Xavier: Experience with a Layered Robot Architecture

Reid G. Simmons, Richard Goodwin, Karen Zita Haigh, Sven Koenig, Joseph O'Sullivan, Manuela M. Veloso
Computer Science Department
Carnegie Mellon University
Pittsburgh PA 15213-3891
{reids, rich, khaigh, skoenig, josullvn, mmv}@cs.cmu.edu

Abstract

Office delivery robots have to perform many tasks such as picking up and delivering mail or faxes, returning library books, and getting coffee. They have to determine the order in which to visit locations, plan paths to those locations, follow paths reliably, and avoid static and dynamic obstacles in the process. Reliability and efficiency are key issues in the design of such autonomous robot systems. They must deal reliably with noisy sensors and actuators and with incomplete knowledge of the environment. They must also act efficiently, in real time, to deal with dynamic situations. To achieve these objectives, we have developed a robot architecture that is composed of four layers: obstacle avoidance, navigation, path planning, and task planning. The layers are independent, communicating processes that are always active, processing sensory data and status information to update their decisions and actions. A version of our robot architecture has been in nearly daily use in our building since December 1995. As of January 1997, the robot has traveled more than 110 kilometers (65 miles) in service of over 2500 navigation requests that were specified using our World Wide Web interface.

1 Introduction

We have been working towards the goal of building autonomous robotic agents that are capable of planning and executing high-level tasks. Our framework consists of the integration of Xavier the robot [24, 31] with the PRODIGY planning system [16, 33] in a setup where users can post tasks for which the planner generates appropriate plans, sends them to the robot, and monitors their execution.

This office delivery robot will have to move autonomously in an office building reliably performing office tasks such as picking up and delivering mail and computer printouts, returning and picking up library books, and carrying recycling cans to the appropriate containers.

To carry out such tasks, the robots must determine the order in which to visit offices, plan paths to those offices, follow paths reliably, and avoid static and dynamic obstacles while travelling. Several issues arise for such autonomous office delivery robots, the main ones being how to perform the tasks reliably and efficiently in the face of uncertainty and incomplete information. Moreover, the robots have to act in real-time to cope with a dynamic environment, such as moving people and changing delivery requests.

While many techniques exist for handling various parts of the delivery problem, comparatively little work has been done on building complete robot architectures. Only complete architectures, however, allow researchers to study interactions between the layers. Our architecture is based on layers of increasing abstraction. Upper layers in the architecture provide guidance to lower layers, while lower layers handle details that the upper layers have abstracted away. We show that each individual layer provides reliable and efficient behaviour, and argue that the overall architecture achieves synergistic effects by suitably partitioning system functionality.

A version of our robot architecture has been in almost daily use in our building since December 1995. As of January 1997, the robot has served over 2500 navigation requests, travelling a total of more than 110 kilometers (65 miles). These experiments demonstrate that our robot architecture leads to fast and reliable navigation. The robot can travel at speeds of up to 60 centimeters per second in peopled environments. Its planning time is negligible and its task completion rate is now about 95 percent. The robot's travel speed is currently limited only by the cycle time of its sonar sensors, while tasks fail mainly because of problems with the wireless network at CMU—both problems are unrelated to the robot architecture.

Xavier, the robot used in these experiments [22, 24], is built on top of a 24 inch diameter RWI B24 base, which is a four-wheeled synchro-drive mechanism that allows for independent control of the translational and rotational velocities (Figure 1).

The sensors on Xavier include bump panels, wheel encoders, a sonar ring with 24 ultrasonic sensors, a Nomadics front-pointing laser light striper with a 30 degree field of view, and a color

Figure 1: Xavier

camera on a pan-tilt head. The odometer, the sonars and the laser are the primary sensors during navigation, while vision is used for final fine-positioning and location verification at the destination, and the bump panels enable to prevent harm in emergencies. Control, perception, and planning are carried out on three on-board 486 computers. The computers are connected to each other via thin-wire Ethernet and to the outside world (for communicating with the User Interface) via a Wavelan wireless Ethernet system [17].

Section 2 presents an overview of our architecture and a scenario that illustrates how the various parts of the architecture work. Sections 3 and 4 discuss each layer in detail, including its functionality and its interface to the other layers. Section 5 presents the related work, and Section 6 discusses issues of user interaction with the robot. The paper concludes with performance data on the overall architecture and presents some of the lessons learned.

2 Overview of the Robot Architecture

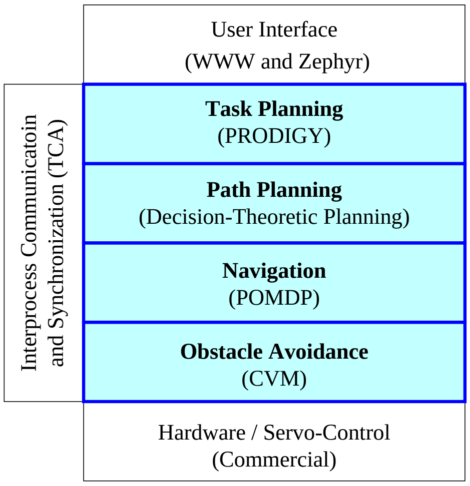


Figure 2: The Robot Architecture

Our robot architecture (Figure 2) consists of four layers: obstacle avoidance, navigation, path planning, and task planning. The layers are implemented as separate code modules (processes).

The robot architecture is implemented as a number of asynchronous processes, distributed over the three on-board computers and an off-board computer permanently running the web-server providing the user interface. The processes are integrated using the Task Control Architecture (TCA) [29]. TCA provides message passing facilities and facilities to support task decomposition, task sequencing, execution monitoring, and exception handling.

Other parts of the architecture include real-time servo control, an interprocess-communication package, and a user interface. The servo control package is provided with the commercially available hardware (robot base and pan-tilt head). Interprocess communication and synchronization is provided by our Task Control Architecture (TCA) [29]. TCA is a general-purpose framework for *task-level control*, by which we mean the coordination of planning, sensing and execution. It provides facilities for interprocess communication (message passing), coordination, and synchronization of the distributed, concurrent modules. Other modules use these facilities to indicate when subtasks should be active and how they should be monitored to determine whether they succeed or fail. The user interface uses the World Wide Web. It allows users to specify requests and monitor robot progress.

While this division of functionality is certainly not novel, each module offers novel approaches with solid theoretical foundations. Obstacle avoidance is performed by our Curvature-Velocity Method (CVM) [30]. Navigation is done using Partially Observable Markov Decision Process models [32]. Path Planning uses our decision-theoretic generate, evaluate and refine strategy that is based on ideas from sensitivity analysis [15]. Task planning is performed using the planning system PRODIGY [16]. We illustrate the layers of our navigation architecture using a typical delivery scenario.

The **user interface** module allows users, such as secretaries, to enter delivery requests (including desired delivery times and priorities). It also provides a means for monitoring the progress of the robot, such as its current position and delivery request being carried out. Assume that there is one delivery request pending, a delivery from A to B (Figure 3), when another user enters a new delivery request: to carry a print-out from C to D.

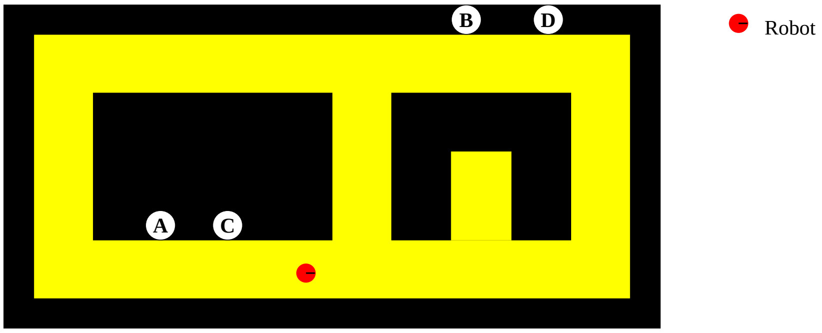


Figure 3: A Navigation Scenario. What order should the robot visit the four locations?

The **task planning** module integrates the new asynchronous request into the current plan, prioritizing tasks and opportunistically achieving compatible tasks. It has to determine, among other things, the order in which to interleave the actions required for each task. In this example, it needs to determine the order in which to visit the four locations. Possible orders are ABCD, ACBD, ACDB, CABD, CADB, and CDAB. It consults with the path planner to determine the expected travel time between any two locations. Based on this information, it selects the route CABD.

The **path planning** module determines how to travel efficiently from one location to another. Actuator and sensor uncertainty complicates path planning since the robot may not be able to follow a path accurately, and the shortest distance path is not necessarily the fastest. Consider, for example, the two paths from A to B shown in Figure 4.

Although Path 1 is shorter than Path 2, the robot could miss the first turn on Path 1 and have to backtrack. This problem cannot occur on the other path since the end of the corridor prevents the robot from missing the turn. The longer path might take less time on average. The path planner uses a decision-theoretic approach to choose plans with high expected utility and uses sensitivity analysis to determine which alternatives to consider.

The **navigation** module generally follows the path suggested by the path planner. It may deviate from the desired path since it, too, has to deal with sensor and actuator uncertainty. For example, if the path planner chooses Path 1 from A to B, and the robot indeed overshoots the turn, then it could mistake the dead-end corridor for the correct corridor. When it reaches the end of the dead-end and discovers its mistake, it issues corrective actions that turn the robot around and let it make progress toward the goal. Our navigation module uses a Partially Observable Markov Decision Process model to maintain a probability distribution of where the robot is at all times,

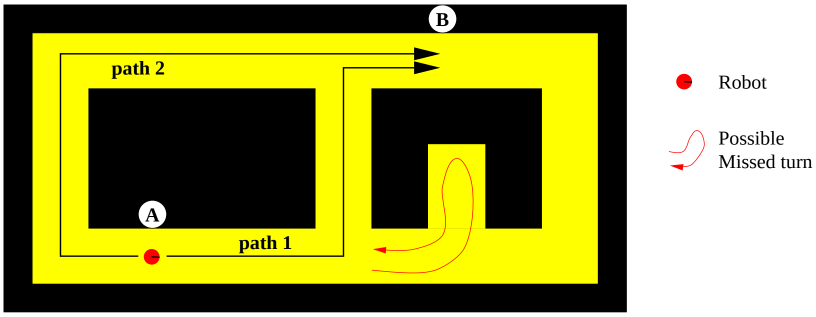


Figure 4: Two Paths from A to B. Path 1 is shorter, but the robot might miss the turn and get lost in the dead-end.

choosing actions based on that distribution.

The **obstacle avoidance** module keeps the robot moving in the desired direction, while avoiding static and dynamic obstacles (such as tables, trash cans, and people). It provides high-speed, safe motion by taking the robot's dynamics into account and by optimizing, in real time, an objective function that combines safety, speed and progress along the desired heading.

To a large extent, the architecture achieves reliability and efficiency by using reliable and efficient component modules. Reliability and efficiency are also achieved through the interaction of the layers: each layer works with more abstract representations of the data from lower levels (Figure 5).

- Higher layers can "guide" the lower layers into regions of the environment where safe and efficient navigation can take place. For instance, the path planning module takes the probability of execution failure into account when planning paths—keeping the robot away from

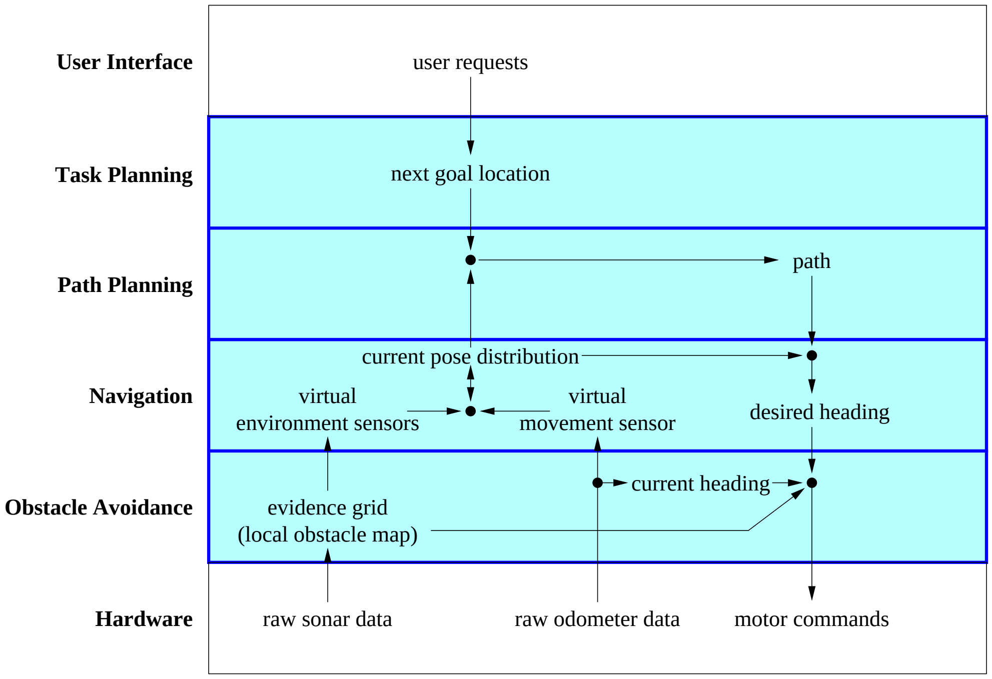


Figure 5: Flow of Information

areas where it may have difficulty navigating.

- Lower layers can take care of details abstracted out by higher layers. For example, while the navigation layer specifies headings for the robot to follow, the obstacle avoidance layer can steer the robot in a different direction, if needed to avoid local obstacles.
- Lower layers propagate failures up to higher layers when they find that they cannot handle certain exceptional situations. For example, the obstacle avoidance module can indicate when it thinks it is stuck in a local minimum. By "failing cognizantly" [14], the lower layers can provide information that enables higher layers to determine how to handle the situation.

The next two sections describe the layers of our architecture in more detail. They also describe the interfaces between the layers and present the data abstractions used for the tasks they perform.

3 Low Level Layers: Obstacle Avoidance and Navigation

3.1 Obstacle Avoidance

Above all else, the robot must travel safely to protect itself and the people with which it shares the environment. In addition, we desire fast (walking speed) travel, both to make the robot useful as a delivery agent and to make it a more socially acceptable "inhabitant" of our building.

The obstacle avoidance layer avoids static and dynamic obstacles, while it moves in the direction provided by the navigation layer. It uses our Curvature Velocity Method [30] to optimize, in real time, an objective function that combines safety, speed, and progress along the desired heading. This way it provides high-speed, safe motion that takes the dynamics of the robot into account.

The input to the obstacle avoidance module is either a desired heading or a goal point. The obstacle avoidance module tries to either maintain this heading or head towards the goal point, while avoiding collisions. Without an explicit goal, the robot wanders while avoiding obstacles. If it gets stuck, the obstacle avoidance module will signal the other modules with a description of the problem encountered.

Previous obstacle avoidance schemes [2, 5] neglected dynamics by assuming the robot could turn instantaneously, more recent schemes enable high speed travel by taking current velocities and feasible accelerations into account. For example, the "dynamic window" approach [13] takes robot dynamics into account by operating in velocity space, and chooses robot velocities by trying to optimize an evaluation function. Our curvature-velocity method (CVM) [30], which is very similar to this method, was developed somewhat later, but independently. CVM poses the obstacle avoidance problem as one of constrained optimization in the velocity space of the robot. The objective function contains terms for safety (distance to the nearest obstacle), speed, and progress (heading in the desired goal direction). By adjusting the coefficients of the objective function, we can easily enable the robot to trade off efficiency for reliability in its travels. Figure 6 illustrates how the robot travels in our corridors (the desired heading is to the right—the robot travels down until it finds an opening, and then travels in the desired heading).

3.2 Navigation

The role of the navigation module is to direct the robot to a given goal location. The path planner sends, through TCA, a desired path for the robot to follow. The navigation module then estimates the robot's current location, calculates the desired heading for that location, and then specifies that heading to the obstacle avoidance module.

The module must work reliably in spite of noisy sensors and actuators ("dead-reckoning uncertainty") as well as incomplete knowledge of the environment (such as uncertainty about the exact

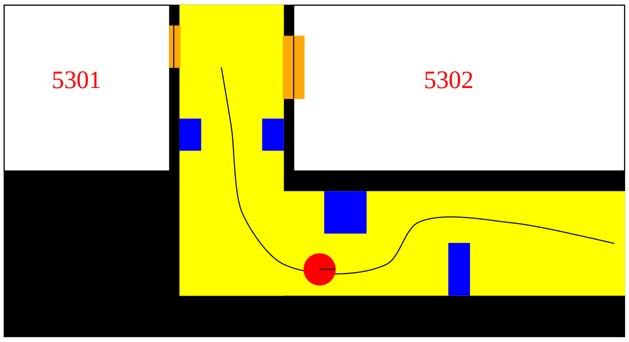


Figure 6: Local Obstacle Avoidance at 60 cm/sec

lengths of corridors). Its performance directly affects the task completion rate and the number of times the robot "gets lost." It can also negatively influence the robot's travel speed if the navigation module runs too slowly.

3.2.1 Estimating Position

In order to avoid getting completely lost, our navigation module maintains a probability distribution over the current *pose* (position and orientation) of the robot. Figure 7 shows two possible probability distributions of the robot, where the size of the circle is proportional to the probability that the robot is in that corridor.

Given new sensor information about the environment or how the robot has moved, plus the current distribution over all possible poses, the navigation module uses Bayes' rule to update the pose distribution. The updated probabilities are based on probabilistic models of the actuators, sensors, and the environment.

> *The navigation module follows the path generated by the path planner. It maintains estimates of the robot's current location and orientation using a POMDP, and then selects a desired heading that will follow the path. It gracefully recovers from noisy sensors and actuator uncertainty.*

This information, together with a prepared topological map, is automatically compiled into a Partially Observable Markov Decision Process (POMDP). The POMDP produced by our system discretizes the pose of the robot: orientation is discretized into the four compass directions (relying on the rectilinear nature of most buildings) and location is discretized with a precision of one meter. There is a trade-off: a coarse discretization leads to smaller memory and runtime requirements, but at reduced precision. For instance, Nourbakhsh [23] uses a similar model, but represents each corridor segment as a single Markov node. Figure 8 shows the Markov state representation and the representation of the actuator transitions between states and robot orientations.

Discretizing the pose allows us to abstract the raw sensor data. The raw, odometer data is discretized into virtual movement reports (e.g., "moved forward one meter" or "turned left ninety degrees"). The virtual movements abstract away low level control aspects, such as circumnavigating obstacles, by reporting the straight-line distance in the desired heading. Similarly, an evidence grid (obstacle maps centered on the robot), which integrates raw sonar data over time [20], is used to derive virtual sensors that report on the environment. For example, we model three sensors (a front, left, and right sensor) that report features such as walls and openings of various sizes (small, medium, and large). These abstract virtual sensor reports are less noisy and more closely approximate the probabilistic independence assumptions needed by Bayes' rule since several related raw sensors can be combined into one virtual sensor.

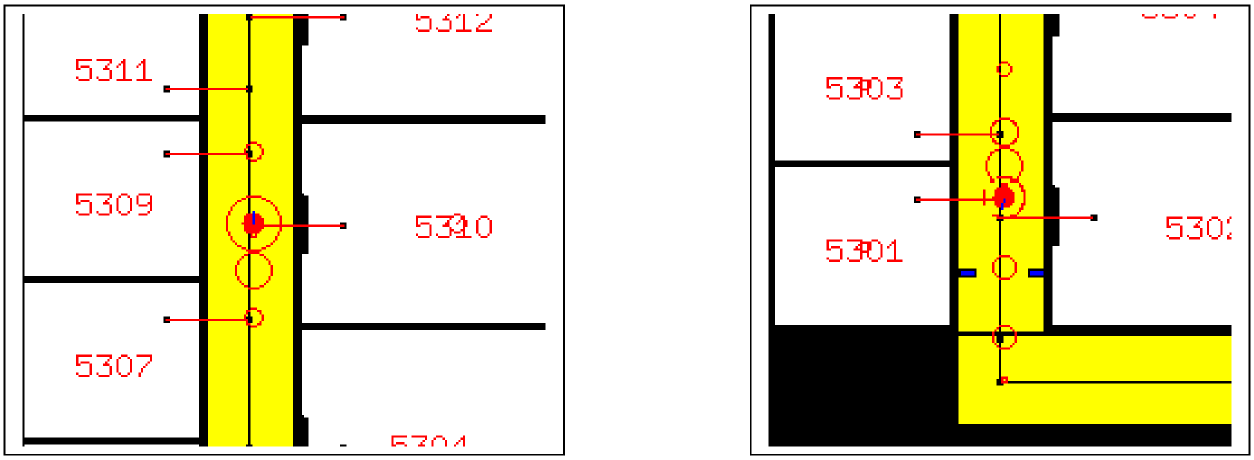


Figure 7: Two possible probability distributions (larger circles indicate higher probability). In each, the robot is shown in the most likely location.

3.2.2 Calculating Desired Headings

The navigation module takes the path produced by the path planning module and converts it to a mapping from pose distributions to desired headings. During navigation, the navigation module can then quickly determine the desired heading by indexing this mapping with the current pose distribution. Thus, it is very reactive to unexpected sensor reports and can gracefully recover from sensor noise and misjudgements about landmarks. For example, even if the robot strays from the desired path, it will automatically execute corrective actions once it realizes its mistake.

Assume, for instance, that the robot takes Path 1 in Figure 4, but misses the first turn and turns into the dead-end instead. Initially, most of the probability mass is in the corridor, since the robot believes that it made the correct turn. However, when it reaches the end of the dead-end, the sensor readings will make the probability mass shift to the correct position. The desired heading within the dead-end is South, so the robot turns, heads out of the dead-end, and then

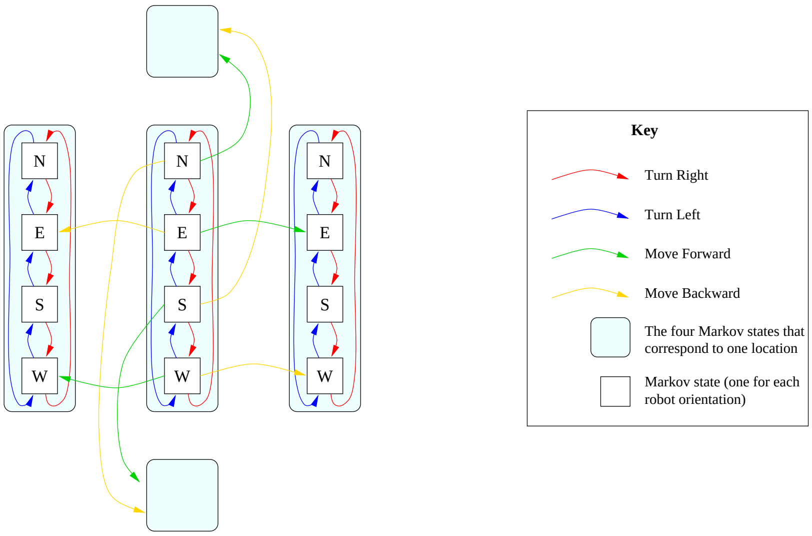


Figure 8: POMDP state representation.

turns right back onto the desired path. Note that this behavior is not triggered by any explicit exception handling mechanism, but results automatically from the way the pose estimation and heading selection interact.

4 High Level Layers: Path Planning and Task Planning

4.1 Path Planning

The task of the path planner is to take a pair of locations and create a policy that can be used by the navigation module to guide the robot. It also provides the task scheduler with estimated travel times between locations. By taking into account that some paths can be followed more easily than others, the planner can choose a path that steers the robot away from regions of the building where navigation is difficult.

The path planner determines how to travel efficiently from one location to another. Using a modified A search, it takes into account probabilities that corridors may be blocked, and the recovery cost of missing turns, yet still identifies the best path quickly [15].*

Our planner generates a path to the goal location and then the navigation module uses that path to seed a policy for navigating the robot to the desired location, allowing it to select the path efficiently.

The planner uses a generate, evaluate and refine strategy to efficiently find the path with the lease expected travel time, taking into account the difficulty of traversing each region, the possibility of missing turns, and the possibility of encountering closed doors or blockages in corridors. Paths are generated incrementally using a modified A* search algorithm that creates a sequence of paths from shortest to longest. As each path is generated, it is evaluated using a forward projection to determine the expected travel time, assuming that all doors are open and no corridors are blocked. The projection takes into account the possibility of missed turns, such as the second turn of Path 1 in Figure 9 since it occurs in the middle of a corridor.

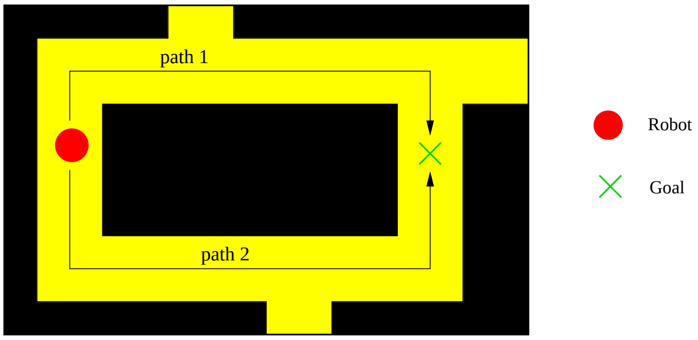


Figure 9: The expected travel time for path 2 is the same as the distance of path 2 divided by the speed of the robot, while the expected travel time for path 1 includes the possibility of missing the turn. The planner can conclude that it has the best path, without exploring any more alternatives.

In cases where a path can be blocked by a closed door, upper and lower bounds are calculated on the travel time needed to recover from encountering a closed door. In order to find the best path, the planner refines its plan by generating more paths and planning for contingencies, like closed doors, until the planner can prove that it has found a path with the least expected travel time. For example, in figure 9, the expected time to traverse path 2 is the same as the distance divided by the speed of the robot, while the expected time to traverse path 1 includes a recovery

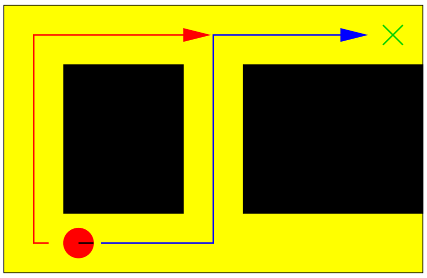


Figure 10: Trading off path length for the ability to follow a path. The robot can miss the turn in the middle of the lower corridor.

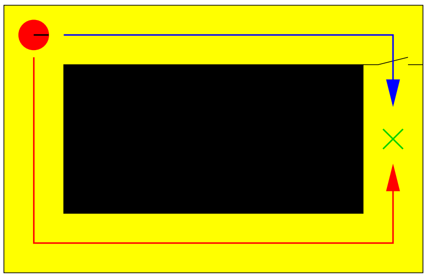
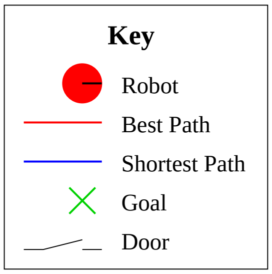


Figure 11: Taking the probability of blockages into account. The door on Path 1 is closed with a 50% probability.



cost for missing the turn. As a result, the planner can conclude that path 2 has the lowest expected travel time.

The use of expected travel time allows the planner to effectively trade off travel distance for the ease of traversing a route. Figures 10 and 11 illustrate two of these tradeoffs. In the first example, the planner selects a slightly longer path, with turns only at the ends or corridors since these turns are harder to miss. In the second example, the planner selects a longer route because the shortest route passes through a door with a 50% probability of being closed.

4.2 Task Planning

Delivery requests may come from the users at any time. The task planner must consider how each task will affect the others in the queue, and then find an interleaving of the requests which maximizes the satisfaction of all users. The simple approach to handle tasks in a first-come, first-served manner leads to inefficiencies and lost opportunities for combined execution of compatible tasks. In addition, the task planner must also know when actions fail and replan to achieve them since the robot operates in a dynamic world that is not completely known. The task planner effectively handles the multiple asynchronous goals and also monitors the execution of plans, compensating for failures.

The task planner creates plans for user requests [16]. It relies on PRODIGY*, a classical planning system [33]. The task planner can*

- *easily integrate asynchronous requests,*
- *prioritize goals,*
- *easily suspend and reactivate tasks,*
- *recognize compatible tasks and opportunistically achieve them,*
- *execute actions in the real world, integrating new knowledge which may help planning, and*
- *monitor and recover from failure.*

The task planning module is based on PRODIGY4.0 [33]. PRODIGY is a domain-independent nonlinear problem solver that uses means-ends analysis and backward chaining to reason about multiple goals and multiple alternatives of achieving them. It has been extended to support real-world execution of its symbolic actions. The planning cycle involves several decision points, including which goal to select from the set of pending goals, and which applicable action to execute. Dynamic goal selection from the set of pending goals enables the planner to interleave tasks, exploiting common subgoals and addressing issues of resource contention.

When a new task request arrives, PRODIGY creates a new top-level goal and starts planning to achieve it. PRODIGY uses its domain knowledge to create a series of actions that will achieve the goal. Usually, the task planner will be in the middle of executing another task. PRODIGY must process incoming requests and interleave them appropriately with existing tasks. It prioritizes the tasks and identifies when different requests can be combined and achieved opportunistically. It is

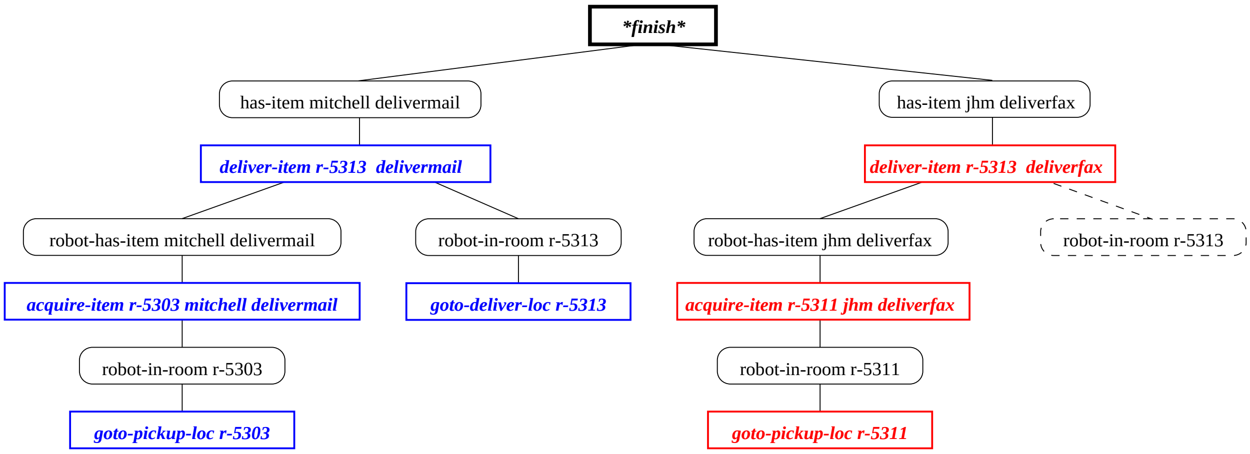


Figure 12: Search Tree for two task problem; goal nodes in ovals, required actions in rectangles.

also able to temporarily suspend lower priority tasks, resuming them when appropriate.

The plan tree shown in Figure 12 shows how PRODIGY expands the two goals `(has-item mitchell delivermail)` and `(has-item jhm deliverfax)`.

To find a good execution order of these actions, the planner selects the one that minimizes the expected total traveled distance from the current location. Actions that opportunistically achieve goals of other tasks are not repeated, e.g. both `<goto-deliver-loc jhm r-5313>` and `<goto-deliver-loc mitchell r-5313>` achieve the same goal, namely `(robot-in-room r-5313)`, so therefore only one of the actions will be executed. Figure 13 shows one possible execution sequence.

Figure 14 shows a trace of the interaction between the planner and the robot for a simple four-step plan. Each time PRODIGY selects an action for execution, the task planner maps it into a sequence of actions supported by the lower layers, most commonly navigation, but also including vision and speech (which are not described in this paper). Each line marked `SENDING COMMAND` indicates a command sent through the TCA interface to one of Xavier's other modules. Each of these commands is supported by one of the lower layers in the architecture, and is treated as an atomic action by PRODIGY. By abstracting away the details of how each request is achieved (e.g. which path the robot takes to a specified goal location), the task planner can more fully address issues arising from multiple interacting tasks, such as efficiency, resource contention, and reliability.

```
Solution:
      <goto-pickup-loc mitchell r-5303>
   [arrival of second request]
      <acquire-item r-5303 mitchell delivermail>
      <goto-pickup-loc jhm r-5311>
      <acquire-item r-5311 jhm deliverfax>
      <goto-deliver-loc mitchell r-5313>
      <deliver-item r-5313 jhm deliverfax>
      <deliver-item r-5313 mitchell delivermail>
```

Figure 13: One possible Execution Sequence

```
<GOTO-PICKUP-LOC MITCHELL R-5303>
   SENDING COMMAND (TCAEXPANDGOAL "navigateToG" #(TCA::MAPLOCDATA 567.0d0 2316.5d0))
   ...waiting...
   Action NAVIGATE-TO-GOAL-ACHIEVED finished.

   Verifying Location:  R-5303
   SENDING COMMAND (TCAEXECUTECOMMAND "verifyLocation" 5303)
   LOCATION-VERIFIED R-5303

<ACQUIRE-ITEM R-5303 MITCHELL DELIVERMAIL>
   SENDING COMMAND (TCAEXECUTECOMMAND "C_say" "Please place Tom Mitchell's mail on my tray.")
   SENDING COMMAND (TCAEXECUTECOMMAND "C_say" "Please indicate on my keyboard when you are finished.")
Are you finished placing Tom Mitchell's mail on my tray?  (y/i(mpossible)):  y
   COMPLETED-ACTION (ACQUIRE-ITEM 1 "Tom Mitchell's mail")

<GOTO-DELIVER-LOC MITCHELL R-5313>
   SENDING COMMAND (TCAEXPANDGOAL "navigateToG" #(TCA::MAPLOCDATA 567.0d0 4115.0d0))
   ...waiting...
   Action NAVIGATE-TO-GOAL-ACHIEVED finished.

   Verifying Location:  R-5313
   SENDING COMMAND (TCAEXECUTECOMMAND "verifyLocation" 5313)
   LOCATION-VERIFIED R-5313

<DELIVER-ITEM R-5313 MITCHELL DELIVERMAIL>
   SENDING COMMAND (TCAEXECUTECOMMAND "C_say" "Please take Tom Mitchell's mail from my tray.")
   SENDING COMMAND (TCAEXECUTECOMMAND "C_say" "Please indicate on my keyboard when you are finished.")
Are you finished taking Tom Mitchell's mail from my tray?  (y/i(mpossible)):  y
   COMPLETED-ACTION (DELIVER-ITEM 1 "Tom Mitchell's mail")
```

Figure 14: Planner/Robot Interaction. Blue indicates planner actions. Green indicates user-interaction.

4.2.1 Monitoring Execution

In any real-world domain, the outcome of executed actions is not guaranteed. The domain model underlying the plan is bound to be incompletely or incorrectly specified. Not only is the world more complex than a model, but it is also constantly changing in ways that cannot be predicted. Therefore any agent executing in the real world must have the ability to monitor the execution of its actions, detect when the actions fail, and compensate for these problems.

In our domain, the navigation module operates using probabilistic information, and therefore occasionally may get confused and report a success even in a failure situation. Thus, the planner always verifies the location with a more computationally expensive secondary test, namely vision or human interaction. (Note that after each `navigateToG` command in Figure 14, the planner verifies the outcome of the action.)

Since PRODIGY's planning algorithm is state-based, it examines the current state before making each planning decision. If the preconditions for a given desirable action are not true, PRODIGY must create subgoals to achieve them. Therefore, when an action fails, the *actual* outcome of the action is not the same as the *expected* outcome, and PRODIGY will attempt to find another solution. If, in Figure 14, the verification step had failed, the planner would, and select a new action to achieve its goals—in this case, it would re-execute the `navigateToG` command.

In a similar manner, PRODIGY is able to detect when an action is no longer necessary. If an action unexpectedly achieves some other necessary part of the plan, then that knowledge is added to the state information. Any actions that require that precondition will no longer require subgoaling to achieve it. Also, when an action accidentally *dis*achieves the effect of a previous action (and the change is detectable), the planner deletes the relevant precondition and PRODIGY will be forced to create a new subgoal to reachieve it.

In this manner, the task planner is able to detect simple execution failures and compensate for them. By interleaving planning and execution, the planner can acquire additional domain knowledge to make more informed planning decisions. For example, it can prune alternative outcomes of a non-deterministic action, notice external events (e.g. doors opening or closing), monitor limited resources (e.g. battery level), and notice failures. As a result, the information lost by abstraction can be reconstructed when unexpected (infrequent) situations arise. The interleaving of planning and execution reduces the need for replanning during the execution phase and increases the likelihood of overall plan success. It allows the system to adapt to a changing environment where failures can occur.

5 Related Work

Consensus is building in the mobile robotics community on the advantages of a layered architecture, consisting of behaviors, task-sequencing, and planning—the range of architectures adopting variants of this approach grows rapidly, including AURA [3], Rhino [7], SSS [10], ATLANTIS [14], RAPs [12, 19], Python [25] and SAPHIRA [28]. Generally such hybrid architectures combine low-level reactive control mechanisms with one or more deliberative layers. Our robot delivery architecture is consistent with such approaches, but takes an orthogonal cut at the decomposition—by task function rather than by architectural capability. In particular, the obstacle avoidance layer is essentially behavioral, the navigation layer contains behavioral and task-sequencing aspects, and both the path planning and task scheduling layers combine planning and task sequencing.

In character, our architecture has many similarities to behavior-based approaches advocated in the literature [6, 9, 18]. Lower layers are always running, even when higher layers are inactive, or not present. For instance, the obstacle avoidance module can keep the robot wandering safely, even without any "desired heading" input from the navigation module. In addition, lower layers are free (within some bounds) to ignore the input received by higher layers, essentially treating higher level plans and commands as "advice" [1]. For example, the local obstacle avoidance module can ignore the current desired heading to steer the robot around obstacles.

The architecture differs from traditional behavior-based approaches in that it makes heavy use of models and internal representations. This actually has the effect of *improving* efficiency and reliability, since the representations explicitly model the capabilities and limitations of the robot, and take uncertainty and incomplete information into account. By combining prior information (models) with current percepts, the robot is able to maintain representations that best reflect its current belief in the state of the world, given that it receives noisy, and often incorrect, sensor information.

Shakey the robot [21] was the first system to use a planning system on a robot. However, the underlying architecture was not as reliable as in Xavier, and therefore it operated in a very simple near-static world. The range of failures that could occur were very limited, and goals were not very challenging. There was little need for complex high-level reasoning. More recent work on putting planning systems on robots include CIRCA [4] Flakey [8], Dervish [23] and NRMA [26]. These systems all demonstrate the need for reliable, modular components where planning and forethought increase reliability and efficiency.

The deployment of Xavier is perhaps the most unique aspect of the our system. There are typically two types of interactions with robot agents. In the most traditional, exemplified by

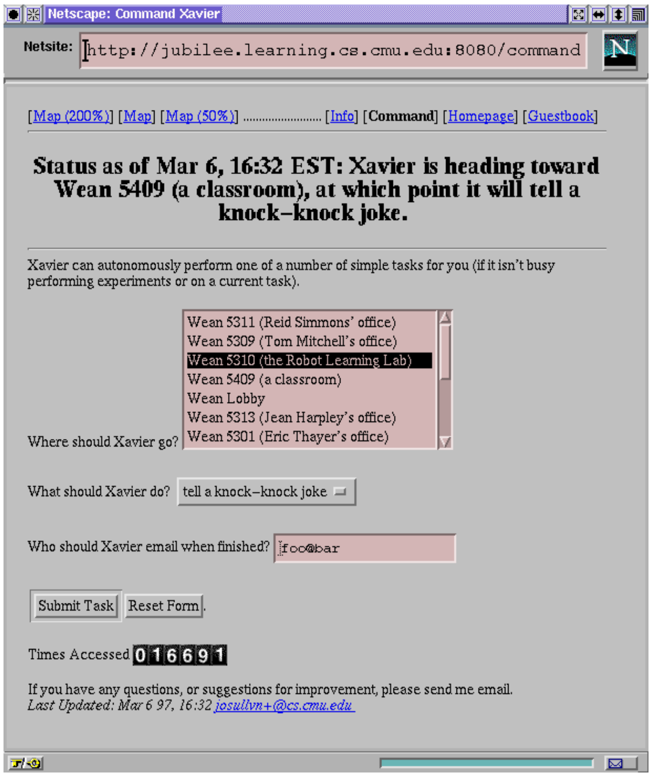


Figure 15: The World Wide Web Interface. The page to control the robot.

systems such as HelpMate [11] and other service robots [27], authorized users interact on a one to one basis with the agent. More recently, as "net robots" such as the USC Robotic Tele-Excavation, Chicago's Labcam are made available, a larger audience can control the agent. In these systems, the robots are teleoperated agents, with each command a low level control mechanism, monitored to prevent harm to the agent and to the enviroment. Xavier represents the first bridge between such extremes. Xavier is an autonomous agent, not simply a teleoperated robot. User requests are tasks to be performed, the actual decisions as to when and how are decided by the task scheduler and the other layers of the architecture—in one stroke both the need for teleoperation and for authorized users are removed. Operating autonomously, Xavier carries out tasks in an uncontrolled, inhabited office building—not interfering with, being dissuaded by, or worst of all harming the the people it encounters during task execution. Xavier operates remotely, communicating over a wireless link. Thus it may not be accessible at all times, and needs to be capable of operating without external intervention.

6 Results and Conclusions

The version of our robot architecture described in this paper has been in almost daily use since December 1995, mainly controlled by our World Wide Web interface, at `http://www.cs.cmu.edu/~Xavier`, and shown in Figures 15 and 16.

In the period from December 1, 1995 through January 31, 1997 (Table 1), Xavier received nearly 11,000 job requests. It attempted 2603 separate tasks (simultaneous requests to the same location were attempted together, and count as only one task), and reached its intended destination

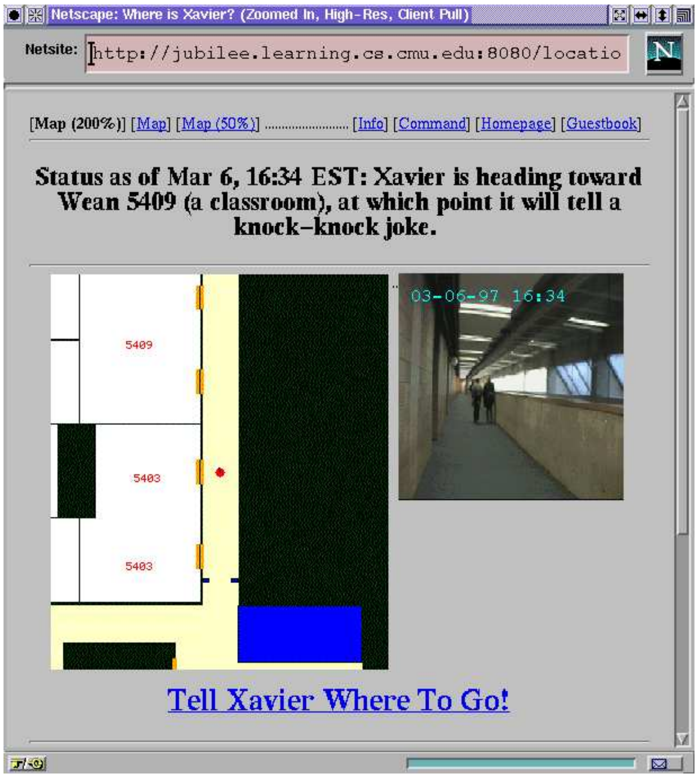


Figure 16: The World Wide Web Intercace. The page to monitor the robot.

Month	Days in Use	Jobs Attempted	Jobs Completed	Completion Rate	Distance Traveled (approx.)
December 1995	13	262	250	95%	7.65 km
January 1996	16	344	310	90%	11.37 km
February 1996	15	245	229	93%	11.55 km
March 1996	13	209	194	93%	10.06 km
April 1996	18	319	304	95%	14.11 km
May 1996	12	192	180	94%	7.90 km
June 1996	7	179	170	95%	8.24 km
July 1996	7	122	121	99%	5.42 km
September 1996	15	178	165	93%	8.16 km
October 1996	31	168	151	90%	7.78 km
November 1996	29	228	219	96%	10.48 km
December 1996	29	172	167	97%	8.83 km
January 1997	31	157	154	98%	7.53 km
Total	327	2,603	2,447	94%	110.25 km

Table 1: Performance Data for World Wide Web Tasks

in 2447 cases (94%). Each job required Xavier to move 42 meters (38 yards) on average for a total travel distance of over 110 kilometers (65 miles). The success rate is slowly climbing (from 90% to about 95%) as we find and correct bugs and refine the individual modules (December 1995 was an anomaly, since tasks in that first month were largely confined to a single corridor of the building). Many of the remaining failures are attributable to problems with our hardware (boards shaking loose) and the wireless communication—while the robot system itself runs on-board, the user interface (which includes the statistics-gathering software) operates off-board, connected by a wireless radio link.

More important than the raw data, our extensive experience with this architecture—both in development and in use—have taught us some valuable lessons about the construction of autonomous mobile robot systems:

- It is important to have solid components.
- Layering increases reliability.
- A modular architecture is easy to refine.
- Users want feedback.

More discussion on these points can be found elsewhere [31].

Acknowledgments

The authors would like to thank Greg Armstrong, Lonnie Chrisman, Domingo Galardo, Soshi Iba, Tom Mitchell, Sebastian Thrun, Hank Wan, and numerous others for their help developing the robot and providing feedback on our research.

This research was supported in part by NASA under contract NAGW-1175 and by the Wright Laboratory and ARPA under grant number F33615-93-1-1330. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations and agencies.

References

[1] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-87)*, pages 268–272, Seattle, WA, 1987. Morgan Kaufmann, San Mateo, CA.

[2] Ron C. Arkin. Motor schema-based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112, 1989.

[3] Ron C. Arkin. Integrating behavioral, perceptual and world knowledge in reactive navigation. *Robotics and Autonomous Systems*, 6:105–122, 1990.

[4] Ella M. Atkins, Edmund H. Durfee, and Kang G. Shin. Detecting and reacting to unplanned-for world states. In *Papers from the 1996 AAAI Fall Symposium "Plan Execution: Problems and Issues"*, pages 1–7, Boston, MA, 1996. AAAI Press, Menlo Park, CA.

[5] Johann Borenstein and Yoram Koren. The vector field histogram – fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991.

[6] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1986.

[7] Joachim Buhmann, Wolfram Burgard, Armin B. Cremers, Dieter Fox, Thomas Hofmann, Frank E. Schneider, Jiannis Strikos, and Sebastian Thrun. The mobile robot rhino. *AI Magazine*, 16(2):31–38, Summer 1995.

[8] Clare Congdon, Marcus Huber, David Koertenkamp, Kurt Konolige, Karen Myers, Alessandro Saffiotti, and Enrique H. Ruspini. CARMEL versus FLAKEY: A comparison of two winners. *AI Magazine*, 14(1):49–57, Spring 1993.

[9] Jonathan H. Connell. A behavior-based arm controller. *IEEE Journal of Robotics and Automation*, 5(6):784–791, 1989.

[10] Jonathan H. Connell. SSS: A hybrid architecture applied to robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 92)*, pages 2719–2724, Nice, France, 1992. IEEE Press, New York, NY.

[11] John M. Evans. HelpMate: An autonomous mobile robot courier for hospitals. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS 94)*, pages 1695–1700, Munich, Germany, 1994. IEEE Press, New York, NY.

[12] James R. Firby. An investigation into reactive planning in complex domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-87)*, pages 202–206, Seattle, WA, 1987. Morgan Kaufmann, San Mateo, CA.

[13] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 4(1):23–33, 1997.

[14] Erann Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 809–815, Seattle, WA, 1992. MIT Press, Cambridge, MA.

[15] Richard Goodwin. *Meta-Level Control for Decision-Theoretic Planners*. PhD thesis, School of Computer Science, Carnegie Mellon University, October 1996. Also avilable as technical report CMU-CS-90-109.

[16] Karen Zita Haigh and Manuela M. Veloso. High-level planning and low-level execution: Towards a complete robotic agent. In W. Lewis Johnson, editor, *Proceedings of First International Conference on Autonomous Agents (Agents '97)*, pages 363–370, Marina del Rey, CA, 1997. ACM Press, New York, NY.

[17] Alex Hills and David B. Johnson. A wireless data network infrastructure at Carnegie Mellon University. *IEEE Personal Communications*, 3(1):56–63, 1996.

[18] Maja J. Mataric. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8(3):304–312, 1992.

[19] Drew McDermott. Planning reactive behavior: A progress report. In K. Sycara, editor, *Innovative Approaches to Planning, Scheduling and Control*, pages 450–458. Morgan Kaufmann, San Mateo, CA, 1990.

[20] Hans Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61–74, Summer 1988.

[21] Nils J. Nilsson. Shakey the robot. Technical Report 323, AI Center, SRI International, Menlo Park, CA, 1984.

[22] Illah Nourbakhsh, Sarah Morse, Craig Becker, Marko Balabanovic, Erann Gat, Reid Simmons, Steven Goodridge, Harsh Potlapalli, David Hinkle, Ken Jung, and David Van Vactor. The winning robots from the 1993 robot competition. *AI Magazine*, 14(4):51–62, Winter 1993.

[23] Illah Nourbakhsh, Rob Powers, and Stan Birchfield. Dervish: An office navigating robot. *AI Magazine*, 16(2):53–60, 1995.

[24] Joseph O'Sullivan, Karen Zita Haigh, and G. D. Armstrong. *Xavier Manual (Version 0.3)*. School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1994. Unpublished internal report. Available via `http://www.cs.cmu.edu/~Xavier/`.

[25] David W. Payton, Julio K. Rosenblatt, and David M. Keirsey. Plan-guided reaction. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(6):1370–1382, 1990.

[26] Barney Pell, Douglas E. Bernard, Steve A. Chien, Erann Gat, Nicola Muscettola, P. Pandurang Nayak, Michael D. Wagner, and Brian C. Williams. An autonomous spacecraft agent prototype. In *Proceedings of First International Conference on Autonomous Agents (Agents '97)*, pages 253–261, Marina del Rey, CA, 1997. ACM Press, New York, NY.

[27] Catherine Rafflin and Alain Fournier. Learning with a friendly interactive robot for service tasks in hospital environments. *Autonomous Robots*, 3(4):399–414, 1996.

[28] Allessandro Saffiotti, Kurt Konolige, and Enrique H. Ruspini. A multivalued-logic approach to integrating planning and control. *Artificial Intelligence*, 76(1–2):481–526, 1995.

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

[30] Reid Simmons. The curvature-velocity method for local obstacle avoidance. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 96)*, pages 3375–3382, Minneapolis, MN, 1996. IEEE Press, New York, NY.

[31] Reid Simmons, Richard Goodwin, Karen Zita Haigh, Sven Koenig, and Joseph O'Sullivan. A layered architecture for office delivery robots. In W. Lewis Johnson, editor, *Proceedings of First International Conference on Autonomous Agents (Agents '97)*, pages 245–252, Marina del Rey, CA, 1997. ACM Press, New York, NY.

[32] Reid Simmons and Sven Koenig. Probabilistic robot navigation in partially observable environments. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1080–1087, Montréal, Québec, Canada, 1995. Springer-Verlag, Berlin, Germany.

[33] Manuela M. Veloso, Jaime Carbonell, M. Alicia Pérez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1):81–120, 1995.

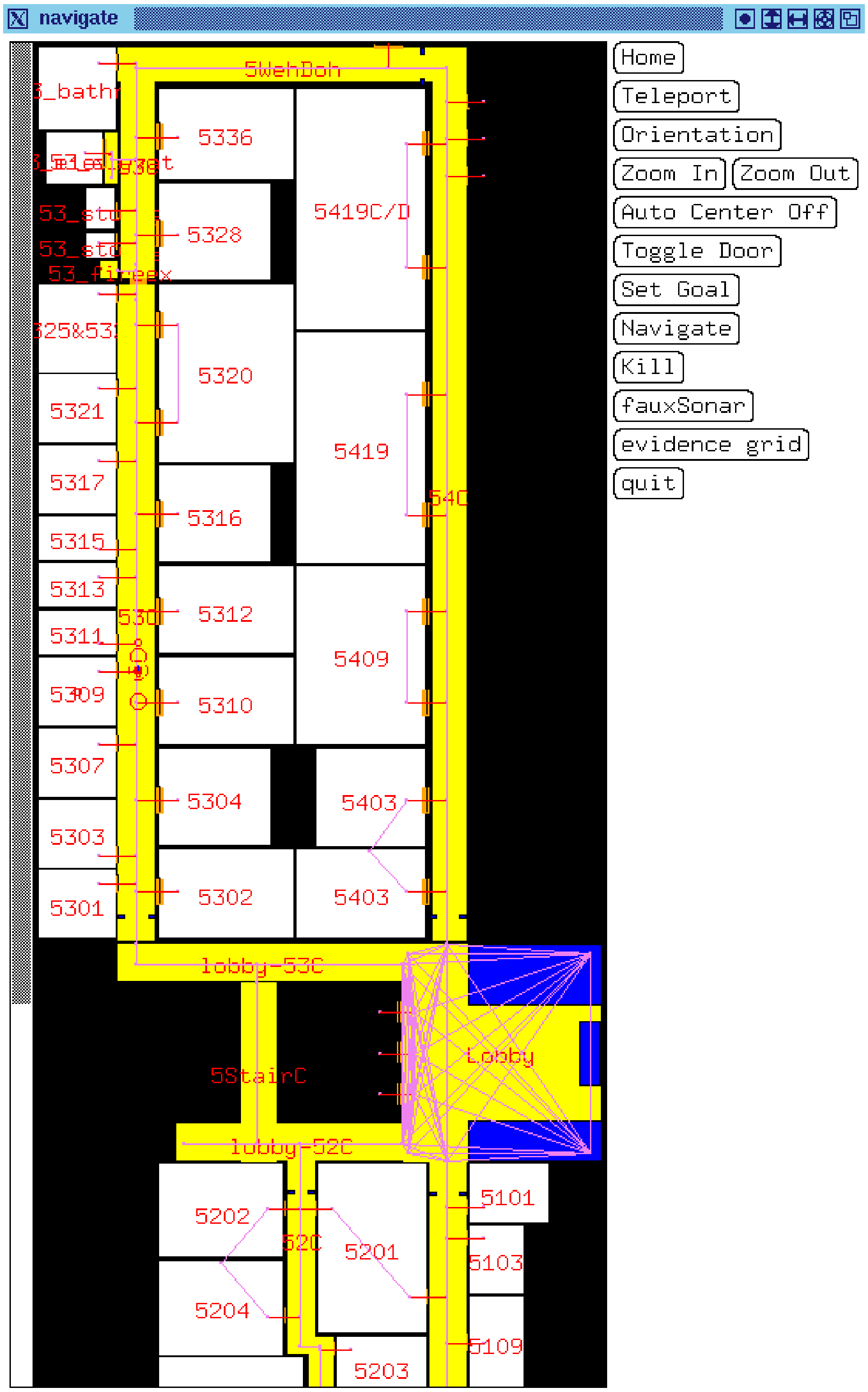


Figure 17: Navigate Window (Xavier's beliefs about the world)

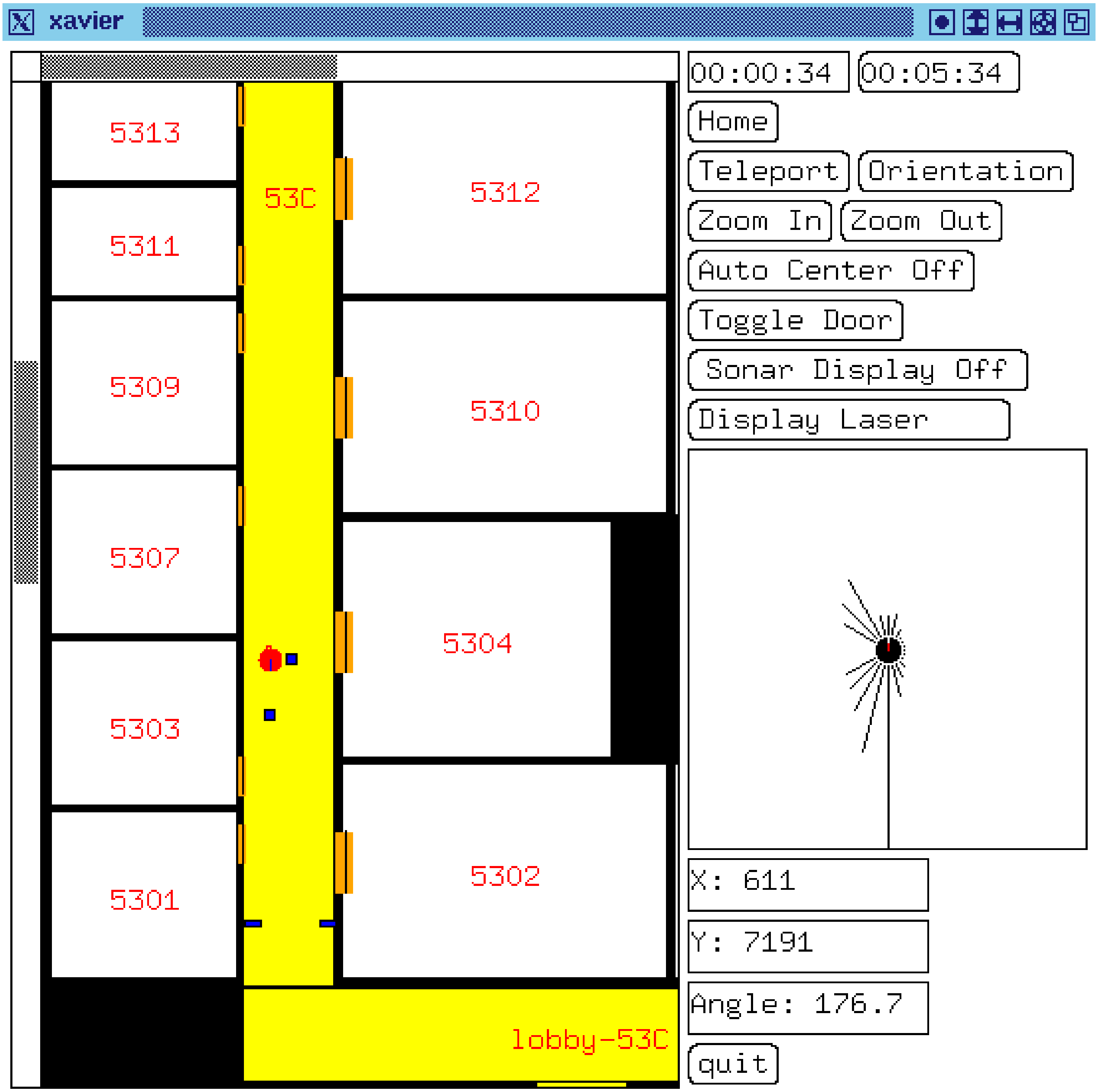


Figure 18: Xavier's World (as shown in the robot simulator)

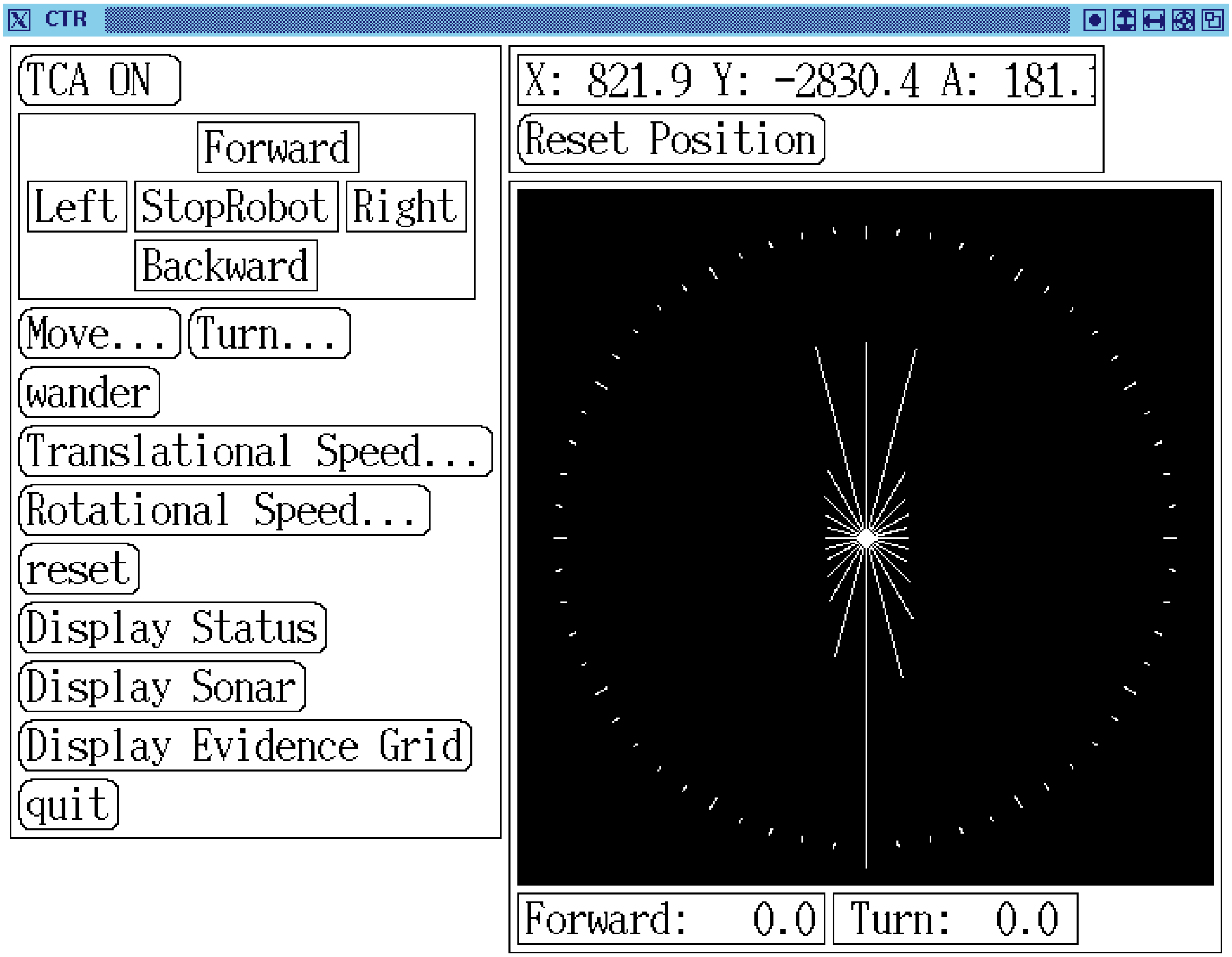


Figure 19: Controller Window, showing sonar readings

Figure 20: Camera Image shown in Figure 15.