The CMU System for Mobile Robot Navigation

Yoshimasa Goto

Anthony Stentz

The Robotics Institute

Carnegie-Mellon University

Pittsburgh, PA 15213

Abstract

This paper describes the current status of the Autonomous Land Vehicle research at Carnegie-Mellon University's Robotics Institute, focusing primarily on the system architecture. We begin with a discussion of the issues concerning outdoor navigation, then describe the various perception, planning, and control components of our system that address these issues. We describe the CODGER software system for integrating these components into a single system, synchronizing the data flow between them in order to maximize parallelism. Our system is able to drive a robot vehicle continuously with two sensors, a color camera and a laser rangefinder, on a network of sidewalks, up a bicycle slope, and through a curved road through an area populated with trees. Finally, we discuss the results of our experiments, as well as problems uncovered in the process and our plans for addressing them.

1. Introduction

The goal of the Autonomous Land Vehicle group at Carnegie-Mellon University is to create an autonomous mobile robot system capable of operating in outdoor environments. Because of the complexity of real-world domains and the requirement for continuous and real-time motion, such a robot system needs system architectural support for multiple sensors and parallel processing. These capabilities are not found in simpler robot systems. At CMU, we are studying mobile robot system architecture and have developed the navigation system working at two test sites and on two experimental vehicles [2] [3] [4] [8] [10] [11]. This paper describes current status of our system and some problems uncovered through real experiments.

1.1. The Test Sites and Vehicles

We have two test sites, the Carnegie-Mellon University campus and an adjoining park, Schenley Park. The CMU campus test site has a sidewalk network including intersections, stairs and bicycle slopes (see Figure 1). The Schenley Park test site has curved sidewalks in an area well populated with trees (see Figure 2).

Figure 3 shows our two experimental vehicles, the NAVLAB used in the Schenley Park test site, and the Terregator used in the CMU campus test site. Both of them are equipped with a color TV camera and a laser rangefinder made by ERIM. The NAVLAB carries four general purpose computers (SUN-3s) on board. The Terregator is linked to SUN-3s in the laboratory with radio communication. All of the SUN-3s are interconnected with a EtherNet. Our navigation system works on both vehicles in each test site.

[1]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 U.S. 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 the Defense Advanced Research Projects Agency or the United State Government.

1.2. Current System Capabilities

Currently, the system has the following capabilities.

- Able to execute a prespecified user mission over a mapped network of sidewalks, including turning at the intersections and driving up the bicycle slope.
- Able to recognize landmarks, stairs and intersections.
- Able to drive on unmapped, curved, ill-defined roads using assumptions about local road linearity.
- Able to detect obstacles and stop until they move away.
- Able to avoid obstacles.
- Able to drive continuously at 200mm/sec.

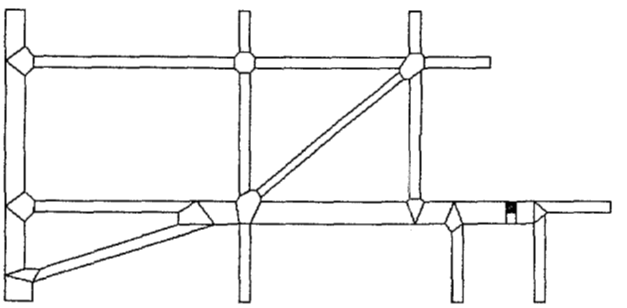


Figure 1: Map of the CMU Campus Test Site

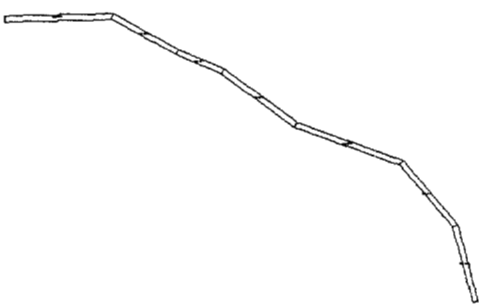
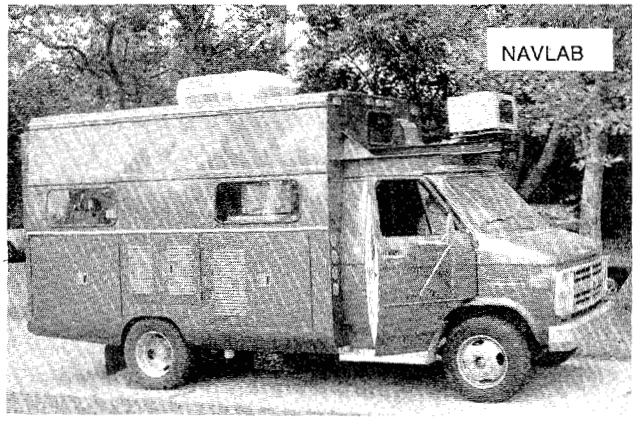


Figure 2: Map of the Schenley Park Test Site

2. Design of the System Architecture

In this section we describe the goals of our outdoor navigation system and the design principles, followed by an analysis of the outdoor navigation task itself. We describe our system architecture as it is shaped by these principles and analysis.



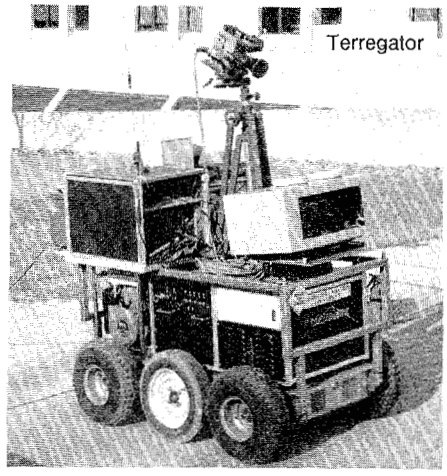


Figure 3: The NAVLAB and Terregator

2.1. Design Goals and Principles

The goals of our outdoor navigation system are:

- **map-driven mission execution:** The system drives the vehicle to reach a given goal position.
- **on- and off-road navigation:** Navigation environments include not only roads but also open terrain.
- **landmark recognition:** Landmark sightings are essential in order to correct for drift in the vehicle's dead-reckoning system.
- **obstacle avoidance**
- **continuous motion in real time:** Stop and go motion is unacceptable for our purposes. Perception, planning, and control should be carried out while the vehicle is moving at a reasonable speed.

In order to satisfy these goals, we have adopted the following design principles.

- **sensor fusion:** A single sensor is not enough to analyze complex outdoor environments. Sensors include not only a TV camera and a range sensor but also an inertial navigation sensor, a wheel rotation counter, etc.
- **parallel execution:** In order to process data from a number of sensors, make global and local plans , and drive the vehicle in real-time, parallelism is essential.
- **flexibility and extensibility:** This principle is essential because the whole system is quite large, requiring the integration of a wide range of modules.

2.2. Outdoor Navigation Tasks

Outdoor navigation includes several different navigation modes. Figure 4 illustrates several examples. On-road vs. off-road is just one example. Even in on-road navigation, *turning at the intersection* requires more sophisticated driving skill than *following the road*. In road following, the assumption that the ground is flat makes perception easier, but *driving through the forest* does not satisfy this assumption and requires more complex perception processing.

According to this analysis we decompose outdoor navigation into two navigation levels: *global* and *local*. At the global level, the system tasks are to select the best navigation route to reach the destination given by a user mission, and to divide whole route into a sequence of *route segments*, each corresponding to a uniform driving mode. The current system supports the following navigation modes: *following the road, turning at the intersection, driving up the slope.*

Local navigation involves driving within a single route segment.

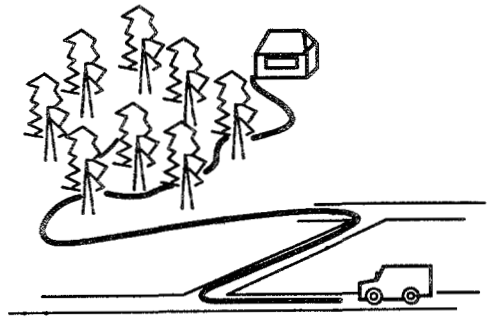


Figure 4: Outdoor Navigation

The navigation mode is uniform and the system drives the vehicle along the route segment continuously, perceiving objects, planning path plans, and controlling the vehicle. The important thing is that these tasks, perception, planning, and control, form a cycle and can be executed concurrently.

2.3. System Architecture

Figure 5 is a block diagram of our system architecture. The architecture consists of several modules and a communications database which links the modules together.

2.3.1. Module Structure

In order to support the tasks described in the previous section, we first decomposed the whole system into the following modules:

- **CAPTAIN** executes user mission commands and sends the destination and the constraints of each mission step to the MAP NAVIGATOR one step at a time, and gets the result of mission step.
- **MAP NAVIGATOR** selects the best route by searching the Map Database, decomposes it into a sequence of route segments, generates a route segment description which includes objects from the Map visible from the route segment, and sends it to the PILOT.
- **PILOT** coordinates the activities of PERCEPTION and the the HELM to perform local navigation continuously within a single route segment.
- **PERCEPTION** uses sensors to find objects predicted to lie within the vehicle's field of view. It estimates the vehicle's position if possible.
- **HELM** gets the local path plan generated by the PILOT and drives the vehicle.

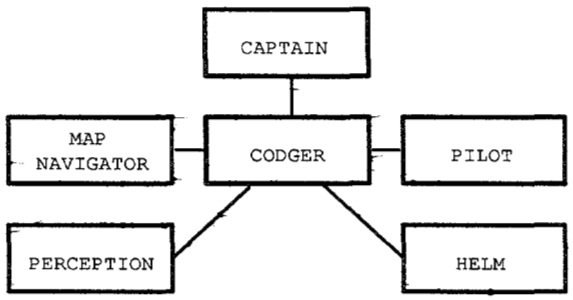


Figure 5: System Architecture

The PILOT is decomposed into several submodules which run concurrently (see Figure 6).

- **DRIVING MONITOR** decomposes the route segment into small pieces called *driving units*. A driving unit is 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. The DRIVING MONITOR creates a *driving unit description* , which describes objects in the driving unit, and sends it to the following submodules.
- **DRIVING UNIT FINDER** functions as an interface to PERCEPTION, sending the driving unit description to it and getting the result from it.
- **POSITION ESTIMATOR** estimates the vehicle position using both of the result of PERCEPTION and dead-reckoning.
- **DRIVING UNIT NAVIGATOR** determines the admissible passage in which to drive the vehicle.
- **LOCAL PATH PLANNER** generates the path plan within the driving unit, avoids obstacles and keeps the vehicle in the admissible passage. The path plan is sent to the HELM.

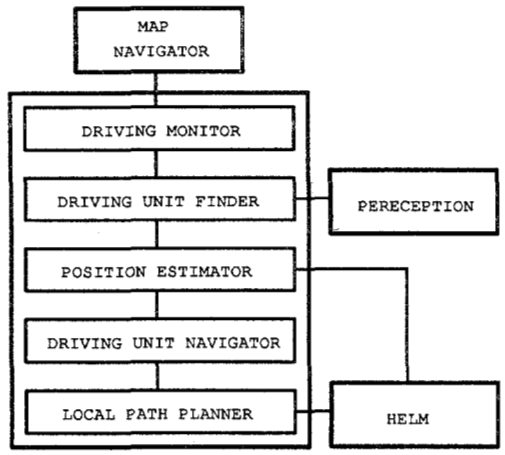


Figure 6: Submodule Structure of the PILOT

2.3.2. CODGER

The second problem in the system architecture design is connecting the modules. Based on our design principles, we have created a software system called *CODGER* (COmmunications Database with GEometric Reasoning) which supports parallel asynchronous execution and communication between the modules. We describe CODGER in detail in the next section.

3. Parallelism

3.1. The CODGER System for Parallel Processing

In order to navigate in real-time, we have employed parallelism in our perception, planning, and control subsystems. Our computing resources consist of several SUN-3 microcomputers, VAX minicomputers, and a high-speed, parallel processor known as the WARP interconnected with an EtherNet. We have designed and implemented a software system called CODGER (COmmunications Database with GEometric Reasoning) [9] to effectively utilize this parallelism.

The CODGER system consists of a central database (*Local Map*), a process that manages this database (*Local Map Builder or LMB*), and a library of functions for accessing the data (*LMB interface*) (see Figure 7). The various perceptual, planning, and control modules in the system are compiled with the LMB interface and invoke functions to store and retrieve data from the central database. The CODGER system can be run on any mix of SUN-3s and VAXes and handles data type conversions automatically. This system permits highly modular development requiring recompilation only for modules directly affected by a change.

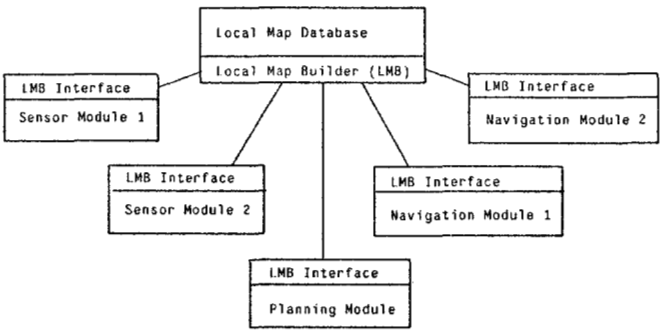


Figure 7: The CODGER Software System

3.1.1. Data Representation

Data in the Local Map is represented in *tokens* consisting of lists of *attribute-value* pairs. Tokens can be used to represent any information including physical objects, hypotheses, plans, commands, and reports. The token types are defined in a *template file* which is read by the LMB at system startup time. Attribute types may be the usual scalars (e.g., floats, integers), sets of scalars, or geometric locations. Geometric locations consist of a two- dimensional, polygonal *shape* and a reference coordinate *frame*. The CODGER system provides mechanisms for defining coordinate frames and for automatically converting geometric data from one frame to another, thereby allowing modules to retrieve data from the database and representing it in a form meaningful to them. Geometric data is the only data interpreted by the CODGER system; the interpretation of all other data types is delegated to the modules that use them.

3.1.2. Synchronization

The LMB interface provides functions for storing and retrieving data from the central database. Tokens can be retrieved using *specifications*. Specifications are simply boolean expressions evaluated across token attribute values. A specification may include computations such as mathematical expressions, boolean relations, and comparisons between attribute values. Geometric indexing is of particular importance for a mobile robot system. For example, the planner needs to search a database of map objects to locate suitable landmarks or to find the shortest path to the goal. The CODGER system provides a host of functions including those for computing the distance and intersection of locations. These functions can be embedded in specifications and matched to the database.

The CODGER system has a set of primitives to ensure that data transfer between system modules is synchronized and runs smoothly. The synchronization is implemented in the data retrieval mechanism. Specifications are sent to the LMB as either one-shot or standing requests. For one-shot specs, the calling module blocks while the

LMB matches the spec to the tokens. Tokens that match are retrieved and the module resumes execution. If no tokens match, either the module 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, the PATH PLANNER may use a one-shot to find obstacles stored in the database *before* it can plan a path. In contrast, the HELM, which controls the vehicle, uses a standing spec to retrieve tokens supplying steering commands *whenever* they appear.

3.2. Parallel Asynchronous Execution of Modules

Thus far we have run our scenarios with four SUN-3s interconnected with an EtherNet. The CAPTAIN, MAP NAVIGATOR, PILOT, and HELM are separate modules in the system, and PERCEPTION is two modules (range and camera image processing). All of the modules run in parallel; they synchronize themselves through the LMB database.

3.2.1. Global and Local Navigation

A good example of parallelism in the system is the interaction between the CAPTAIN, MAP NAVIGATOR, and PILOT. The CAPTAIN and MAP NAVIGATOR search the map database to plan a global path for the vehicle in accordance with the mission specification. The PILOT coordinates PERCEPTION, PATH PLANNING, and control through the HELM to navigate locally. The global and local navigation operations run in parallel. The MAP NAVIGATOR monitors the progress of the PILOT to ensure that the PILOT's transition from one route segment to the next occurs smoothly.

3.2.2. Driving Pipeline

Another good example of parallelism is within the PILOT itself. As described earlier, the PILOT monitors local navigation. For each driving unit, the PILOT performs four operations in the following order: predict it, recognize with the camera and scan it for obstacles with the rangefinder, establish driving constraints and plan a path through it, and oversee the vehicle's execution of it. In the PILOT, these four operations are separate modules linked together in a pipeline (see Figure 8). While in steady state, the PILOT is predicting a driving unit 12 to 16 meters in front of the vehicle, recognizing a driving unit and scanning it for obstacles (in parallel) 8 to 12 meters in front, planning a path 4 to 8 meters in front, and driving to a point 4 meters in front. The stages of the pipeline synchronize themselves through the CODGER database.

The processing times for each stage vary as a function of the navigation task. in navigation on uncluttered roads, the vision subsystem requires about 10 seconds of real-time per image, the range subsystem requires about 6 seconds, and the local path planner requires less than a second. In this case, the stage time of the pipeline is that of the vision subsystem: 10 seconds. In cluttered environments, the local path planner may require 10 to 20 seconds or more, thereby becoming the bottleneck. In either case, the vehicle is not permitted to drive on to a driving unit unit it has propagated through all stages of the pipeline (i.e., 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 in order to see the next driving unit. Once the vehicle reaches the corner, it must stop while waiting for the vision, scanning, and planning stages to process the driving unit before driving again.

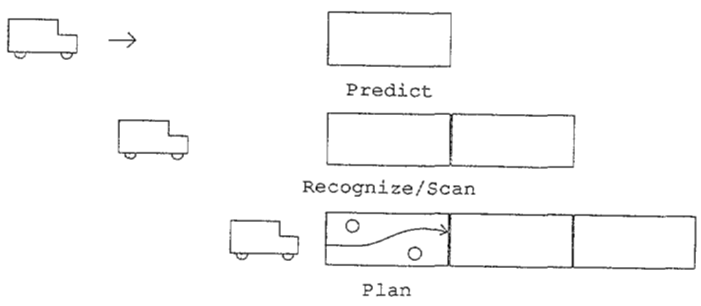


Figure 8: Driving Pipeline

4. Sensor Fusion

4.1. Types of Sensor Fusion

The NAVLAB and Terregator vehicles are equipped with a host of sensors including color cameras, a laser rangefinder, and motion sensors such as a gyro and shaft-encoder counter. In order to obtain a single, consistent interpretation of the vehicle's environment, the results of these sensors must be fused. We have identified three types of sensor fusion [8]:

- **Competitive:** Sensors provide data that either agrees or conflicts. This case arises when sensors provide data of the same modality. In the CMU systems, the task of determining the vehicle's position best characterizes this type of fusion. Readings from the vehicle's dead-reckoning system as well as landmark sightings provide estimates of the vehicle's position.
- **Complementary:** Sensors provide data of different modalities. The task of recognizing three-dimensional objects illustrates this kind of fusion. In the CMU systems, a set of stairs is recognized using a color camera and laser rangefinder. The color camera provides *image* information (e.g., color and texture) while the laser rangefinder provides *three-dimensional* information.
- **Independent:** A single sensor is used for each task. An example of a task requiring a single sensor is distant landmark recognition. In this case, only the camera is used for landmarks beyond the range of the laser rangefinder.

4.2. Examples of Sensor Fusion Tasks

4.2.1. Vehicle Position Estimation

In our road following scenarios, vehicle position estimation has been the most important sensor fusion task. By vehicle position, we mean the position and orientation of the vehicle in the ground plane (3 degrees of freedom) relative to the world coordinate frame. In the current system, there are two sources of position information. First, dead-reckoning provides vehicle-based position information. The CODGER system maintains a history of the steering commands issued to the vehicle, effectively recording the trajectory of the vehicle from its starting point.

Second, landmark sightings directly pinpoint the position of the vehicle with respect to the world at a point in time. In the campus test site, the system has access to a complete topographical map of the sidewalks and intersections on which it drives. The system uses a color camera to sight the intersections and sidewalks and uses these sightings to correct the estimate of the vehicle's position. The intersections are of rank three, meaning that the position and orientation of the vehicle with respect to the intersection can be determined fully (to three degrees of freedom) from the sighting. Our tests have shown that such landmark sightings are far more accurate but less reliable than the current dead-reckoning system, that is, landmark sightings provide more accurate vehicle position estimates; however, the sightings occasionally fail. If the vehicle position estimates from the sighting and dead-reckoning disagree drastically, the *conflict* is settled in favor of the dead-reckoning system; otherwise, the result from the landmark sighting is used. In this case, the CODGER system adjusts its record of the vehicle's trajectory so that it agrees with the most recent landmark sighting, and discards all previous sightings.

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

4.2.2. Pilot Control

Complementary fusion is grounded in the Pilot's control functions. The Pilot ensures that the vehicle travels only where it is permitted and where it can. For example, the color camera is used to segment road from nonroad surfaces. The laser rangefinder scans the area in front of the vehicle for obstacles or unnavigable (i.e., rough or steep) terrain. The road surface is fused with the free space and is passed to the local path planner. Since the two sensor operations do not necessarily occur at the same time, the vehicle's dead-reckoning system also comes into play.

4.2.3. Colored Range Image

Another example of complementary fusion of camera and range data is the colored range image. A colored range image is created by "painting" a color image onto the depth map of a range image. The resultant image is used in our systems to recognize complicated three dimensional objects such as a set of stairs. In order to avoid the relatively large error in the vehicle's dead-reckoning system, the vehicle remains motionless while digitizing a corresponding pair of camera and range images [2].

4.3. Problems and Future Work

We have plans for improving our sensor fusion mechanisms. Currently, the CODGER system 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 (e.g., landmark sighting) can easily throw the vehicle off track. Second, measurements can reinforce each other. By discarding old measurements, useful information is lost. A weighting scheme is needed for combining competing sensor data. In many cases, it is useful to model error in sensor data as gaussian noise. For example, error in dead-reckoning may arise from random error in the wheel velocities. Likewise, quantization error in range and camera images can be modeled as gaussian noise. A number of schemes exist for fusing such data ranging from simple Kalman filtering techniques to full-blown Bayesian observation networks [1] [7].

5. Local Control

In this section we discuss some of the control problems in local navigation.

5.1. Adaptive Driving Units and Sensor View Frames

Management of driving units and sensor view frames is essential in local control. As described in section 2, the driving unit is a minimum control unit, a unit to perceive objects, generate a path plan, and drive 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, an appropriate driving unit size is essential for stable control. For example, the sensor view frame cannot cover a very large driving unit. Conversely, small driving units place rigid constraints on the LOCAL PATH PLANNER, because of the short distance between the starting point and the goal point. The aiming of the sensor view frame determines the point at which to digitize an image and to update the vehicle position and path plan.

In the current system, the sensor view frame is always fixed with respect to the vehicle. The size of the driving unit is fixed for driving on roads (4~6 meters length), and is changed for turning at intersections so that the entire intersection can be see in a single image and to increase driving stability (see Figure 9). This method works well in almost all situations in current test site.

For intersections requiring sharp turns (about 135 degrees), the current method does not suffice. Because there is only one driving unit at the intersection, the system digitizes an image, estimates the vehicle's 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 the intersection until the vehicle has turned through the intersection. Though actual path generated is not so bad, it is 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 plan to improve our method for managing driving units. Our new idea is:

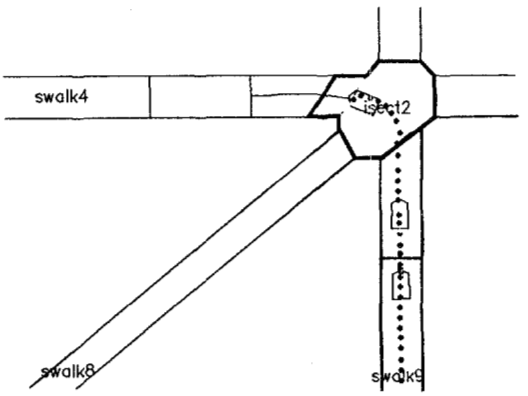


Figure 9: Intersection Driving Unit

- **Length of the driving unit:** The length of the driving unit is bounded at the low end by the LOCAL PATH PLANNER's requirements for generating a reasonable path plan, and at the high end by the view frame required by PERCEPTION for recognizing a given object.
- **driving unit interval:** The *driving unit interval* is the distance between the centers of adjacent driving units. Adjacent driving units can be overlapped, that is, they can be placed such that their interval is shorter than their length. Figure 10 illustrates this situation.
- **adjusting size and interval of driving unit:** If the passage is simple, the length and interval of the driving unit is long. If the passage is complex, for example, in the case of highly curved roads or intersections, or in the presence of obstacles, the length and interval of driving unit are shorter. And if the required driving unit interval must be shorter than the length of driving unit, the driving

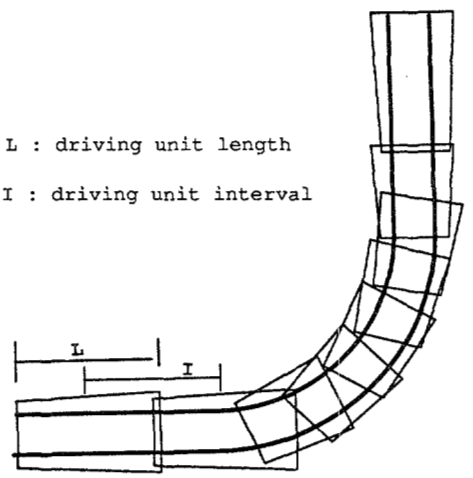


Figure 10: Adaptive Driving Units

units are overlapped. Therefore, the vehicle's position is estimated and a local path is planned more frequently so that the vehicle drives stably (see Figure 10).

- **adjusting sensor view frame:** The sensor view frame with respect to the vehicle, the distance and the direction to the driving unit from the vehicle, 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 the processing time of the

PERCEPTION and the PILOT is constant, the longer distance means a higher vehicle speed. But the longer distance produces less accuracy in perception and vehicle position estimation. Therefore, the distance is determined for the required accuracy, which depends on the complexity of passage. Using the pan and tilt mechanism, PERCEPTION can digitize an image at the best distance from the driving unit, since the sensor's view frame is less rigidly tied to the orientation and position of the vehicle.

5.2. Vehicle Speed

It is an important capability for autonomous mobile robot to adjust the vehicle's speed automatically so that the vehicle drives safely at the highest possible speed. The current system slows the vehicle down in turning to reduce driving error.

The delay in processing in the LOCAL PATH PLANNER and communication between the HELM and actual vehicle mechanism gives rise to error in vehicle position estimation. For example, because of continuous motion and non-zero processing time, the vehicle position used by the LOCAL PATH PLANNER as a starting point differs slightly from the vehicle position when the vehicle starts executing the plan. Because the smaller turning radii give rise to larger errors in the vehicle's heading, which are more serious than displacement errors, the HELM slows vehicle for the smaller turning radii. This method is useful for making the vehicle motion stable.

We are going to add the capability to the system for adjusting the vehicle speed to the highest possible value automatically. Our idea is the following:

- **schedule token:** The modules and the submodules working at the local navigation level store their predicted processing times in a *schedule token* in each cycle. PERCEPTION is the most time consuming module, and its processing time varies drasticly from task to task.
- **adjusting vehicle speed:** Using the path plan and the predicted processing time stored in the schedule token, the HELM calculates and adjusts vehicle speed so that the speed is maximum and the modules can finish processing the driving unit before the vehicle reaches the end of the current planned trajectory.

5.3. Local Path Planning and Obstacle Avoidance

Local path planning is the task of finding a trajectory for the vehicle through admissible space to a goal point. In our system, the vehicle is constrained to move in the ground plane around obstacles (represented by polygons) while remaining within the driving unit (also a polygon). We have employed a configuration space approach [5] [6]. This algorithm, however, assumes that 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.

6. Navigation Map

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

6.1. Map Structure

The navigation map is a set of descriptions of physical objects in navigation world. It is composed of two parts, the geographical map and the object data base. The geographical map stores object locations with their contour polylines. The object data base stores object geometrical shapes and other attributes, for example, the navigation cost of objects. Though, in the current system, all objects are described with both of the geographical map and the object data base, in general, either of them can be unused. For example, the location of *stairs A* is known, but its shape is unknown.

The shape description is composed of two layers. The first layer stores shape attributes. For example, the width of the road, the length of the road, the height of the stairs , the number of steps, etc. The second layer stores actual geometrical shapes represented by the surface description. It is easy to describe incomplete shape information with only the first layer.

6.2. Data retrieval

The map data is stored in the CODGER data base as a set of tokens forming tree structure. In order to retrieve map data, parents tokens have indexes to children tokens. Because current CODGER system provides modules with a token retrieval mechanism that can pick 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 combined with an inheritance mechanism.

7. Other Tasks of the System

Navigation is just one goal of a *mobile robot system.* Generally speaking, however, navigation itself is not an end, but actually a means to achieve the final goals of the autonomous mobile robot system, such as carrying baggage, exploration, or refueling. Therefore, the system architecture must be able to accommodate tasks other than navigation.

Figure 11 illustrates one example of an extended system architecture which loads, carries and unloads baggage. The whole system is comprised of four layers, *mission control, vehicle resource management, signal processing,* and *physical hardware.* The 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, there are different modules working for different tasks. Although their tasks are different, they all work in a symbolic domain and do not handle the physical world directly. These modules oversee mission execution, generate plans, and pass information to modules in the signal processing layer. Through CODGER, they can communicate with each other, if necessary. The MAP NAVIGATOR and the PILOT, parts of the navigation system, are included in the vehicle resource management layer. The MANIPULATOR makes a plan (e.g., how to load and unload baggage with the arm) and sends it to the ARM CONTROLLER.

The modules in the signal processing layer interact with physical world using senors and actuators. For example, PERCEPTION processes the signals from sensors, the HELM drives the physical vehicle, and the ARM CONTROLLER operates the robot arm. The bottom level contains the real hardware, even if it includes some primitive controller. The sensors, the physical vehicle, and the robot arm are included in this layer.

Because our current system architecture is built on the CODGER system it will be easy to expand it to include these additional capabilities.

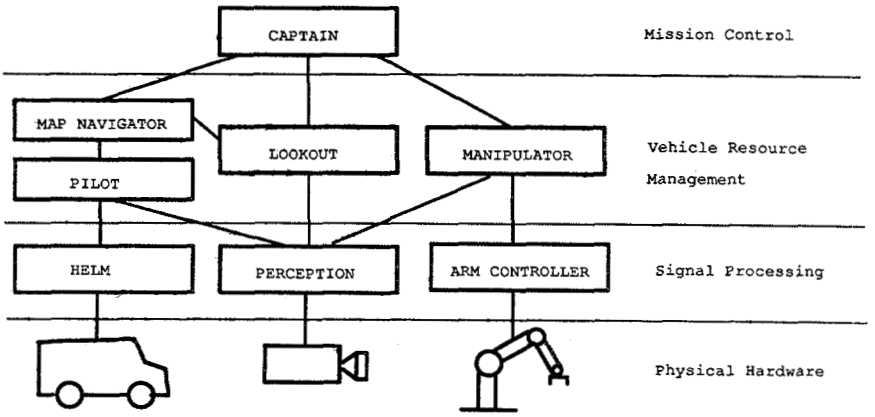


Figure 11: Extended System Architecture

8. Conclusions

In this paper, we have described the CMU architecture for autonomous outdoor navigation. The system is highly modular and includes components for both global and local navigation. Global navigation is carried out by a route planner that searches a map database to find the best path satisfying a mission and oversees its execution. Local navigation is carried out by modules that use a color camera and a laser rangefinder to recognize roads and landmarks, scan for obstacles, reason about geometry to plan paths, and oversee the vehicle's execution of a planned trajectory.

The perception, planning, and control components are integrated into a single system through the CODGER software system. CODGER provides a common data representation scheme for all modules in the system with special attention paid to geometry. CODGER also provides primitives for synchronizing the modules in a way that maximizes parallelism at both the local and global levels.

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

9. Acknowledgements

The design of our architecture was shaped by contributions from the entire Autonomous Land Vehicle group at CMU. We extend special thanks to Steve Shafer, Chuck Thorpe, and Takeo Kanade.

I. References

[1] Durrant-Whyte, H.
Integration, Coordination and Control of Mult-Sensor Robot Systems.
PhD thesis, University of Pennsylvania, 1986.

[2] Goto, Y., Matsuzaki, K., Kweon, I., Obatake, T.
CMU Sidewalk Navigation System.
In *FJCC-86.* 1986.

[3] Hebert, M. and Kanade, T.
Outdoor Scene Analysis Using Range Data.
In *Proc. 1986 IEEE Conference on Robotics and Automation.* April, 1986.

[4] Kanade, T., Thorpe, C., and Whittaker, W.
Autonomous Land Vehicle Project at CMU.
In *Proc. 1986 ACM Computer Conference.* Cincinnati, February, 1986.

[5] Lozano-Perez, T., Wesley, M. A.
An Algorithm for Planning Collison-Free Paths Among Polyhedral Obstacles.
Communications of the ACM 22(10), October, 1979.

[6] Lozano-Perez, T.
Spatial Planning: A Configuration Space Approach.
IEEE Transactions on Computers C-32(2), February, 1983.

[7] Mikhail, E. M., Ackerman, F.
Observations and Least Squares.
University Press of America, 1976.

[8] Shafer, S., Stentz, A., Thorpe, C.
An Architecture for Sensor Fusion in a Mobile Robot.
In *Proc. IEEE International Conference on Robotics and Automation.* April, 1986.

[9] Stentz, A., Shafer, S.
Module Programmer's Guide to Local Map Builder for NAVLAB.
1986.
In Preparation.

[10] Wallace, R., Stentz, A., Thorpe, C., Moravec, H., Whittaker, W., Kanade, T.
First Results in Robot Road-Following.
In *Proc. IJCAI-85.* August, 1985.

[11] Wallace, R., Matsuzaki, K., Goto, Y., Webb, J., Crisman, J., Kanade, T.
Progress in Robot Road Following.
In *Proc. IEEE International Conference on Robotics and Automation.* April, 1986.