IEEE EXPERT
Intelligent Systems and their Applications

Mobile Robot Navigation: The CMU System

Yoshimasa Goto and Anthony Stentz
Carnegie Mellon University

Focusing primarily on system architecture, this article describes the current status of autonomous land vehicle (ALV) research at Carnegie Mellon University's Robotics Institute. We will (1) discuss issues concerning outdoor navigation; (2) describe our system's perception, planning, and control components that address these issues; (3) examine Codger, the software system that integrates these components into a single system, synchronizing the dataflow between them (thereby maximizing parallelism); and (4) present the results of our experiments, problems uncovered in the process, and plans for addressing those problems.

Carnegie Mellon's ALV group has created an autonomous mobile robot system capable of operating in outdoor environments. Using two sensors—a color camera and a laser range finder—our system can drive a robot vehicle continuously on a network of sidewalks, up a bicycle slope, and over a curved road through an area populated with trees. The complexity of real-world domains and requirements for continuous and real-time motion require that such robot systems provide architectural support for multiple sensors and parallel processing—capabilities not found in simpler robot systems. At CMU, we are studying mobile robot system architecture and have developed a navigation system working at two test sites and on two experimental vehicles.[1-6]

t s [illegible]

We use two test sites, the CMU campus and Schenley Park—a city park adjoining the campus. The campus site contains a sidewalk network including intersections, stairs, and bicycle slopes (see Figure 1). Schenley Park has sidewalks curving through a treed area (see Figure 2).

Figure 3 presents our two experimental vehicles—NavLab (used in the Schenley Park test site) and Terregator (used in the CMU campus test site). Each is equipped with a color TV camera, plus a laser range finder made by ERIM. NavLab carries four general-purpose Sun-3 computers on board. Terregator links by radio to Sun-3s in the laboratory. The SUN-3s interconnect with an Ethernet. Our navigation system works on both vehicles in both test sites.

Current system capabilities. Currently, the system

- **Executes** a prespecified user mission over a mapped network of sidewalks, including turns at intersections and driving up the bicycle slope;
- **Recognizes** landmarks, stairs, and intersections;
- **Drives** on unmapped, curved, or ill-defined roads using assumptions about local road linearity;
- **Detects** obstacles and stops until they move away;
- **Avoids** obstacles; and
- **Travels** continuously at 200mm per second.

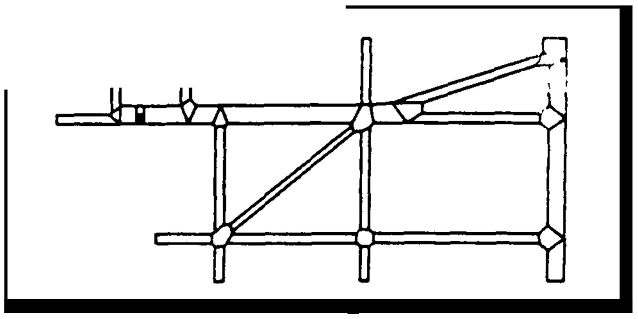


Figure 1. The CMU campus **test** site

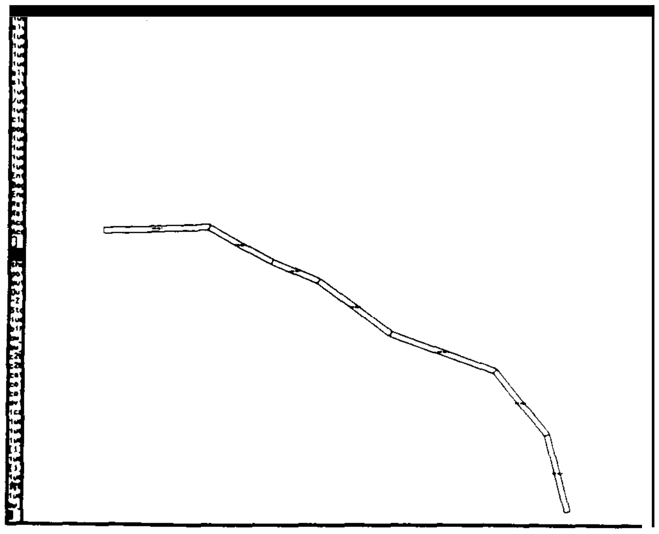


Figure 2. The Schenley Park test site.

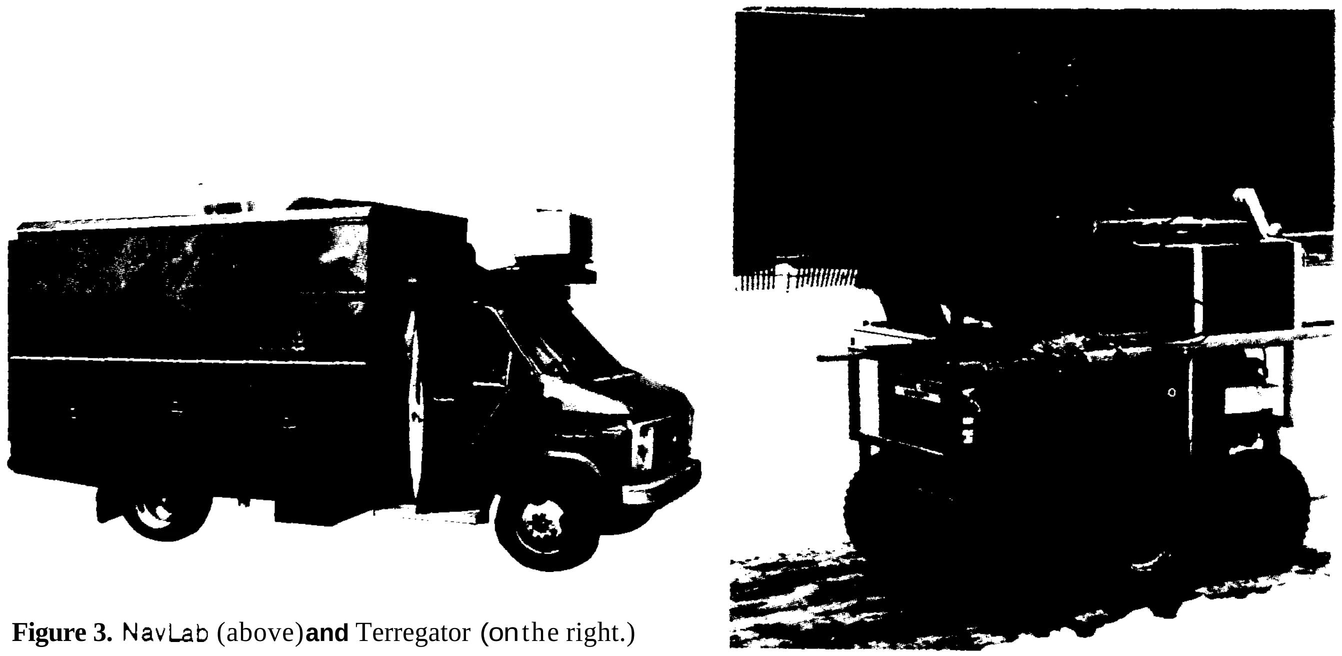


Figure 3. NavLab (above) **and** Terregator (on the right.)

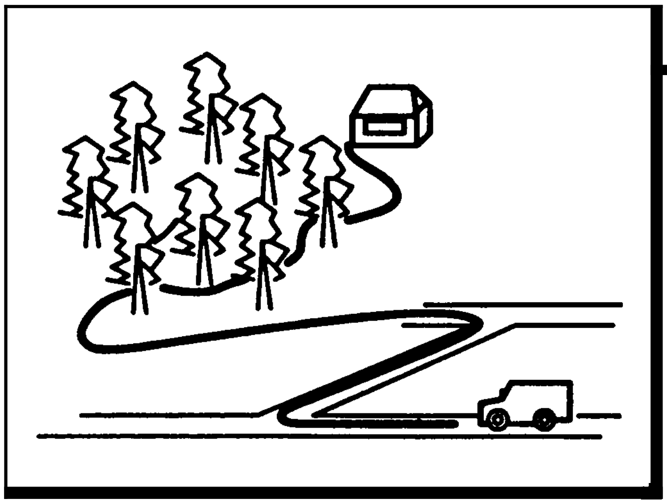


Figure 4. Outdoor navigation.

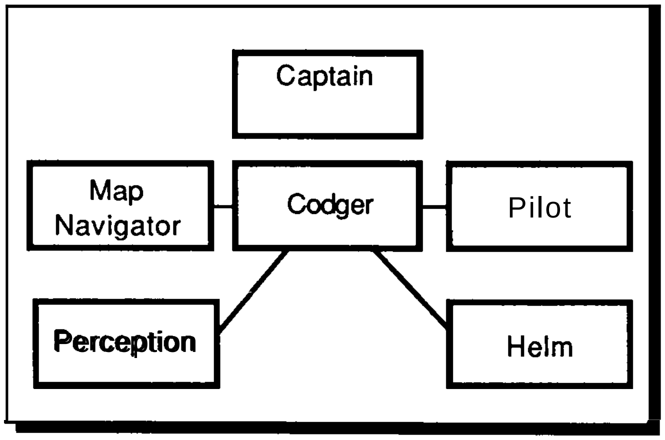
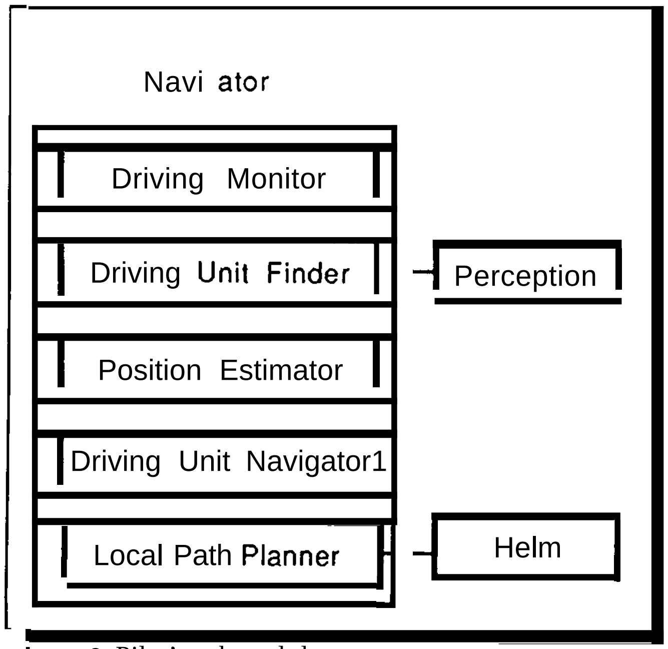


Figure 5. The **CMU** system architecture.



igure 6. Pilot's submodule structure.

System architecture

This section defines goals of our outdoor navigation system and its design principles, and analyzes the outdoor navigation task itself. We will describe our system architecture as it is shaped by these principles and analyses.

Design goals and principles. Our outdoor navigation system seeks the following goals:

- Map-driven mission execution—The system drives the vehicle to a given position (goal);
- On- and off-road navigation—Navigation environments include roads and open terrain;
- Landmark recognition—Landmark sightings are essential when correcting for drift in the vehicle's dead-reckoning system;
- Obstacle avoidance—As wise for robots as for humans; and
- Continuous motion in real time—Stop-and-go motion is inadequate for our purposes. Perception, planning, and control should be carried out while the vehicle is moving at a reasonable speed.

To satisfy these goals, we have adopted the following design principles:

- Sensor fusion—A single sensor is not enough to analyze complex outdoor environments. In addition to a TV camera and range finder, sensors include an inertial navigation sensor and a wheel rotation counter;
- Parallel execution—Parallelism is essential when processing data from many sensors, making global and local plans, and driving the vehicle in real time;and
- Flexibility and extensibility—Also essential since the whole system is quite large, requiring the integration of many modules.

Outdoor navigation tasks. Outdoor navigation includes different navigation modes—Figure **4** illustrates several examples. On-road versus off-road is just one example. Even during on-road navigation, turning at intersections requires more sophisticated driving skill than road following. In road following, assuming that the ground is flat makes perception easier. But driving through the forest does not satisfy this assumption, requiring more complex perception processing.

According to this analysis, we decompose outdoor navigation into two navigation levels—global and local. At the global level, system tasks are (1) to select the best route to reach destinations given by user missions, and (2) to divide the route into segments, each corresponding to a uniform driving mode. The current system supports three navigation modes—following roads, turning at intersections, and driving up slopes.

Local navigation involves driving within a single route segment. The navigation mode is uniform. The system drives the vehicle along the route segment continuously, perceiving objects, planning paths, and controlling the vehicle. These important tasks—perception, planning, and control—form a cycle and can be executed concurrently.

System architecture. Figure 5 presents a block diagram of our system architecture, consisting of several modules and a communications database linking the modules together.

Module structure. To support tasks described in the previous section, we first decomposed the whole system into the following modules:

- **Captain** executes user mission commands and sends each mission's destination and constraints step-by-step to the Map Navigator, then awaits the result of each mission step;
- **Map Navigator** searches the map database, selects the best route, decomposes it into a route segment sequence, generates a route segment description including mapped objects visible from the route segment, and sends all of this to the Pilot;
- **Pilot** coordinates the activities of Perception and Helm, performing local navigation continuously within a single route segment. Pilot is decomposed into several submodules that run concurrently (see Figure 6);
- **Perception** uses sensors to find objects predicted to lie within the vehicle's field of view, and estimates vehicle position when possible;
- **Helm** gets the local-path plan generated by Pilot and drives the vehicle;
- **Driving Monitor** decomposes the route segment into small pieces called driving units. **A** driving unit comprises the basic unit for perception, planning, and control processing at the local navigation level. For example, Perception must be able to process a whole driving unit with a single image. Driving Monitor creates a driving unit description describing objects in the driving unit, and sends that description to the following submodules:
  - **Driving Unit Finder** functions as an interface, sending the driving unit description to—and getting the result from—Perception;
  - **Position Estimator** estimates vehicle position, using the results of Perception and dead reckoning;
  - **Driving Unit Navigator** determines admissible passages through which to drive the vehicle; and
  - Local **Path Planner** generates path plans within the driving unit, avoids obstacles, and keeps the vehicle in its admissible passage. The path plan is sent to Helm.

Codger. The second system architecture design problem is module connection. Based on our design principles, we have created a software system called Codger (communications database with geometric reasoning) that supports parallel asynchronous execution and communication between modules. The next section describes Codger in detail.

Parallelism

We have employed parallelism in our perception, planning, and control subsystems to navigate in real time. Our computing resources consist of several Sun-3 microcomputers, **VAX** minicomputers, and a high-speed parallel processor known as the Warp—all interconnected with an Ethernet. We have designed and implemented the Codger software system to effectively utilize this parallelism.

The Codger system for parallel processing. Codger consists of a central database (local map), a process called LMB (local map builder) that manages this database, and the LMB interface (a function library for accessing data, as shown in Figure 7). The LMB interface compiles the system's perceptual, planning, and control modules; the modules, in turn, invoke functions to store and retrieve data from the central database. We can run Codger on any mix of Sun-3s and VAXs to handle data type conversions automatically, permitting highly modular development that requires recompilation only for modules directly affected by changes.

Data representation. Tokens—lists of attribute-value pairs—represent local map data. We can use

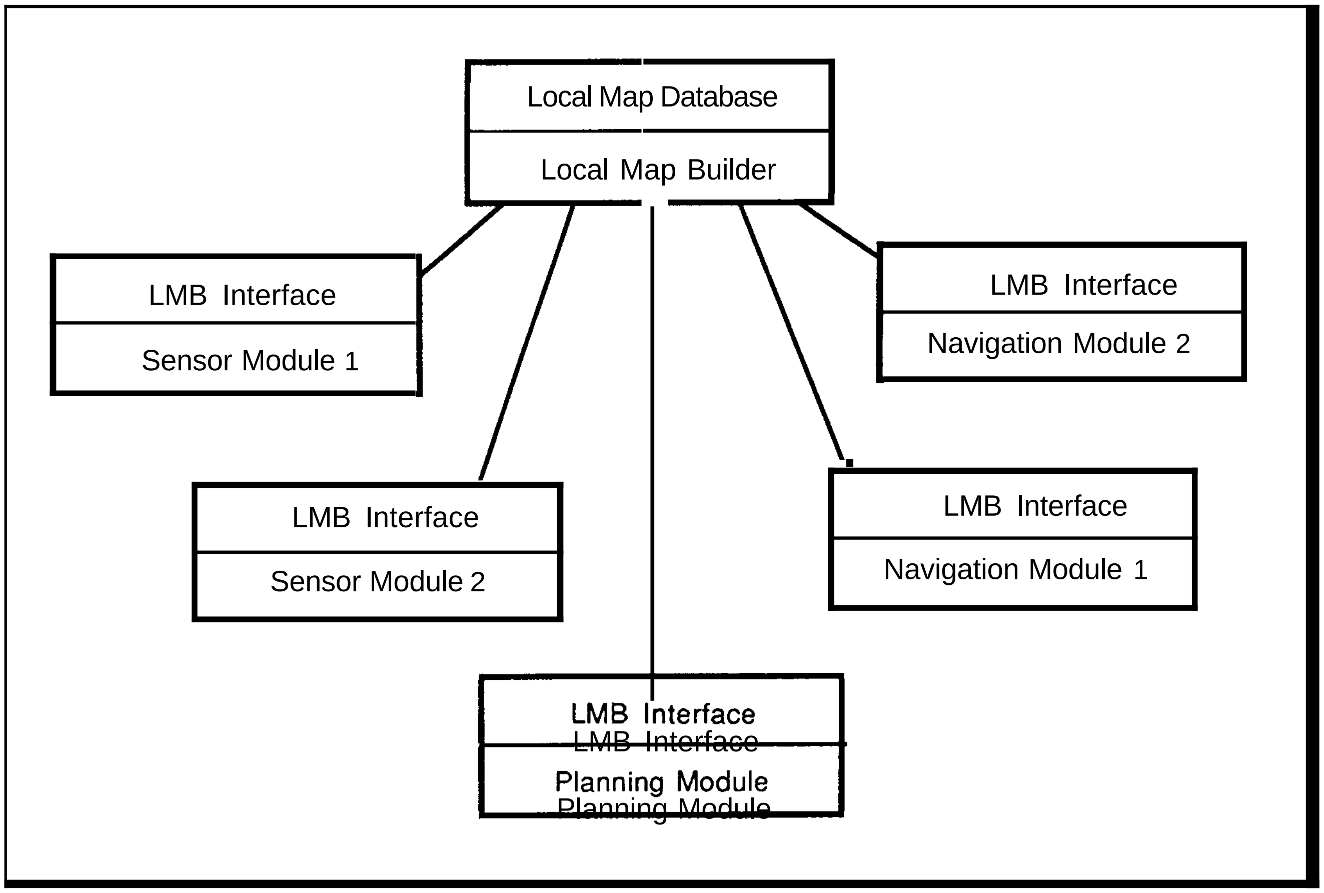


Figure 7. The Codger software system.

tokens to represent physical objects, hypotheses, plans, commands, and reports. A template file read by the LMB at system startup time defines token types. Attribute types can be the usual scalars (for example, floats and integers), sets of scalars, or geometric locations. Geometric locations consist of a two-dimensional polygonal shape and a reference coordinate frame. Codger provides mechanisms for defining coordinate frames and automatically converting geometric data from one frame to another, thereby (1) enabling modules to retrieve data from the database and (2) representing that data in a form meaningful to the modules. Geometric data is the only data that Codger interprets; all other data types are interpreted by the modules using them.

Synchronization. The LMB interface provides functions for storing and retrieving data from the central database. Tokens can be retrieved using specifications (Boolean expressions evaluated across token attribute values). Specifications can include computations such as mathematical expressions, Boolean relations, and comparisons between attribute values. Geometric indexing is particularly important for mobile robot systems. For example, planners must search a map object database to locate suitable landmarks or to find the shortest path to goals. Codger provides many functions, including those for computing distance and intersections of locations—functions that can be embedded in specifications and matched to the database.

Codger embeds a set of primitives synchronizing and smoothing data transfer between system modules. The data retrieval mechanism implements synchronization. Modules send specifications to the LMB as either one-shot or standing requests: The calling module blocks for one-shot specs, while the LMB matches the spec to the tokens and retrieves matching tokens, and the module resumes execution. If no tokens match, the module either stays blocked until a matching token appears in the database—or an error is returned and the module resumes execution—depending on an option specified in the request. For example, before it can plan a path, the path planner may use a one-shot request to find obstacles stored in the database. In contrast, Helm (controlling the vehicle) uses a standing spec to retrieve tokens that supply steering commands whenever those tokens appear.

Parallel asynchronous execution of modules. Thus far, we have run our scenarios with four Sun-3s interconnected through an Ethernet. Captain, Map Navigator, Pilot, and Helm are separate modules in the system; Perception comprises two modules (range and camera image processing). The modules run in parallel, synchronizing themselves through the LMB database.

Global and local navigation. The interaction between Captain, Map Navigator, and Pilot exemplifies Codger's parallelism. Captain and Map Navigator search the map database to plan the vehicle's global path in accordance with mission specifications. Pilot coordinates Perception, Path Planning, and control through Helm to navigate locally. Global and local navigation operations run in parallel. Map Navigator monitors Pilot's progress to ensure that Pilot's transition from one route segment to the next occurs smoothly.

Driving pipeline. Another good example of parallelism occurs within Pilot itself. As described earlier, Pilot monitors local navigation. For each driving unit, Pilot performs operations in the following order: Pilot predicts the driving unit, recognizes it with the camera and scans it for obstacles with the range finder, plans a path through it, oversees the vehicle's execution of it, and establishes driving constraints. These four operations are separate modules in Pilot, linked together in a pipeline (see Figure 8). While in steady state, Pilot (1) predicts a driving unit 12 to 16 meters in front of the vehicle, (2) recognizes a driving unit, (3) scans it for obstacles (in parallel) eight to 12 meters in front, (4)plans a path four to eight meters in front, and (5) drives to a point four meters in front. The stages of the pipeline synchronize themselves through Codger's database.

Processing times vary for each stage as a function of the navigation task. The vision subsystem requires about 10 seconds of real time per image when navigating on uncluttered roads, the range subsystem requires about six seconds, and Local Path Planner requires less than a second. In this case, the pipeline's stage time equals the vision subsystem's—specifically, 10 seconds. In cluttered environments, Local Path Planner may require 10 to 20 seconds or more—thereby becoming a bottleneck. In either case, Helm does not permit the vehicle to drive onto a driving unit until that driving unit has propagated through all stages of the pipeline (that is, until all operations have been performed on it). For example, when driving around the corner of a building, the vision stage must wait until the vehicle reaches the corner to see the next driving unit. And once the vehicle reaches the corner, it must wait for the vision, scanning, and planning stages to process the driving unit before driving again.

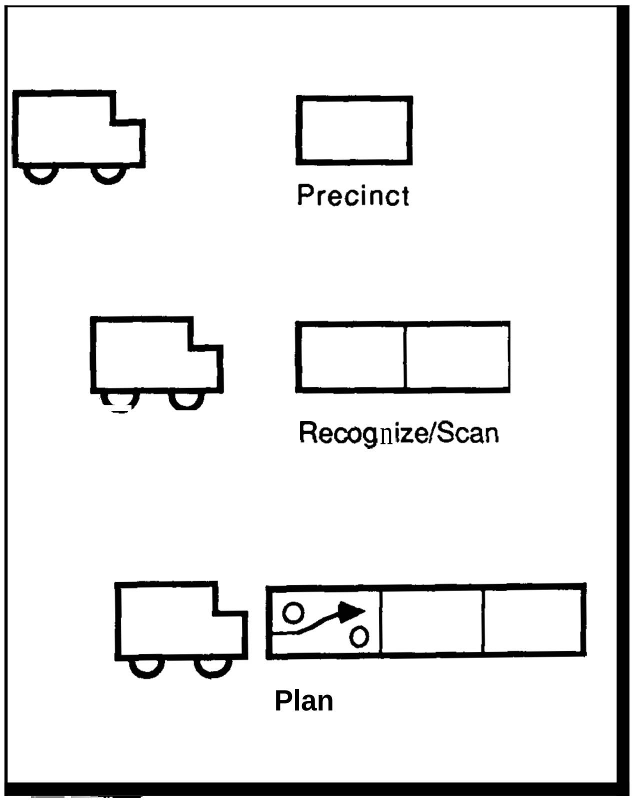


Figure 8. The driving pipeline.

Sensor fusion

NavLab and Terregator are equipped with many sensors including a laser range finder, color cameras, and motion sensors such as a gyro and shaft-encoder counter. To obtain a single, consistent interpretation of the vehicle's environment, sensor results must be fused.

Types of sensor fusion. We have identified three types of sensor fusion:'

- Competitive fusion—Sensors provide data that either agrees or conflicts. Both cases arise when sensors provide data of the same modality. In CMU's systems, determining the vehicle's position best characterizes this type of fusion. Readings from the vehicle's dead-reckoning system and landmark sightings provide vehicle position estimates.
- Complementary fusion—Sensors provide data of different modalities. Recognizing three-dimensional objects illustrates this kind of

fusion. Using a color camera and laser range finder, CMU systems recognize a set of stairs. The color camera provides image information (such as color and texture) while the laser range finder provides three-dimensional information.

- Independent fusion—CMU systems use a single sensor for each task. For example, distant landmark recognition requires a single sensor. In this case, only the camera is used for landmarks beyond the range of the laser range finder.

Examples of sensor fusion tasks. Vehicle position estimation and landmark sighting exemplify sensor fusion tasks.

Vehicleposition estimation. In our road-following scenarios, vehicle position estimation has been the most important sensor fusion task. By vehicle position, we mean the vehicle's position and orientation in the ground plane (three degrees of freedom) relative to the world coordinate frame. The current system has two sources of position information.

First, dead reckoning provides vehicle-based position information. Codger maintains a history of steering commands issued to the vehicle, effectively recording the vehicle's trajectory from its starting point.

Second, landmark sightings directly pinpoint the vehicle's position with respect to the world at a given time. In the campus test site, the system has access to a complete topographical map of sidewalks and intersections on which it drives; it uses a color camera to sight the intersections and sidewalks, and uses these sightings to correct the vehicle's estimated position. Intersections are of rank three, meaning that the vehicle's position and orientation with respect to the intersection can be determined fully (to three degrees of freedom) from the sighting.

Our tests have shown such landmark sightings to be far more accurate—but less reliable—than the current dead-reckoning system; that is, landmark sightings provide more accurate vehicle position estimates but the sightings occasionally fail. If vehicle position estimates from landmark sighting and dead-reckoning disagree drastically, Codger settles the conflict in favor of the dead-reckoning system; otherwise, the landmark sighting is used. In such cases, Codger adjusts the vehicle trajectory record to agree with the most recent landmark sighting and discards all previous sightings.

Codger can handle landmark sightings of less than rank three. The sidewalk on which the vehicle drives is our most common landmark. Since a sidewalk sighting provides only the orientation and perpendicular distance of the vehicle to the sidewalk, the correction is of rank two. Therefore, the vehicle's position is constrained to lie on a straight line. Codger projects the vehicle's position from dead reckoning onto this line, using the projected point as a full (rank three) correction. This approximation works well since most vehicle motion error is lateral drift from the road.

Pilot control. Complementary fusion is grounded in Pilot's control functions. Pilot ensures that the vehicle travels only where it is permitted and where it is able. For example, the color camera segments road from nonroad surfaces. The laser range finder scans the area before the vehicle for obstacles or unnavigable (that is, rough or steep) terrain. The road surface is fused with free space and passed to Local Path Planner. Since the two sensor operations do not necessarily occur simultaneously, the vehicle's dead-reckoning system also comes into play.

Colored range image. Another example of camera and range data complementary fusion is the colored range image, created by "painting" a color image onto a range image depth map. Our systems use the, resultant image to recognize complicated three-dimensional objects (such as a set of stairs). To avoid relatively large error in the vehicle's dead-reckoning system, the vehicle remains motionless while digitizing a corresponding pair of camera and range images.'

Problems and future work. We plan to improve our sensor fusion mechanisms. Currently, Codger handles competing sensor data by retaining the most recent measurement and discarding all others. This is undesirable for the following reasons: First, a single bad measurement (for example, a landmark sighting) can easily throw the vehicle off track. Second, measurements can reinforce each other. By discarding old measurements, Codger loses useful information. The system needs a weighting scheme to combine competing sensor data. In many cases, it's useful to model error in sensor data as Gaussian noise. For example, dead-reckoning error can arise from random error in wheel velocities. Likewise, quantization error in range and camera images can be modeled as Gaussian noise. Various schemes exist for fusing such data, ranging from simple Kalman filtering techniques to full-blown Bayesian observation networks."'

Local control

Management of driving units and sensor view frames is essential in local control. This section discusses control problems in local navigation.

Adaptive driving units and sensor view frames. For each driving unit (each minimum control unit), the CMU system perceives objects, generates a path plan, and drives the vehicle. The Perception module digitizes an image in each driving unit, and the vehicle's position is estimated and its trajectory is planned once in each driving unit. Therefore, stable control requires an appropriate driving unit size. For example, the sensor view frame cannot cover avery large driving unit. Conversely, small driving units place rigid constraints on Local Path Planner because of the short distance between starting point and goal point. Aiming the sensor view frame determines the point at which to digitize an image and to update vehicle position and path plan.

Our current system's sensor view frame is always fixed with respect to the vehicle. Driving unit size is fixed for driving on roads (four to six meters in length) and is changed for turning at intersections so that the entire intersection appears in a single image (for easy recognition) and to increase driving stability (see Figure 9). In current test sites, this method almost always works well.

For intersections requiring sharp turns (about 135 degrees), the current method does not suffice. Because there is only one driving unit at intersections, the system digitizes an image, estimates vehicle position, and generates a path plan only once for a large turn. Furthermore, since the camera's field of view is fixed straight ahead, the system cannot see the driving unit after an intersection until the vehicle has turned through the intersection. Though actual paths generated are not so bad, they are potentially unstable.

This experimental result indicates that the system should scan for an admissible passage, and update vehicle position estimation and local path plan more frequently when the vehicle changes its course faster. We have the following plan to improve our method for managing driving units:

- Driving unit length—The length of the driving unit is bounded at the low end by Local Path Planner's requirements for generating reasonable path plans, and at the high end by the view frame that Perception requires for recognizing given objects.

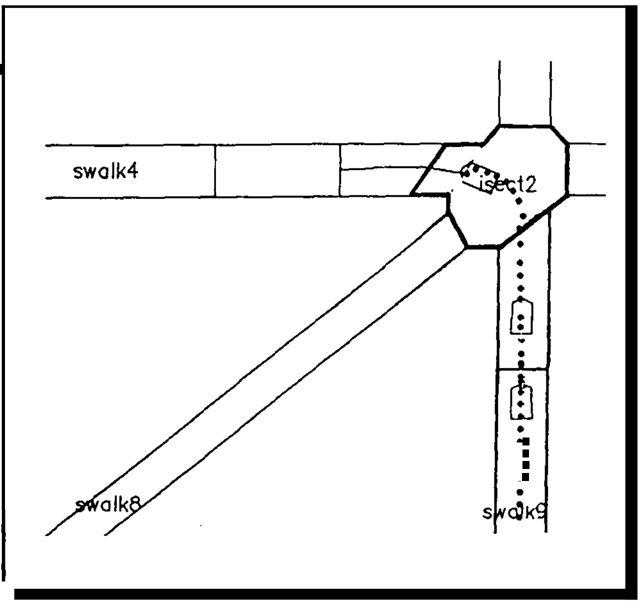


Figure **9. An intersection driving unit.**

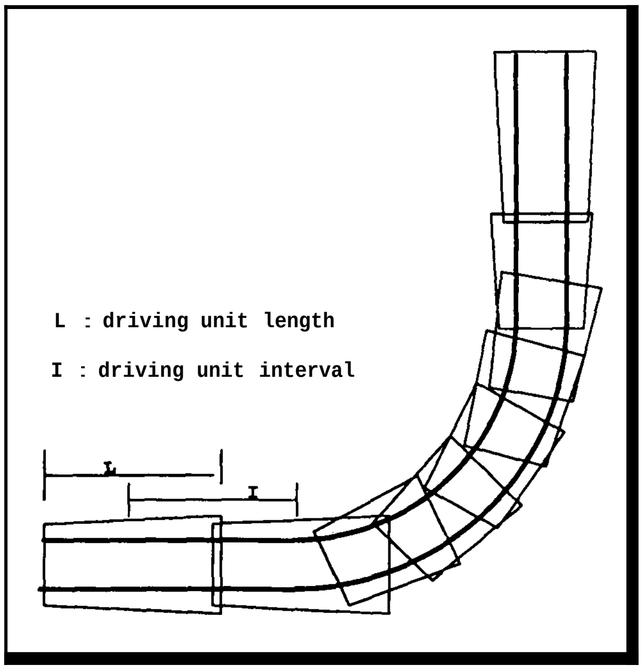


Figure **10. Adaptive driving units.**

- Driving unit interval—The distance between centers of adjacent driving units is the driving unit interval. Adjacent driving units can be overlapped: that is, they can be placed such that their interval is shorter than their length (see Figure 10).
- Adjusting driving unit length and interval—In simple passages, the lengths and intervals of driving units are long. If the passage is complex (for example, on highly curved roads and intersec-

tions or in the presence of obstacles) the lengths and intervals are shorter. And if the required driving unit interval must be shorter than the driving unit length, driving units are overlapped. Therefore, the vehicle's position is estimated and a local path is planned more frequently so that the vehicle drives stably, as Figure 10 illustrates.

- Adjusting sensor view frame—The sensor view frame with respect to the vehicle (that is, the distance and direction from the vehicle to the driving unit) is adjusted using the pan-and-tilt mechanism of the sensor. In most cases, a longer distance to the next driving unit allows a higher vehicle speed. If Perception and Pilot processing times are constant, the longer distance means a higher vehicle speed. But the longer distance produces less accuracy in perception and vehicle position estimation. Therefore, distance is determined for required accuracy, depending on the passage's complexity. Using the pan-and-tilt mechanism, Perception digitizes an image at the best distance from the driving unit, since the sensor's view frame is less rigidly tied to the vehicle's orientation and position.

Vehicle speed. Autonomous mobile robots must be able to adjust vehicle speed automatically so that vehicles drive safely at the highest possible speed. The current system slows the vehicle in turning to reduce driving error.

Delays in processing in Local Path Planner and communication between Helm and the actual vehicle mechanism cause error in vehicle position estimation. For example, because of continuous motion and nonzero processing time, vehicle position used as a starting point by Local Path Planner differs slightly from vehicle position when the vehicle starts executing the plan. Because smaller turning radii give rise to larger errors in vehicle heading, which are more serious than displacement errors, Helm slows the vehicle for smaller turning radii—a useful method for stabilizing vehicle motion.

We have the following method for enabling our system to adjust vehicle speed to the highest possible value automatically:

- Schedule token—In each cycle, modules and submodules working at the local navigation level store their predicted processing times in a schedule token. Perception is the most time-consuming module, and its processing time varies drastically from task to task.
- Adjusting vehicle speed—Using the path plan and predicted processing time stored in the schedule token, Helm calculates and adjusts vehicle speed to maximum acceleration and the modules can finish processing the driving unit before the vehicle reaches the end of its planned trajectory.

Local path planning and obstacle avoidance. Local path planning finds a trajectory for the vehicle through admissible space to a goal point. Our vehicles are constrained to move in ground planes around obstacles (represented by polygons) while remaining within a driving unit (also a polygon). We have employed a configuration space approach.[9,10] However, this algorithm assumes the vehicle is omnidirectional. Since our vehicles are not, we smooth the resultant path to ensure that the vehicle can execute it. The smoothed path is not guaranteed to miss obstacles. We plan to overcome this problem by developing a path planner that reasons about constraints on the vehicle's motion.

The navigation map

Some a priori information about the vehicle's environment must be supplied to the system—even if that information is incomplete, and even if it is nothing more than a data format for storing explored terrain. For example, the user mission "turn at the second cross intersection and stop in front of the three oak trees" does not make sense to the system without environmental description. The navigation map is a database storing the environment description needed for navigation.

Map Structure. The navigation map—a description of physical objects in the navigation world—is composed of two parts: the geographical map, and the object database. The geographical map stores object locations with their contour polylines. The object database stores object geometrical shapes and other attributes; for example, the navigational cost of objects. Our current system describes all objects with both the geographical map and the object database; in general, however, either of them can be unused. For example, the location of a particular flight of stairs is known but its shape is unknown.

Shape descriptions are composed of two layers. The first layer stores shape attributes (such as road width, road length, stair height, or number of steps). The

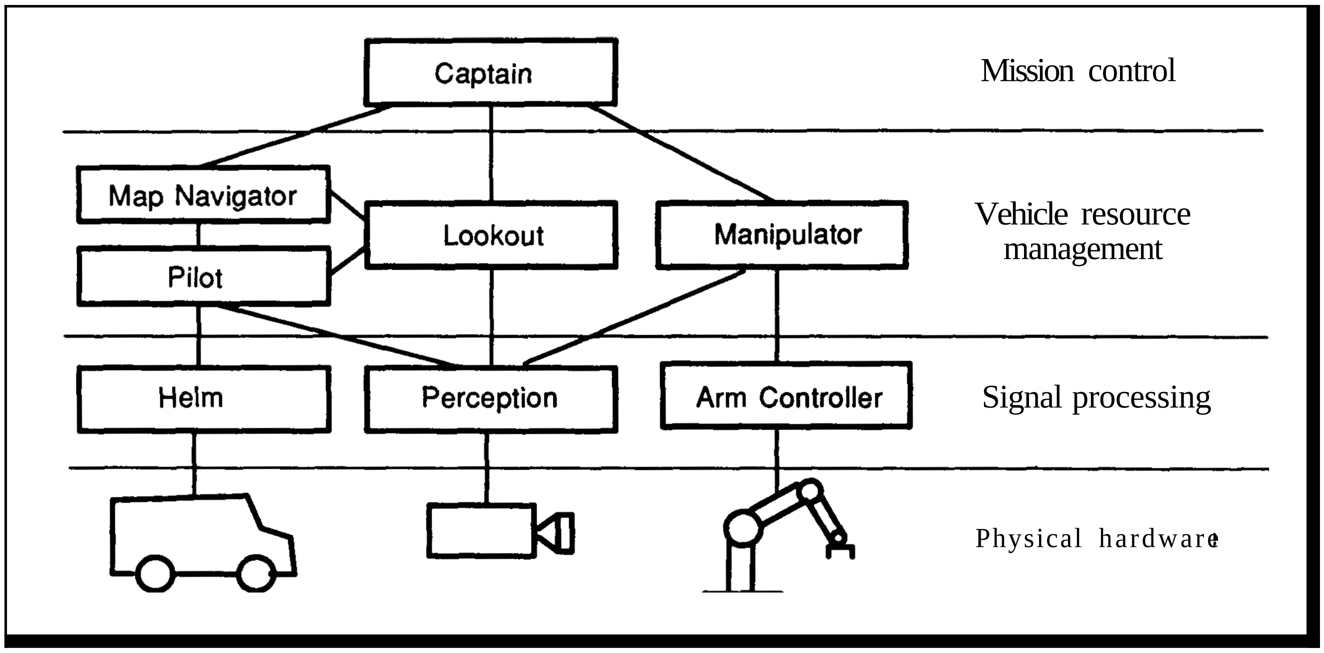


Figure 11. Extended system architecture.

second layer stores actual geometrical shapes represented by the surface description. It is easy to describe incomplete shape information with only the first layer.

Data retrieval. Map data is stored in the Codger database as a set of tokens forming tree structure. To retrieve map data, parent tokens contain indexes to children tokens. Because Codger currently provides modules with a token retrieval mechanism that picks up only one token at a time, retrieving large portions of the map is cumbersome. We plan to extend Codger so that it can match and retrieve larger structures, possibly combining that with an inheritance mechanism.

While navigation is one goal of a mobile robot system, navigation itself is not an end; instead, it is a means for achieving the final goals of the autonomous mobile robot system—goals such as exploration, refueling, and carrying baggage. Therefore, system architecture must accommodate tasks other than navigation.

Figure 11 illustrates one example of an extended system architecture that loads, delivers, and unloads baggage. Four layers comprise the whole system—mission control, vehicle resource management, signal processing, and physical hardware. Captain (only one module in the mission control layer) stores the user mission steps, sends them to the vehicle resource management layer one by one, and oversees their execution.

In the vehicle resource management layer, different modules work for different tasks. Although their tasks are different, modules work in a symbolic domain and do not handle the physical world directly. They oversee mission execution, generate plans, and pass information to modules in the signal processing layer. Through Codger, they communicate with each other if necessary. Included in the vehicle resource management layer are Map Navigator and Pilot, parts of the navigation system. Manipulator makes a plan (for example, how to load and unload baggage with the arm) and sends it to Arm Controller.

Using sensors and actuators, modules in the signal processing layer interact with the physical world; for example, Perception processes sensor signals, Helm drives the physical vehicle, and Arm Controller operates the robot arm. The bottom level contains the real hardware, even if it includes some primitive controller. This layer includes the sensors, the physical vehicle, and the robot arm.

Since we built our current system architecture on the Codger system, we can easily expand it to include these additional capabilities.

We have described the CMU architecture for autonomous outdoor navigation—a highly modular architecture including components for both global and local navigation. A route planner carries out global navigation, searching a map database to find the path best satisfying a mission, and overseeing its execution. Modules carry out local navigation, using a color camera and a laser range finder to recognize roads and landmarks, scanning for obstacles, reasoning about geometry to plan paths, and overseeing the vehicle's execution of a planned trajectory.

A single system integrates perception, planning, and control components through the Codger software system. Codger provides a common data representation scheme for all modules in the system, paying special attention to geometry. Codger also provides primitives for synchronizing modules to maximize parallelism at both local and global levels.

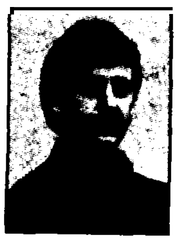
We have demonstrated our system's ability to drive around a network of sidewalks and along a curved road, to recognize complicated landmarks, and to avoid obstacles. Future work will focus on improving Codger to handle more difficult sensor fusion problems. This work will seek better schemes for local navigation and will strive to reduce our dependence on map data.

Acknowledgments

CMU's entire **ALV** group helped shape our architecture's design. We extend special thanks to Steve Shafer, Chuck Thorpe, and Takeo Kanade.

This research was supported by the Strategic Computing Initiative of the Defense Advanced Research Project Agency (DoD) through ARPA Order 5351, and monitored by the US Army Engineer Topographic Laboratories under contract DACA76-85-C-0003. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of DARPA or the United States government.

Yoshimasa ***Goto,*** a rerearcher for the corporate engineering division of Matsushita Electric Industrial Company, Ltd., is a visiting scientist at Carnegie Mellon University's Robotics Institute. For the last two years, he has worked on CMU's autonomous land vehicle project. His interests include map-driven navigation and system architecture for autonomous robots. He received his BA and MS in mechanical engineering from Nagoya University in Japan.



Anthony Stentz is a doctoral student in computer science at Carnegie Mellon University. **His** interests include mobile robots, path planning, computer vision, system architecture, and AI. He received his BS in physics from Xavier University, and his **MS** in Computer Science from Carnegie Mellon University.

The authors can be reached at the Computer Science Dept., Carnegie Mellon Univ., 5000 Forbes Ave.. Pittsburgh, PA 15312-3890.

References

1. Y. Goto et al., "CMU Sidewalk Navigation System," ***Proc. FJCC-86,*** Dallas, Tex.. Nov. 1986.
2. M. Hebert and T. Kanade, "Outdoor Scene Analysis Using Range Data," ***Proc. 1986 IEEE Int'l Conf. Robotics and Automation,*** San Francisco, Calif., Apr. 1986.
3. T. Kanade, C. Thorpe, and W. Whittaker, "Autonomous Land Vehicle Project at CMU," ***froc. 1986 ACM Computer Con!.,*** Cincinnati, Ohio, Feb. 1986.
4. S. Shafer, A. Stentz, and C. Thorpe, "An Architecture for Sensor Fusion in a Mobile Robot," ***Proc. IEEE Int'l Conf. Robotics and Automation,*** San Francisco, Calif., Apr. 1986.
5. R. Wallace et al., "First Results in Robot Road-Following," ***Proc. IJCAI-85,*** Los Angeles, Calif., Aug. 1985.
6. R. Wallace et al., "Progress in Robot Road Following," ***Proc. IEEE Int'l Conf. Robotics and Automation,*** San Francisco, Calif., Apr. 1986.
7. H. Durrant-Whyte, ***Integration. Coordination, and Control of Multi-Sensor Robot Systems,*** PhD dissertation, University of Pennsylvania, Philadelphia, Pa. 19104, 1986.
8. E.M. Mikhail and F. Ackerman, ***Observations and Leas/ Squares,*** University Press of America, Lanham, Md., 1976.
9. T. Lozano-Perez and M.A. Wesley, "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," ***Comm. ACM,*** **Vol.** 22, No.10, Oct. 1979.
10. T. Lozano-Perez, "Spatial Planning: A Configuration Space Approach," ***IEEE Trans. Computers,*** Feb. 1983.