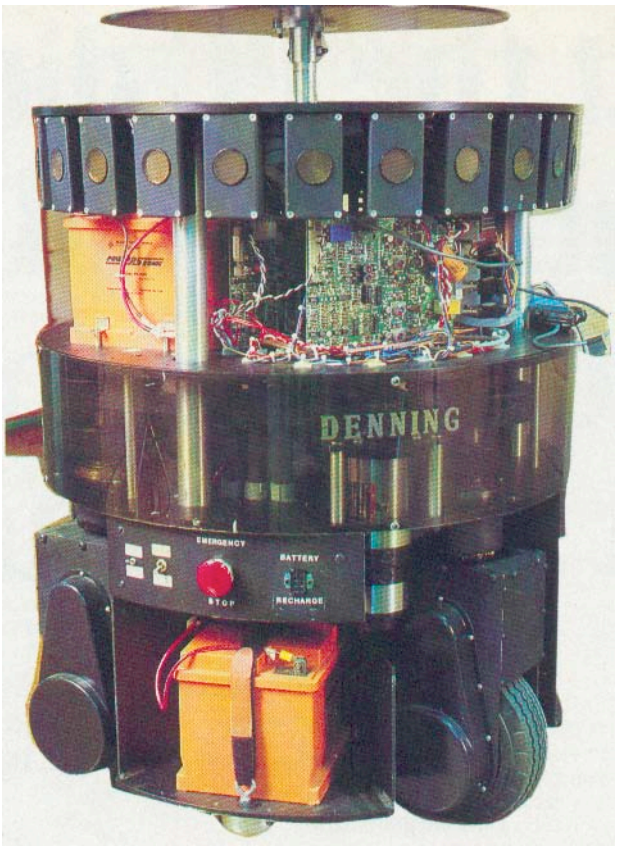

Coordinaton of Action and Perception in a Surveillance Robot

James L. Crowley

Institut National Polytechnique de Grenoble

This article describes a technique for coordinating action and perception within a mobile surveillance robot employing an architecture composed of twin navigation and perception hierarchies. A knowledge-based supervisor controls these hierarchies. This architecture has enabled us to explore the interface between heuristic and procedural programming techniques for perception and navigation. Results illustrate that most low-level perception and navigation tasks are algorithmic in nature. At the highest levels, on the other hand, decisions regarding which actions to perform are based on knowledge relevant to each situation. Such decisions are heuristic, and relevant knowledge is naturally encoded as production rules (organized as contexts).

Coordinating action and perception

Research reported in this article is part of an effort to develop a family of intelligent mobile surveillance robots—"intelligent agents" operating in and interacting with the real world. Developing intelligent agents poses important scientific questions for robotics and AI, such as the relationship between heuristic knowledge and procedural skills. In particular, where and how should the boundary between heuristic and procedural programming occur in an intelligent agent? A second problem concerns the relationship between planning and plan execution: How much planning can we develop in advance? And how should the agent respond when the plan does not succeed?

0885-9000/87/1100/0032/ $1.00©1987 IEEE

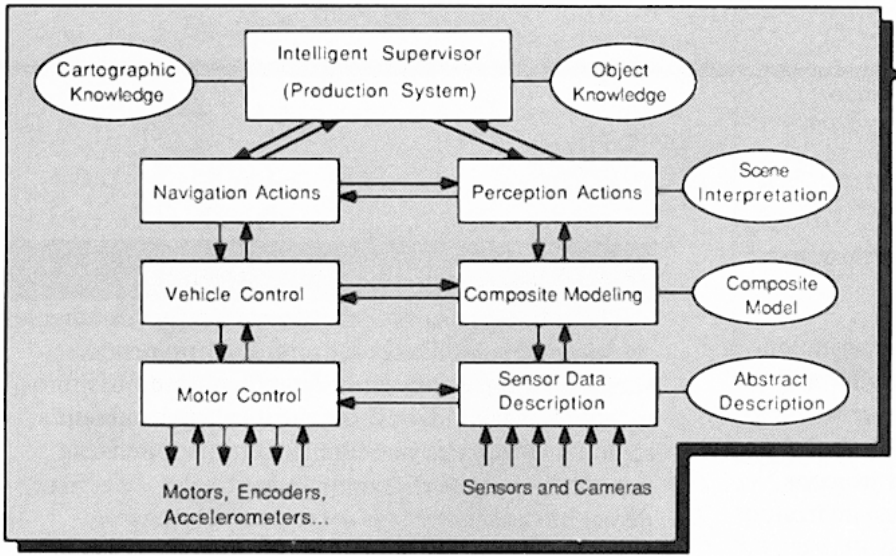

Figure 1. The system architecture: An intelligent supervisor controls parallel architecture for navigation and perception.

To explore such issues, we have studied these problems within a specific application domain; namely, planning and executing surveillance missions for a mobile robot. We have developed systems for dynamic world modeling and for navigation and locomotion of a surveillance robot within our laboratory.[1] We have investigated knowledge-based coordination of action and perception using a simulated version of our surveillance robot and a database of simulated environments. This simulation imitates the functional behavior of our surveillance robot equipped with a set of surveillance sensors in a changing environment.

Knowledge-based coordination. Our investigation has led to an architecture in which a production system sits at the top of twin hierarchies for perception and navigation. Lower levels of these hierarchies assure (1) control at vehicle level, and (2) signal integration from environmental sensors into a composite model of the immediate environment (see Figure 1).

The composite model and vehicle level controller are data-driven processes at roughly the same level of abstraction within both hierarchies. Above this level, a number of perception and navigation abilities exist at what we refer to as the "action level" in the hierarchies. These action level abilities correspond roughly to human skills. Attempts to implement these abilities as bodies of rules within the production system soon convinced us that such an approach was inappropriate. Skills are fundamentally procedural in nature and are better suited for programming in a traditional programming language.

The result is an architecture in which rules within the production system trigger procedures for navigation and perception. Navigation at this level often requires considerable information from the perception hierarchy. As a result, much communication between navigation and perception occurs without production system awareness. Procedures using the composite model of the environment control actions such as following a wall or maneuvering among obstacles.

System Architecture

Control of navigation and perception in an unknown environment presents difficult problems. Earlier projects in which we implemented this supervisory level with procedural programming techniques led to systems with rigid behavior.[2] Such research revealed that top-level control requires the application of poorly organized knowledge. Thus, we have developed a system architecture in which the production system controls perception and navigation hierarchies.

System organization. Figure 1 illustrates system organization. Twin hierarchies for navigation and perception are organized as a set of levels according to the abstraction of processed information and the speed with which responses are required.

Motors and sensors. At the lowest level, each hierarchy asynchronously processes raw signals. In the navigation hierarchy, processing involves closed-loop control of motors (to maintain a specified velocity) as well as capture of proprioceptive sensor signals for estimating position and velocity. In the perception hierarchy, processing involves sensor signal acquisition and conversion to an initial symbolic representation in vehicle coordinates.

Vehicle control and the composite model. At an intermediate level, both hierarchies represent their information at an abstraction level based on the vehicle and its environment. At the center of the navigation hierarchy is a vehicle level controller that accepts asynchronous commands to move and turn the vehicle. The vehicle level controller also maintains a robot position-and-velocity estimate, sharing this with the perception hierarchy. The perception hierarchy projects the description of new sensor signals into a common coordinate system, using the projected information to update a "unified" composite model of the environment. As a side effect of the update process, the perception hierarchy detects errors in the esti-

mated position and relays these to the vehicle controller.

The action level. Collected procedures operate on information provided by the vehicle controller and the composite model. The navigation hierarchy includes procedures for such tasks as following hallways, following walls, or traveling towards distant beacons. These procedures necessarily depend on asynchronous access to perceptual information, made available by a set of interface procedures for the composite model. Significant navigation and perception coordination occurs between procedures at the action level. In human terms, these action level procedures correspond to learned human skills such as driving a car or understanding the environment in terms of rigid objects.

Surveillance procedures also exist at the action level—procedures permitting the supervisor to scan surveillance sensors while continuing with other activities.

The supervisory level. Controlling twin navigation and perception hierarchies involves selecting appropriate perception and navigation actions to accomplish some set of high-level goals. We cannot easily organize such activity as procedures. Each situation implies appropriate knowledge—knowledge naturally expressed as rules organized into contexts. Within each context, internal facts representing things (such as goals, external events, or descriptions of the environment) trigger rules.

Specifications. Each level in these hierarchies exists as an independent, asynchronous process. Our surveillance robot uses dedicated microprocessors for motor control and vehicle control, plus sensor data acquisition and interpretation. Communication occurs by message passing over serial communication lines. Modules for perception actions, navigation actions, and the supervisor currently reside on an off-board VAX. These three modules operate as independent processes that communicate by message passing, using the socket mechanism provided by UNIX BSD 4.3. The modules have been designed so that they can be transferred to on-board microprocessors.

The supervisor is written in OPS-5 running under Lisp. C functions loaded into Lisp accomplish communication with perception, navigation, and surveillance modules. We have modified the OPS-5 Lisp code so that messages can be received asynchronously. Action level procedures can send messages that add objects to the OPS-5 working memory between recognize-act cycles.

The perception system

The robot's sensors form two classes: environmental sensors and surveillance sensors. Environmental sensors, operating autonomously and continuously, provide information describing the local environment's structure—whereas surveillance sensors, operating asynchronously when commanded by the supervisor, detect intruders.

Environmental sensor data drives a process that constructs and maintains a composite environmental model describing geometric, dynamic, and surface features within the local environment. This composite model is the interface between data-driven and knowledge-driven perception processes.

Integrating sensor data—the composite model. A dynamically maintained data structure, the composite model is the perception system's heart. Our current system model provides a two-dimensional geometric description of the limits to free space.[3] A separate project has recently developed a three-dimensional composite model using motion and stereo.[4]

The composite model is a geometric description; it contains no labels for "recognized" objects. The supervisor, using domain knowledge and procedures at the action level, interprets structures within the composite model as known objects.

Parts of the composite model's matching and update processes have complexities approximating the square of the number of elements. To keep cycle time fast, we have restricted composite model contents to a few tens of primitives. The system quickly removes elements from the corresponding composite model when their presence is not reinforced by sensor signals or needs of the action level processes. A recency mechanism mediates such purging. Older elements are purged from the composite model to restrict the number of elements to a fixed limit.

The action level interface to the composite model. Composite model contents are never directly available to other parts of the system. Instead, these contents are accessible through interface procedures. Versions of these primitives exist for both the two-dimensional composite model[2] and the three-dimensional composite model.[3]

- *Visible*—Given points A and B, return the identity of the surface closest to point A that intersects the line from A to B. A return value of nil indicates that point B is visible from point A. Figure 2 illustrates procedure Visible.

- *Correspond*—Given primitive element P, with a particular position and orientation and a tolerance for position and orientation, return the identity of the primitive element in the model that "best" corresponds to the primitive. A return value of nil indicates that no such primitive can be found in the model.
- *FreePath*—Given positions A and B as well as a tolerance, indicate whether the robot may safely travel from A to B without coming within the tolerance of an object. Return either nil or the identity of the first object that the robot might strike. Figure 3 illustrates procedure FreePath, which is accomplished by a "clipping" function.
- *FindPrimitive*—Given a primitive element with a set of position-independent attributes and uncertainty tolerances for each of those attributes, return a list of all occurrences of the primitive in the composite model.
- *Recall*—Recall, from a prestored global model, elements that should be visible from a specified location. The process adds these new elements with a very low confidence to the model. In the next update cycle, the normal update mechanism removes elements for which no correspondence is found in the sensor data.[2] Segments in the global model are typically more complete than currently observed segments.

These interface procedures, accessible to the supervisor and to the action level navigation procedures, are also used in developing more sophisticated action level procedures, such as

- *FindPath*—Given a start point and a goal point, determine a set of straight-line path segments that will take the robot from the start point to the goal point without intersecting any of the limits to free space described in the composite model. FindPath constructs a path composed of a sequence of three straight-line movements[1] and similar to the two segment paths.[2] The two alternative paths are tested in the composite model. If both paths are clear, the shorter path is returned. If only one path is clear, then it is returned. If neither path is clear, FindPath returns a value of false. Figure 4 illustrates procedure FindPath.
- *FindObject*—Given a description of an object as a composition of primitives, return a list of all such objects in the composite model. FindObject is implemented by prediction-verification using pose-clustering.[3]

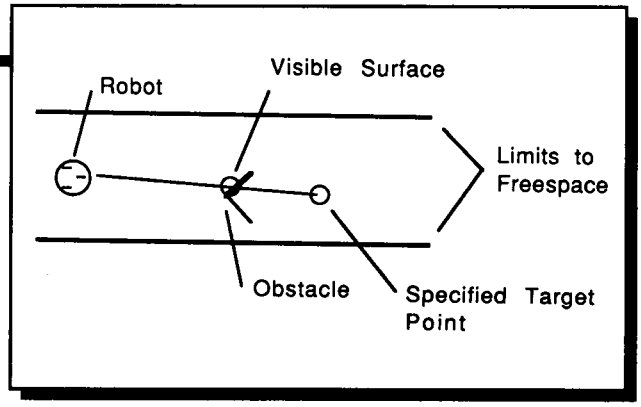

Figure 2. The interface procedure Visible determines whether a specified point should be visible. If an element of the composite model intersects the line from the composite model to the target point, then the location of the intersection and the identity of the element are returned.

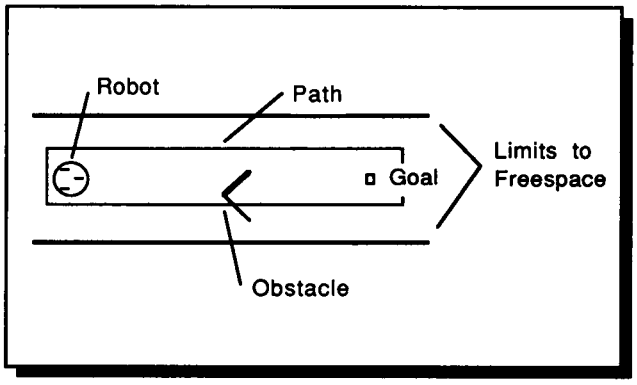

Figure 3. Procedure FreePath determines whether a projected path intersects with any element in the composite model.

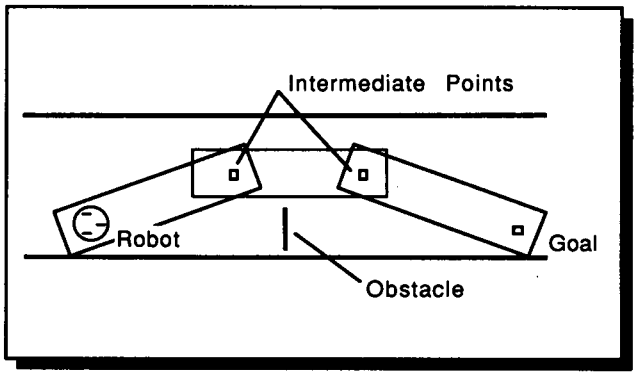

Figure 4. Procedure FindPath determines a new path to a goal point. The path is composed of two intermediate avoidance points that take the robot around the obstacle and on to the goal. The large rectangles indicate the test, performed by FreePath, verifying that a proposed path does not intersect or contain a limit to free space within the composite model. The small rectangles indicate the two avoidance points and the goal.

which changes in orientation depend on changes in position (that is, a car's geometry), Scott Harmon (in a personal communication with the author) has suggested that time-based parameters for Turn may be replaced by distance-based parameters. In such cases, Turn parameters are expressed in degrees per meter, which translate directly to a steering angle.

Control of trajectory—by specifying degrees per meter (or degrees per second at a known velocity)—leads to movement over a set of analytic curves known as "clothoid" curves.[6] Tom Binford refers to this family as "Euler Spirals," remarking that Euler is the first to have reported their existence.

Action level navigation procedures. Control of local vehicle movements is inherently procedural. This level of control corresponds to "local navigation."[2] Our system currently uses the following six such procedures:

- *Straight-line travel* is given a point in its local environment, expressed in an external coordinate system. After verifying that a free path exists to the point, the robot turns toward the point and then travels in a straight line. A simple finite-state automaton assures straight-line travel. The procedure continues to assure that the path remains free during movement, using procedure FreePath (described earlier).

- *Pursue*—Based on techniques described by Wallace et al.,[5] the robot dynamically adjusts its orientation velocity to pursue a specified point.

- *Wall following*—The robot travels a specified tolerance to the right or left of a wall for a specified distance. The target point is determined by projecting a target point in front of the vehicle and then measuring the perpendicular distance to the wall, as well as the orientation of the wall at that point, using the composite model. A new target point is computed at the specified distance from the wall and the Pursue function is called with the target point. The procedure fails if it is unable to find the wall to the side of the target point.

  Wall measurements are made in the composite model using function Visible. The procedure also uses function FreePath to assure that the path is not blocked by an obstacle.[1]

- *Hall following* resembles wall following, except that the next goal point is determined by measuring the center point between the walls in front of the robot. If either wall is missing, a center point is estimated using the second wall. The procedure fails if no walls are found.

- *Doorway traversal* locates the two sides of a doorway in the composite model. It then determines a pair of points defining a path to travel through the center of the door, and uses procedure Straight-line travel along the path.

- *Approach*—Given a starting location, a target location, and an approach distance, this procedure determines a path taking the robot to a point that is the approach distance from the target. A goal point is computed by projecting a straight line to the target from the starting point. Procedure FreePath is used to see if a straight-line path exists to this goal point. If not, procedure FindPath is used to determine a set of straight-line path segments to the goal points. If a path is found, the robot executes the path using straight-line travel.

Straight-line travel—based on the use of estimated position maintained by the vehicle level controller and corrected by the perception hierarchy—is most useful for navigating in domains where the composite model is completed by information from a prestored global model. Pursue is a form of direct pursuit to a point specified by the supervisor. Hall following and wall following use direct pursuit to a point determined from the composite model, and are most useful when the global model is known to be incomplete. The supervisor uses Approach to approach and warn an intruder to leave a restricted area.

The choice of appropriate local navigation procedures depends on the local environment. Such a choice is best stated as a set of heuristic rules, depending on knowledge about the environment and the robot's mission. In our surveillance robot, each route in the cartographic knowledge base contains a suggested local navigation mode for traveling along the route. The supervisor reads the suggested navigation mode from the cartographic database and calls the appropriate local navigation procedure. When a navigation procedure is unable to continue, the procedure halts and notifies the supervisor.

The surveillance-robot supervisor

The supervisor's behavior is goal oriented; it is given a mission as a sequence of navigation and surveillance tasks to accomplish. The mission is decomposed into a sequence of actions by a planning stage. During execution, the supervisor accomplishes each task by executing the actions of the plan. The supervisor adapts and modifies the plan as needed to accomplish the mission goal.

The supervisor is implemented as a production system using the OPS-5 language.[7] The supervisor's rule base is organized as contexts according to tactical situation and current task. The right-hand side of OPS-5 rules can include function calls in Lisp, providing the interface to action level procedures for navigation, perception, and surveillance as well as access to the cartographic database.

The supervisor uses operational knowledge for both mission planning and mission execution. A major use of the planning process is to signal impossible tasks. If specified tasks cannot be completed within allowed constraints, the operator is notified. The supervisor can remedy problems by increasing the value of the maximum constraint or by changing the mission definition. In mission execution, the supervisor uses its operational knowledge to react to unforeseen events.

The following subsections describe cartographic and operational knowledge bases, and outline the specification, planning, and execution of a mission.

Cartographic knowledge. Much knowledge required to plan and execute navigation tasks is cartographic, organized naturally as a network of objects representing places and routes. A cartographic database is implemented as a network of structures accessible from Lisp functions. The supervisor can recall the contents of a place or route by calling the Lisp functions "get-place" or "get-route," resulting in the creation of an OPS-5 object that contains attributes of the route or place. In planning, the supervisor uses the external function "generate-nodes." This function generates objects representing various forms of travel on each route leading from a place.

A place contains the following fields:

Name—An alphanumeric name for the place;
Location—A Cartesian location for the place;
Tactical zone—The tactical situation at the place; and
Adjacent places—A list of pairs (place, route) for adjacent places.

A route is composed of

Name—A name for the route;
Navigation-action—A suggested navigation procedure for this route;
Length—The length of the route;
Maximum speed—The maximum speed for the route;
Efficient speed—An efficient speed for the route;
Quiet speed—A speed for minimizing operational noise; and
Tactical zone—The current tactical zone.

Each route specifies a suggested navigation action. When the robot travels on a route, it uses this navigation action to specify an action level procedure. If the procedure fails, the robot may attempt to continue using position estimation maintained by odometry and inertial guidance and assisted by position estimation using the perception hierarchy.

Each place and route in the cartographic knowledge base is labeled with an attribute identifying the tactical zone for that place or route. The tactical zone, in turn, is an important factor in determining the robot's behavior. Each zone contains a risk factor, multiplied by the time spent in the zone, and used as a constraint in mission planning. Similarly, fuel consumption for a route is computed by multiplying the distance by a fuel consumption rate (presented in a table).

The tactical zones are

Friendly—The robot is permitted to travel freely in this zone. No surveillance activity is required. The risk factor is 1;
Restricted—The robot can travel freely in this zone, but maintains vigilance to detect intruders. The risk factor is 1;
Unknown—The tactical situation is unknown in such zones. The robot maintains maximum surveillance. The risk factor is 2;
Dangerous—The robot maintains maximum surveillance in such zones. The risk factor is between 2 and 10; and
Hostile—The robot avoids travel in hostile zones unless explicitly instructed otherwise by the mission statement. The risk factor is a number greater than 10.

Operational knowledge. The supervisor uses operational knowledge during planning and plan execution to determine the robot's behavior. After conversations with an expert on military security, we developed the operational knowledge described here as a concept demonstration. However, the expert did not

Table 1. This illustrates compatibility between navigation modes and tactical zones. Preferred speeds are shown in boldface. Mission constraints may force the selection of an alternative speed. The supervisor's planning rules determine the choice of speed, but may modify speeds during mission execution. An X marks a speed that may not be used in the specified zone.

	Zones				
Mode:	Friendly	Restricted	Unknown	Dangerous	Hostile
Efficient	**efficient**	**efficient**	efficient	X	X
	fast	fast	fast	X	X
	quiet	quiet	quiet	X	X
Rapid:	efficient	efficient	efficient	X	X
	fast	**fast**	**fast**	**fast**	**fast**
	quiet	quiet	quiet	quiet	quiet
Discreet:	efficient	efficient	efficient	X	fast
	fast	fast	fast	fast	X
	quiet	**quiet**	**quiet**	**quiet**	**quiet**
Reconnaissance:	efficient	efficient	efficient	efficient	efficient
	X	X	X	X	X
	quiet	**quiet**	**quiet**	**quiet**	**quiet**

participate in developing the rule base. The operational knowledge base presented below should be seen only as a concept demonstration.

The robot can travel at any of three speeds specified to the robot during mission planning: fast, efficient, and quiet. Table 1 summarizes the permitted speeds in the different tactical zones for different navigation modes. Mission planning selects the speed if mission constraints are not violated. Constraints may force selection of another speed during the planning process as described below. An X indicates that a speed is not permitted in the specified zone.

The current set of modes includes

Efficient—The robot travels at a rate that optimizes fuel consumption. No surveillance activity occurs;

Rapid—The robot travels as fast as possible. No surveillance activity occurs;

Discreet—The robot, seeking to avoid detection by traveling at its quietest speed, maintains a passive surveillance activity using only the infrared and acoustic sensors in unknown, restricted, hostile, or dangerous zones; and

Reconnaissance—In friendly zones, the robot travels efficiently with no surveillance. In unknown, restricted, hostile, or dangerous zones, the robot travels at its quietest speed and maintains surveillance activity using the infrared, acoustic, and ultrasonic sensors. In case of detection, the robot notifies the base and enters one of the reaction modes (described below).

The following reaction modes are available when the robot has detected an intruder:

(1) Monitor—The robot remains stationary and tracks all intruders, continuously relaying the positions of all known intruders to the base;

(2) Warn—If it detects an intruder in a restricted zone, the robot selects and travels to the intruder and warns it to leave the restricted zone. Continuously relaying the position of all known intruders to the base, the robot then remains within a short distance until the intruder has left the zone; and

(3) Tail—The robot locks onto and observes an intruder. When the intruder is stationary, the robot periodically scans the environment to locate other intruders. If the intruder flees, the robot follows. If the intruder advances, the robot flees. The robot continuously relays the position of all known intruders to the base.

If it is in a reaction mode when the time arrives to execute its next task, the robot will quit the reaction and commence the task.

Mission specification and planning. Planning seeks to (1) decompose the mission into a set of navigation and surveillance actions, (2) verify that the mission can be accomplished within specified time, fuel, and risk constraints, and (3) ensure that the robot has the necessary cartographic knowledge to execute the mission.

The main problem in planning is determining the routes, navigation actions, speeds, and surveillance actions to be used during the mission. Planning is accomplished by a form of "graphsearch" implemented in a set of production rules.[8] Mission planning

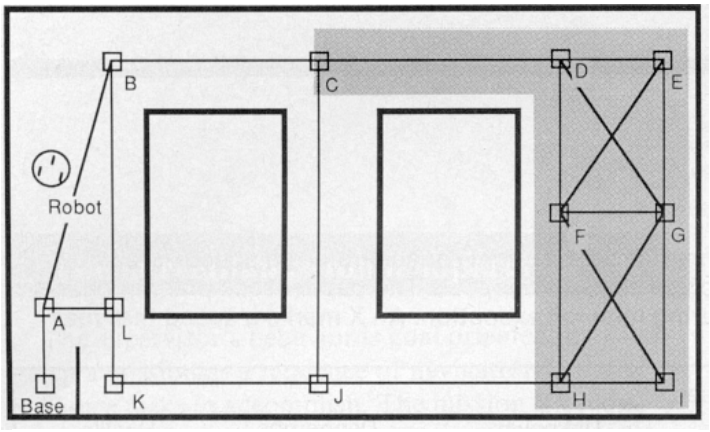

Figure 5. A sample domain for a surveillance mission: The small boxes and circles labeled Base and A through L are places in the network of places. Lines connecting places indicate routes. The shaded area is a restricted zone.

is a process of heuristically guided generate and test. The external procedure "generate-nodes" uses the cartographic database to generate nodes representing travel at efficient, fast, and discreet speeds from the selected place. Operational knowledge then "edits" these new nodes to eliminate nodes incompatible with the tactical zone of the selected place.

The following navigation tasks are possible:

Go-To-Place—At the specified time, the robot will depart for the specified place. Go-To-Place actions are planned by forward chaining from the start place to the goal place.

Arrive-At-Place—The robot will arrive at the specified time and place. Arrive-At-Place actions are planned by backward chaining from the specified place and time to determine the appropriate starting time.

In addition, the navigation task Remain-At-Place exists in which the robot remains at its current location until the time for its next navigation task.

Navigation tasks contain a start time and a completion time: The start time is specified for the Go-To-Place task, and planning determines completion time. However, the next task's start time places a hard constraint on the total time available for the task. Similarly, completion time is specified with the task Arrive-At-Place, and task planning determines start time. The completion time of the previous task determines a hard constraint. Adjacent navigation tasks determine the start and completion times for the task Remain-At-Place.

A potential problem exists when an Arrive-At-Place task follows a Go-To-Place task. Currently, the time constraint for planning the Go-To-Place task is furnished by the completion time of the Arrive-At-Place task.

Specification of a surveillance mission. We currently specify missions by responding to questions from a mission specification rule base. Mission tasks include both navigation and surveillance tasks. We plan to replace this specification rule base with a menu-driven specification system. In either case, specification results in a set of constraints and navigation tasks describing the mission.

Figure 5 shows a sample domain illustrating a surveillance mission. The following is a sample mission definition in this domain expressed as sentences in pseudo English:

Mission Constraints:
Time = 120, Fuel = 20, Maximum risk = 20, Maximum distance 1000
If detection, use mode Warn.
(1) At time = 540 Go to place C in mode Efficient.
(2) At time = 560 Go to place E in mode Surveillance.
(3) Remain at place in mode Surveillance.
(4) At time = 600 Arrive at Base in mode Efficient.

This example's first navigation task generates an object of type NavTask with a task-type of Go-To-Place, a start-place of the current place (Base), and a goal-place of A. The values for goal-zone and goal-loc are obtained from the external database. Constraints for distance, time, risk, and fuel are copied from mission constraints.

Mission planning: Generating possible actions. We plan the mission by assembling a sequence of navigation and surveillance actions for each task, conducting an A* graphsearch that uses route information from the cartographic database. Each search tree node contains the current value of a constraint vector composed of the set (distance, time, risk, and fuel).

For each route leaving the selected place, three nodes are generated representing travel at a speed that minimizes fuel, time, or risk. The function generate-nodes builds nodes containing values for the attribute's place and zone using route information in the cartographic database. A rule set removing newly generated nodes enforces incompatibilities between navigation modes and tactical zones.

Constraints on distance, time, risk, and fuel. For the search to be A*, we must use a heuristic cost that satisfies optimality criteria identified by Nilsson.[8] Each navigation task mode specifies the constraint to be minimized. Each constraint is based on a linear function of the distance traveled; therefore, each can be estimated by a function of the form

h(node, goal) = C * distance(node, goal)

where C is a positive constant. The Cartesian distance between two places is less than or equal to the route distance. Thus, Cartesian distance yields an optimum search (the first path found is the shortest path). Similarly, any heuristic based on the product of a constant multiplied by the distance also produces an optimal search. Thus, any constraint and any linear combination of constraints may serve as the heuristic cost for an optimal search. However, constants must not change during search.

We can minimize time, risk, and fuel costs during planning. Graph search uses two cost estimates at each node—g(start, node) and h(node, goal). The function h() is estimated using the vehicle's maximum speed. Costs for g() are accumulated using the speed proposed in the node. A rule for the specified navigation mode copies a value for time, risk, and fuel into the values for g and h. Table 2 presents formulas for estimating costs.

For each node, we compute estimated values for time, speed, and fuel use by the external procedure generate-nodes, based on the values of distance found in the external database. The fuel efficiency factor is based on a table of estimated values. The function generate-nodes adds costs computed for the route to cumulative costs. Thus, values in the newly created nodes represent the cumulative value from the mission's start. The navigation mode determines which cost is minimized in the search.

In addition to their potential use as a cost function, the constraints also serve as a hard limit for a task. Any node for which an element of the constraint vector exceeds a limit is removed. A set of rules compares these values in nodes of status new to the constraints for the task. Nodes for which any of these values exceed the maximum value are removed. Thus, if mission constraints cannot be satisfied using the specified default navigation modes, branches of the search tree using other modes are considered.

The mission plan. In the above mission, the first navigation task was "At time 540 Go to place C in mode Efficient." This generated a set of navigation actions of the form

straight-line to Place A in mode Efficient
straight-line to Place B in mode Efficient
follow-hall to Place C in mode Efficient

The execution of a surveillance mission. Executing the surveillance mission is a process of executing navigation and surveillance tasks at the specified times. The program executes each navigation task by executing navigation actions for that task, and executes each navigation action by simply calling an external procedure at the action level and then monitoring the procedure's status. Navigation procedures use perception procedures to detect obstacles and to maintain an estimate of the robot's position. As long as a navigation action does not fail, the supervisor need only call a new navigation action as each navigation action finishes.

Table 2. The cost formulas, which are minimized for each navigation mode.

Mode	Cost Formula
Rapid:	time = distance /speed
Discreet:	risk = risk-factor * time
Efficient:	fuel = fuel-efficiency(speed) * time

Rules based on the current surveillance task and tactical zone determine surveillance activity specified in the plan. We will next describe (1) the execution of navigation actions and reaction to blocked paths, and (2) surveillance actions and reaction to the detection of intruders.

Executing the plan. While moving, the supervisor continually calls the external function GetRobotStatus to monitor the action procedure's execution. Navigation procedures at the action level directly control navigation perception processes. Perception information does not pass through the supervisor's working memory.

In the planning example above, the third navigation action was to travel from place B to place C. The cartographic database for the route from place B to C indicates a suggested navigation action of follow-hall. Thus, the supervisor calls the navigation action level procedure follow-hall to place C. During execution, the supervisor periodically requests the navigation status by sending a message to the navigation procedure. The navigation procedure replies with a message indicating the robot's current position, orientation, action, and status. This message is translated into an OPS-5 object of type robstatus, which is put into the supervisor's working memory.

The supervisor also calls a perception action ("recall") that constructs a description of the expected local environment from the global model and proposes this description to the composite model as a set of hypothesized composite model elements. If the hypothesized elements match the existing elements in the composite

model, then these are used as if they had been perceived by the composite model. If the recalled elements do not match elements currently in the composite model, then they are not added to the model.

A navigation action terminates with a status of "finished," "failed," or "blocked." If a procedure returns a finished status, the supervisor so marks the current navigation action and marks the next navigation action with a status of "active." When a navigation action encounters difficulty by reporting a blocked or failed status, the supervisor must determine the cause of the difficulty and specify a new navigation action. The principal cause of action failure is a blocked path; however, the supervisor uses the same technique to respond to failures in wall or hall following.

Going around obstacles. When a path is blocked, the robot tries to construct a path around the blockage by calling an action level perception procedure FindPath. If it finds a path, then the two intermediate goal points are returned to the supervisor. The current navigation action is marked "suspended," and straight-line-travel NavActions are created for the two intermediate goal points.

If the perception procedure FindPath fails, then the supervisor must rely on the network of places to find a new path. The supervisor issues a command to the cartographic database to mark the current route as temporarily blocked, and replans the current navigation task. If the replanning cannot produce new navigation actions that respect mission constraints, then the robot requests permission from the base station to abandon the mission and return to base.

Surveillance tasks. The supervisor controls surveillance actions as calls to action level surveillance procedures. The procedure "scan" operates asynchronously from the supervisor. Upon termination of a scan, the procedures generate an object of the type Scan-Terminated. The supervisor can then initiate a new scan by calling the surveillance action.

The supervisor keeps track of targets with an object list of type Target. When surveillance actions detect a target, they insert an object of type Detection into working memory, giving the target's estimated position and identity as well as the sensors that detected the target and the time of detection. If the target's identity is known, the supervisor updates the target object; otherwise, it creates a new target object.

The surveillance scan does not halt when detection occurs, but continues asynchronously from the supervisor until the scan has completed. The supervisor keeps track of detections using an object of the type Target with attributes for identity, kind, status, and location.

When a target is matched to a detection, the supervisor updates the target's attributes; otherwise, the supervisor creates a new target object with a status of New. This triggers a call to an external database containing a list of known false alarms with the sensor that responded, the Cartesian location, and an uncertainty zone. If the target matches a target in the false-alarm database, the supervisor marks it with a status of Ignore and it is ignored; otherwise, the supervisor notifies the base of the detection. A command from the base station can order the robot to ignore this target. A second command can add the target to the false target database.

To avoid false alarms, we have discussed implementing a system of badges for authorized personnel. This would require badge recognition, perhaps via a coded electromagnetic or ultrasonic signal.

Reaction mode Monitor. In reaction mode Monitor, the supervisor marks the targets with a status of Watching at the end of each scan of directional sensors and then repeats the scan. During the scan, detection of a known target changes that target's status to Detected. Targets in mode Watching at the end of a scan are marked with a status of Questionable. Targets in mode Questionable at the end of a scan are removed. Thus, a target must be missed by two consecutive scans to be removed.

Reaction mode Warn. In reaction mode Warn, the supervisor locks onto the target for approach by marking the target with a status of Tracking. Targets in mode Tracking trigger a call to the action level navigation procedure Approach for approaching and warning the target to leave.

While approaching the target, the supervisor uses a narrow angle scan to continuously observe the target with its directional sensors. Before each scan, the function Visible is first used to verify that the target should be visible from the current location. The scan is not performed until Visible reports that the target's position should be visible. If the target location changes, then the supervisor computes a new approach path and the current goal is superseded by the first goal on the path. When it is within approach distance of the target, the robot warns the target to leave the area, repeats this warning every 30 seconds, and follows the target until it leaves the restricted zone.

Reaction mode Tail. When detecting a target in mode Tail, the supervisor halts any movement and marks the target with a mode Tailing. As long as the target remains visible, the robot remains stationary and the sensors perform a narrow angle search to track the target. If the target disappears, the robot travels toward the target's last known position while scanning the direction towards that position. The path to the position is determined by procedure FindPath.

We have described a program that coordinates action and perception in a mobile surveillance robot, and have presented an architecture in which twin hierarchies for navigation and for perception are controlled by a production system. Much of the system's functionality is provided by action level procedures at the hierarchy's highest levels. In particular, navigation procedures make heavy use of procedures within the perception hierarchy. An important source of coordination between action and perception is thus provided by these procedures.

A supervisor implemented as a production system provides task level control. This supervisor chooses the navigation and perception procedures according to the mission plan and the local environment. The supervisor develops the mission plan in a planning stage before the mission begins. This planning stage (1) decomposes the mission into subgoals that can be translated into procedure calls to the task level procedures, and (2) verifies that the mission can be accomplished within stated constraints. ■

Acknowledgments

Work described in this article is the evolutionary result of a long series of projects. Early versions of the navigation and perception hierarchies were developed at Carnegie Mellon University's Robotics Institute under the sponsorship of Commodore Business Machines, Denning Mobile Robotics, and the state of Pennsylvania. Recent versions have been constructed at the Institut National Polytechnique de Grenoble's LIFIA, in collaboration with two teams of student programmers from the software engineering program DESS Genie Informatique. Special credit is due Renaud Zigmann, Herve Fontaine, Franck Hocquet, and Pierre Bomel for producing the current version of the navigation and perception hierarchies.

The first version of the supervisor was constructed in collaboration with Societé AERO as part of a study for the French Army's DRET office. Its conception resulted from a series of discussions and working sessions with a team headed by J.P. Pouplard of Societé AERO. Contributors from LIFIA include Jean Pierre Muller and Thierry Granier. The first version of the supervisor was written by Thierry Fraichard. More recently, enhancements have been made by Kok-Wei Fong. The supervisor and the navigation and perception hierarchies are currently supported by project MITHRA (EUREKA 110) in collaboration with ITMI, AID, and Societé Bertin.

References

1. J.L. Crowley and J. Coutaz, "Navigation et Modelisation pour un Robot Mobile," (in French), *Technique et Sciences en Informatique,* Oct./Nov. 1986.
2. J.L. Crowley, "Navigation for an Intelligent Mobile Robot," *IEEE J. Robotics and Automation,* Mar. 1985.
3. J.L. Crowley, "Using a Composite Surface Model for Perceptual Tasks," *Proc. Fourth IEEE Conf. Robotics and Automation,* IEEE Service Center, 445 Hoes La., Piscataway, N.J. 08854-4150, Apr. 1987.
4. J.L. Crowley and F. Ramparany, "Mathematical Tools for Representing Uncertainty in Perception," *AAAI Workshop on Spatial Reasoning and MultiSensor Fusion,* AAAI, 445 Burgess Dr., Menlo Park, Calif. 94025, Oct. 1987.
5. R.W. Wallace et al., "First Results in Road Following," *Proc. IJCAI 85,* Morgan Kaufmann, Los Altos, Calif., Aug. 1985.
6. Y. Kanayama, "Trajectory Generation for Mobile Robots," *Third Int'l Symp. Robotics Research,* Paris, France, Oct. 1985.
7. L. Brownston et al., *Programming Expert Systems in OPS-5,* Addison-Wesley, Reading, Mass., 1985.
8. N.J. Nilsson, *Principles of Artificial Intelligence,* Tioga/William Kaufmann, Los Altos, Calif., 1980.

James L. Crowley has been a visiting professor at the Institut National Polytechnique de Grenoble since February, 1985. He received his BSEE from Southern Methodist University in 1975 and his MSEE and PhD from CMU in 1977 and 1982, respectively. He participated in the founding of CMU's Robotics Institute, from which he is on extended leave and where he has held a research faculty position since May, 1980.

His research activities span the areas of vision, mobile robotics, and expert systems. In vision, his current research includes multiple resolution representation and matching of images, dynamic modeling of 3D scenes, and shape inference by motion and stereo. In mobile robotics, he has developed a series of systems for navigation and perception using a rotating ranging device, and more recently a ring of ultrasonic sensors. In expert systems, he has recently directed the development of ELOISE—an OPS-5-like expert system shell based on an object-oriented rete algorithm written in LOOPS. ELOISE has been used to develop XNS-Config, an expert system for configuring Ethernet servers.

The author's address is LIFIA (INPG), 46 Avenue Felix Viallet, 38031 Grenoble, France.