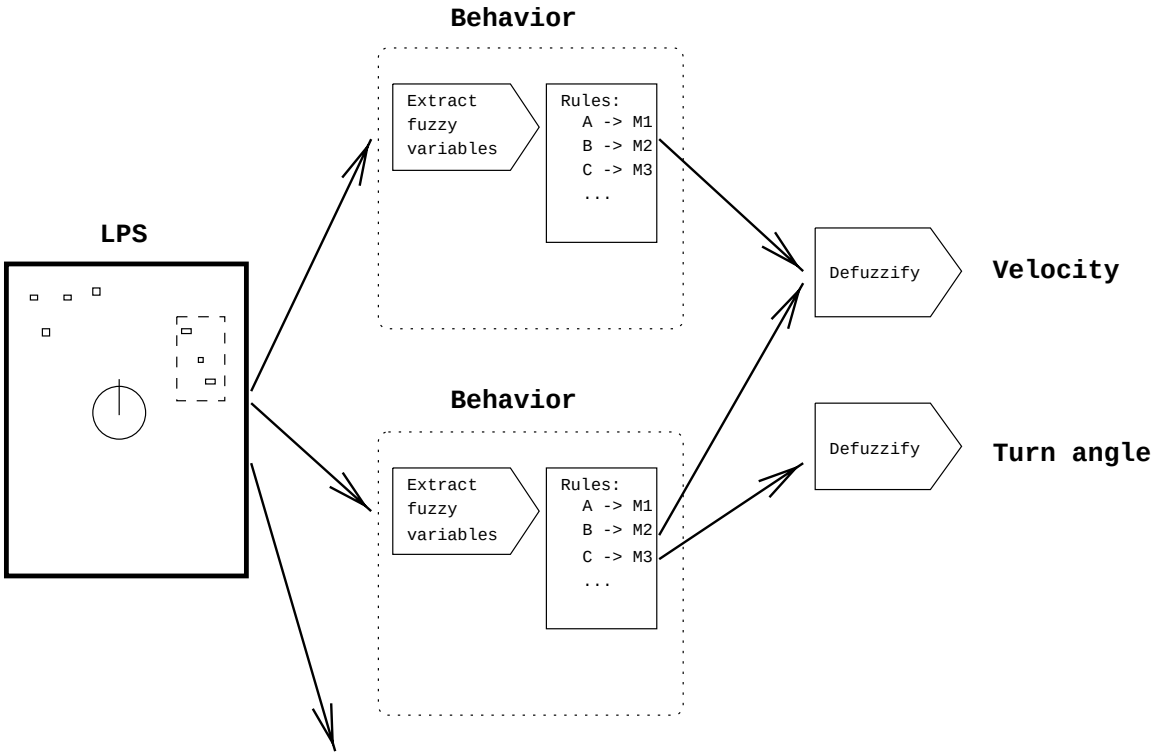
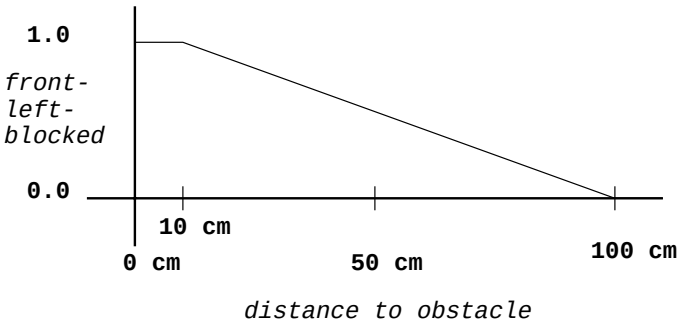


Figure 5: Fuzzy behaviors. Each behavior contains a function for calculating appropriate fuzzy variables from the LPS, and a set of rules that give the desirability of possible actions. The outputs of the behaviors are combined and a control value is chosen for each control channel.

variable. Suppose we consider the left front of the robot to be completely blocked if there is an obstacle 10 cm away, and unblocked if there is nothing within 100 cm. In between we would like the value of *front-left-blocked* to be intermediate between 0 and 1. Its transfer function looks like this:



The fuzzy variables are the inputs to a set of fuzzy control rules of the form

$$A \rightarrow C\,,$$

where A is a fuzzy formula composed by fuzzy predicates and the fuzzy connectives AND, OR and NOT, and C is a control action. For example, the rules for obstacle avoidance could be written as:

front-left-blocked AND (NOT *front-right-blocked*) AND (NOT *side-right-blocked*)	→	turn right
front-right-blocked AND (NOT *front-left-blocked*) AND (NOT *side-left-blocked*)	→	turn left
front-right-blocked OR *front-left-blocked*	→	slow down

Note that these rules refer to both the heading and velocity of the robot. In general, the robot may have many *control channels*, each with the ability to perform some motion function. Flakey and all robot servers for Saphira have at least the ability to turn to a heading and maintain a velocity; other functions may include grippers or manipulators, pan/tilt heads for a camera, and so on. A behavior can affect as many of these control channels as it needs to, while leaving the others alone.

The results of a rule application is a *desirability function*, which states how much the behavior would like to see a particular control value. For each control channel, the results of all the behavior rules for that channel are averaged to give the final control value. The averaging includes a weighting function based on two properties: priority of a behavior and its activation context. Behaviors that are more important are given higher priorities, e.g., avoiding obstacles has a higher priority than wall-following, since we don't want the robot to bump into anything. But letting the collision-avoidance behavior have complete control over the robot, even when there are no obstacles, is not reasonable, since in this context it is wall-following that should take precedence. So while priorities are fixed, the context of activation changes, and the controls of the behaviors are blended according to the priority of the behavior and how active it is. We call this weighting *context-dependent blending*, since it takes into account how much a behavior should respond in a particular context.[3] Figure 6 shows an example of context-dependent blending in operation as the robot follows a wall and avoids an obstacle.

Context-dependent blending has proven to be an effective technique for coordinating reactive and goal-oriented behaviors. Behaviors are not just switched on and off: rather, their preferences

[3] For averaging to make sense, the rules in a behavior should not suggest dramatically opposite actions in the same state. Our coding heuristic has been to make sure that rules with conflicting consequents have disjoint antecedents. Other authors have preferred to use more involved choice functions (e.g., [34]).

Figure 6: Context-dependent blending of following and avoiding behaviors. After point (a), the Keep-Off behavior starts to become active, and its higher priority dominates the Follow behavior. At point (b), the obstacle is passed, and Follow resumes.

are combined into a tradeoff desirability. An important consequence is that the preferences of the goal-seeking behaviors are still considered during reactive maneuvers, thus biasing the control choices toward the achievement of the goals. In the example above, suppose that the obstacle is right in front of the robot (e.g., the robot ended up facing the wall) and can thus be avoided by either turning right or left; then, the combined behavior prefers the side that better promotes corridor following.

It is interesting to compare context-dependent blending with the artificial potential field technique, first introduced by Khatib [14] and now extensively used in the robotic domain [17], [2]. In the potential field approach, a goal is represented by a potential measuring the desirability of each state from that goal's viewpoint. For example, the goal of avoiding obstacles is represented by a potential field having maximum value around the obstacles; and the goal of reaching a given location is represented by a field having minimum value at that location. At each point, the robot responds to a pseudo-force proportional to the vector gradient of the field. Although there are some technical differences, in general the fuzzy rule approach can also be seen as generating a *desirability gradient* showing the best direction for the robot to move at each point in the field. The main difference, however, is in how the complex control problem is decomposed. The fuzzy control approach allows one to partition control actions according to desired behaviors of the robot, and then combine the behaviors via context-dependent blending. By contrast, the potential-field method forces one to think about how to change a global potential field to achieve a particular result, conceptually a much harder task. Interestingly, the motor schema approach [2] brings a behavior orientation to potential fields, by decomposing the global field into a set of sub-fields, each designed to accomplish a particular goal.

3.2 Coherence

Reactive behaviors such as obstacle avoidance often can take their input directly from sensor readings, perhaps with some transformation and filtering. More goal-directed behaviors can often benefit from using *artifacts*, internal representations of objects or object configurations. This is especially true when sensors give only sporadic and uncertain information about the environment. For example, in following a corridor, a robot will not be able to sense the corridor with its side sonars when traversing open doorways or junctions. It would be foolish to suspend the behavior at this point, since over a small distance the robot's dead-reckoning is good enough to follow a "virtual corridor" until the opening is passed.

In other situations, an artifact may represent an artificial geometric entity that guides the behavior. Such situations occur frequently in human navigation, e.g., in crossing a street one tends to stay within a lane defined by the sidewalks on either side, even when there is no painted crosswalk. Similarly, in the follow-corridor behavior, the robot is guided by a lane artifact that is positioned a

foot or so in from the corridor walls.

In accordance with these behavioral strategies, artifacts in Saphira come from three sources:

- From *a priori* information. Typically, the robot will start with a map of the corridors and offices in its environment.
- From perceptual features. When a perceptual process recognizes a new object, it may add that object to the list of artifacts.
- Indirectly, from other artifacts or goal information. For example, if the user gives the command, "Move 3 feet forward," a goal artifact is created at a position three feet in front of the robot.

Artifacts always have a class (WALL, CORRIDOR, POSITION, etc), geometric information about their position and extent, and a unique identity, so that no two artifacts in the same class are the same. They may also have information about geometrical dependencies with other artifacts.

Normally, artifacts in the LPS are updated based on the robot's dead-reckoning mechanism, which is reliable only over short distances. *Coherence* is the property of updating artifact positions in the LPS based on perception, so that the robot's model of the environment stays registered with the robot's position as it moves. To understand how this works, we refer to the following diagram:

$$\text{feature} \iff \text{object hypothesis} \iff \text{artifact}$$

Features are constructs based on sensor information, usually representing surface information. A typical feature would be a linear surface, represented by a straight segment in the LPS.

An *object hypothesis* is a set of features that could correspond to a real-world object. For example, two breaks in a linear surface could be a doorway, and these two features might be grouped together to form a doorway hypothesis. Object hypotheses do not have any particular identity, i.e., a doorway hypothesis does not have information that it is the doorway to a particular office.

All three types of representations — features, object hypotheses, and artifacts — coexist in the LPS. As the diagram implies, there is no strict relationship in how one is created or manipulated by another. Depending on the task, it is possible to go in several different directions. In map-building, for example, features are grouped into object hypotheses, which are then recognized as artifacts. Although this process is mostly bottom-up, there is also an important top-down component, in which previously-recognized artifacts are matched against current hypotheses, and only those hypotheses which refer to possible new objects are given the status of artifacts.

In practice, we have found that the introduction of artifacts greatly simplifies the design of behaviors, by allowing us to decouple the problem of control from the problems of interpreting noisy sensor data. Our behavior-writing methodology has been to first write small rulesets for elementary types of movements based on simple artifacts, like follow a line, or reach a location; and then focus on the strategies to keep these artifacts anchored to the right features in the environment. The resulting behaviors often proved to be more robust than purely reactive controllers.

In this paper, we cannot describe all of the ways in which the feature to artifact connection is made. We will concentrate on two areas: feature extraction from perceptual information, and the process of *anchoring*, in which current artifacts are kept coherent with the environment by matching against features or object hypotheses.

3.2.1 Extracting Features

To navigate through extended regions, Saphira uses a global map that contains imprecise spatial knowledge of objects in the domain, especially walls, doorways, and junctions of corridors. Using

a map depends on reliable extraction of object information from perceptual clues, and we (as well as others) have spent many frustrating years trying to produce object interpretations from highly uncertain sonar and stereo signatures (see, for example, [6, 18, 21]). The best method we have found is to use extended aperture sonars readings, perhaps augmented with depth information from the stereo system. As Flakey moves along, readings from the side sonars are accumulated as a series of points representing possible surfaces on the side of the robot. This gives some of the resolution of a sensor with a large aperture along the direction of motion. By running a robust linear feature algorithm over the data, we can find walls segments and doorways with some degree of confidence.

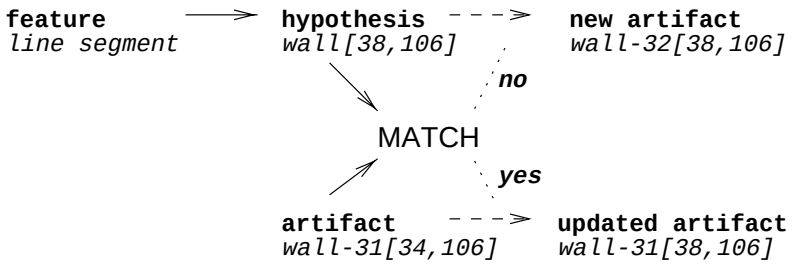
The utility of this technique is that it yields reliable linear features with almost no false positives. False positives are difficult to deal with, because using them for localization puts the robot at the wrong position in its internal map, and subsequent matches against features will fail. In general it's better to miss a few features and err on the conservative side in accepting features, since dead-reckoning works reliably for short periods.

Extracting wall and doorway features makes it easy to build a global map automatically, by having Flakey explore an area. The map is imprecise because there is error in the dead-reckoning system, and because the models for spatial objects are linear, e.g. corridors are represented as two parallel, straight lines. As features are constructed they can be combined into object hypotheses, matched against current artifacts, and promoted to new artifacts when they are not matched. In practice, we have been able to reliably construct a map of the corridors in the Artificial Intelligence Center, along with most of the doorways and junctions. Some hand editing of the map is necessary to add in doorways that were not found (because they were closed, or the robot was turning and missed them), and also to delete some doorway artifacts that were recognized because of odd combinations of obstacles.

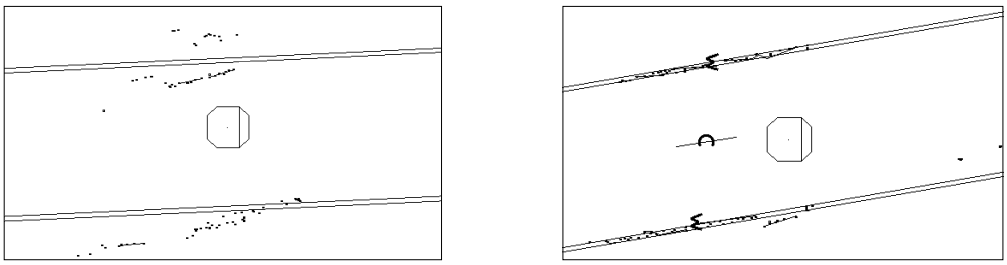
3.2.2 Anchoring

Artifacts exist as internal representations of the environment. When the physical object that an artifact refers to is perceived by the sensors, we can use this information to update the position of the artifact with respect to the robot. This is necessary to guarantee that behavior using the artifact operate with respect to the actual object, rather than with respect to an *a priori* assumption. We call *anchoring* the process of (1) matching a feature or object hypothesis to an artifact, and (2) updating the artifact using this perceptual information (see [27] for more on anchoring).

In Saphira, the structure of decision-making for the anchoring problem takes the following form:



As features are perceived, Saphira attempts to convert them to object hypotheses, since these are more reliably matched than individual features. These hypotheses are matched against artifacts existing in the LPS. If they match against an artifact, the match produces information for updating (anchoring) the artifact's position. If not, they are candidates for inclusion as new artifacts in the map.



(a) (b)

Figure 7: Anchoring a corridor artifact to sensor readings for corridor-following.

If an artifact that is in view of the perceptual apparatus cannot be matched against an object hypothesis, then Saphira tries to match it against individual perceptual features. This is useful, for example, when the robot is going down a hallway and trying to turn into a doorway. Only one end of the doorway is initially found because the other end is not in view of the side sonars. This information is enough to anchor the doorway artifact, and allow the robot to proceed with the door-traversing behavior.

Artifacts that have no perceptual support when they should (i.e., when they are in the range of the cameras or sonars), are candidates for reaping from the LPS. In general the reaping process must take into account the environment of the robot. For example, walls and corridors are fixed structures that are always present, while doors can be closed (and therefore not show up as features). The most obvious candidates for reaping are transient objects such as obstacles. At present we do not have any general theory of how to represent the dynamic aspect of objects and its relation to artifact recognition, but rely on special-purpose algorithms for each type of object.

Figure 7 shows an example of anchoring in Saphira. The picture shows the LPS in two consecutive moments during corridor following. The corridor-following behavior acts with respect to a corridor artifact, represented by the two double lines. This artifact is initially placed by the planning and executive levels based on map information (a). Note that the position of the artifact does not correspond to that of the actual wall, visible from the clusters of sonar readings (small dots): this may be because the map is incorrect or, more commonly, because of the inaccuracy of self-localization. As enough sonar readings are gathered, Saphira's perceptual routines infer the existence of two parallel walls (marked by "W") and hence form a corridor hypothesis ("C"). The artifact is then anchored to this hypothesis (b), and the movement now proceeds with respect to the actual corridor. If one wall is obscured by obstacles or a doorway, then the other will be used to anchor the corridor. Anchoring provides a closed-loop response whenever the relevant sensor data are available. When data are not available, e.g., if the walls become obscured, artifacts act as assumptions for movement.

Currently, anchoring is successfully used by individual behaviors or activity schemas that control behavior. For example, it is used by the doorway-traversing behavior to track the location of the doorway as it moves into or out of a room. It is also used to keep the robot globally registered with respect to a map. The registration process is sufficiently robust that the original map the robot makes does not have to be very precise (and it can't be, if dead-reckoning is poor). The anchoring process will keep the robot's position updated with respect to the relevant objects in the environment. This assumes, of course, that the robot can find and match objects of the right sort. For instance, going down a long corridor with no features, the robot will stay correctly registered in the center of the hallway, but the error in its longitudinal position will grow. So it's important

to find doorways or breaks in the corridor at reasonable intervals. Using the anchoring strategy on corridors, doorways, and junctions, we have been able to keep the robot localized for arbitrarily long periods (i.e., until the batteries run low) as it navigates the corridors of the Artificial Intelligence Center.

Anchoring thus helps to correct uncertain prior information by using perception, and keep the robot localized on a map as it navigates. But there are still several problems we have not addressed with this scheme. One is that sonar percepts are, in general, not adequate to keep the robot localized in complex environments. For example, in crowded rooms, the robot quickly becomes confused because it cannot find a sufficient set of long linear segments to match. To solve this problem, we are considering correlation-based algorithms using the stereo vision system. By keeping track of a number of small surface patches that are sufficiently distinctive as features, we hope to be able to use the same registration algorithms to reliably localize the robot in complex indoor and outdoor environments.

A second problem is that of *place recognition*: localization when the robot has no knowledge of its current position. Alessandro Saffiotti, who worked with the authors on Flakey's fuzzy behaviors, has ideas for solving this task based on the use of fuzzy sets to represent locational uncertainty [31, 28].

3.2.3 Tracking people

Saphira incorporates a simple people-tracker based on stereo information. The tracking algorithm does not perform correlations on successive temporal frames to keep track of the object. Rather, it looks for person-like objects in each successive frame without reference to previous ones, formulates a hypothesis, and then passes it to the LPS registration routines. "Person-like objects" are detected by a matching process running on the stereo data. Three bands of range information at torso height are examined for a shape profile the width of an adult. If two of the three bands agree, then a person is detected at the center of the shape. Because we are only using a portion of the stereo information, we can run the person-tracker at a 10 Hz rate.

The registration algorithm makes a decision about whether the hypothesized person is the same as one represented by an artifact, and either updates the artifact position or creates a new one. We keep a simple velocity model of person artifacts, which helps the registration process to make better matching decisions. It is also used by a *centering* process which can keep the camera center locked on the person being tracked. If a person artifact does not receive support for a period of time, it is removed from the LPS.

Interestingly, the registration process makes it possible to follow people around corners and through doorways, where they may be temporarily lost from sight. The artifact represents their most likely position, and the vision routines keep searching this area until the person is re-acquired, or cannot be found.

One limitation of the current system is its inability to distinguish different people. The range resolution of the system is coarse enough so that distinguishing people by shape is not possible, even without the complication of motion, arm position, and so on.

3.3 Controlling Executive: PRS-lite

Behaviors provide low-level situated control for the physical actions effected by the system. Above that level, there is a need to relate behaviors to specific goals and objectives that the robot should undertake. This management process involves determining when to activate/deactivate behaviors as part of the execution of a task, as well as coordinating them with other activities in the system.

PRS-Lite [24], a reactive controller based loosely on the Procedural Reasoning System (PRS-CL) [11, 23] fills this role within Saphira.

Many of the core capabilities provided by PRS-Lite are shared by most current-generation task-controllers. These include the smooth integration of goal-driven and event-driven activity, timely response to unexpected changes in the world, and the hierarchical decomposition of tasks. Several additional capabilities, however, distinguish PRS-Lite from other reactive control systems (including PRS-CL). These include the management of interacting continuous processes, a rich set of declarative control mechanisms, and a generalized goal semantics that replaces concepts of success or failure with levels of goal satisfaction.

3.3.1 Overview

The representational basis of PRS-Lite is the *activity schema*, a parameterized specification of procedural knowledge for attaining a declared objective. This procedural knowledge is represented as an ordered list of *goal-sets*, whose successive satisfaction will yield the overall objective of the schema. A goal-set is composed of one or more *goal statements* (or *goals*), each consisting of a goal operator applied to a list of arguments. A goal-set can be optionally identified by a label unique to its schema. Intuitively, a goal-set corresponds to an ordered sequence of goals that are to be achieved as an atomic unit. A compiler transforms the schema specifications into parameterized finite-state machines (performing optimizations where appropriate), whose arcs are labeled with individual goal-sets to be achieved. Activity schemas are launched by instantiating their parameters and *intending* them into the system. Such instantiated schemas are referred to as *intentions*. Multiple intentions can be active simultaneously, providing a multitasking capability.

Different modalities of goals are supported, as summarized in Figure 8. Broadly speaking, goal types can be categorized as *action* or *sequencing*. Action goals ground out in either executable functions (called *primitive actions*), tests on the state of the environment (as represented in the internal world model), or the activation/deactivation of intentions and behaviors. This last ability enables the hierarchical decomposition of goals. Sequencing goals provide conditional goal execution and sophisticated goal ordering beyond the default of linear processing, as well as various forms of parallelism.

Overall, PRS-Lite can be used to generate and manage a forest of directed graphs whose nodes each represent a goal-set from some activity schema. The root node of each graph represents a top-level goal, with its successors generated by hierarchical goal refinement. We refer to the set of active graphs at a given point in time as the current *intention structures* for the system. The leaf nodes of the intention structures are called the *current nodes*, and their associated goal-sets the *current goal-sets*. Note that an intention structure can have multiple current nodes because of the inclusion of parallel sequencing goals in the schema definition language.

An *executor* manages intentions at runtime. It repeatedly operates a short processing cycle in which it considers the current goal-sets in each intention structure, performs any actions required to achieve their constituent goals, and updates the set of current nodes (as appropriate). The decision to limit processing to a single goal-set for each leaf node in an intention structure ensures overall responsiveness to new events. Given the granularity of processing, responsiveness is dependent on the number of active intentions, the degree of parallelism in those intentions, the size of goal-sets, and the underlying primitive actions that are executed. The design has proven adequate for the tasks considered to date (as discussed later).

Action Goals:

Test	check a condition
Execute	execute a primitive action
=	assignment of local variables
Wait-for	wait for a condition
Intend	dispatch intentions (block/nonblock)
Unintend	terminate intentions

Sequencing Goals:

If	conditional goals
And	parallel goals (with join)
Split	parallel goals (without join)
Goto	branching to labeled goals

Figure 8: PRS-Lite Goal Modalities

3.4 Goal Modalities

Action goals supply the most basic operations in the system. **Test** goals provide the ability to test beliefs about the current state of the external world. Within Saphira, beliefs are characterized by a combination of the Local Perceptual Space and a set of environment variables. **Execute** goals provide for the execution of primitive actions, which may perform internal bookkeeping, the setting of environment variables, or the triggering of specific external actions by the system. External actions for Flakey include the generation of an utterance by the speech synthesis module, and the invocation of a route planner. = goals enable the binding of local variables within an intention.

Intend goals lead to the activation of both intentions and behaviors. As such, they enable the hierarchical expansion of intentions through repeated goal refinement. **Unintend** goals provide the complementary ability to explicitly terminate active intentions before they run their full course. This ability is critical when operating in dynamic, unpredictable domains, where rapid switching among activities is essential. Intentions can be assigned priorities that determine the order in which they are processed by the executor. Intentions can also be named when activated, allowing them to be referenced by other intentions (in particular, **Unintend** goals).

A critical feature of **Intend** is that it supports the invocation of intentions in either *nonblocking* or *blocking* mode. In nonblocking mode, the intention is activated and control proceeds to the next goal. In essence, the nonblocking intention is spawned as an independent process within the context of the parent intention; the nonblocking intention will persist either until it completes or its parent intention terminates. In contrast, blocking mode disables updating of the current goal within the parent intention until the child completes. Any intentions activated earlier by the parent intention will continue to be processed. Degrees of blocking are also supported: an intention can be blocked until a designated success criteria is satisfied. This capability is valuable for controlling behaviors implemented as fuzzy rules, which provide a natural metric for defining degrees of success (namely, the fuzzy predicates that model the world state). As a simple illustration, Flakey has a `face-direction` behavior that orients the robot toward a designated heading. This behavior is invoked with different thresholds of blocking in different contexts, depending on how critical it is to be precisely oriented to that heading.

Wait-for goals enable limited suspension of an intention until a certain condition or event occurs.

The **Wait-for** goal modality is critical in the framework in that it enables synchronization among concurrent intentions through the use of shared variables. An illustration is provided in the next section.

The sequencing goals enable more sophisticated goal-ordering and selection mechanisms than does the default of linear processing of goal-sets. Sequencing goals can be nested to arbitrary depths, yielding a rich framework for specifying control strategies. **If** goals support conditional activation of a goal. **Goto** goals support non-linear flow of control within an activity schema, by allowing a current goal-set to be mapped to any other labeled goal-set in the schema. Iteration can be specified through appropriate combinations of **If** and **Goto** goals. Two forms of parallelism are provided via the **Split** and **And** goal modalities. **Split** parallelism spawns sets of independent concurrent goals, with control in the parent intention proceeding to the successor goal-set. Each thread of activity for the spawned goals continues until it completes, or the parent intention terminates (similar in spirit to the nonblocking mode of intending). In contrast, **And** parallelism treats the parallel goals as a unit; processing of the parent intention suspends until each of the threads occasioned by the **And** subgoals terminates.

3.5 Methodology and Use

Goal-directed behavior is produced by intending schemas for satisfying individual tasks. Reactive, event-directed behavior is produced by launching intentions that employ **Wait-for** goals to suspend until some condition or event transpires.

A common idiom for the design of activity schemas is to define an umbrella intention for a specific objective, which in turn invokes both a lower-level intention for achieving the objective, and one or more 'monitor' intentions (thus combining goal- and event-driven activities). Monitors use **Wait-for** goals to detect changes in the world that could influence the actions required to achieve the overall objective of the top-level schema. Certain monitors identify failure conditions that would invalidate the approach being taken for the current task. Others provide reactivity to unexpected events that require immediate attention. Monitors can also check for serendipitous effects that eliminate the need for certain goals in active intentions, and modify the intention structures accordingly.

To illustrate the use of the various goal modalities and idioms, Figure 9 presents simplified versions of activity schemas used by Flakey to perform basic navigation tasks. The schema `:plan-and-execute` encodes a procedure for generating and robustly executing a plan to navigate to a designated destination. The destination is specified as a parameter to the schema, and is represented as an artifact in the LPS, thus linking abstract notions of place to the robot's beliefs about its environment.

The initial goal-set in the schema (with the label `:plan`) employs an **And** goal applied to three subgoals to perform certain initializations. The first **Execute** subgoal invokes a function `say` that performs a speech generation command. The second **Execute** goal initializes the environment variable `*failed-execution*`, which is used to encode information about the status of plan execution. This variable is an example of state information within PRS-Lite that provides coordination among intentions (as described further below). The final subgoal invokes a function `find-path` that produces a topological plan for navigating from the current locale to the destination. Navigation within Saphira is at the level of regions (doors, junctions, hallways); the route planner produces a sequence of such regions that should be traversed to reach the target destination.

After performing the necessary initializations, the schema intends a nonblocking monitor intention `:monitor-planex`, followed by a blocking intention `:follow-path`, in the spirit of the umbrella idiom described above. This latter intention (not shown here) cycles through the computed path, launching various lower-level intentions as required to navigate between successive regions in the generated path. The lower-level intentions may encounter difficulties, which they signal by setting the

```
(defintention :plan-and-execute
  :params (dest)
  :goals
  `((:plan
     (AND
      (EXECUTE (say "Planning path to ~a" dest))
      (EXECUTE (setq *failed-execution* nil))
      (= plan (find-path *cur-region* dest)))  )
    (IF (null plan) (GOTO :finale))
    (INTEND :monitor-planex () :blocking nil)
    (INTEND :follow-path ((path . plan))
    :blocking t :name follow-it)
    (IF *failed-execution* (GOTO :plan))
    (:finale
     (IF (null plan)
       (EXECUTE (say "No passable routes"))) )  ) )

(defintention :monitor-planex
  :params ()
  :goals
  `((:monitor (WAIT-FOR *failed-execution*))
    (:cleanup (UNINTEND 'follow-it)) ))
```

Figure 9: Activity Schemas for Directed Navigation

environment variable `*failed-execution*`. The **Wait-for** goal in the `:monitor-planex` intention would detect such an event, and then process the goal **(Unintend** `'follow-it`**)**. Satisfying this goal would deactivate the `:follow-path` intention, with the monitor intention then terminating. If no lower-level intention signals a failure, the blocking intention `:follow-path` will eventually complete, enabling processing of the remainder of the `:plan-and-execute` intention. The `:monitor-planex` intention is terminated automatically when its parent intention `:plan-and-execute` terminates.

Figure 10 displays a snapshot of the intention structures at a point during a run in which Flakey uses the above schemas to execute a delivery task.[4] Each line in the display consists of: an initial marker, indicating whether the intention is blocking (∗) or nonblocking (○), the name of the activity schema (*e.g.*, `Deliver-Object`), a unique identifier for the particular instantiation of the schema (*e.g.*, a label such as `Follow-It` if one was specified in the **Intend** goal, else an assigned name such as `I3674`), and either the next state of execution (for an intention) or `B` (for a behavior). At the instant captured by this display, PRS-Lite has two intentions active at the highest-level (corresponding to two distinct user-specified objectives): `Deliver-Object` and `Avoid`. The `Avoid` intention has only one active thread at this point, namely the behavior for avoiding collisions (`Avoid-Collision`). Note though that in the past or future, this intention may trigger many other activities. Of more interest is the state of execution for the `Deliver-Object` intention. At its topmost level, this parent intention

[4] To improve the understandability of the robot's actions, PRS-Lite maintains an *intention display* that summarizes the intention structures at the end of each execution cycle. The intention display provides a concise overview of the motivation for the actions being undertaken by the system at any point in time, thus conveying to an observer *why* the robot is behaving in a certain manner.

* Avoid (Avoid1) END
  * Avoid-Collision (Collide) B

* Deliver-Object (I3674) S171
  * Plan-And-Execute (I3675) S146
    - Monitor-Planex (I3676) CLEANUP
    * Follow-Path (Follow-It) S138
      * Corridor-To-Junction (I3677) END
        * Follow-To-Target (I3678) END
          - Follow (Follow-To-Target) B
          - Orient (Orient-To-Target) B
          - Keep-Off-With-Target (Keep-Off) B

Figure 10: Snapshot of the Intention Structures during Execution of a Delivery Task

has the single child intention `Plan-and-Execute`, which in turn is executing the `:follow-path` schema while simultaneously monitoring for execution failures (via `Monitor-Planex`). As part of the path-following schema, the robot is currently moving from a corridor to a junction, which in turn has activated an intention to move toward a specific target. At the lowest level, three behaviors are activate simultaneously, namely `Follow`, `Orient`, and `Keep-Off-With-Target`.

3.5.1 Discussion

PRS-Lite provides a powerful and natural framework in which to specify and manage the purposeful activities of a robot. The system itself is compact (<500 lines of LISP code, including executor, compiler, and display manager), especially in comparison to PRS-CL. It has proven to be sufficiently fast over a broad range of tasks, with the intention execution loop easily fitting into Saphira's overall cycle time of 100 ms. Precise figures for the cycle time of the PRS-Lite executor are not available, but the combination of behavior and task control lies somewhere in the range of 5 to 30 ms (on a Sparc-2 processor). This is very fast, considering that on average there are 10 to 15 intentions in operation, monitoring various conditions and coordinating behaviors.

One novel capability of PRS-Lite within Saphira is its ability to manage behaviors implemented as fuzzy rules. A key feature of such behaviors is the smoothness of activity that results from rule blending. However, sequencing languages (including activity schemas) model task-level events as discrete actions connected by explicit transitions, even when the tasks themselves ground out in continuous processes. This separation of task and behavior levels leads to discontinuities in the resultant activities of the system. As an illustration, consider the task of navigating to an office in a given corridor, and then entering it. Flakey has activity schemas defined for the navigation and doorway-crossing, each of which employs an appropriate set of behaviors. A straightforward approach to the overall task would be to execute a navigation intention to reach the office, and then a doorway-crossing intention to enter the office. But the termination of the corridor navigation intention prior to the activation of the doorway-crossing intention leads to jerky, unnatural motion by the robot. For smoother operation, the doorway-crossing behaviors must be activated before the robot reaches the office (i.e., while the corridor behaviors are still active). The corridor navigation behaviors should terminate once the robot is sufficiently far down the corridor to enable successful completion of the doorway-crossing intention. Such intention-level blending can be specified in the

PRS-Lite goal language, using a combination of monitors and thresholded blocking of intentions.

4 Communication

At this point, our ideas about communicating have been strongly influenced by the Natural Language Understanding community, especially the plan-based theory of speech acts [1]. Eventually, we would like to have a complete understanding system that would perform intention recognition on the speaker's utterance, and form appropriate plans. For the scenario, we concentrated on the area of referent identification, matching the speaker's terms to objects in the immediate environment of the robot. In this section we describe the speech input system, and the schemas that carry out advice-taking and tasking commands.

4.1 Speech input

We were fortunate to have an excellent speech-recognition system, developed at SRI, called CORONA. CORONA has speaker-independent recognition models that need no training, although it will have better accuracy with individually-tuned models. CORONA accepts a BNF grammar of phrases, and produces strings of the words spoken. On a Sparc 10-51, it operates at about twice realtime on the simple grammar we used, i.e., a 5-second sentence would take about 10 seconds to recognize.

One of the hardest problems in voice communication is letting the supervisor know when Flakey has heard an utterance, and what the state of its understanding is. We employed several techniques:

Keying. There can be a lot of extraneous chatter during interactions. The grammar was created with a keyword initiation, so that only phrases beginning with the word "Flakey" were recognized. This worked extremely well; it was natural to address commands and statements to the robot in this fashion. A nice additional feature would be to use directional interpretation of sound sources for getting Flakey's attention; we currently do not have this capability.

Process status. CORONA employs an "endpointer," a period of low speech energy, to signal the end of a speaker phrase. From this point there is a delay until the speech is processed; the speaker can be confused about whether his input was received, and whether it was recognized. We implemented a simple end-of-speech flag: Flakey says "um" when the endpoint is reached. Thus the speaker can tell that his input was received and is being processed.

Results. It is important to give feedback to the speaker about the interpretation of his input. For example, if the speaker says "turn left," and it is interpreted as "go forward" (a not altogether unknown occurrence), the speaker will be mystified by the robot's behavior. So Flakey would acknowledge each spoken command, either by paraphrasing it, or nodding its cameras in recognition. If the phrase was not recognized, Flakey said "duh?" or "what?" Crude, but effective.

4.2 Gestures

We were ambitious enough to want gesture recognition as part of the attention process. The idea was to use a mixture of speech and gesture for reference, e.g., "This office (*point to the right or left*) is Karen's." We did manage to extract enough information from the stereo system to recognize left or right-pointing arms, but did not have enough time to integrate it correctly with the speech input, even for simple reference. This would be a good project for future work, with a more elaborate natural-language understanding system.

We implemented a simple but surprisingly sufficient set of activity schemas to perform the scenario. They fall into several categories: direct motion commands ("turn around"), sensor-based movement ("follow me", "follow the corridor"), tasks ("get the file from Karen"), and information ("this is John's office").

4.3 Direct Motion

These are direct commands to move forward or turn, or to look (move the cameras) in certain directions, e.g., "look behind you." Direct motion commands are implemented as simple sequential behaviors with well-defined ending conditions and short timeouts. For example, if Flakey is told to move forward while facing a wall, it will turn away from the wall and move forward about 1 meter, since collision avoidance is active at all times. It would be smarter, of course, to recognize that it is not possible to move forward, and state this fact. But in our current implementation Flakey does not check for possibility of carrying out commands; it only tries to do the best it can.

4.4 Attending and Following

These commands involve coordination of sensing and acting, mediated by artifacts in the LPS. Even very simple activity schemas, when coordinated with sensing, can give the appearance of a well-trained robot with human-like capabilities.

The Attending schema finds the closest person-like object and brings Flakey to face the person at a distance of about 1 meter. The command "look at me" or "find me" triggers this schema. Flakey performs a scan from the current camera position to the extreme left and right position of the pan/tilt head, giving a full 360 degree field of view. The first person-like object detected by the stereo tracking algorithm (Section 3.2.3) is placed in the LPS, and an activity schema positions the robot to face the object and keep the cameras centered on it. Success of the Attending schema is indicated by saying "here I am;" failure, after a full scan, by "I can't find you!".

The Following schema uses the same kind of movement/sensor control to track a person. If there is no person currently in view, the Attending schema is invoked. Once found, the person is kept centered in the cameras, using the spatial information returned from the tracking algorithm. The activity schema keeps Flakey about 1 - 1.5 meters from the person as much as possible (top speed of 300mm/sec), while avoiding obstacles. If the person is lost, Flakey looks in the most likely area for awhile, but does not attempt to re-attend, since if there are other people around, it may acquire the wrong one. Flakey's vision system cannot distinguish individuals.

Both these behaviors worked well in practice. Mr. Alda was able to lead Flakey through several corridors and doorways at SRI, with only minor problems in re-acquisition.

4.5 Information

This is one of the most interesting areas for robot/human interaction. The hard problem is relating robot-centered representations of the world with human-centered ones. For example, when the supervisor says "..this office...", the robot must understand that there is an object in the current focus of dialogue that is being referred to. The point of contact, for the robot, is in the artifacts of the LPS. Flakey has enough background knowledge to infer that doors can lead to offices, even though it cannot recognize an office *per se*. So the phrase "this office" is linked to the nearest doorway in the robot's perceptual space.

In general the problem of reference resolution can be phrased in terms of abductive inference: find an object that, if assumed as the referent, would make the speaker's phrase carry reasonable

information [12]. For the scenario demonstration, we did not use a general inference method to determine reference, since we decided beforehand that we would only refer to offices, people, and certain other objects. But in general the problem of reference for robots will require some of the same tools that have been used in computational linguistics. However, robots do enjoy one advantage: since they also perceive the world, they can draw on a repertoire of perceived objects as referents (in philosophical terms, they are *situated* in the word). Instead, the problem becomes one of matching the artifacts of the robot's representation with the linguistic expressions of the speaker. We have formulated a theoretical foundation for this interchange [22]; but it was not applied in the scenario. Other more complex theories of knowledge and action have been developed in a logical framework [20, 19]; we expect to see more application of these theories as robots become more sophisticated in their interactions.

Another type of information used by the robot is knowledge of how to locate people, since delivery tasks often involve finding someone. In general this is a planning problem, with reasoning about where someone is, or who might have information about where that person is. For the scenario, we implemented a simple routine that would first check the person's office (using speech output and waiting for a reply), and then start looking for people who might know where the person is. More sophisticated routines might check a personal calendar, or compile a list of likely places, or perform inference about knowledge prerequisites for action, and so on. Here again the problem of reference emerges. Phrases such as "Karen is around the corner" would be hard to interpret, so we settled on a set of place-names that could be used: offices, corridors, and open areas such as the library. These were all places that Flakey could learn from the supervisor, since there were artifacts (doors, corridors, junctions) that could be identified with the area.

One lesson we learned from attempting to give Flakey advice of this sort is that the closer a match between the robot's perceptual categories and the human's, the easier and more foolproof the exchange of information. For example, if Flakey had no concept of a corridor, it would be almost impossible to tell it the name of the corridor (e.g., "J-wing") and have it internalized in an effective way. Suppose Flakey were to store its current position in association with the corridor name. Then the command "Go to J-wing" would cause Flakey to go back to the same location, even if another location in the corridor were much closer. Commands such as "follow the corridor" wouldn't make any sense at all.

4.6 Tasking

A robot is supposed to perform useful tasks. For the scenario, Flakey was a delivery robot, whose main task was to deliver messages and manuscripts. While performing these tasks, Flakey maintained the goals of obstacle avoidance and localization. These required no overt planning; PRS-Lite invoked the requisite behaviors automatically. In more complicated situations, for example where the robot must explore to find a new route, there might be explicit planning for localization.

Navigation plans were computed by graph search on the learned topological map, and then executed by PRS-Lite. More complicated procedures were constructed as conditional navigation plans: find person X, ask him/her where person Y is, and then go to that location. These plans were simple enough that we just created PRS-Lite schemata for them. A major failure in the plan (e.g., person X not found) caused PRS-Lite to instantiate an alternative schema. The most complicated plan Flakey executed was going to Karen's office and finding out she wasn't there; then going to John's office, asking him where Karen was, finding and getting a report from her, and returning to deliver it.

5 Discussion

There are two basic areas where our work on Saphira has differed in design and emphasis from others. The first is in the area of coherence: by mediating interacts between perception and action through the LPS, we have been able to abstract away some of the difficulties found in interpreting goal-directed behavior relative to the perceptual state of the robot. This is especially true of tasks that cannot rely on current perceptions to formulate correct action sequences. Typical here is the task of following a person around a corner. As the person turns the corner, the perceptual system can no longer track him: there is nothing in the perceptual space on which to base a "follow" behavior. By providing an artifact that represents the best estimate of the person's position, Saphira can still perform the follow behavior on the expectation of re-acquiring perceptual information after a short while.

The second distinctive aspect of our work in in the area of coordination, both in coordination of basic behaviors, and scheduling of behaviors to perform high-level tasks. Context-dependent blending is a general methodology to form complex controllers by composing basic behaviors. Basic behaviors are simple functions on a local state written to satisfy a single goal over a small range of environments (the context). As such, they are relatively easy to write and to debug — they should also be easier to learn in the future. Complex behaviors to achieve multiple goals or operate over wider environmental conditions are composed out of simpler ones by using fuzzy context rules. This is in general an easy operation if the preconditions of the basic behaviors have been clearly stated. Interestingly, this compositional methodology can be formally analyzed using the tools of multivalued logics. In a related work [29], we have shown several properties that link behavior composition to goal decomposition: for example, we can prove, under certain assumptions, that if two behaviors individually promote two goals, then their conjunctive blending promotes the conjoint goal in the conjoint context. Formal groundedness is an important feature of our compositional methodology to form complex behaviors.

Obviously, our approach also has its problems. Fuzzy behaviors implement a local "greedy" method, gradient descent, that is highly reactive and simple to compute. Like any local technique, however, fuzzy behaviors can be trapped in local minima. It is the responsibility of higher-level intentions to only instantiate behaviors for which gradient descent is appropriate based on some global analysis, or to monitor execution to detect local minima and failures. In practice, it is not always easy to find the right contextual condition for a behavior.

A related difficulty in our experience has been that tuning the parameters of the fuzzy rules may be difficult. Although the decomposition of complex behaviors into simpler ones makes them simpler to write, some behaviors required many days of experimental debugging. We are currently exploring the possibility to automatically synthesize basic behaviors from specifications; and the use of learning techniques to improve a behavior's performance.

Numerous task-level controllers for managing the activities of mobile robots have been built in recent years [9, 10, 25, 26, 33], many of which provide services similar to those in Saphira. However, two aspects of the PRS-Lite controller within Saphira set it apart from other systems. One is that PRS-Lite builds many key control constructs into the schema language itself, rather than leaving them implicit in the procedural workings of the controller. The second is its thresholded goal semantics, which enables task-level blending of fuzzy behaviors.

In our experiences of writing over 50 schemas for a variety of different tasks, the expressiveness of PRS-Lite has proven to be mostly adequate. However, certain extensions would simplify the task-control effort. One would be to add a database to provide explicit, declarative representations of beliefs about the world, thus enabling more general reasoning capabilities. Currently, this state information is stored in a combination of the Local Perceptual Space and a set of environment

variables. A more significant change would be to add some limited deliberation to enable selection among multiple candidate schemas, rather than direct dispatch of a single designated schema. The database and deliberation capabilities were explicitly excluded from the original system for fear that their computational overhead might eclipse the limited processing time available for each perceive-act cycle. Given that there is available cycle time, it would be interesting to modify the system to be more declarative in this manner.

Acknowledgments

We have worked closely with Enrique Ruspini and Alessandro Saffiotti, who are primarily responsible for the ideas and code behind the fuzzy control behaviors of Saphira. They are also responsible for many of the most interesting features of Saphira, and we are grateful for their participation in this project. We would also like to thank John Woodfill of Interval Corporation for allowing us to use his stereo vision programs on Flakey.

References

[1] J. F. Allen, Recognizing intentions from natural language utterances, in: *Computational Models of Discourse* (MIT Press, Cambridge, MA, 1983) 107–166.

[2] R. C. Arkin, Integrating behavioral, perceptual and world knowledge in reactive navigation, *Robotics and Autonomous Systems* **6** (1990) 105–122.

[3] R. A. Brooks, A layered intelligent control system for a mobile robot, in: *Proceedings of the IEEE Conference on Robotics and Automation* (1986) 14–23.

[4] J. Connell, *Minimalist Mobile Robotics: A Colony-style Architecture for an Artificial Creature* (Academic Press, 1990).

[5] J. Connell, SSS: A hybrid architecture applied to robot navigation, in: *Proceedings of the IEEE Conference on Robotics and Automation* (1992) 2719–2724.

[6] M. Drumheller, Mobile robot localization using sonar, A. I. Memo 826, Massachusetts Institute of Technology (1985).

[7] C. C. et. al., CARMEL versus FLAKEY: A comparision of two winners, *AI Magazine* **14** (1) (1993) 49–57.

[8] R. J. Firby, An investigation into reactive planning in complex domains, in: *Proceedings of the Conference of the American Association of Artificial Intelligence* (1987) 202–206.

[9] R. J. Firby, Task networks for controlling continuous processes, in: *Proceedings of the Second International Conference on AI Planning Systems* (AAAI Press, 1994).

[10] E. Gat, Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots, in: *Proceedings of the Conference of the American Association of Artificial Intelligence* (1992).

[11] M. P. Georgeff and F. F. Ingrand, Decision-making in an embedded reasoning system, in: *Proceedings of the Conference of the American Association of Artificial Intelligence*, Detroit, MI (1989) 972–978.

[12] J. R. Hobbs, M. Stickel, D. Appelt, and P. Martin, Interpretation as abduction", *Artificial Intelligence* **63** (1–2) (1993) 69–142.

[13] J. L. Jones and A. M. Flynn, *Mobile Robots:: Inspiration to Implementation* (A. K. Peters, Wellesley, MA, 1993).

[14] O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots, *The International Journal of Robotics Research* **5** (1) (1986) 90–98.

[15] K. Konolige, Designing the 1993 ncai robot competition, *AI Magazine* (Spring 1994).

[16] K. Konolige, Erratic competes with the big boys, *AI Magazine* (Summer 1995).

[17] J. C. Latombe, *Robot Motion Planning* (Kluwer Academic Publishers, Boston, 1991).

[18] J. Leonard, H. Durrant-Whyte, and I. J. Cox, Dynamic map building for an autonomous mobile robot, in: *IROS* (1990) 89–95.

[19] Y. Lespérance and H. J. Levesque, Indexical knowledge and robot action — a logical account, *Artificial Intelligence* **73** (1–2) (February 1995).

[20] R. C. Moore, Reasoning about knowledge and action, PhD thesis, Massachusetts Institute of Technology, Cambridge, MA (1980).

[21] H. P. Moravec and A. E. Elfes, High resolution maps from wide angle sonar, in: *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, Washington, D. C. (1985) 116–121.

[22] K. Myers and K. Konolige, Reasoning with analogical representations, in: B. Nebel, C. Rich, and W. Swartout, eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR92)*, San Mateo, CA (Morgan Kaufmann, 1992).

[23] K. L. Myers, User's guide for the procedural reasoning system, Technical note, SRI Artificial Intelligence Center, Menlo Park, California (1993).

[24] K. L. Myers, A procedural knowledge approach to task-level control, in: *Proceedings of the Third International Conference on AI Planning Systems* (AAAI Press, 1996).

[25] N. J. Nilsson, Teleo-reactive programs for agent control, *Journal of Artificial Intelligence Research* **1** (1994) 139–158.

[26] D. W. Payton, J. K. Rosenblatt, and D. M. Keirsey, Plan guided reaction, *IEEE Trans. on Systems, Man, and Cybernetics* **20** (6) (1990).

[27] A. Saffiotti, Pick-up what?, in: C. Bäckström and E. Sandewall, eds., *Current Trends in AI Planning* (IOS Press, Amsterdam, Netherlands, 1994).

[28] A. Saffiotti, Using fuzzy logic for robot navigation, PhD thesis, Université Libre de Bruxelles, Brussels, Belgium (1996).
In preparation.

[29] A. Saffiotti, K. Konolige, and E. H. Ruspini, A multivalued-logic approach to integrating planning and control, *Artificial Intelligence* **76** (1-2) (1995) 481–526.

[30] A. Saffiotti, E. H. Ruspini, and K. Konolige, Integrating reactivity and goal-directedness in a fuzzy controller, in: *Procs. of the 2nd Fuzzy-IEEE Conference*, San Francisco, CA (1993).

[31] A. Saffiotti and L. P. Wesley, Perception-based self-localization using fuzzy locations, in: *Procs. of the International Workshop on Reasoning with Uncertainty in Robotics*, University of Amsterdam (1995).

[32] R. Simmons, The 1994 AAAI robot competition and exhibition, *AI Magazine* (Summer 1995).

[33] R. G. Simmons, Structured control for autonomous robots, *IEEE Transactions on Robotics and Automation* **10** (1) (1994) 34–43.

[34] J. Yen and N. Pfluger, A fuzzy logic based robot navigation system, in: *Procs. of the AAAI Fall Symposium on Mobile Robot Navigation*, Boston, MA (1992) 195–199.

[35] R. Zabih and J. Woodfill, Non-parametric local transforms for computing visual correspondence, in: *3rd European Conf. Computer Vision*, Stockholm (1994).