

Toward Autonomous Driving: The CMU Navlab

Part II — Architecture and Systems

Charles Thorpe, Martial Hebert, Takeo Kanade, and Steven Shafer
Carnegie Mellon University

AN AUTONOMOUS MOBILE robot requires perception, planning, and control to act intelligently. Our goal in the Navlab project is to build a robot that can operate autonomously under a variety of conditions. Once we had developed the first versions of reliable perception software (detailed in the article on p. 31), we developed novel planning methods for rough terrain, and we designed and built systems software to forge the separate perception and planning modules into integrated systems. By directly confronting the central areas of perception, planning, and system building, we are filling in the missing links that will enable us to build reliable high-speed mobile robots.

Although they are important components, such other technologies as vehicle design, high-speed computing, and control theory are not the main bottlenecks; they have been or are being developed by other groups, often outside the mobile robotics community.

We designed and built the Navlab vehicle in 1986 as a testbed for vision and navigation experiments (see Figure 1). It is based on a standard commercial van, with a rooftop air conditioner, plus at least one video camera and a laser range finder mounted over the cab. Inside, the Navlab is a computer room, with five electronics

racks, 20 kilowatts of on-board power, and miscellaneous consoles and monitors. Over the course of our experiments, the vehicle has carried Sun 3 and 4 workstations, several generations of the Carnegie Mellon University/General Electric Warp supercomputer, various specialized real-time controllers, gyrocompasses, an inertial navigation system, and a satellite positioning system. We currently use only a small portion of the available rack space and electrical power. Our current real-time controller occupies four slots in a VME cage, and our general-purpose computers consist of three Sun 3s in a single cage and two Sun 4s in another. The operating system is Sun OS, so we have a standard environment for development and debugging.

Planning

The role of planning is to generate trajectories that meet goal requirements (such as positioning to see a landmark) without endangering the robot. It must also make sure that the robot is kinematically able to execute these trajectories, all in the presence of uncertainty in the robot's control and environment.

A number of systems address a subset of these issues. Early planners were designed for indoor mobile robots and assumed that the environment could be modeled as a flat surface with polygonal or polyhedral objects. The robot also was assumed to be circular and omnidirectional. Later, Laumond¹ and Jacobs² relaxed the omnidirectional

A MOBILE ROBOT'S SOFTWARE ARCHITECTURE ASSEMBLES ITS SENSING, PLANNING, AND CONTROL COMPONENTS INTO A COHERENT SYSTEM. EDDIE, THE ARCHITECTURE FOR THE NAVLAB MOBILE ROBOT, PROVIDES A TOOLKIT THAT LETS US BUILD SPECIFIC SYSTEMS QUICKLY AND EASILY.

requirement by modeling a car-like robot with a minimum turning radius. Hughes later developed a system that relaxed the indoor environment constraint, and a planner that planned paths in off-road environments.³ However, none of these systems could reason about sophisticated goal requirements and uncertainty in perception and control. The theoretical foundation in this area was laid by Smith, Self, and Cheeseman,⁴ but without actually building a usable planner. Planning with uncertainty has been explored in manipulation, but it is unclear how to apply the domain-specific nature of these techniques to planning for mobile robots.

While these contributions are important, it is often difficult to extend them to address the remaining issues or to generalize to other robots or environments. Our planner addresses these problems by providing a framework for building efficient planners for different types of robots, environments, goals, and uncertainty models.

The first step in building a planner is defining the constraints that must be used to compute a safe trajectory and to reach the goal. We have defined three types of constraints: sensing, environmental, and kinematic. First, we select positions at which the robot has to take some action, such as registering its position relative to the world or acquiring new sensor data. These positions are intermediate goals that give the planner additional constraints.

Second, we identify placements (configurations) in the environment that will incapacitate the robot or render it unable to move. These are environmental constraints that a trajectory must satisfy. Such configurations include those that bring the robot in contact with other objects in the environment (as have been modeled in traditional indoor robotics). Outdoor robots face other hazards: Configurations that cause the robot to tip over or place it in situations where it cannot propel itself forward are also to be avoided. Figure 2 shows a set of environmental constraints.

Third, we define kinematic constraints. Most robots are not omnidirectional. They cannot travel between two arbitrary configurations within given bounds. For example, car-like vehicles cannot move directly sideways. (The Navlab's minimum turning radius is seven meters.)

In addition to these three basic constraints, we must also account for uncer-

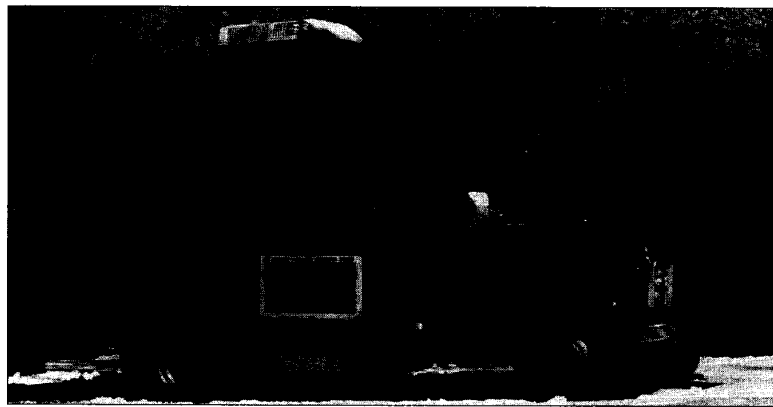


Figure 1. The Navlab.

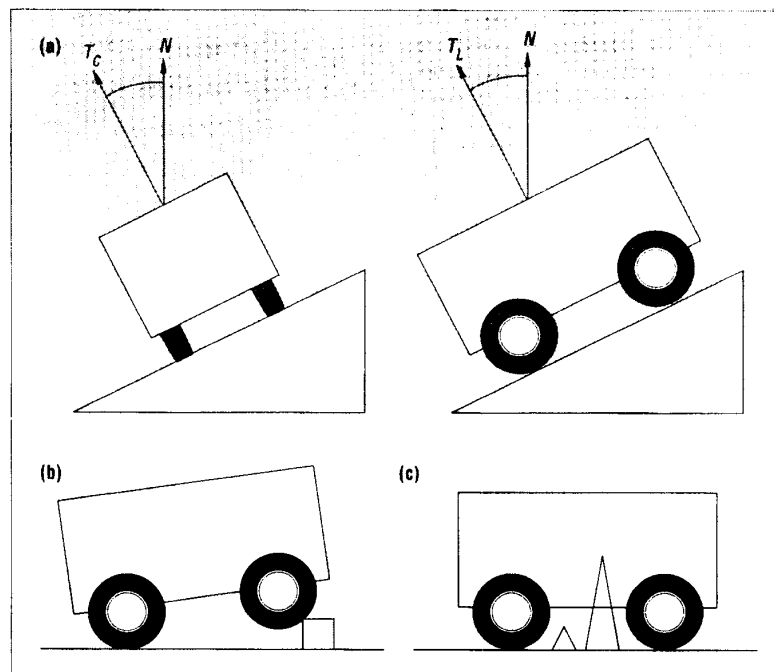


Figure 2. Environmental constraints: (a) tilt constraint; (b) support constraint; (c) body clearance constraint.

tainty in the robot's position. Sources of uncertainty range from random error in the robot's control to gross errors such as wheel slippage. Our local-path planner accounts for control-based uncertainty to avoid collisions and to guarantee attaining the goal.

The planner generates trajectories to the next sensing point using a range map of the terrain in front of the robot (acquired from

a laser range finder). Since the Navlab's pose can be represented by two translational parameters and one heading parameter, the planner must find an admissible trajectory through a 3D configuration space. Conceptually, each constraint is represented by a functional inequality of the form $f(\mathbf{p}) < K$, where \mathbf{p} is the vector of robot configuration parameters. The constraint is satisfied if the inequality is satisfied.

Applying the constraints divides the configuration space into admissible subspaces. The sensing positions form a subspace of this configuration space that comprises the goal of the path. The environmental constraints form a subspace representing unsafe configurations for the robot. The kinematic constraints dictate the functional form of the trajectory. The uncertainty constraints dictate an envelope about the trajectory guaranteed to contain the robot.


Analytic approaches to the problem are infeasible given the complexity of some of the constraint functions. Furthermore, the constraints depend on the terrain itself, which does not have a functional form. A straightforward approach is to tessellate the space into pixel-sized points, evaluate the constraints at each point, and search the resultant lattice. However, even for moderately-sized planning spaces, the number of points (states) makes the search prohibitively expensive.

Instead, our planner finds paths using a parameter resolution hierarchy. In this hierarchy, all constraints (sensing, environmental, kinematic, uncertainty, and so on) are evaluated across a subspace of configurations at a time (rather than individual configuration points), thus reducing the number of states in the search. Sensing and environmental constraints are evaluated across 3D voxels in configuration space, kinematic constraints are enforced between faces of the voxels, and uncertainty constraints determine the voxel expansion needed to bound the robot's pose.

The planner finds a trajectory by searching connected sequences of voxels. For a given subspace, it evaluates each constraint to determine whether all, none, or some configurations in the subspace satisfy the constraint. The planner begins by considering large subspaces. Passage through the subspace is permitted if all constraints are satisfied for all configurations. If at least one constraint fails for all configurations, the entire subspace is untraversable and is removed from further consideration. In the remaining case (at least one constraint is not satisfied by at least one configuration), the subspace may be traversable, so the planner subdivides the subspace into smaller spaces and continues to plan at a higher resolution. Most of the constraints are modeled uniformly as functional inequalities. Thus, the planner can classify a subspace into one of the three cases by computing

the upper and lower bounds for the function across the subspace and comparing them to a constant. A cost can be assigned to each subspace, and a standard depth-first, breadth-first, or heuristic search can be used.

The quadtree representation of the terrain described above provides an efficient way to implement the hierarchical search. Relevant terrain parameters, such as the minimum and maximum elevation, are maintained for each quadrant so that the planner does not have to go back to the



***A MOBILE ROBOT'S
SOFTWARE ARCHITECTURE IS
THE FRAMEWORK THAT
ASSEMBLES THE SEPARATE
COMPONENTS (FOR SENSING,
PLANNING, AND CONTROL)
INTO A COHERENT SYSTEM.***

highest resolution map to evaluate its constraints. Figure 3 shows one slice of the constraint space generated by running the cross-country planner on the elevation map of Figure 4. The crossed areas represent inadmissible terrain areas, that is, areas on which it is illegal to place the center of the vehicle. Due to the uniform way in which the constraints are modeled and the resolution hierarchy is built, we can apply the framework used in this planner to other classes of robots, environments, and goal specifications. Future work will include building a complete system around the planner to autonomously drive the Navlab off-road, implementing algorithmic improvements and using faster computer hardware to increase performance, and extending this work to operate on more capable off-road vehicles.

Architectures and systems

A mobile robot's software architecture is the framework that assembles the separate components (for sensing, planning, and control) into a coherent system. Simple

robots, performing simple tasks, often have an architecture that consists of a fixed sequence of subroutine calls, repeated without variation. More complex robots and missions require more structure to allow changing behaviors and conflicting subgoals and to specify functions and interfaces so groups of researchers can contribute to building the system.

The Navlab's current architecture is based on a toolkit called the Efficient Decentralized Database and Interface Experiment (EDDIE), which provides communications and a tight interface to our low-level vehicle control. On top of EDDIE we have built tools such as the annotated map, a mechanism for storing object and mission information. Our most ambitious system, the Autonomous Mail Vehicle, uses EDDIE and annotated maps for navigating suburban streets.

Background. The most conventional architectures separate robot software into separate modules for sensing, thinking, and control. This has the advantage of giving one module control of the vehicle, another control of all sensors, and a third control of modeling and planning. This decomposition groups design tasks according to the likely areas of expertise of separate research groups. However, this approach does not allow for high-speed special-purpose reflexes that require sensing, thinking, and control all in one tightly integrated module.

Brooks' subsumption architecture typifies the opposite approach.⁵ In his robots, each module covers the complete range from sensory input to control output. He divides his modules into a hierarchy of functions, each "subsuming" the lower levels. The first module watches sensor data and moves the vehicle away from obstacles. The next layer moves the vehicle randomly, unless the lowest layer takes over to avoid hitting an object. Higher layers add purpose to the wandering (moving toward open doorways, for example), look for objects of interest, and so forth. Each layer is relatively simple to build, and (in principle) mostly decoupled from adjacent layers. But with no central world model, it takes careful design to ensure that various modules are not working at cross purposes. Related ideas include reactive or reflexive planning, which emphasizes quick responses rather than careful preplanning,

and behaviors, which package sensing and control modes appropriate for specific situations.⁶

Several attempts have been made to build architectures that combine the best of both approaches. These systems typically propose a hierarchy in which sensor interpretation at each level feeds into both same-level planning and higher-level sensor interpretation.⁷ Plans at each level are decomposed into lower-level steps and given to the next lower level for execution. The hierarchies are often structured by time (from quick reflexes at the low level to slower processes at higher levels), data abstraction (from raw signals to symbolic reasoning), and space (from local effects to global databases). In trying to encompass all possible systems, these general-purpose architectures lose their prescriptive power. Their main contribution might instead be descriptive: providing a common vocabulary in which to discuss the differences between architectures.

For the Navlab, our first real architecture was the Communications Database for Geometric Reasoning, or Codger.⁸ Codger is a centralized architecture, focused on a module called the local-map builder. We designed the architecture to handle all communications and geometric transforms, making it easier to build and interface individual modules. Communications are anonymous; modules send data and requests to the local-map builder, and they receive responses when available, without knowing which other modules are generating or using data or where those modules are running. The local-map builder stores all geometric objects and maintains a history list of vehicle motion and position updates. Codger uses this list to answer geometric queries involving multiple coordinate frames. The local-map builder can take a location, specified relative to the vehicle at a particular time, and return the coordinates of that point either in the world frame or relative to the vehicle at a different time.

At the time, we did not know the nature of the modules and their interactions, and our major concern was to not preclude any conceivable system design. Thus, we made Codger very general and provided easy reconfiguration through anonymity of data storing and access. Once we knew the specific configuration of low-level modules needed to run the Navlab and how they

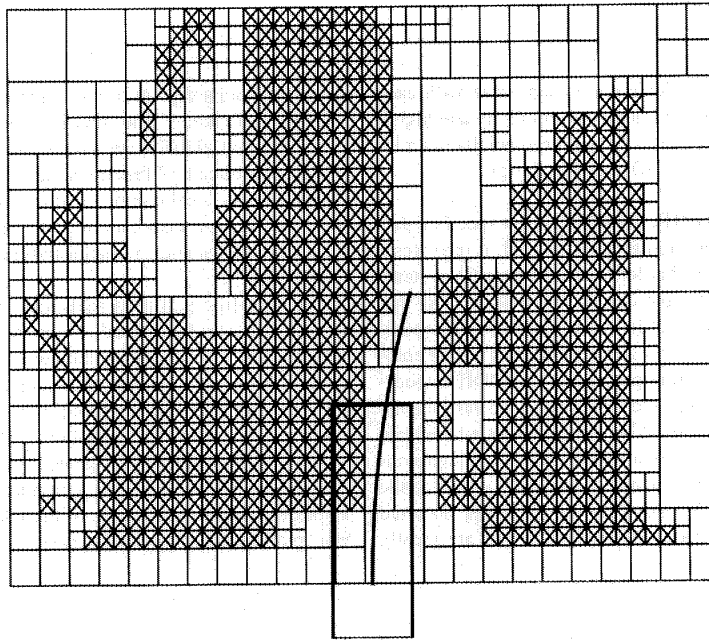


Figure 3. A planned path through cross-country terrain. The crossed squares are inadmissible regions, while the empty squares are passable areas.

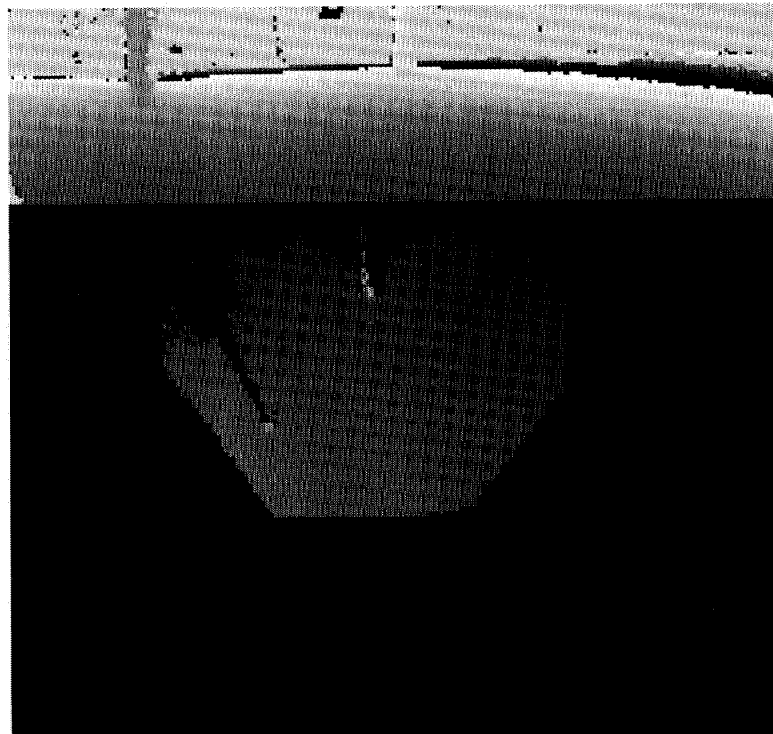


Figure 4. Range image and corresponding elevation map.

communicate and synchronize with each other, we sought the simplicity and higher performance of a more specialized architecture. EDDIE is that design.

EDDIE. EDDIE does not specify a particular architecture. Instead, it provides a toolkit that lets us build specific systems quickly and easily. Much of the data that Codger put in a central database properly belongs in a single module or pair of communicating processes, as EDDIE encourages. Vehicle positions — the most important models that an architecture maintains — are maintained by the lowest-level controller, which has the closest access to the vehicle and therefore the most accurate information. Communications are greatly

simplified and are point to point, increasing their efficiency. The map is divided into local and global representations. By splitting architectural functions into separate pieces for local communications, vehicle history, and map handling, the individual modules are much smaller and easier to maintain.

EDDIE's first part is its real-time controller. This module handles low-level vehicle control, manages communication with higher-level modules, and maintains the current vehicle position. Vehicle motion commands arrive at the controller labeled as either "immediate" or "queued." The controller parses incoming commands, handles the queue, and talks to the hardware motion controller to set new steering-

wheel positions and vehicle velocities. By frequently querying the vehicle's encoders, the controller can maintain an accurate dead-reckoned position estimate. EDDIE does not maintain the vehicle's position history. The only time EDDIE needs to know the vehicle's position is when new data is acquired or during trajectory planning. It is easier and more accurate to dispense with history mechanisms and instead to query the controller for the current vehicle position each time an image is digitized and whenever a planner needs to know the location.

The vehicle controller uses different tracking strategies to keep the vehicle on the desired path. It can also follow a previously recorded map if the perception clients are temporarily unable to navigate the vehicle. This keeps the vehicle on a safe path while it turns sharp corners that are outside the camera's field of view or while it travels through featureless or confusing visual scenes. Another safety consideration is smoothly regulating velocity, trading some reduction in velocity accuracy for smooth accelerations and reduced vehicle roll around sharp curves. The controller warns against system failures and records a log of events for future reference. This is extremely valuable in system configuration and debugging. The controller also uses data from the inertial navigation system and the encoder to find the best estimate of the vehicle's current position, and it relays this estimate to external clients through the Ethernet. Figure 5 shows an accurate estimation of the vehicle's position using the inertial navigation system (solid line), and a less accurate but still stable estimation using only dead reckoning (dotted line). The clients are managed by a software server that prioritizes the connections to meet the needs of many clients without degrading the performance level required by critical system components.⁹

Closing all position-estimation loops through the controller allows transparent path modifications. We have implemented a joystick interface that lets the user modify commanded trajectories. Joystick input is simply summed with computer input, so the user has the sensation of "nudging" the vehicle away from its planned path. We are also equipping the Navlab with a "soft bumper," a ring of ultrasonic range sensors to detect nearby objects

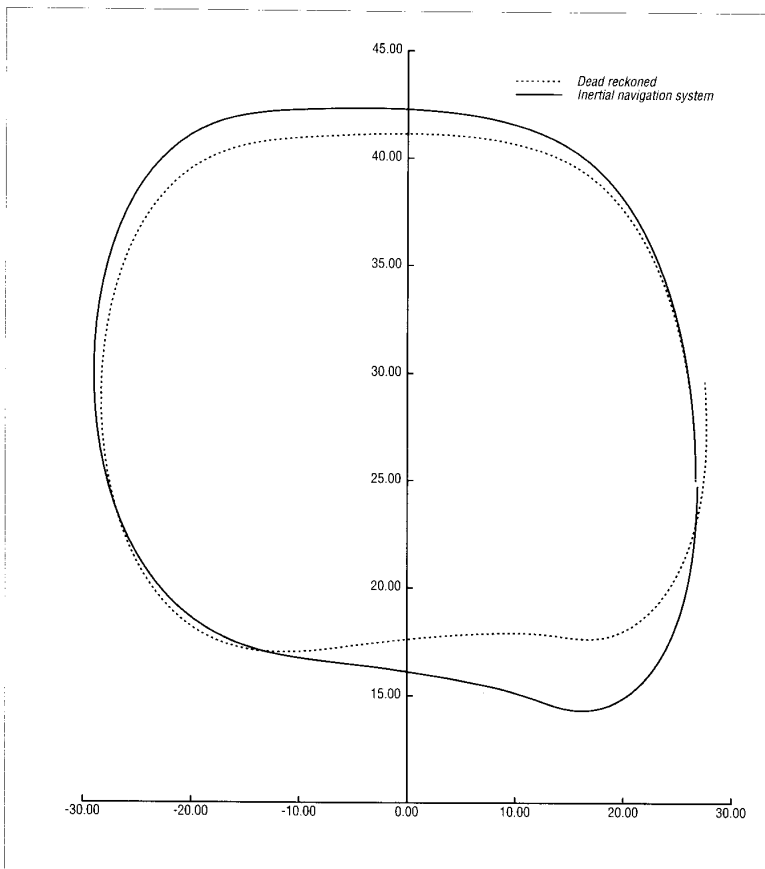


Figure 5. Position estimation during a robot run. The solid line shows the accurate vehicle track given by inertial navigation sensors. The dotted line shows the less accurate vehicle track estimated by dead reckoning.

before collision. When completed, the soft bumper will interact with the controller in the same manner as the joystick (by adding its control input to the planning input), but it will have progressively higher gains as the time to collision decreases. Previous systems would have been destroyed by this subversion of planned paths, since Codger kept the vehicle's position history by an open-loop expectation of perfect path tracking. In EDDIE, all position queries are handled directly by the low-level controller and are therefore answered correctly, even if the path has been modified.

Communications in EDDIE are unexciting and uninteresting, but fast, with point-to-point connections. We currently use transmission-control protocol/Internet protocol over the Ethernet, but we could use shared memory or other protocols for particular connections, as needed. Instead of building special-purpose synchronization mechanisms, EDDIE simply uses a blocking read to pause module execution until data arrives.

Annotated maps. EDDIE does not have a global map at its center. It uses local positions only for obstacle avoidance or path following, and it never writes them into a map. Global, permanent maps are handled by a separate mechanism, called annotated maps.

Annotated maps start with a geometric representation of objects, such as roads, intersections, and landmarks. The annotations are tied to particular locations or objects and contain a wide variety of information — both procedural (actions and methods) and declarative (data) — not usually contained in maps. Annotations can range from high level ("church") to geometric ("steeple height 25 meters") to sensor specific ("look for long, nearly-vertical edges") to raw data ("color R1 G1 B1"). The knowledge in an annotation can come from many sources, such as human experts, mission-planning software, and even the vehicle's own observations and experiences on previous missions.

A map-manager module controls the annotated map. The module provides two forms of access: queries and triggers. Queries let a module fetch information on demand, and they return all annotations of the requested type within a specified polygon. Typical queries ask for descriptions of landmarks or for which recognition methods

have worked for this landmark on previous vehicle runs. Triggers are a special form of annotation, monitored by the map manager. When the vehicle reaches a trigger's location, the map manager automatically sends a specified message to a named module. Triggers can be set up during mission planning and used to awaken "sleeping" processes at specified locations or to alert a running module to a change in conditions. In a typical run, triggers tell the vehicle when and where to look for landmarks and

***WE BEGAN WITH THE FIRM
CONVICTION THAT THE BEST
WAY TO MAKE REAL PROGRESS
WAS TO BUILD COMPLETE
SYSTEMS AND TO ELIMINATE
THE BOTTLENECK OF
INADEQUATE PERCEPTION.***

when to switch from road-following code to the slower intersection-navigation code.

Annotated maps are not designed to be a master control. Rather, they serve as a scratch pad (for queries) and alarm clock (for triggers). Annotations have a standard format for header information, such as type and location. The format for the rest of the annotation is defined by the modules that post and retrieve the annotations, and need not be interpreted by the map manager.

Annotated maps provide a convenient framework for organizing knowledge. Tying the knowledge in annotations to particular locations in the map lets the system preplan difficult mission segments and retrieve that information efficiently during execution. This framework enables missions that would not otherwise be possible due to real-time constraints and limits in processing and algorithmic power.

Autonomous Mail Vehicle. We have built several systems on top of EDDIE and the annotated maps. The Navlab's road-following system is the Autonomous Mail Vehicle. This system draws its inspiration from postal deliveries in suburban and rural areas, which follow the same route

day after day, deterred by neither "rain nor snow nor dark of stormy night." The mail carriers drive at relatively slow speeds, often on many different kinds of roads. They perform gross navigation through a network of roads and intersections, and perform fine-position "servoing" to mail boxes.

This type of system is part of a broader class of applications that focuses on map building and reuse, positioning, road following, and object recognition. Our AMV project is investigating these issues, including strategies for using different sensors and image-understanding operators for the perception components.

Our most ambitious mission so far is a 0.4-mile run on unmodified suburban streets in Pittsburgh's North Hills area. This involved

- driving along curving suburban streets with no pavement markings, including many different types of driveways;
- traversing four intersections, at two of which the Navlab had to make a 90-degree left turn;
- stopping for unexpected obstacles and resuming motion when clear; and
- locating landmarks for position updates and for finding the destination.

We built an annotated map of the route, driving the Navlab by hand and using the laser scanner to record the location of 3D objects. Object positions were measured in multiple images, to discard moving objects (pedestrians, cars, dogs) and to improve the accuracy of measured positions. The map was then annotated with triggers that controlled execution of the vehicle path. During the run, the vehicle began slowly as it found landmarks to initialize its position. A trigger then caused the vehicle to speed up until it approached the first turn. At that point, triggers caused various modules to slow the Navlab, find 3D objects, match them against the map, and update the vehicle's position estimate. Through the turn, the vision sensors could not see the road, so another trigger caused the vehicle to rely on dead reckoning until the vehicle was lined up with the next road, when the road was again in the field of view and vision could resume control. The run proceeded in this fashion until the final triggers, which matched the mailbox at the destination with the map and brought the vehicle to a stop.

Figure 6 shows the map used in this system. The roads are parallel lines, and the map shows two intersections. The small circles and dots are landmark annotations, each indicating the location of an object used for matching. Triggers are indicated by line segments across the road. The vehicle takes appropriate action whenever it crosses one of the lines.

Contributions and lessons

We began the Navlab project six years ago with the firm conviction that the best way to make real progress on outdoor mobile robots was to build complete systems and to concentrate our efforts on eliminating the bottleneck of inadequate perception. We still agree with and follow those convictions. However, we have been surprised (usually unpleasantly) by several other aspects of building mobile robots:

sensor problems, the difficulty of using experimental computers, questions of how to evaluate our work and compare it with results from other groups, and the critical importance of simplicity and of defining the environment in which the vehicle must operate.

Contributions. The Navlab experiments have validated and demonstrated several new ideas. Cross-country trajectory planning requires not only a representation of obstacles, but also reasoning about vehicle capabilities, limits, and inaccuracies. These constraints can be combined efficiently and powerfully to guide the vehicle up to the limits of its sensory and mechanical abilities. Simple architectures work best. Dictating the structure of the data and control flow is not needed. It is better to build a toolkit that provides communication, synchronization, map-data handling, and clean interfaces to the low-level control, and to

let individual system builders tailor the system structure to their own needs.

Lessons. Mobile robots operate in a certain environment to carry out a certain task. There is currently no such thing as a completely general-purpose robot, a universal vision system, or a generic architecture. Tracking a highway requires substantially different processing from driving cross-country. Some of the concepts are shared (local-map building, control), and some systems use shared modules (such as neural nets) which adapt to different situations. But the way to build mobile robot systems now is to incorporate knowledge of the task and the environment in the design, from the beginning. Too often, neat ideas are investigated in perception or planning and then artificially matched to an environment and a task. While this is great for demonstrating new research results, it usually does not contribute much to mobile robots.

Simplicity. The simplest approach is always the best. Designing a complex system does not solve any problems, especially if the components of the system have not been considered yet. The research community is full of proposed architectural standards that needlessly complicate mobile robots and that are not based on experience with working perception systems. Simpler is better. For example, the approach we have followed in our AMV system is to

- (1) Define the task — Track roads with the help of a map, and perform actions at specific locations.
- (2) Develop and analyze the necessary components (road following, object detection, map building).
- (3) Build and evaluate the components separately to understand their limitations — For example, we first built a smaller system that tracks a road map and stops at specific objects; then we expanded to annotated maps and the AMV.
- (4) Define representations that match the task, such as the annotated maps.
- (5) Assemble components and representations in a system configured for the task— The system is “simple” in that it includes only the functionality needed for the task using the selected components.
- (6) Experiment — The experimental phase should be used to evaluate how well

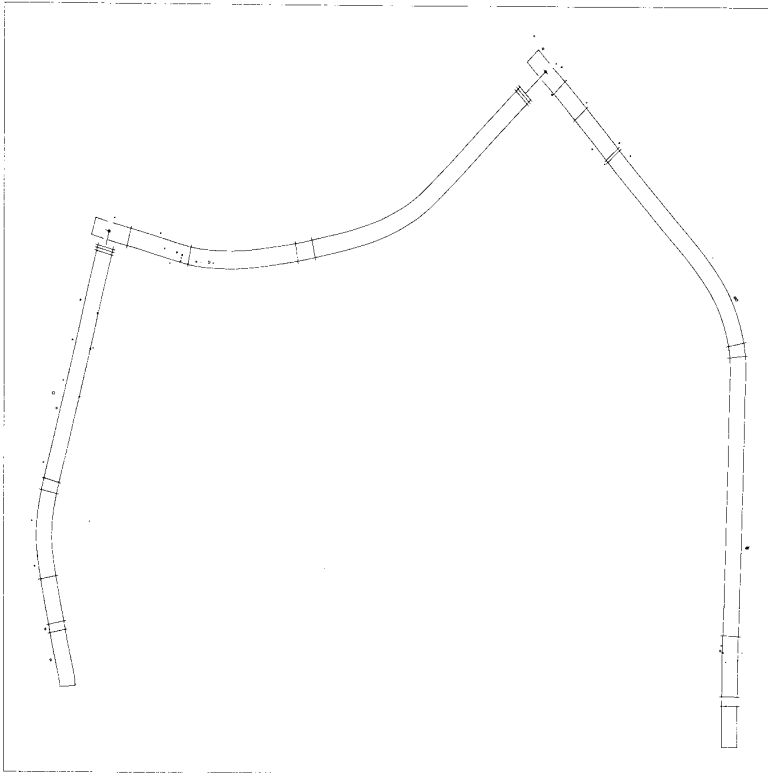


Figure 6. An annotated map of a suburban neighborhood, showing roads, intersections, landmark annotations (small circles and dots), and trigger annotations (lines across the road).

the mission is carried out and perhaps to add new perception components or modify the representations. It should not be used for debugging a large, complex system.

Computation. Of course, fast computation is of great help in building a mobile robot system. Not only does it improve the performance of the final system, it also holds the promise of more images processed, faster runs, more experiments, and thus faster progress in the basic research. We have found, however, that faster computation should not be the highest priority. Especially in the early stages of a mobile-robot project, researchers should try many different approaches to perception. It is more important to have easy-to-use computers with well-supported and efficient compilers than to have the ultimate in running speed. Good support for I/O is also crucial, both for image digitization and for communicating results. After six years, our algorithms are now stable enough that we can properly take advantage of non-standard high-speed machines. Still, those machines should be stable and well supported. It is difficult to perform robotics research at the same time as hardware or operating-systems research.

Vehicles. The vehicle itself is an integral part of a mobile robot system, not just a platform on which experiments are conducted. The Navlab was specialized for our early systems, and it provides the high-accuracy motion and slow speeds we needed.¹⁰ It was not designed for rough-terrain motion, nor for highway speeds. We are now building new testbed vehicles that will be capable of the higher speeds that our perception and control systems can now handle. It will also be more capable of rough-terrain operation. We are selecting and modifying our testbed vehicles to complement the capabilities of our sensors, perception algorithms, and planners.

Controllers. Real-time mobile robot controllers need to integrate a wide range of capabilities beyond just control theory: position estimation, path mapping and tracking, human interfaces, fast communication, multiple-client support, and monitoring vehicle status for safety and debugging. Most mobile robots do not push the limits of current control theory. However, the major issue in controller design is not

control theory, but rather, design for system integration.

Debugging and monitoring. At slow speeds, it is easy to watch a system's performance. Our first color road trackers, for instance, ran in tens of seconds, which gave us ample opportunity to watch graphics, save files to disk, note the vehicle's responses, and so forth. It is much more difficult to debug a system running at higher speeds. Our YARF system (for "yet another road follower"; see the article

THE SIMPLEST APPROACH IS ALWAYS THE BEST. DESIGNING A COMPLEX SYSTEM DOES NOT SOLVE ANY PROBLEMS, ESPECIALLY IF THE SYSTEM'S COMPONENTS HAVE NOT BEEN CONSIDERED YET.

on p. 31) now runs in less than a second, which is faster than we can write an image to disk (for later examination), examine the debugging graphics, or even read text output. As a corollary, YARF can now process hundreds of images in a typical run, or thousands of images during a day's experiments, which makes examining the output by hand tedious at best. We need both better technology (faster disks, better video recorders, and so on) and better ideas for debugging complex real-time systems.

Experimental evaluation. But even with proper tools to monitor a system, it is difficult to measure progress. The basic problem is answering the questions "Does it work?" and "Does it work better?" Some systems are easy to measure: Did an obstacle-avoidance system run over a tree or not? Others are more difficult: Did the vehicle clip a corner because of bad calibration, bad trajectory planning, bad image processing, or bad control? The problems become worse when comparing work from different research groups. All papers on road following claim success. Most are missing crucial details that would let others evaluate competing algorithms. Even

where all the details of the software are spelled out, crucial differences in hardware (processing rates, camera capabilities, vehicle and camera control, and so on) make head-to-head comparisons difficult. Common image databases provide only a small part of the solution, since different algorithms and vehicles might need different sensor vantage points, image-collection frequency, auxiliary data, and so forth.

WE ARE STILL IN THE EARLY stages of understanding how to build reliable outdoor mobile robots, both at CMU and in the community as a whole. It is too early to define standards for most modules or architectures. We are still far from being able to design a robot top-down from general specifications, and from being able to build perception algorithms with specified performance on demand.

The progress in our group and others is largely attributable to the experimental approach and the emphasis on building complete systems. Mobile robot research is not just research in perception algorithms, or sensors, or architectures, or computers, or vehicles, or controllers. Many fine modules, developed in isolation in the laboratory, have proved difficult to use or incomplete in the context of real outdoor systems. Our greatest advances have come by developing modules to fit a certain system need, using real vehicle data for development and debugging, and then testing the modules in a complete vehicle running realistic experiments.

This experimental approach will continue to be fruitful. In our project's first six years, we have gone from excruciatingly slow motion in benign conditions (two centimeters per second along clean sidewalks) to the vehicle's top speed (20 miles per hour) on a variety of real roads. Big challenges remain, both in driving on roads (changing lighting conditions, changing road shapes and lane markings, traffic) and in driving cross-country (higher speeds, mapping terrain, avoiding obstacles). We are working in both these areas. In addition to our perception research, we are refining our software architecture, continuing to develop maps and planning systems, and building new testbed vehicles for both on- and off-road systems.

While general-purpose systems are still far off, the large amount of experimental work over the past few years has brought several mobile-robot research groups to the threshold of applications in limited domains. Prototype robots are being proposed or built for several environments. Barren terrain, such as planetary surfaces or some hazardous-waste sites, allows easier perception. Limited-access environments, such as underground or strip mines, decrease the need for safety checks and eliminate unknown moving obstacles. The task of convoy following relies on a person driving the lead vehicle to avoid difficult situations, while subsequent robotic vehicles have the much simpler task of tracking the leader. Other applications involve a human supervising one or more semi-autonomous vehicles, so the vehicles can handle routine cases and decrease operator workload. All these applications will not only be useful in themselves, but also will continue to build the components needed for the truly intelligent autonomous vehicles of the future.

Acknowledgments

Navlab work is the product of many people. Planning and systems have been done by Tony Stentz and Eddie Wyatt. The new controller is the work of Omead Amidi. Dave Simon built the first AMV prototype, and Jay Gowdy continues its development. Karl Kluge is following structured roads with explicit models, while Jill Crisman and Didier Aubert work on unstructured roads with simple appearance models. Dirk Langer is working on the "soft bumper." Ken Rosenblatt is developing new system integration approaches. Dean Pomerleau, a student of Dave Touretzky, does neural nets on the Navlab. Thanks also to those who keep the Navlab alive and productive, especially Jim Frazier, Bill Ross, Jim Moody, and Eric Hoffman. This article benefited from comments and figure contributions from many people, especially Dirk Langer, Didier Aubert, Karl Kluge, Omead Amidi, Jill Crisman, Dean Pomerleau, and Jay Gowdy.

This research is sponsored in part by contracts from DARPA (titled "Perception for Outdoor Navigation" and "Development of an Integrated ALV System"), by NASA under contract NAGW-1175, by the National Science Foundation contract DCR-8604199, and by the Digital Equipment Corporation External Research Program.

References

1. J.-P. Laumond, "Finding Collision-Free Smooth Trajectories for a Nonholonomic Mobile Robot," *Proc. Int'l Joint Conf. Artificial Intelligence*, 1987, Morgan Kaufmann, San Mateo, Calif.
2. P. Jacobs and J. Canny, "Planning Smooth Paths for Mobile Robots," *Proc. IEEE Int'l Conf. Robotics and Automation*, CS Press, Los Alamitos, Calif., 1989, pp. 2-7.
3. M. Daily, J. Harris, and K. Reiser, "An Operational Perception System for Cross-Country Navigation," *Proc. Image-Understanding Workshop*, Morgan Kaufmann, San Mateo, Calif., 1988.
4. R. Smith and P. Cheeseman, "On the Representation and Estimation of Spatial Uncertainty," *Int'l J. Robotics Research*, Vol. 5, No. 4, Winter 1986, pp. 56-67.
5. R. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE J. Robotics and Automation*, Vol. RA-2, No. 1, 1986, pp. 14-23.
6. D. Payton, "An Architecture For Reflexive Autonomous Vehicle Control," *Proc. IEEE Int'l Conf. Robotics and Automation*, CS Press, Los Alamitos, Calif., 1986, pp. 1,838-1,845.
7. J. Albus, H. McCain, and R. Lumia, "NASA/NBS Standard Reference Model for Tele-robot Control System Architecture," Tech. Report 1235, National Bureau of Standards, 1987.
8. A. Stentz, "The Codger System for Mobile Robot Navigation," in *Vision and Navigation: The Carnegie Mellon Navlab*, C. Thorpe, ed., Kluwer Academic Publishers, Norwell, Mass., 1990, pp. 187-200.
9. O. Amidi, "Integrated Mobile Robot Control," tech. report, Carnegie Mellon Univ. Robotics Inst., Pittsburgh, Pa., 1990.
10. K. Dowling et al., "Navlab: An Autonomous Navigation Testbed," in *Vision and Navigation: The Carnegie Mellon Navlab*, C. Thorpe, ed., Kluwer Academic Publishers, Norwell, Mass., 1990, pp. 259-282.

Charles Thorpe is a senior research scientist at Carnegie Mellon University's Robotics Institute, where he directs research on the Navlab. His photo appears on p. 42. He is also involved with robots for planetary exploration and underwater mapping. His research interests include computer vision, planning, and the control of robot vehicles in unstructured outdoor environments. He is also editor of the book, *Vision and Navigation: The Carnegie Mellon Navlab*.

Thorpe received his PhD in computer science from CMU in 1984.

Martial Hebert is a research scientist at Carnegie Mellon's Robotics Institute, where his current projects include modeling environments from range data for autonomous vehicles, supervised mapping of plants, and object modeling for manipulation in unstructured environments. His photo appears on p. 42. His research interests include building and recognizing 3D environment models from sensor data.

Takeo Kanade is a professor of computer science, codirector of the Robotics Institute, and chair of the PhD program in robotics at Carnegie Mellon. His photo appears on p. 42. He is the principal or coprincipal investigator for the university's projects on image understanding, vision systems for autonomous land vehicles, and the NASA Mars Rover.

Kanade received his PhD in electrical engineering from Kyoto University, Japan, in 1974. He is an AAAI fellow and a founding editor of the *International Journal of Computer Vision*.

Steven Shafer is an associate professor at Carnegie Mellon, and associate director of the PhD program in robotics. His photo appears on p. 42. His work in the Calibrated Imaging Lab of the Robotics Institute examines how a computer can analyze images using optical models of illumination, reflection, and the imaging process. His research interests also include architectures for mobile robot perception, planning, and control.

Shafer received his PhD in computer science from Carnegie Mellon in 1983.

Readers can contact the authors at the Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213.