# Development of an Integrated Mobile Robot System at Carnegie Mellon University: June 1988 Annual Report

**Steve Shafer and William Whittaker**

CMU-RI-TR-89-22

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

July 1989



This research is sponsored by the Defense Advanced Research Projects Agency, DoD, through DARPA order 5682, and monitored by the U.S. Army Engineer Topographic Laboratories under contract DACA76-86-C-0019. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or of the U.S. Government.

# Table of Contents

## List of Figures

# Abstract

This report describes progress in development of an integrated mobile robot system at the Carnegie Mellon Robotics Institute from July 1987 to June 1988. This research was sponsored by the Defense Advanced Research Projects Agency and monitored by the US Army Engineer Topographic Laboratories under contract DACA76-86-C-0019.

Our program includes a broad agenda of research in the development of mobile robot vehicles, focused on the *NAVLAB* computer-controlled van. In the year covered by this report, we addressed major issues in both hardware and software for autonomous mobile robots:

- **Evolution of the NAVLAB Vehicle.** We built the NAVLAB mobile robot vehicle in our previous work under this contract, by outfitting a commercial truck chassis with computer-controlled drive and steering controls and a set of on-board computer workstations. The NAVLAB serves as a mobile navigation laboratory that allows researchers to interact intensively with the system during testing and execution. This year has seen a continued evolution and improvement of the NAVLAB mechanism, sensors, controller, and Virtual Vehicle interface to higher-level planning and perception software.
- **Evolution of the CODGER Blackboard.** Last year, as part of this research program, we designed and implemented the CODGER blackboard system for robot perception and reasoning on a distributed collection of processors. This year, in response to our experience in using CODGER for mobile robot control, we have upgraded it to deal with geometric models and uncertainty in perception and map data.
- **Experiments With the Driving Pipeline.** To control the NAVLAB and Terregator mobile robot vehicles, we developed the Driving Pipeline architecture last year for coordinating road following, obstacle avoidance, and vehicle motion control. In our ongoing research, we have performed numerous experiments with this system that demonstrate its value.

This hardware and software is the basis for the **New Generation System** (NGS) for robot vision and navigation, which integrates many independent technologies to produce an integrated mobile robot system.

## Acknowledgements

This research has been a team effort involving many people, including: William Whittaker, Steve Shafer, Takeo Kanade, Chuck Thorpe, Paul Allen, Gary Baun, Mike Blackwell, Kevin Dowling, Thad Druffel, James Frazier, Eric Hoffman, Ralph Hyre, James Ladd, James Martin, Clark McDonald, Jim Moody, Henning Pangels, David Simon, Bryon Smith, Eddie Wyatt, Yoshi Goto, Taka Fujimori, Inso Kweon, Doug Reece, and Tony Stentz.

# Section I

# Introduction

## Introduction and Overview

This report reviews progress at Carnegie Mellon from July 1, 1987, to June 30, 1988, on research sponsored by the Strategic Computing Initiative of DARPA, DoD, through ARPA Order 5682, and monitored by the US Army Engineer Topographic Laboratories under contract DACA76-85-C-0019, titled "Development of an Integrated Mobile Robot System." This report consists of an introduction and overview, and detailed reports on specific areas of research.

In our previous work under this contract, we developed a computer-controlled mobile robot, the NAVLAB, as a tool and testbed for research in robot navigation, and we developed a software framework for integrating vision, planning, and control modules into a single working system. The modules themselves are under development through a related research effort in "Road Following", which is also sponsored by DARPA. The total system has been demonstrated in outdoor navigation runs without human intervention, on a road in Schenley Park, Pittsburgh, near the Carnegie Mellon campus.

This year, we have made progress in several areas of the NAVLAB hardware and software:

- **Evolution of the NAVLAB Vehicle.** We built the NAVLAB mobile robot vehicle in our previous work under this contract, by outfitting a commercial truck chassis with computer-controlled drive and steering controls and a set of on-board computer workstations. This year has seen a continued evolution and improvement of the NAVLAB mechanism, sensors, controller, and Virtual Vehicle interface to higher-level planning and perception software.
- **Evolution of the CODGER Blackboard.** Last year, as part of this research program, we designed and implemented the CODGER blackboard system for robot perception and reasoning on a distributed collection of processors. This year, in response to our experience in using CODGER for mobile robot control, we have upgraded it to deal with geometric models and uncertainty in perception and map data.
- **Experiments With the Driving Pipeline.** To control the NAVLAB and Terregator mobile robot vehicles, we developed the Driving Pipeline architecture last year for coordinating road following, obstacle avoidance, and vehicle motion control. In our ongoing research, we have performed numerous experiments with this system that demonstrate its value.

**Summary: Evolution of the NAVLAB Vehicle.**

In this year, Robotics Institute researchers logged over 900 hours of mobile robot experiments abord the NAVLAB. Significant maintenance efforts have been carried out to support this demanding schedule. In addition, improvements have been made in several aspects of the NAVLAB itself, including the NAVLAB hardware, new sensors, and improvement of the Virtual Vehicle Interface.

Ongoing development of the hardware has been aimed at improvement of the power generation, reliability, and driveability of the NAVLAB. One problem was the falloff of power during uphill runs. This was solved by replacing the original throttle with an analog engine speed control system to provide constant engine speed and thus constant power. We also re-configured the on-board computers to

facilitate support of the WARP supercomputer on the NAVLAB. The motion control boards were redesigned to provide smoother driving. All of these and other upgrades of the vehicle are placing an ever-increasing load on the air conditioning and weight limits of the vehicle.

We have also installed and integrated several new sensors on the NAVLAB. Two of these, a Global Positioning System satellite receiver and an inertial navigation unit, are joined together into a subsystem for vehicle position determination. Another sensor is specialized for collision avoidance: a single-scan-line (1D) laser range scanner. This laser scanner allows us to implement a rapid-response clearance check for obstacles in the environment. This is necessary even with perfect 3D terrain and obstacle sensing, because the control error in the vehicle can cause it to deviate from the planned path through the terrain and obstacles.

The Virtual Vehicle Interface was also improved this year. This interface is the command set through which the high-level software for perception and planning can communicate with the vehicle control subsystem. Improvements to the Virtual Vehicle Interface include a new mode of operation that executes commands immediately instead of queuing them in order of receipt, and providing more feedback to the high-level software concerning vehicle status and the execution of commands. Also, the control software can now handle variable-length driving units, which was an important feature for conducting the Driving Pipeline experiments described below.

This research is described in more detail in Section II: "Evolution of the NAVLAB Vehicle".

**Summary: Evolution of the CODGER Mobile Robot Blackboard.**

In the first year of this contract, the CODGER mobile robot blackboard was developed and used to control the NAVLAB. CODGER implements a distributed database with a central database manager module, and features data values and operators to support geometric reasoning for robot navigation. In the last year, we have developed CODGER II, which is based on CODGER but includes new features to address important issues in mobile robot integration.

The first set of new features in CODGER II were added to suport map representation. A robot that is navigating using a map needs to make many different kinds of queries about the data, such as "what is the next road segment?" and "are there any visible obstacles in this region?". While CODGER I had facilities for geometric representation of polygons, it did not posess a mechanism for answering topological questions about connectivity and adjacency. We developed a complete 2D geometric modeling capability and added it to CODGER for use in representing and utilizing map data.

In a mobile robot vehicle, the geometric relationships between the vehicle and the world are constantly changing, and the vehicle itself may have moving parts such as vehicle suspension and pan-tilt mounts for sensors. The systems needs to be able to maintain both the current relationships and a complete history of the geometric relationships among objects. To facilitate this, CODGER II introduces the concept of *frame generators* that represent time-varying geometric transformations between objects. The objects themselves are organized into *affixment groups* of relatively stationary objects. Within an affixment group, all geometric transforms are stationary; across affixment groups, the transforms vary over time.

Representing map information is very important for vehicle navigation, but map data is not always

complete and accurate. It is very important for the vehicle to be able to update map information as it makes new observations about the world. This requires that each observation about the world be recorded, and be marked as an observation so that it can be used to incrementally revise the pre-stored map information. Observations are a particular type of frame generator. Whenever an observation is made, the other frame generators are updated to resolve any inconsistency between the old and the new data. This same approach is used to resolve multiple, possibly inconsistent, sources of information about vehicle position itself; this provides a capability for landmark navigation that integrates on-board motion sensors with landmark recognition.

CODGER II will be a capable framework for continued experimentation in the integration of symbolic and quantitative map data with observations from a robot vehicle in the field. This research is described in more detail in Section III: "Evolution of the CODGER Blackboard".

**Summary: Experiments With the Driving Pipeline.**

Mobile robot vehicles must control the execution of numerous perception and planning processes to navigate successfully in complex environments. In the past, most mobile robot systems have utilized "stop-and-go" control schemes that avoid addressing the driving control problem, or have used fixed control schemes that do not allow for the changing environment and field of view of the vehicle. This report presents our architecture for mobile robot control called the "Driving Pipeline", that integrates multiple perception and planning processes and provides continuous motion with adaptive control. The Driving Pipeline has been implemented and tested on numerous versions of two vehicles: the Terregator and the NAVLAB. It has proven to be a flexible and powerful mechanism for building integrated software for mobile robot perception and planning.

The Driving Pipeline is based on the principle of dividing the navigation area into small (5-10m) pieces called *driving units*. By dividing the ground into driving units, each unit can be processed separately by the various sensors and planning systems on the vehicle.

The processing steps themselves include vision and range sensing, analysis of the environment, and trajectory planning. Each step must be executed in turn before the vehicle actually traverses each driving unit. Since the steps are sequential and the vehicle travels sequentially over the driving units, the steps can be executed in parallel on the successive driving units ahead of the vehicle. This arrangement provides fast enough throughput to allow continuous motion of the robot vehicle.

The driving units are not always the same length. When the vehicle approaches a curve or intersection, the field of view of the sensors does not completely overlap the road. This reduces the distance that the vehicle can look ahead; therefore, smaller driving units will be used in such places. Since the vehicle travels each driving unit in approximately constant time, the result is that the vehicle automatically and smoothly slows down when the vehicle turns.

When the vehicle has a map available in advance, the Driving Pipeline can operate as just described. However, if there is no map, then the environmental analysis for one driving unit must be completed before the next driving unit can begin to be processed. This reduces the ability of the system to execute multiple functions in parallel, and naturally results in a slower vehicle speed. Thus, the availability of a map allows the vehicle to move faster.

This research is described below in Section IV: "Experiments With the Driving Pipeline".

**Summary: Experimentation on the ALV**

Our experience at Carnegie Mellon includes both the integrated NAVLAB system (this contract) and basic research on perception and planning (the related Road-Following contract). This has given us at CMU a rather unique perspective on the interaction between the two. At the DARPA Autonomous Land Vehicle workshop in Vail, Colorado (April 1988), the subject of discussion was how basic research and integrated system development can interact most profitably for both. Because of our experience in both domains, we were asked by DARPA to prepare a summary after the workshop for use as a planning document by the ALV and Strategic Computing Vision communities for future research. We prepared such a document, and it has been used for such research planning within the ALV/SCVision community.

Our experience has been both positive and negative in the interaction among research paradigms (basic v. systems). Our conclusions are:

- Basic research without systems development can make great progress but eventually becomes out of touch with real-world problems.
- Beyond that point, integrated systems research and development is essential for defining the specific problems that need to be addressed by further basic research efforts. Furthermore, the simple act of collecting data for basic research becomes so demanding that only an integrated system can serve as an appropriate data-collection platform.
- When specific problems have been defined through the system development effort, more basic research is then needed. However, because integrated systems are big and have great inertia, they are resistant to easy change. Thus, for purely software engineering reasons, it is wrong to expect that all the new basic research will be fully compatible with existing systems. Rather, the basic research should be allowed to "piggy-back" on the big systems, for example using the system to move a robot vehicle while collecting brand-new data for off-line analysis with the new perceptual techniques.
- Finally, when the basic research has shown how to construct new, more reliable and useful components, then a new integrated system development is appropriate.

These issues are discussed in our report to DARPA, which is reproduced as Appendix I: "Experimentation Issues for Mobile Robot Systems". Although the report specifically talks about the ALV, the issues and conclusions are appropriate for all research in large, integrated robot systems.

## Accomplishments

The key accomplishments of this research in the time period from July 1987 to June 1988 have been:

- Vehicle and controller enhancements in support of 900 experimental hours.
- Fast processing of radial range data for safeguarding by a soft bumper.
- Improvement of the Virtual Vehicle Interface between the high-level and low-level computer systems.
- Development of the CODGER II blackboard with new features for geometric modeling, time-varying coordinate systems, and uncertainty modeling.
- Experiments with the Driving Pipeline and development of variable-sized driving units.
- Demonstrations of complete NAVLAB system with these new features in Schenley Park.

## Technology Transfer

The NAVLAB has a fairly unique status as a robot vehicle whose architecture is suited for research in both integrated robot systems and invidual component technologies (path planning, map navigation, and perception). Thus, the NAVLAB fills an important role in the research community as a focal point for technology transfer operations. The key areas of technology transfer to and from the NAVLAB have been:

- *Exchange of software and concepts for perception and planning:* Image data and visual motion analysis code have been exchanged with the University of Massachusetts. A path planner developed at Hughes is being expanded on for use in the NAVLAB.
- *Export of NAVLAB hardware and software for other robot vehicles:* The CODGER database has been sent to Martin-Marietta for use in the ALV, to other ALV contractors including FMC and ADS, and to non-DARPA sites including NASA-Goddard and DEC. This hardware and software is being used at CMU and elsewhere for space exploration and underwater robots as well as several land vehicles.

## Future Directions

We have identified several problems and issues as likely directions for our research in the next year:

- We need to develop a new generation of the low-level controller system that provides a high-performance UNIX-like environment.
- The vehicle path tracking is not as predictable as we would like. We have begun to develop a new path tracking method based on continuous replanning of quintic arcs to provide more precise vehicle control.
- We will continue development of the x-y-$\theta$ path planner based on the Hughes path planner, and add to it uncertainty management and representation.
- The Driving Pipeline concept has been very serviceable, but it has some key limitations. In particular, the need for all subsystems to operate in a pipeline means that computational and sensing resources are not operated at maximum efficiency. The new path planner may provide a good alternative scheduling mechanism for perception and other activities.

# Section II

# Evolution of the NAVLAB Vehicle

Under this contract, we developed the NAVLAB mobile robot van last year. With on-board sensors and computing, and seating and controls for researchers, the NAVLAB is a self-contained laboratory for research in autonomous mobile robots.

In the year from July 1987 to June 1988, Robotics Institute researchers logged over 900 hours of mobile robot experiments aboard the NAVLAB. In the course of this research, development has continued on several aspects of the NAVLAB itself, including:

- Upgrades of the NAVLAB Hardware
- Integration of New Sensors
- Improvement of the Virtual Vehicle Interface

In addition, significant maintenance has been carried out to support the demanding schedule of live experimentation.

### Ongoing Hardware Development

This year, we made several improvements to the NAVLAB's mechanical systems to enhance its power generation, reliability, and driveability (Figure 1):

- The NAVLAB, as originally designed, suffered from power falloffs during uphill runs, due primarily to inappropriate carburetor design. To correct for this deficiency, we designed, built, and installed an analog engine speed controller, which replaced the original throttle. The new throttle control adjusts the engine carburetion to maintain a constant engine speed regardless of load.
- New computers were installed in early 1988: one rack was re-configured to consolidate three SUN 380s into one enclosure, and a re-worked WARP supercomputer was installed in a VME cabinet. Modifications to the air conditioning system were made to cool these devices. In addition, thermal shutdown sensors were installed in the WARP to prevent overheating. While we experienced no difficulties in providing ample, clean power for these computers, the air conditioning is operating continuously at full capacity and will require an extensive overhaul or replacement in the near future.
- The electrical power generation problems, stemming from generator design, were resolved with the vendor in 1988. A clean, constant power supply is now in place.
- Sound proofing was added to reduce interior noise levels.
- The hydraulic drive controls were upgraded to increase their reliability.
- New motion control boards were designed, built, and installed to improve motion control; driving performance is now smoother and virtually free of oscillation.

The net effect of all of these improvements is to create a more reliable research vehicle, with greater uptime, more predictable behavior, and a better environment for the passenger/researchers. However, there are additional mechanical issues that will need to be addressed in the near future. In particular, some consideration will have to be given to engine performance. The vehicle's weight has doubled since the project began, and as the capability to increase autonomous driving speed increases, so will the demand for more horsepower.

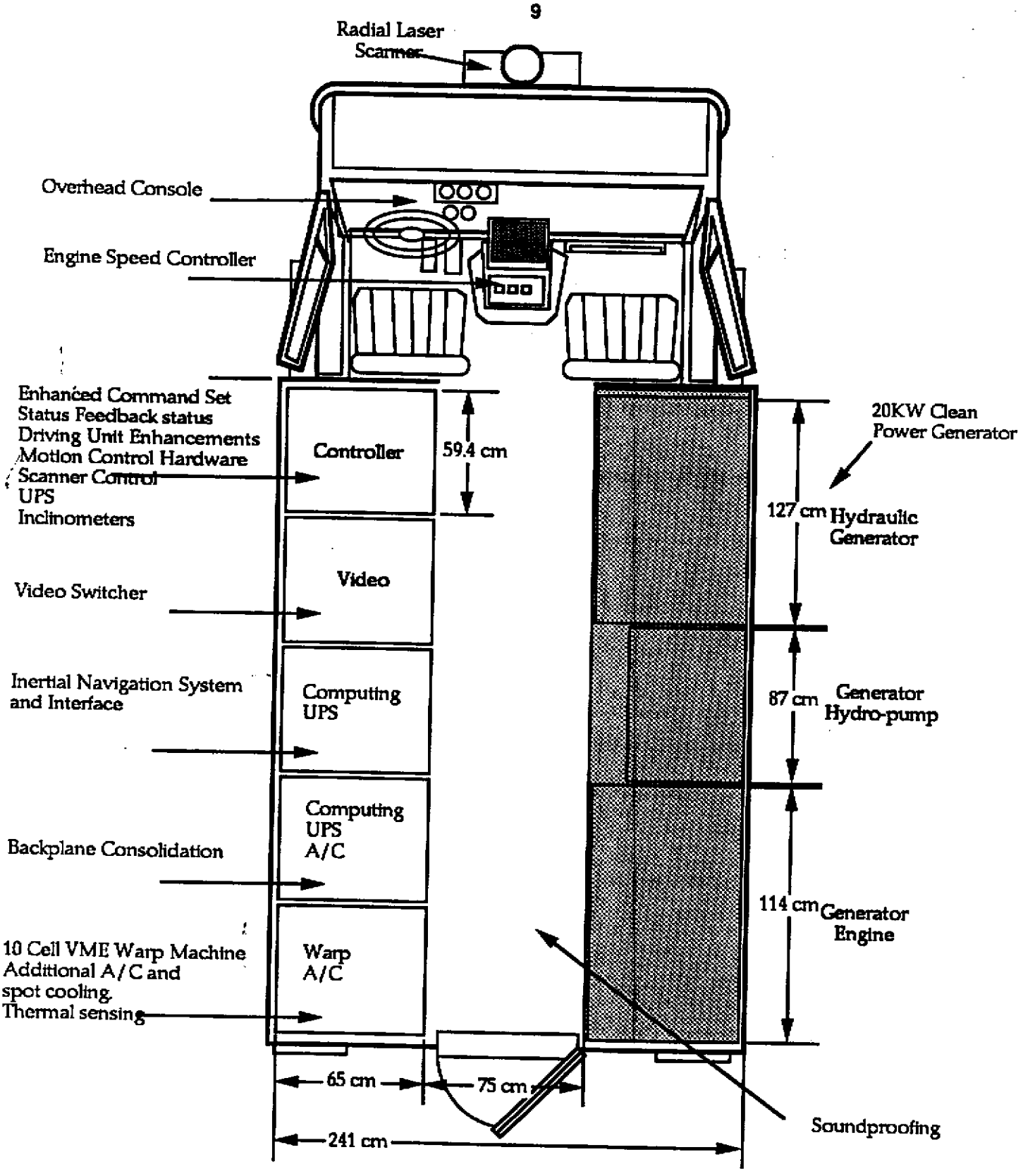



**Figure 1:** Improvements in the NAVLAB Vehicle

### Integration of New Sensors

In conjunction with a related contract to develop high-speed off-road navigation, three new sensors were integrated into the NAVLAB for specific experiments. These include a Global Positioning System satellite receiver and an inertial navigation unit, which together we call the Vehicle Positioning System (VPS), and a front bumper-mounted single-axis (1D) radial laser scanner that provides a "soft bumper" for vehicle safeguard. The inertial navigation unit provides position data that is consistently accurate to 0.5 m. However, inertial measurements tend to drift with distance traveled. Software proprietary to the contractor processes data from the GPS to correct this.

To help with our experimentation, we devoted an extensive effort to developing utilities for imaging inertial and range data. For range data, we developed utilities to store and retrieve scanner measurements. These images can be displayed on either a SUN or an external video monitor. We also developed a utility to store and recall data collected from the VPS system. Data from the VPS can be displayed relative to time or to any other VPS data. Finally, we developed utilities to display a reference path and compare it graphically to the actual path traveled.

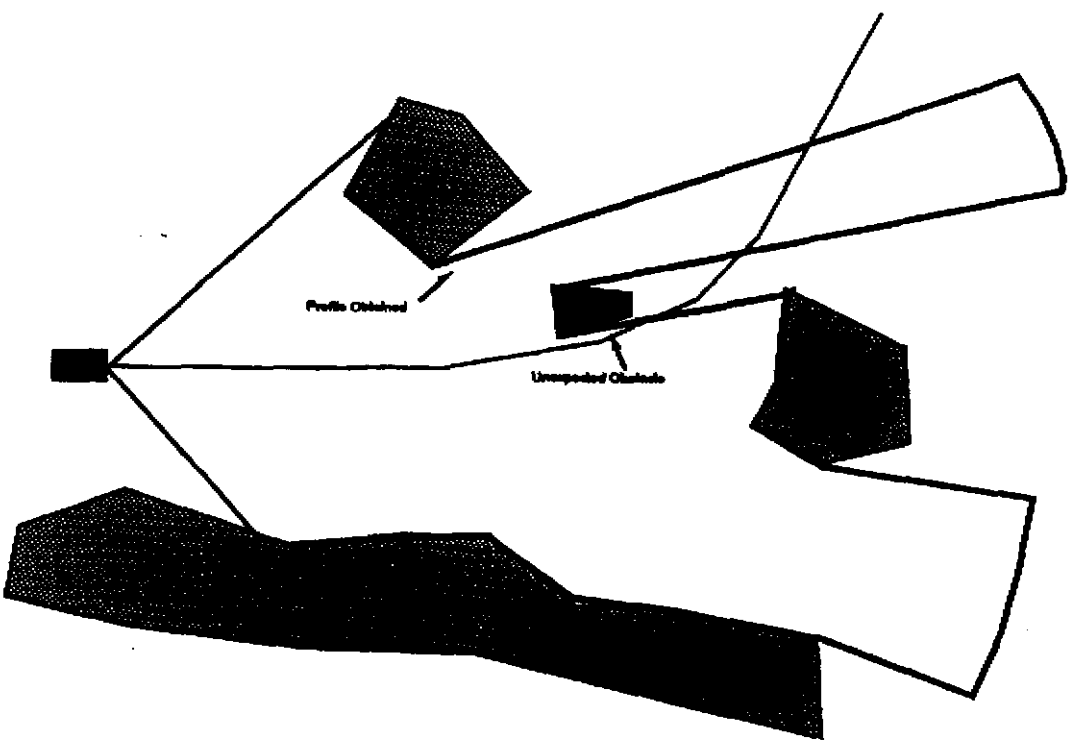


**Figure 2:** Unexpected Obstacle Due to Control Error

With these new sensors, we were able to pursue the idea of *clearance checking* as opposed to the traditional *terrain planning* for obstacle avoidance. In this approach, the space immediately ahead of the vehicle is continually checked for obstacles, instead of relying on strict adherence to a long path planned through cluttered terrain. The problem with the traditional terrain planning approach arises from errors in position estimation relative to a global coordinate frame. Such errors result in the vehicle following an

actual path that deviates from the reference path by the amount of the error. However, since the tracking and collision avoidance schemes use the same position estimate, collision avoidance continues to search about the reference path rather than the actual vehicle path. Thus, even with perfect range data, terrain planning can cause collisions due to imperfect vehicle control. However, with online clearance checking, collision avoidance is controlled at the lowest level by dedicated sensors that move along with the vehicle and thus are centered on the actual path rather than the idealized reference path. This is illustrated in Figure 2, which shows how control error in the vehicle path can cause it to encounter obstacles that could not be predicted from range data, even perfect range data, taken at a distance.

So far, the scheme we have used for collision avoidance presumes a flat and level ground plane. The range sensor scans in a plane horizontal to the ground plane and thus is certain to miss objects lower than the height of the beam. We have so far demonstrated an implementation of collision avoidance using two processors working on the vehicle at 5 mph. Consideration of other schemes is in progress and will be implemented in simulation in the near future.

**Improvement of the Virtual Vehicle Interface**

The Virtual Vehicle Interface (VVI) is the command set through which the high-level planning and perception software communicates with the low-level vehicle control system. In the past year, several aspects of the Virtual Vehicle Interface were improved. One area of improvement was the enhancement of the VVI command set. The new commands allow explicit control of steering angle and drive speed by a host computer. This feature enables high-speed path-tracking algorithms to supply reference signal updates to the vehicle servo controllers at rates of up to 4 Hz. In this mode of operation, no queuing of commands takes place; the reference signals to the servo controllers are updated as soon as the corresponding command is received. We also added status fields to the arc commands to indicate whether a commanded arc was executed normally by the vehicle.

In addition, we improved the ability of the controller software system itself to respond to external events occurring asynchronously. These signals include current gearing (low/high/neutral and forward/reverse), control mode (computer/manual), brake status (on/off), and activation of the kill switch. These hardware status signals have also been made available to the host computer by activating the previously unused "REP" command of the VVI command set.

The VVI was also modified to handle variable-length driving units, which allows it very naturally to control speed at intersections. When the vehicle approaches an intersection to make a turn, the lookahead distance of the sensors is reduced because of the bend in the upcoming path of the vehicle. The NAVLAB can now account for this by shortening the driving unit size. The vehicle naturally slows when passing through the turn, according to the driving unit size. This enhancement to the NAVLAB was dictated by the needs of the Driving Pipeline research, which is described in Section IV of this report.

Finally, we have formulated a likely future enhancement that will be needed to the NAVLAB. There is a need for faster processing of immediate arc commands, which are necessary to control the robot at higher road speeds. During typical operation of the vision/navigation system, several immediate arc commands are issued to the controller, with the intent that the most recent one of them is to supersede all

previous ones. The controller currently queues arcs in the order they are received. The new algorithm, which places highest priority on the most recently received commands, will allow faster execution of immediate arc commands.

# Section III

# Evolution of the CODGER Blackboard

In the previous year of this contract, we designed and implemented the CODGER mobile robot blackboard to serve as the framework for the high-level NAVLAB software. CODGER was successfully built, and with it we have performed many experimental runs with the vehicle. On the basis of these experiences, we have become aware of a number of additional problems in mobile robot system design that have not been raised in the literature to date. Accordingly, we have implemented a new version of CODGER with many fundamental new features that address these issues.

## The CODGER I System

The basic design of CODGER was described in our previous annual report, "June 1987 Annual Report: Development of an Integrated Mobile Robot System at Carnegie Mellon" [8], and will not be repeated in detail here. However, the significance of CODGER's key features has only become clear to us through the last year of research and experimentation, so we will begin with a brief review of CODGER.

CODGER is a "blackboard" of the type that is now fairly common for robot systems. Actually, in traditional terms, it is a distributed database with synchronization facilities. Each module is then a separate program, which communicates with the central database; the modules may all be on one computer, or they may be distributed among machines on a network, or any combination of these. Some other mobile robot systems are based on message-passing, which is not equivalent to using a database: a database system is more powerful then message-passing. To see this, note that there are two types of data communication -- *explicit* passing of data from one module to a specific other module, and *implicit* communication where the data is anonymously recorded, stored indefinitely, and reported to one or more clients upon request. Message-passing systems implement only the explicit communication, but require an outboard "database module" to handle the implicit communication; whereas database systems implement the implicit communication which can carry out explicit communication as a special case. Thus, database systems are more powerful than message-based systems. CODGER is a database system; thus each module is provided with primitive operations to store data, to search for and retrieve data, and to wait for data to arrive (as in producer/consumer dataflows).

CODGER implements a centralized database, with a single program that actually stores the data and handles all communications with the modules. CODGER thus has a "star" architecture with the database module (called the LMB, "Local Map Builder") in the center. Other designs might be to distribute the data by broadcasting and replicating all data, or by partitioning the database among the processing modules; these provide the same functionality as the centralized implementation, and differ only in performance. The centralized implementation of the CODGER database adds a bit of (usually negligible) overhead time to data transfers, but it facilitates the implementation of many of the sophisticated features described below, such as updating observed object locations when the vehicle position is corrected. The star architecture of CODGER is therefore a good choice for a research-oriented mobile robot system.

CODGER is based on a fairly standard database design that implements *tokens* composed of *attribute/value pairs*. The values are generally of common data types such as integer, floating-point

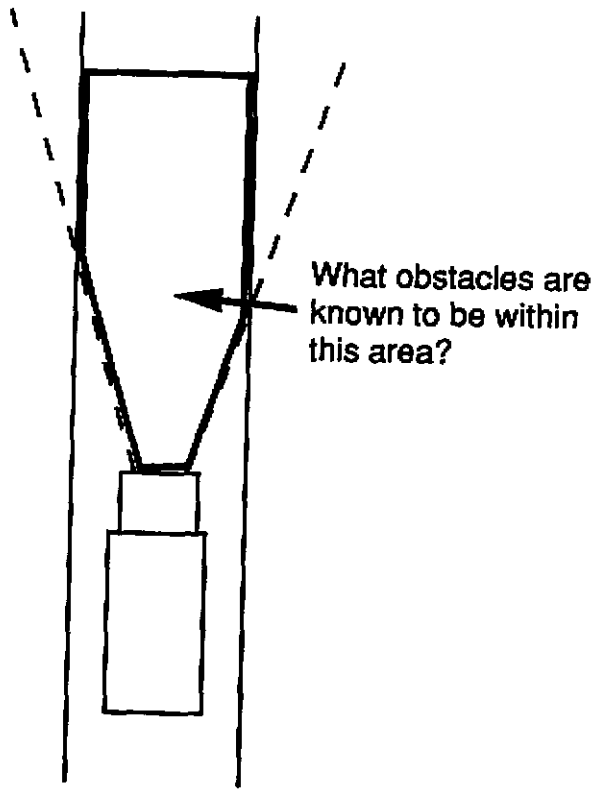



**Figure 3:** The Need for Geometric Values

number, Boolean, string, enumerated type, or array (or set) of any of these. However, CODGER begins to depart from traditional databases by incorporating geometric values as well. The need for geometric values is illustrated in Figure 3. Here, the vehicle is traveling down the roadway, has perceived the road boundaries, and wants to perform path planning. Therefore, the database is requested to provide the set of all obstacles known to be within the area of the roadway, up to the desired planning horizon of the robot. To perform the query, the database must know where to search; thus, three things must be intersected:

- the area of the roadway
- the field of view of the obstacle (range) sensor
- the distance limit of the path planner

The resulting intersected area is the search area for the the data retrieval; then, the Local Map Builder (LMB) must find all obstacles whose area intersects this search area. To solve this problem, CODGER implements data values that are geometric objects of the following types: point, line segment, polygon. The search requests can specify a number of geometric operations such as intersection, union, centroid, convex hull, area. For example, a module can request to find "all objects with **area** > 100" or "all objects whose **location** is within the intersection of *polygon X* and *polygon Y* and whose *distance from the vehicle* is less than 30". Such geometric primitives are necessary for geometric reasoning, which is the heart of "middle-level" mobile robot planning for obstacle avoidance.

CODGER was the first system to implement such geometric reasoning along with other database primitives, but it is no longer unique in that regard. Other systems, such as the SRI Core Knowledge

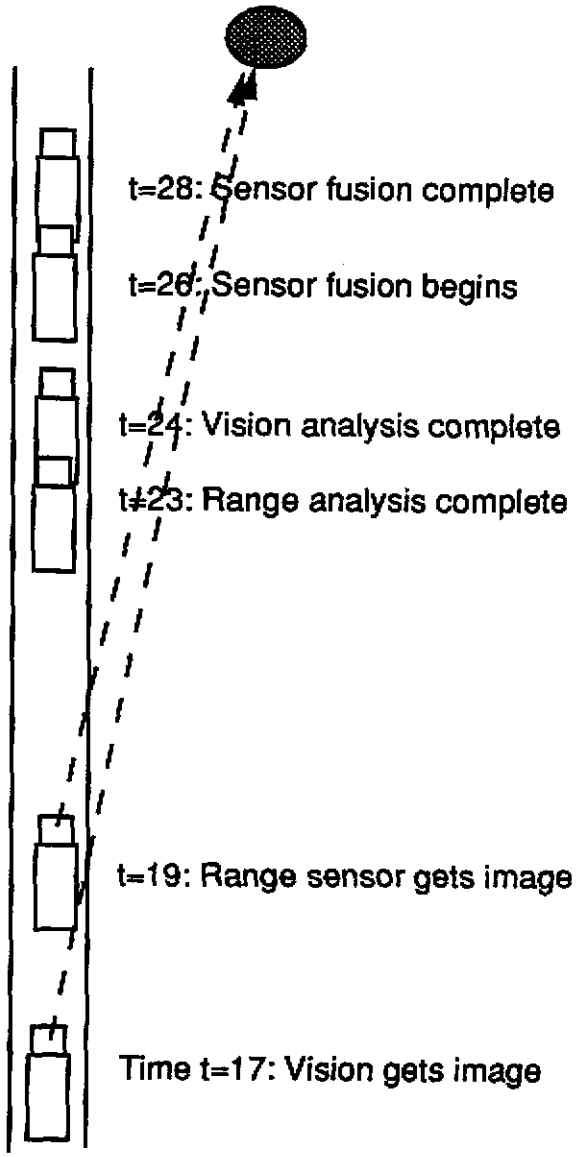



**Figure 4:** Fusion of Data in a Moving Robot

System, also implement geometric reasoning. However, such facilities address only the most basic problem in geometric reasoning. Figure 4 illustrates the additional problem that arises from attempting to fuse data from several sensors in an asynchronous system. Here, vision data from time 17 is analyzed at time 24, while range data from time 19 is analyzed at time 23; both results are fed into a sensor fusion module at time 26, which produces an answer at time 28. The key point is that the data received by this fusion module includes vision data relative to the vehicle's position at time 17, and range data relative to the vehicle's position at time 19. Thus, all data concerning the vehicle/world relationsip must be time-stamped, and the system must continuously maintain the vehicle-to-world transformation. Most systems solve this problem by immediately transforming all data into some absolute world coordinates as soon as it is received. However, this assumes that the vehicle-to-world transformation is always accurate, which is a bad assumption for most robot vehicles.

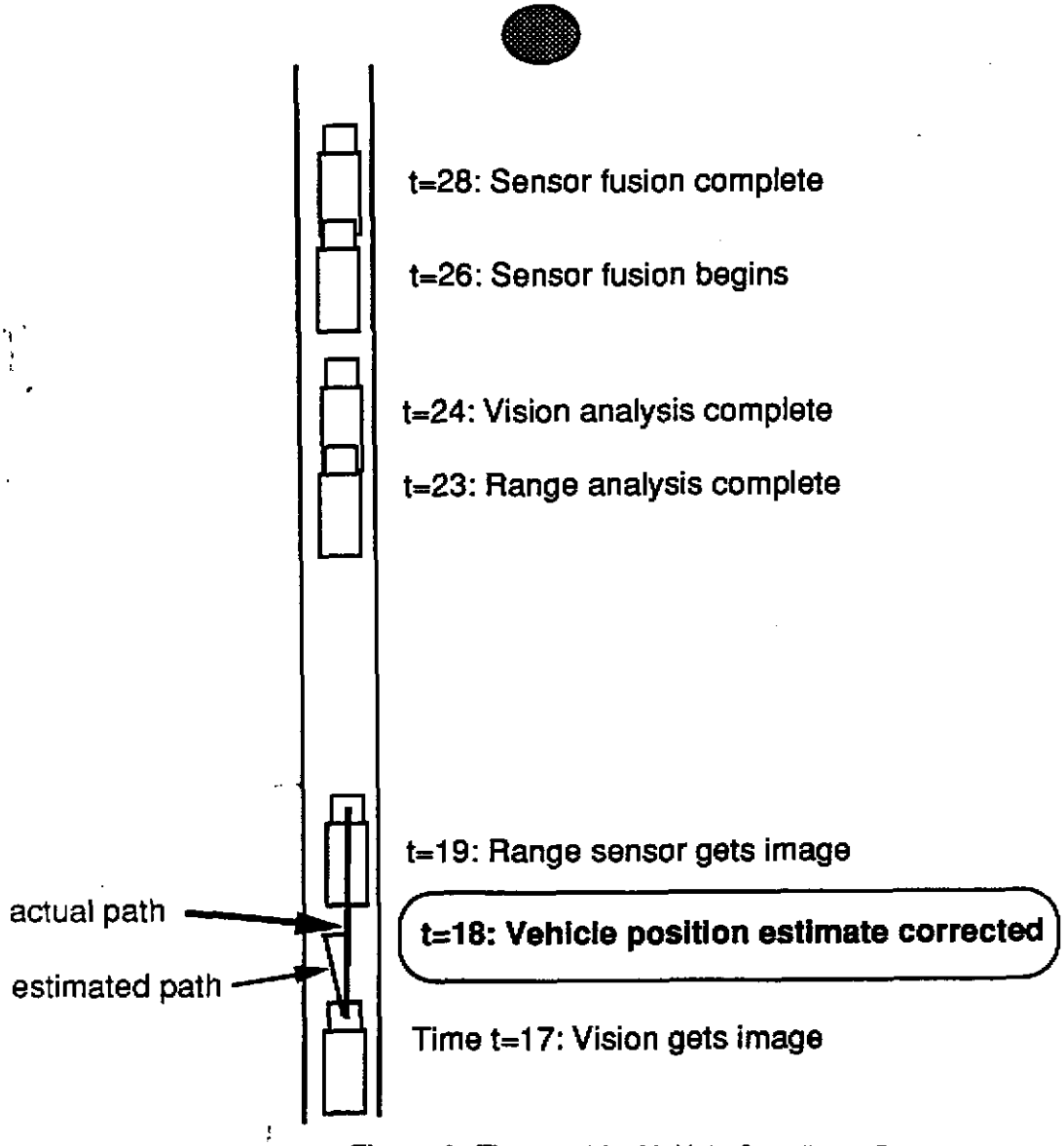



**Figure 5:** The Need for Multiple Coordinate Systems

The problem is illustrated in Figure 5. In this figure, the vehicle had drifted from its ideal path, and the drift was corrected at time 18. If the data were always stored in global world coordinates, then the vision data from time 17 would have to be updated at time 18 when the vehicle position is corrected. However, the vision system has just begun to analyze this data, and won't be finished with it until time 24. Thus, the system has to remember until time 24 that the vision data from time 17 has to be corrected according to the update of time 18! Such chains of geometric corrections quickly become unmanageable. Therefore, CODGER implements a different approach. In CODGER, all data related to sensor observations is stored relative to a "vehicle" coordinate frame, along with its time-stamp. The "vehicle-to-world" transformation is parameterized by time. Thus, the fusion module at time 26 actually receives sensor data from "vehicle at time 17" and "vehicle at time 19"; CODGER provides facilities for performing all necessary coordinate transformations. In this case, the transformation depends only on the relative vehicle motion and is independent of the vehicle-to-world update at time 18. Then, later on, when the

path planner attempts to relate the sensor data to the "world" coordinate system, CODGER will automatically incorporate the entire history of the vehicle-to-world transformation including the update at time 18. Thus, the vehicle-to-world update at time 18 will be automatically taken into account by CODGER and need not be explicitly remembered by the processing modules themselves.

These essential features -- geometric values and retrieval primitives, time-varying coordinate transforms, and multiple coordinate systems -- were all implemented in the original CODGER system a year ago. However, the discussion here has pointed out a number of significant insights about the system that were developed within this past year.

This year, the representational facilities in CODGER have been upgraded to deal with a number of additional problems that we have encountered or that we anticipate as a result of our further experimentation with the NAVLAB vehicle. Together, we call this new version of the system CODGER II.

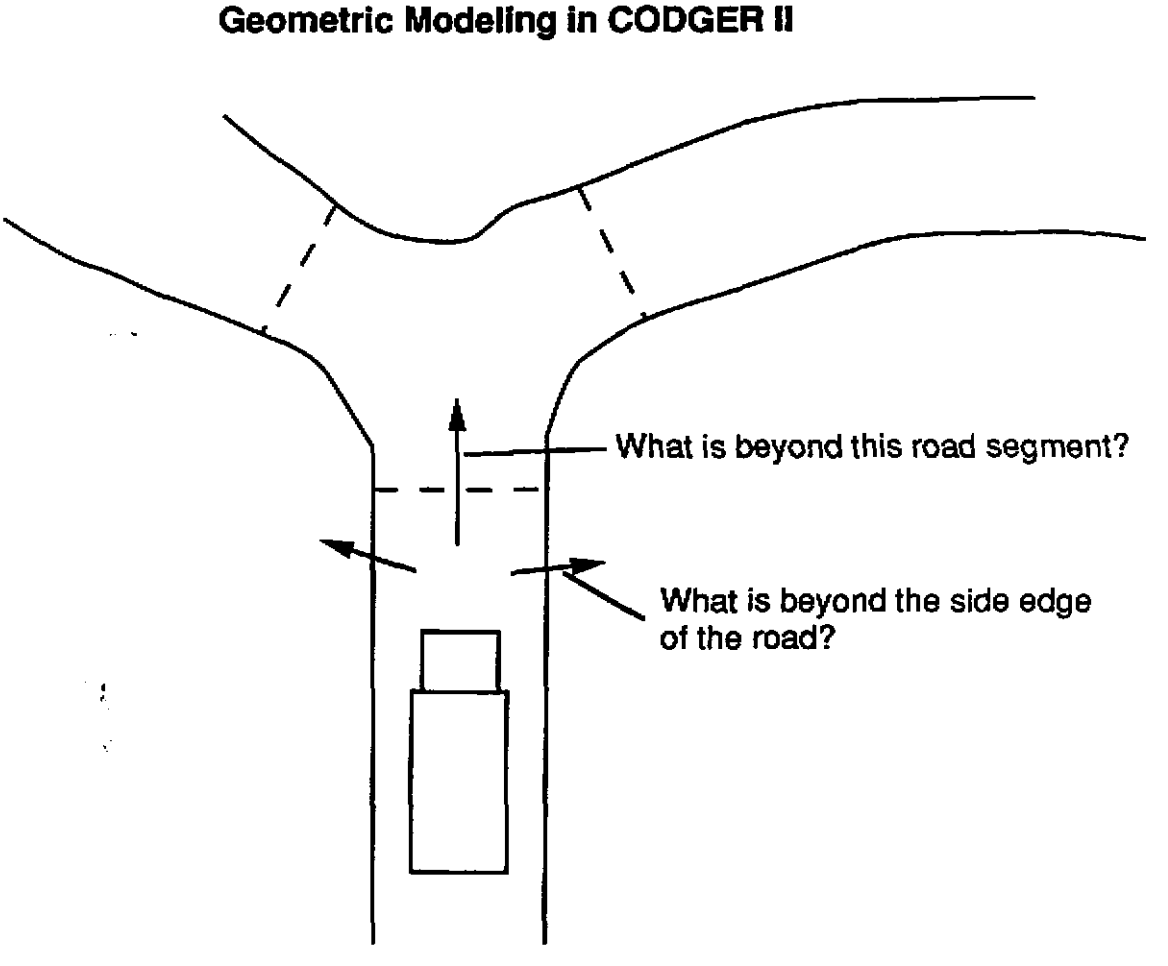



**Figure 6:** The Need for Connectivity Information

In CODGER I, the only information about object locations was contained in the polygon attached to each individual object. This has not proved adequate to represent map information for two reasons, as shown in Figure 6. First, there is a constant need to identify the roadway segments in order, which is very difficult using only geometric operations. The concept of "connectedness" of sequential road segments

needs to be represented in the database itself. Additionally, for tasks such as perceptual identification of road edges, it is necessary to ask what is "beside" the road so that its color can be identified. Such a query requires that the current road segment have identifiable "sides", with connectivity information for each. CODGER I had no facility for representing such connectivity information.

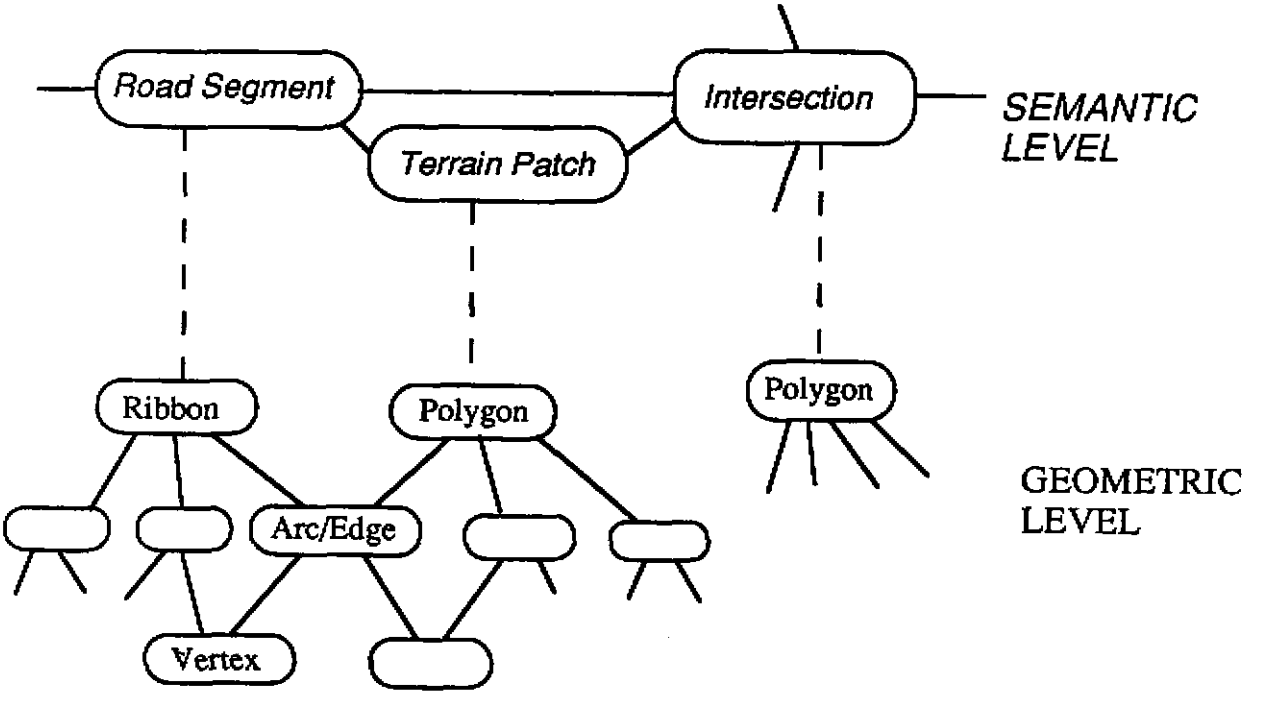



**Figure 7:** The Semantic/Geometric Network in CODGER II

The solution adopted for CODGER II was to implement a combination of semantic and geometric network, as illustrated in Figure 7. In this new representation, there is an upper "semantic" level in which objects are represented symbolically with topological connections. However, there is no actual quantitative geometry at this level of representation. Instead, there is an additional "geometric" level of data objects in which a complete 2D modeling system is implemented. In the geometric level, each object corresponds to a ribbon or polygon, with separate data tokens for each edge and each vertex. Thus, semantic queries such as "what is the next road segment?" can be answered by tracing along the semantic/topological pointers in the database, while metric queries such as "what is the shape of this intersection?" are answered by examining the geometric objects and pointers. Different processing modules may be interested in one or the other, or sometimes both, levels of the system.

The semantic/geometric network works well for representing map information, but it does not address the issues raised by the task of map revision as the vehicle discovers details and corrections to add to the *a priori* map data. Figure 8 shows an example of the problem: the map contains an error in the coordinates of objects A and B. If A and B are stored geometrically in "world coordinates", it is very difficult to decide exactly how to modify those coordinates to reflect the new information. Instead, the solution used in CODGER II is the customary one for geometric modeling systems -- each geometrical entity is assigned its own "intrinsic" coordinate frame, and each geometric link has attached to it a transformation between the intrinsic frames of the objects being linked. With this mechanism, the

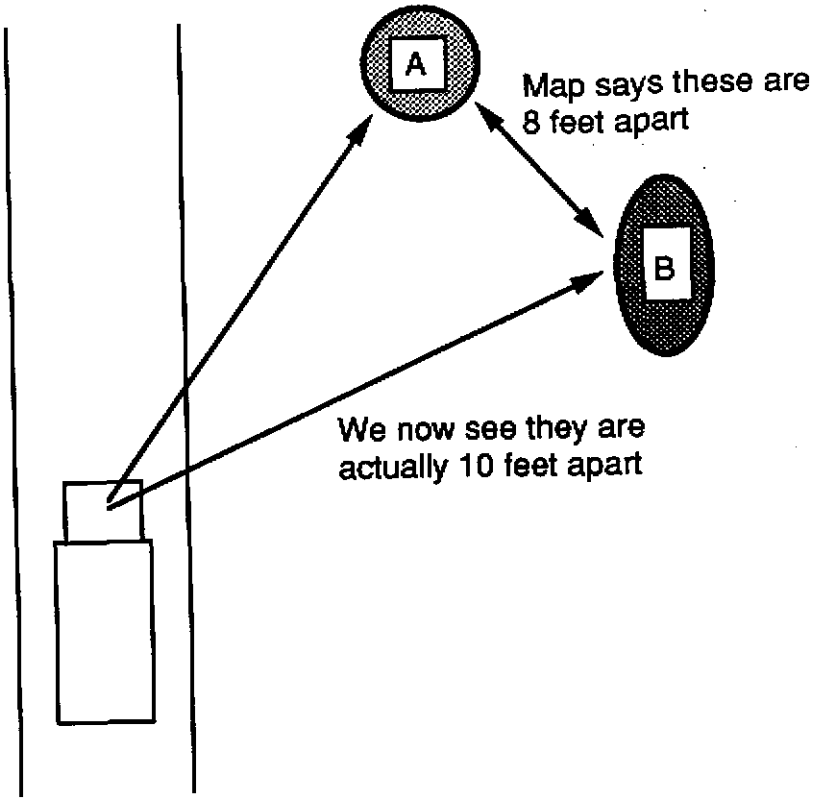



**Figure 8:** Map Revision Requires Local Coordinates

transformation from A to B can be updated as needed. Of course, this may create an inconsistency if A and B are both defined in world coordinates; the inconsistency is handled by uncertainty-modeling techniques described later.

## Time-Varying Transformations in CODGER II

With the new geometric modeling facilities of CODGER II, map information can be stored and revised. However, such facilities are only suitable for a completely static world. When the vehicle moves in the world, there arises a new type of geometric transformation that varies over time -- a dynamic transformation. For example, the vehicle-to-world transformation varies over time, and the vehicle may have a pan-tilt mount whose relationship to the vehicle also varies over time. To deal with time-varying transformations, all transformations should in concept be parameterized by time; thus, rather than asking "what is the distance from A to B?" we should ask "what is the distance from A to B at time T?" To implement this, we introduce the concept of *frame generators.* A frame generator is a function *F(t)* that returns a geometric transform for any given time *t.* Now, each link between geometric objects can have a frame generator attached to it, so that time-varying relationships can be managed.

Several types of frame generators are needed to adequately represent all the necessary relationships in the database. First, there are the truly *time-varying transforms* such as the vehicle-to-world relationship, which we symbolize as *F(t).* However, most objects are stationary in the world and thus the relationship to the world is the same for all times *t.* We call these *constant transforms* and symbolize

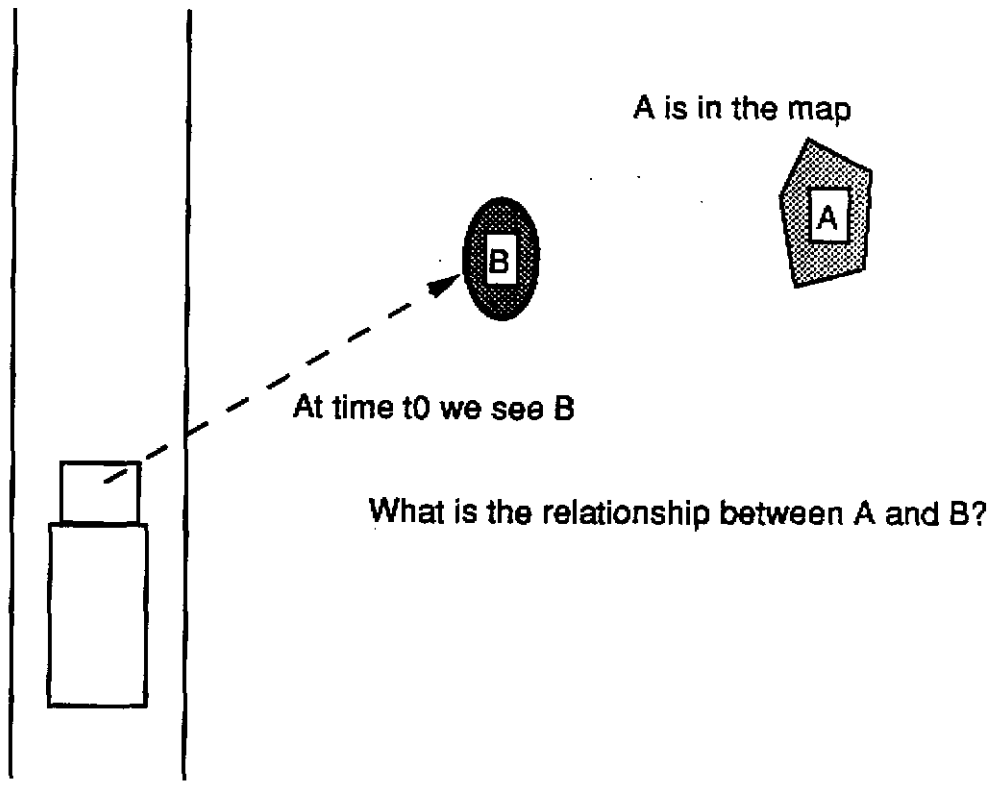



**Figure 9:** Several Types of Time-Varying Relationship

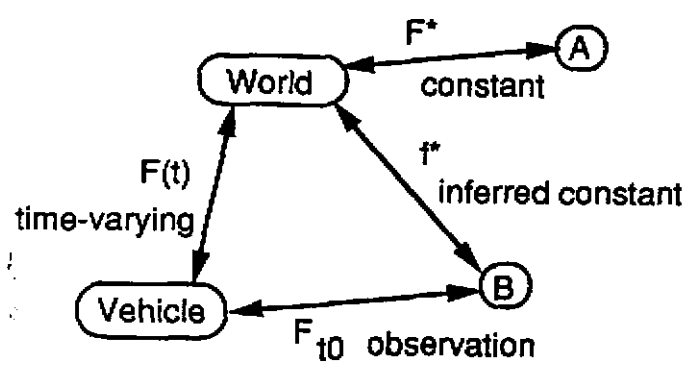



**Figure 10:** Several Types of Frame Generator

them by $F_*$. In Figure 9, object A is stationary and thus has a constant transform to the world coordinate system. In this example, the vehicle observes object B at a specific instant of time $t_0$. We call this an *observation*, and denote its frame generator by $F_{t_0}$. This frame generator can *only* produce an actual transform at the time $t_0$; otherwise, its value is undefined. Finally, although B is detected in sensor data that is relative to vehicle coordinates, we do not believe that B is attached to the vehicle. Instead, we assume it is fixed in the world and infer a frame generator to attach it to world coordinates. This requires a new kind of frame generator that is constant, yet is inferred from observations; we call it an *inferred transform* and denote it by $f_*$. Note that it is determined from the observation and the vehicle-to-world

transform according to $f_* = F_{t_0} F(t_0)$, i.e. the product of the observation of B relative to the vehicle, and the vehicle-to-world transform at that moment $t_0$. The resulting geometric modeling network is shown in Figure 10.

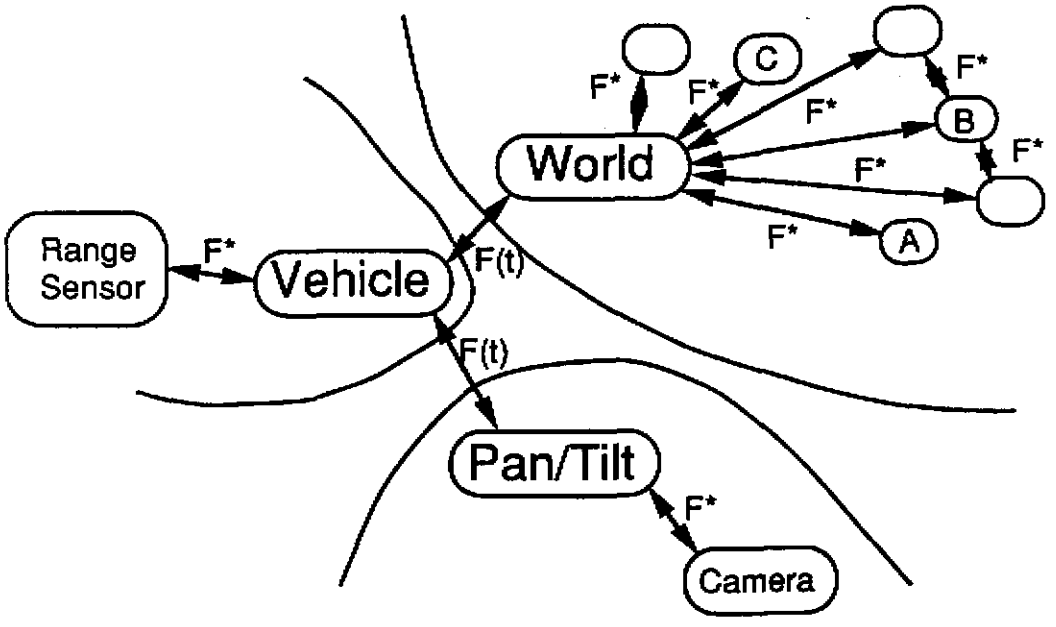



**Figure 11:** Affixment Groups

With this rich set of frame generators, all the important time-varying relationships can be represented. However, there is a danger that the system may degerate into a chaotic spaghetti of geometric relationships, with no clear rules for finding the transform between two arbitrary objects. To eliminate this problem, we have developed the concept of *affixment groups*, which are groups of objects that are assumed to have a constant relationship to each other (Figure 11). We partition all objects into affixment groups of mutually fixed objects; thus, there is one affixment group for the world, containing all objects in the world, and one for the vehicle that includes all vehicle-relative objects. If the vehicle had a pan-tilt mount for a camera, the camera would have its own affixment group. Now, within each affixment group, we create an object called the *affixment object* that simply represents the coordinate frame within which the objects of the affixment group is defined. Each affixment group has a single affixment object, so there is one for the world coordinate frame, one for the vehicle coordinate frame, etc.

Now some simple rules are adopted for the frame generators that link the objects in the database. Every object defined in a coordinate system has a constant frame generator $F_*$ that links it to the corresponding affixment object. Thus, for example, all objects in the map have constant frame generators that specify where they are in world coordinates. So, for any two objects in the world, the transform from one to the other is simply computed from the transforms that link each to the world coordinate frame. No search through the database is required. Where desired, objects within an affixment group may also have constant frame generators that link them directly.

Constant frame generators are not allowed to link objects from different affixment groups, because such objects are assumed to be moving relative to each other. For these calculations, the affixment

objects themselves are linked by a network of time-varying transforms *F(t)*. One of these is the vehicle-to-world transform; another may be the pan/tilt-to-vehicle transform, etc. These time-varying frame generators can produce transforms that depend on time. No other time-varying transforms occur in the database; only those that link affixment objects. Thus, to find the transform between two objects in different affixment groups, at a particular time, one applies the constant transform from each object to its respective affixment object, and the transform between the affixment objects which depends on the time. Again, no search through the database is required to evaluate such relationships.

## Uncertainty Modeling in CODGER II

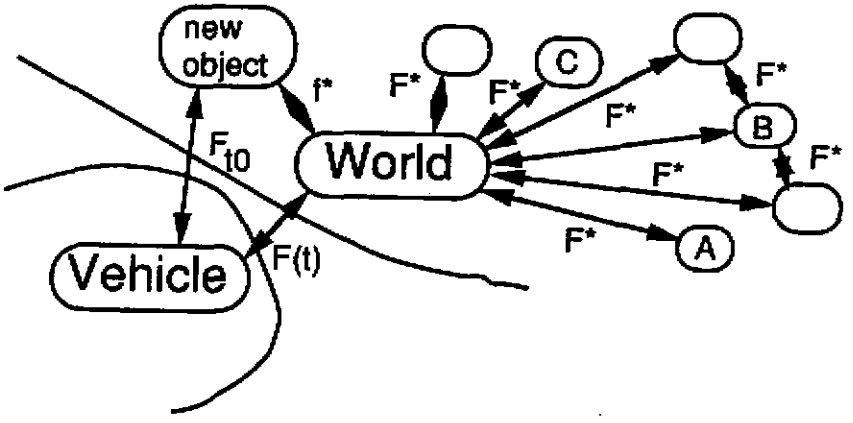



**Figure 12:** Recording an Observation

When an object is read in from a map database, it can be directly attached to its affixment object by a constant transform as described above. However, when it is derived from the robot's perception, a slightly different representation is needed. The situation is illustrated in Figure 12. Here, an object has been seen by the vehicle *but it is assumed to be affixed to the world.* It would be wrong to affix it to the vehicle, because then it would be assumed to move as the vehicle moves. Instead, the object is created within the affixment group of the world, with an observation transform $F_{t_0}$ to relate it to the vehicle coordinate system. To affix the object to the world, an inferred transform $f_*$ must now be created to relate the object to the world coordinate system. This is done by using the observation transform in conjunction with the current value of the vehicle-to-world transform. Thus, while the object is seen by the vehicle, it is stored in relation to the world, using the best estimate of the current vehicle position.

Observations of objects can also form one of the most important sources of information for updating the vehicle-to-world transform, that is, for performing landmark navigation. Such navigation primarily takes the form of correcting for drift and error that has accumulated over time from such other mechanisms including wheel motion encoders, inertial guidance systems, and visual motion analysis. CODGER II includes a complete facility for implementing such navigational updates. The basis for updating is the representation of each geometric transform not only by its value, but also by the covariance matrix that describes the uncertainty with which the value is known. Thus, at all times, geometric uncertainty is recorded throughout the database. In this way, measurements that are slightly in error can be reconciled by weighted averaging of multiple uncertain values.

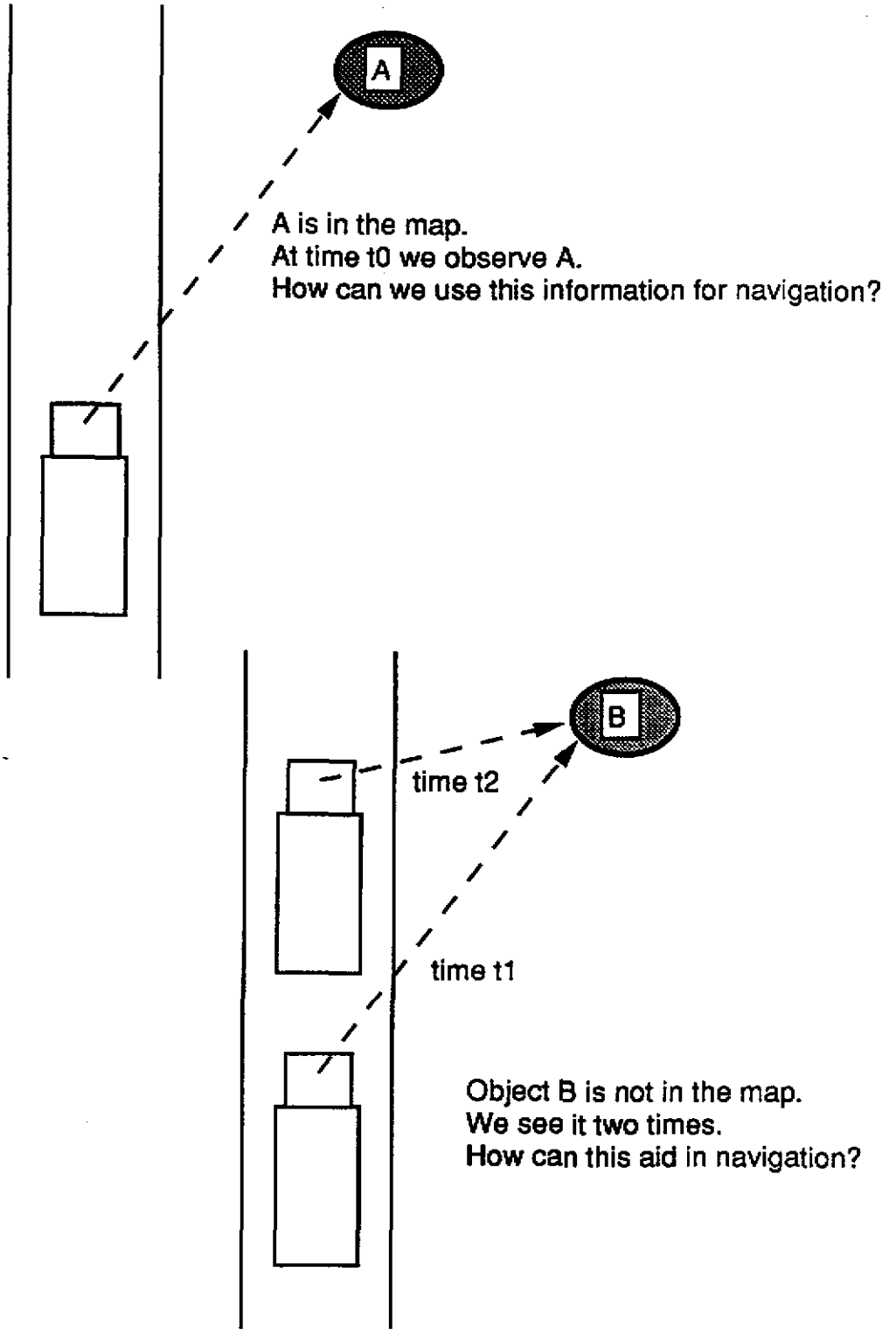
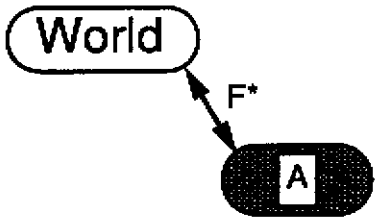


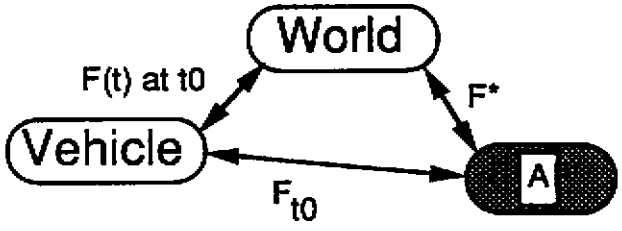

**Figure 13:** Two Scenarios for Landmark Navigation

Two scenarios for landmark navigation are shown in Figure 13. In the first, an object A is known from the map, and the vehicle now observes it. From the map, the representation of Figure 14(a) is created, with a constant transform from the object to world coordinates. Now, when the observation is made, the

(a): Since A is in the map, F* is known from the map data.




(b): When A is seen at t0, a cycle is created from F*, Ft0, and F(t) at t0.




(c): When B is seen at t1, f* is calculated from Ft1 and F(t) at t1.
When B is seen again at t2, f* is recalculated from Ft1, Ft2, and F(t) at t1 and t2

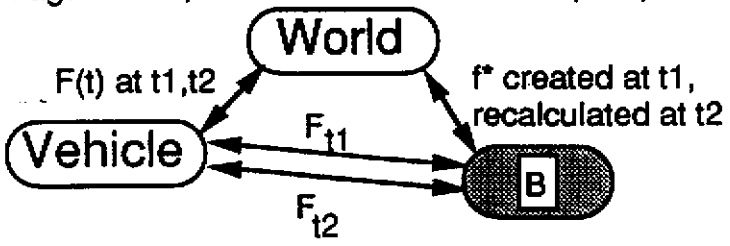



**Figure 14:** Representation of Landmark Observations

situation is shown in Figure 14(b), where a cycle is created between the constant transform from world to object, the observation from object to vehicle, and the vehicle position which is the transform from vehicle to world. Any time there is a cycle in the geometric relationships, inconsistency may arise due to the movement errors cited above and uncertainty in perception. Such cycles of uncertainty can be resolved by classical least-squares methods to yield updated transforms that have optimal values. In this case, the uncertainties that would be weighed against each other are uncertainty in the map data, uncertainty in the perceptual process, and uncertainty in the vehicle position estimate. Most likely, the vehicle position is the least certain; thus, the effect will be to correct the vehicle-to-world mapping at this moment in time.

The second scenario of Figure 13 shows an object that is not in the map, but it is seen twice. The first time it is seen, at time $t_1$, an inferred transform is created to relate it to world coordinates. When it is seen again at time $t_2$, a new observation is obtained as shown in Figure 14(c). This creates a cycle of a different type within the database. First, note that the inferred transform does *not* create a meaningful cycle because it was only the result of the computation from the vehicle-to-world transform and the first observation. However, the second observation and the first observation create a cycle as follows: the observation from the object to the vehicle at time $t_1$, through the vehicle-to-world transform at time $t_1$,

back through the vehicle-to-world transform at time $t_2$, and back to the object through the observation at time $t_2$. This cycle can be resolved to find the new optimal estimate of the inferred location of the object in the world, and also to balance this against the uncertainty in the vehicle-to-world transform itself. Thus, multiple observations of the same object give improved estimates of the vehicle position and motion.

Of course, it is also possible that several observations of an object simply cannot be reconciled consistently with each other and with the estimated vehicle motion. In this case, there is a solid statistical grounds for assuming that the object itself is moving. A new affixment group can be created for that object, and it can now be tracked over time to determine its motion, i.e. the time-varying transform from that object to the world. We have not performed any experiments along these lines, but this at least points the way towards a data representation that can manage information about a dynamic environment.

We may note that each position or motion sensor, as well as the landmark navigation processes just described, all produce "snapshot" estimates of the vehicle motion or the vehicle-to-world transformation. These estimates themselves can be kept in a network, which will have many cycles; the classical algorithms can then be used to provide a least-squares estimate for the entire history of the vehicle motion. In this way, the current vehicle position estimate can be kept continuously up-to-date, and the estimated history of vehicle travel will be smooth. Whenever a new, highly confident estimate is made, such as the sighting of an important landmark, the vehicle's entire estimated history of travel will be smoothly updated instead of producing an instantaneous "jump" to a new position.

# Section IV

# Experiments With the Driving Pipeline

## Introduction

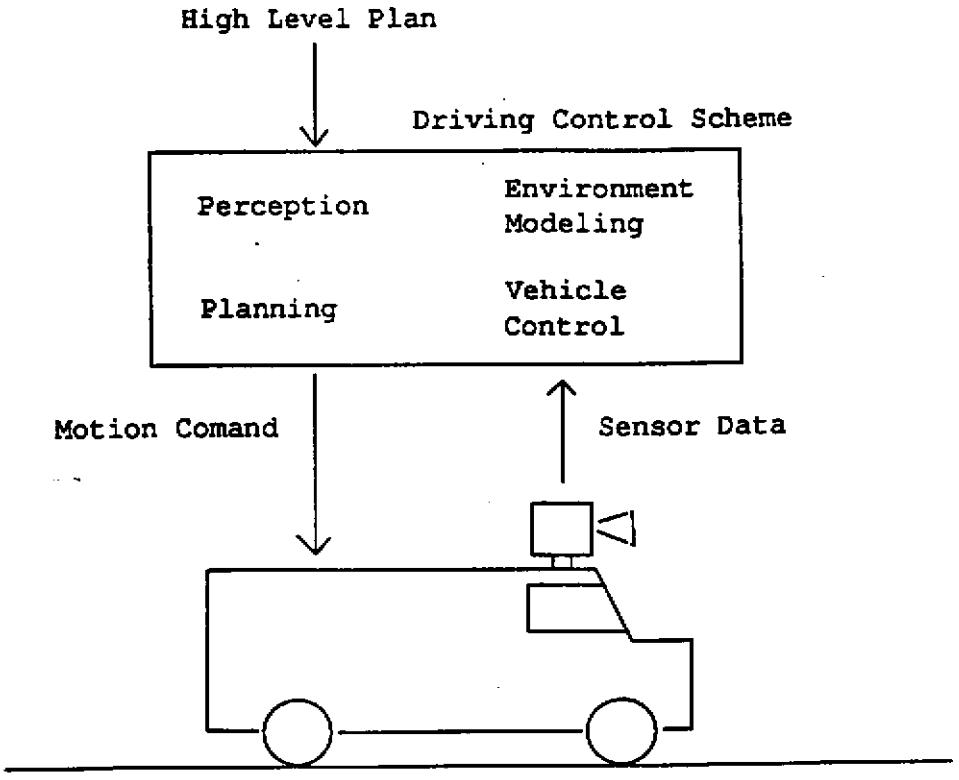



**Figure 15:** Driving Control Scheme

This paper describes a driving control scheme for a mobile robot that drives the robot vehicle outdoors, avoiding obstacles, and keeping the vehicle within a navigable area. As illustrated by Figure 15, the driving control scheme takes a high-level navigation plan from planning modules and sensor data from sensors, and generates vehicle motion commands, performing the necessary computations including perception, environment modeling, path planning, and vehicle control. We have developed a scheme for the coordination of these tasks, which we call the *Driving Pipeline*. This paper describes the Driving Pipeline, the various processes that it coordinates, and the experiments in which the Driving Pipeline has been successfully used for building mobile robot systems.

Our objective is to build an autonomous mobile robot working in the real world in real-time, so we adopted the following design goals:

- **Flexibility:** Other systems have been developed that perform a single navigation task well; however, these systems are not easily extended to handle a broad range of tasks.

- **Continuous Vehicle Motion:** Continuous motion is more desirable than stop-and-go motion, because it produces higher vehicle speeds and smoother control.
- **Adaptive Control:** Driving control must be adaptive to the environment and to the internal condition of the robot vehicle. For example, the vehicle should be able to drive faster using less sensor data on a flat broad ground than on a winding narrow road. The driving control scheme must adjust its computation and maintain effective coordination among numerous perception and planning processes.
- **Parallel Execution:** For real-time motion, driving control requires a large amount of computation in a variety of different procedures. For this end, parallel computing is the most practical solution. In addition to small-grain parallelism such as parallel machines for signal data processing, large-grain parallelism can be used to coordinate the various tasks involved in driving. Parallel computing can take advantage of two kinds of parallelism: parallelism in processing steps and parallelism in data to be processed.

In order to achieve these goals, we developed the **Driving Pipeline**. A *pipeline* is a form of parallelism in which the computation is decomposed into a sequence of processing steps, called *stages*, to be executed in a fixed order. Typically, each stage is a separate processor receiving input data from the previous stage and providing output data to the next stage. A stage commences execution whenever data arrives from the input. A pipeline is used for performing the same computation over a number of different data sets. Since the pipeline can begin processing a second data set before the first has finished, the stages run in parallel. The pipeline processes data sets at the rate of one per *cycle time*. The cycle time is the longest stage time. The total time required to process a given data set (called the *job time*) is the sum of individual stage times. The construction of our pipeline is based on two key ideas:

- **The Driving Unit:** We divide the area in which the vehicle navigates (road, hillside, etc.) into a sequence of small areas called *driving units* so that it can process each driving unit separately. Each processing module for perception and planning will operate successively on each driving unit in turn.
- **Execution Pipeline:** The Driving Pipeline allocates the primitive processing steps along a pipeline so each one can work independently, receiving input data from the previous processing step and passing data to the following processing step.

These two key ideas enable the pipelined execution of the primitive processing steps on the sequence of driving units, which provides enough throughput to allow continuous vehicle motion. As the vehicle encounters changes in the road configuration, it can place driving units with different sizes and intervals by adjusting the sensor view frames, execution intervals, and vehicle speed.

Although several mobile robot systems have been built in the past, they did not address driving control scheme very deeply. Stop-and-go motion, although it does incorporate all of the primitive processing steps, deliberately avoids the problem of continuous motion control [2, 4, 7, 11]. Waxman et al. mentioned the necessity for vehicle speed adjustment using knowledge, but didn't show any method for doing so [12]. Brooks developed a layered control structure that drives a vehicle continuously [1]. However, it does not have the ability to adapt the control to meet the changing needs of perception. Dickmanns and Zapp develped a system for high-speed navigation on the German Autobahn [3]. This system tracks simple visual features (e.g., white lines bordering the road) and cannot be easily extended to handle more difficult perceptual scenarios.

To solve these problems, we have developed the concept of the Driving Pipeline and verified it in two experimental mobile robot systems: the Terregator and the NAVLAB. This paper describes the Driving

Pipeline, including the component concepts of the Driving Unit and the Execution Pipeline, and describes our experiments with these vehicles.

## Processing Steps and the Driving Unit

We divide the computation necessary for driving control into the following primitive processing steps:

- The **Prediction** step plans the area that the vehicle will move into next.
- The **Perception** step detects navigable area boundaries and obstacles using sensor data.
- The **Environment Modeling** step makes a description of the vehicle environment and updates the estimate of the vehicle position.
- The **Local Path Planning** step plans the vehicle trajectory.
- The **Vehicle Control** step drives the vehicle mechanism.

These steps must each execute in turn to process each area of terrain that the vehicle will traverse.

We developed the concept of the *driving unit* to indicate the area that each primitive step will process once in each execution cycle. The vehicle's entire route is divided into driving units which are passed, one at a time, to each of the primitive processing steps. In this way, planning and perception are synchronized to provide driving control.

### Prediction and the Driving Unit

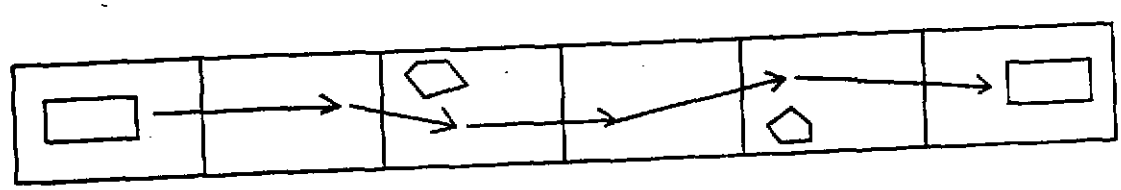


**Figure 16:** Sequence of Driving Units

The Prediction step works as the manager of the Driving Pipeline. It receives the high-level plan from the map navigation level of the system, predicts the next chunk of area into which the robot vehicle should move, and indicates it by defining a new driving unit. Because the driving units are placed in the order that the vehicle travels, the sequence of driving units forms the vehicle passage, which outlines the planned path of the vehicle (Figure 16).

The parameters for placing the driving units are:

- location of the driving unit;
- type of the driving unit : such as *on-road, open-terrain*;
- size of the driving unit : the width and length of the driving unit;
- interval of driving units : the distance between the centers of consequtive driving units along the vehicle trajectory.

The driving unit location is determined based on the high-level plan derived from the navigational map, combined with the vehicle's current position estimate. The type of driving unit can be road or intersection,

depending also on the map and the vehicle position. The factors that determine the size and the interval area are discussed in the following sections.

## Perception and the Driving Unit

The Perception step scans a driving unit with sensors to determine the key objects within it. Perception results will be used by the Environment Modeling step both for determining navigable areas and for updating the vehicle position estimate.

Two parameters, the driving unit and a *scanning position*, direct the Perception step. The driving unit, which is given by the Prediction step, indicates the area that the Perception should see. Because sensor data must cover the driving unit, the sizes of sensor view frames give the upper limit of the driving unit sizes.

The scanning position is the position at which the Perception step should scan the driving unit. Two factors determine the scanning position: the required accuracy of the visual measurement, and the need for specific vehicle position information. The required accuracy of the visual measurement is important because of the reduced accuracy as distance increases. Thus, the vehicle should be close enough to the driving unit to satisfy the accuracy needs of the Environmental Modeling step. The need for specific vehicle position information also constrains the scanning position. The vehicle position estimation is updated with both the perceptual results and dead reckoning from the control system. In general, the perception result gives a more accurate vehicle position estimate. The vehicle position estimated with the perception result will, of course, be a scanning position. Therefore, when the mobile robot system needs an accurate vehicle position estimation at a specific position, this position should be the scanning position.

Once the driving unit and the scanning position are determined, the Perception step can calculate the sensor view frame relative to the vehicle and aim the sensors. This enables Perception to aim the sensors adaptively.

## Environment Modeling and the Driving Unit

By analyzing the perception results, the Environment Modeling step produces an environment description that indicates a navigable area from the current vehicle position toward the end of the last scanned driving unit.

The Environment Modeling step also updates the vehicle position estimation. Because the vehicle is traveling continuously and the scanning positions are discrete, the Modeling step merges the perception result and the dead reckoning updates to estimate the vehicle positions between the scanning positions and beyond the last scanning position.

## Local Path Planning and the Driving Unit

The Local Path Planning step determines the physical vehicle trajectory within the navigable area determined by the Modeling step, from the current vehicle position to the end of the last scanned driving unit.

As shown in Figure 17, the local path plan restricts the minimum size of a driving unit, because the driving unit must be large enough to allow the vehicle to manuever and avoid obstacles.

The Driving Pipeline includes two levels of path planning: the driving passage from the Prediction step

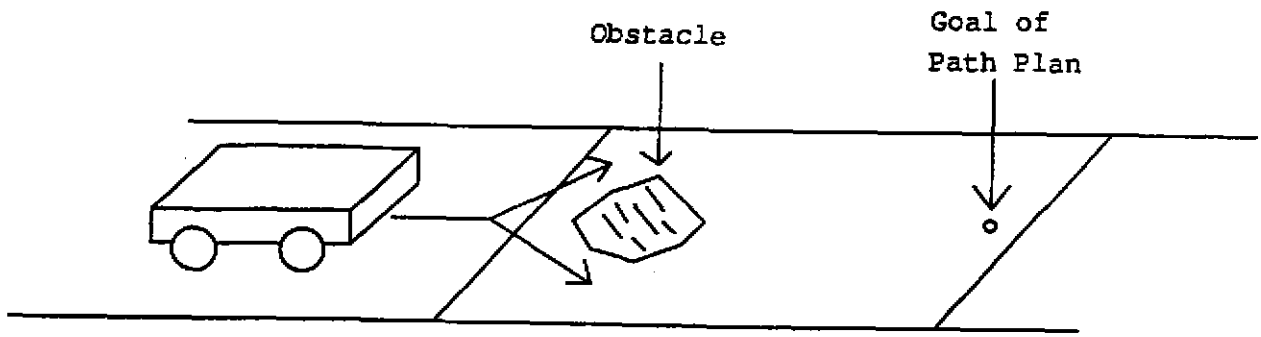



**Figure 17:** Driving Unit Size for Vehicle Maneuvering

and the trajectory from the Local Path Planning step. If the map database is complete, the driving passage can be planned before navigation by consulting the map data. If not, it is determined gradually based on perception results from the previous driving units. This is the reason why we include planning the vehicle passage in the Driving Pipeline level of the system rather than in a higher level.

### Vehicle Control and the Driving Unit

The Vehicle Control step drives the physical vehicle. It generates a set of motion commands for the vehicle mechanism from the trajectory plan given by the Local Path Planning step. Because the trajectory plan ends at the far edge of the last scanned driving unit, the vehicle never moves into an unscanned area. Also, this step adjusts the vehicle speed to be optimal unless the Local Path Planning step gives commands on speeds (such as stopping at a specific place). The details will be described below.

## Continuous Motion, Adaptive Control, and the Driving Pipeline

The simplest control structure for implementing the Driving Unit concept would be for the vehicle to stop at the end of each driving unit, process the next one through each of the primitive steps, then drive across the next driving unit and stop, repeating this cycle over and over. This paradigm is known as the "stop-and-go" model of vehicle control, and it produces very jerky motion as well as being far below the optimum vehicle speed. To remedy these problems, we apply the concept of pipelined execution of the primitive steps to form the Driving Pipeline.

### Pipelined Execution for Continuous Motion

In order to drive the robot vehicle continuously, the Vehicle Control step should work on one driving unit after another without stopping the vehicle. To accomplish this, the Prediction step, the Perception step, the Modeling step, and the Local Path Planning step must have finished processing the next driving unit before the Vehicle Control step finishes the current driving unit. This is the reason that continuous vehicle motion needs a Driving Pipeline to process multiple driving units in parallel.

The Driving Pipeline supports continuous vehicle motion by using *pipelined execution*. As described above, the processing steps are allocated along the pipeline, and the Driving Pipeline executes the processing steps in parallel by passing a sequence of the driving units through this pipeline. Figure 18 illustrates the pipeline execution of the Driving Pipeline as follows:

1. When the vehicle is on Driving Unit 1, the Prediction step places a new prediction for Driving Unit 4.

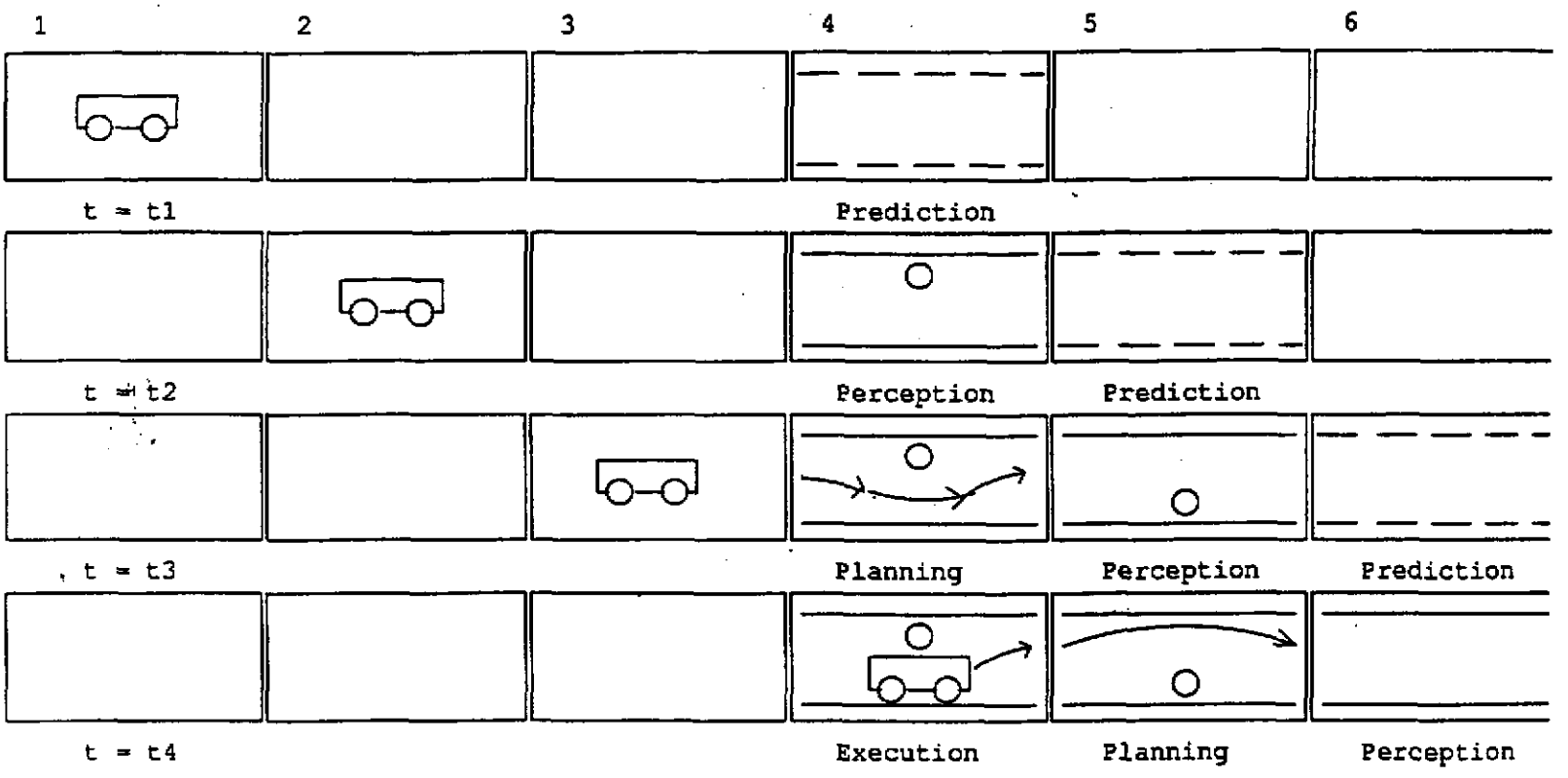



**Figure 18:** *Pipelined Execution of the Driving Pipeline*

2. When the vehicle is on Driving Unit 2, the Perception step works on Driving Unit 4. At the same time, the Prediction step places the next driving unit, Driving Unit 5.
3. When the vehicle is on Driving Unit 3, the Modeling step determines the vehicle passage and the Local Path Planning step plans the path to the end of Driving Unit 4. In parallel, the Prediction step defines Driving Unit 6 and the Perception step works on Driving Unit 5.
4. When the the Vehicle control step drives the vehicle on Driving Unit 4, the Prediction step is defining Driving Unit 7, Perception is working on Driving Unit 6, and the Modeling and the Local Path Planning step are working on Driving Unit 5.

Several key features of the Driving Pipeline make the pipelined execution possible. First is the concept of the driving unit, which is critical because it allows the route ahead of the vehicle to be partitioned into individual units for processing by the successive steps. Because each driving unit specifies an area on which one processing step works, the Driving Pipeline may assign the different processing steps to different areas along the vehicle passage.

The second is the constant flow of the driving units through the processing steps in a prearranged sequence. Each driving unit is created at the Prediction step and is passed through the following steps from one step to the next step ending with the Vehicle Control Step, thus forming the data flow through the processing steps. This flow is always one way and in the same direction; no driving unit skips any processing step or goes back to the previous steps. Therefore, the order of execution of the primitive processing steps can be "hard-wired" into the system without the need for symbolic reasoning to decide what to do next.

The third necessary feature is the independent computation of the processing steps. The computation for driving control is divided into processing steps in such a way that each processing step performs a different function. Each step requires as input only the outputs of the previous steps. Therefore, each step can only work on a driving unit after the previous steps have completed their processing on that driving unit.

The fourth feature is the order of the driving units themselves. Since the driving units are created as the vehicle travels and are placed along the vehicle passage, the order of their generation is always the same as the order in which they are processed by the processing steps. Therefore, the Driving Pipeline can feed the driving units to the processing steps continuously.

Finally, the ability of the sensors to look ahead of the vehicle more than one driving unit's distance is necessary. This permits Perception to be working at a distance beyond the next driving unit. This ultimately limits the distance over which pipelining can be effective.

The existence of all of these features allows pipelined execution in both of the necessary aspects, the processing and the data. The name "Driving Pipeline" comes from the pipeline of processing steps, the sequence of driving units, and the pipelined execution. The following sections provide a more detailed examination of the pipelined execution.

## Execution Intervals of the Driving Pipeline

The "execution interval" of the driving control system refers to how often the mobile robot system executes the cycle of the primitive processing steps. Adjusting the execution interval to be optimal is essential for an autonomous mobile robot system, because the necessary execution intervals depend on driving conditions such as the width, flatness, and curvature of the road. Execution intervals that are too long may cause unstable vehicle motion, because the vehicle position and the path plan are updated only once in each interval. On the other hand, execution intervals that are too short consume unnecessary computation and slow down the vehicle speed because the amount of computation in each interval is roughly constant.

To provide the optimal vehicle speed control, the driving control scheme needs a way to compute and change the execution intervals. In the Driving Pipeline the sizes of the consecutive driving units determine the execution intervals, because each execution cycle works on one driving unit and the number of driving units per unit trajectory length is equal of the number of the execution cycles. Therefore, the Driving Pipeline is able to adjust the execution intervals by changing the driving unit intervals.

If the vehicle could be controlled to exactly follow the planned path, the driving units could be made as long as the range of the effective field of view of the sensors. Unfortunately, the actual vehicle trajectory may differ from the local path plan because of many reasons, particularly the error in the control mechanism and the inaccuracy of dead reckoning. The cumulative error in the control of vehicle motion and the allowed error tolerance in the vehicle position are the factors used to determine the driving unit intervals.

The error in the vehicle position and direction, which grows as the vehicle travels, must be canceled by the execution of the driving pipeline before it surpasses an error tolerance. Therefore, if the accumulated error increases very rapidly, the intervals of the driving pipeline must be shorter. If the accumulated error

increases slowly, they can be longer. For example, because errors in the vehicle direction can produce a larger accumulated error in the vehicle position than errors in the vehicle displacement, the interval must be shorter in turning than in moving straight.

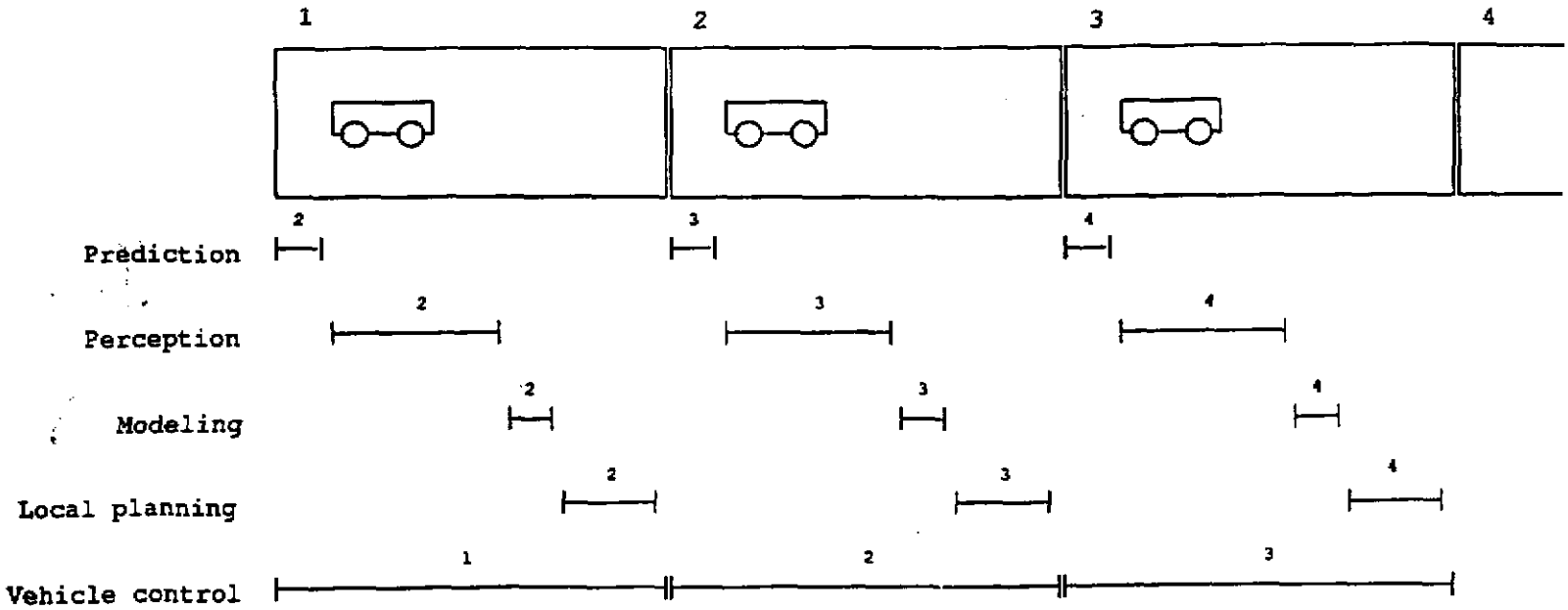



**Figure 19:** Badly-Balanced Execution of the Driving Pipeline

As mentioned above, vehicle maneuverability restricts the minimum size of a driving unit. If a driving unit interval is shorter than a driving unit length, adjacent driving units overlap.

## Parallelism in the Driving Pipeline

Although the pipelined execution allows the processing steps to work in parallel, it does not ensure a high degree of parallelism. Figure 19 illustrates an extreme example in which parallel execution is not well maintained. In this figure, the vehicle speed is too high. This brings the vehicle to the end of the local path plan before the next plan is produced by the Local Path Planning step. The vehicle then has to stop at the end of the current driving unit to wait for the new path plan to be completed. In this example, the Prediction step, the Perception step, the Environment Modeling step, and the Local Path Plan step must work serially without any parallelism. In this section and the next we discuss the parallelism in the Driving Pipeline and a mechanism for keeping it high. This section discusses parallel execution among the Prediction, Perception, Environment Modeling, and Local Path Planning steps. The next section discusses parallelism between these steps and the Vehicle Control step.

The Prediction, Perception, Environment Modeling, and Local Path Planning steps generally work on each driving unit sequentially, with their execution times overlapping each other on consecutive driving units due to the execution pipeline. However, the parallelism among these steps depends on whether or not there exists a sufficiently rich map database. When such a map exists, we call this the *map navigation mode*; if not, the vehicle drives in the *map building mode*. The timing of the start of pipelined execution varies in these these two modes. In the map navigation mode, the map database can offer enough information so that the Prediction step is able to place a new driving unit without using the perception results from the preceding driving unit, relying instead on the map database and the

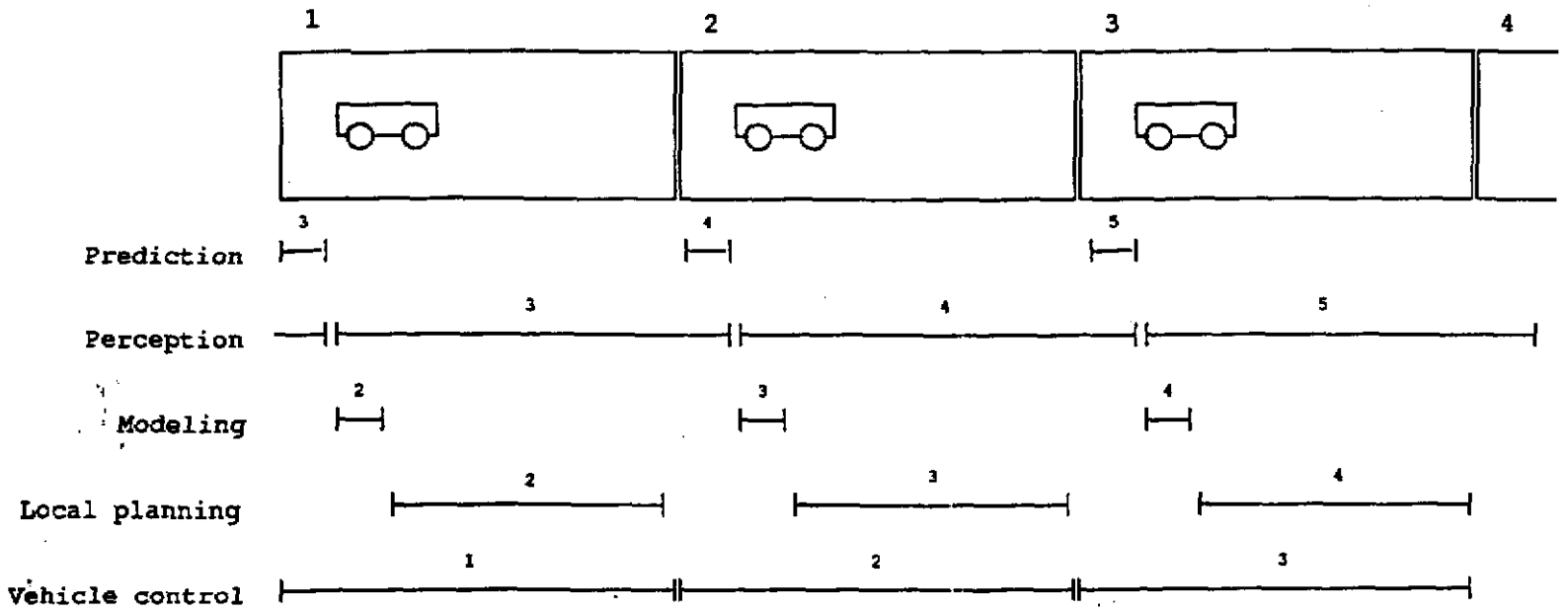



**Figure 20:** Parallel Execution Pattern in the Map Navigation Mode

perception results from earlier driving units. Therefore, the Prediction step can work on the next driving unit before the Perception and the Environment Modeling steps finish the current driving unit. This produces the execution pattern illustrated in Figure 20. In this case, since all processing steps are ready to work on the next driving unit just after finishing the current one, complete pipelined execution is achieved.

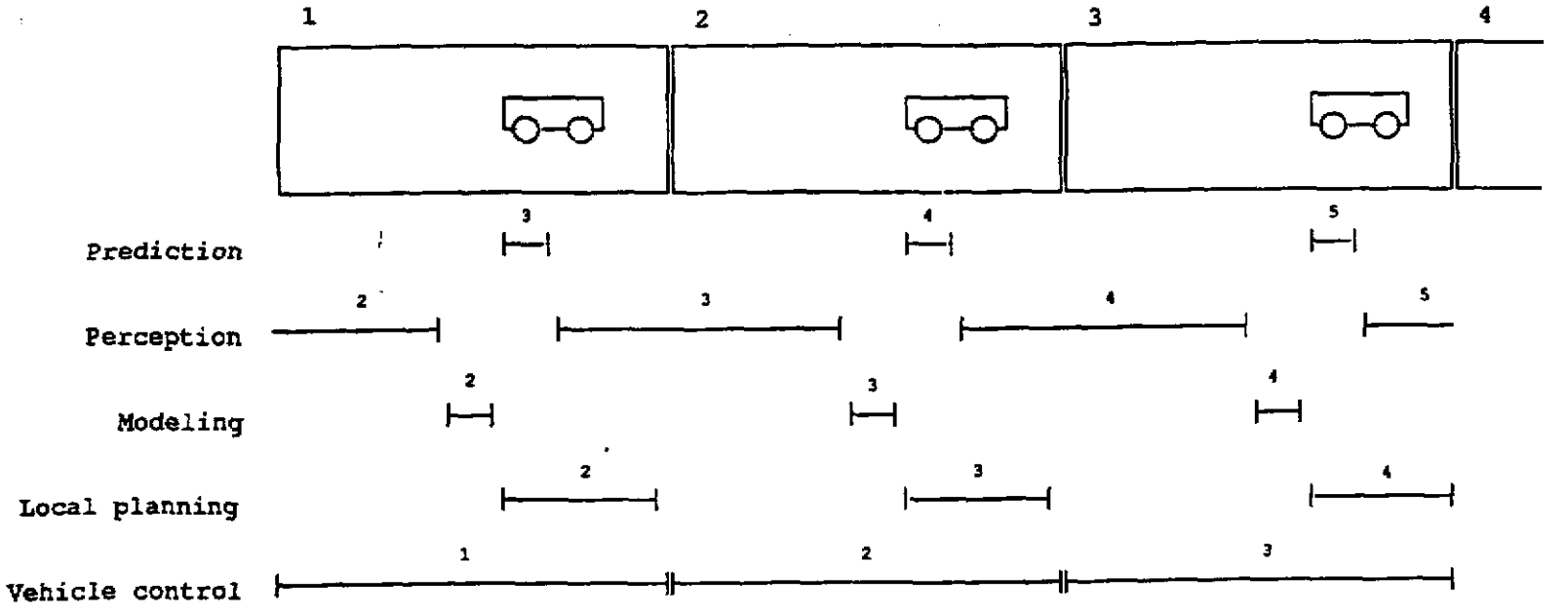



**Figure 21:** Parallel Execution Pattern in the Map Building Mode

In the map building mode, the map database does not have enough information about the unscanned areas, so the Prediction step needs the perception result on the current driving unit in order to place the next driving unit. In this case, the Prediction step has to wait until the Perception step and the Environment Modeling step finish the current driving unit. The resulting execution pattern is illustrated in Figure 21. Consecutive execution cycles overlap less in the map building mode than the map navigation mode.

The difference between the map navigation and map building modes explains one reason that a rich map database results in a higher vehicle speed than the poor map database. In addition, a rich map database allows perception to potentially be faster and more accurate, thus reducing the processing time and/or allowing larger driving units.

In both execution modes, the scanning position is a key factor in maintaining these parallel execution patterns because it regulates the execution patterns. The Environment Modeling step, the Local Path Plan step, and the Vehicle Control step start just after the previous step finishes. The Prediction step starts just after the Perception step finishes in the map building mode, and may start any time in the map navigation mode. So, all of these steps can start at a time independent of the actual vehicle progress. On the other hand, the Perception step can start working only when the vehicle reaches the desired scanning position. The scanning positions that produce the highest parallelism, illustrated in Figures 20 and 21, are given by the following equation:

$$scanning\ distance = \frac{T_p}{T_c} \cdot L_i \tag{1}$$

where

$L_i$ = *driving unit interval*

$T_p$ = *total job time of Perception, Environment Modeling and Path Planning*

$T_c$ = *cycle time of Driving Pipeline*

In this equation, the "scanning distance" is the distance from the scanning position to the driving unit to be scanned. The "cycle time" is the time between consecutive execution cycles, which is the time taken for the vehicle to travel one driving unit. In the map navigation mode, the cycle time is determined as:

$$T_c = T_m \tag{2}$$

whereas in the map building mode, the cycle time is:

$$T_c = Max\ (T_m, T_t) \tag{3}$$

where

$T_m$ = *job time of the most time consuming step*

$T_t$ = *total job time of Prediction, Perception and Environment Modeling*

In the map navigation mode, if the most time consuming processing step works in the whole cycle time, the execution pattern will be the most condensed and will exhibit the highest degree of parallelism. In this execution pattern, the Perception, Environment Modeling, and Local Path Planning steps must work after the vehicle passes the scanning position. That is the derivation of the above equation for the map navigation mode. In the map building mode, the processing for the sequence of the Prediction, Perception, and Modeling steps can not overlap with the processing of this sequence for consecutive

driving units. Therefore, this execution sequence behaves like one individual processing step. That is the reason for the above equation for the map building mode.

## Vehicle Speed and Driving Pipeline

The Vehicle Control step must take into account the execution time of all the processing steps in order to achieve the optimum vehicle speed. Too high a vehicle speed requires the vehicle to stop at the end of each driving unit, as described in the previous section. In this section, we discuss the highest possible vehicle speed and the method to achieve it.

Because the distance that the vehicle moves in one cycle time is equal to the interval of the driving unit, the highest vehicle speed is described by the following equation:

$$vehicle\ speed < \frac{L_i}{T_c} \tag{4}$$

The maximum vehicle speed is less than the driving unit interval divided by the cycle time because distance must be allocated for decelerating the vehicle in the event that some stage of the pipeline requires more time than expected.

If the scanning position is adjusted as described above, the cycle time is given by Equations 2 and 3. Then the above equation can be rewritten as follows:

in the map navigation mode,

$$vehicle\ speed = \frac{L_i}{T_m} \tag{5}$$

and in the map building mode,

$$vehicle\ speed = \frac{L_i}{Max(T_m, T_l)} \tag{6}$$

These equations are based on the highest degree of parallelism among the processing steps and therefore give the highest achievable vehicle speed.

The vehicle speeds given by these equations are possible only when the scanning position is optimally adjusted. The scanning position, however, may be determined by other factors as described previously. For example, the scanning distance may be shorter than the distance given by Equation 1 because the Perception step requires a closer distance for more accurate measurement. If the scanning distance is shorter than the distance given by Equation 1, the speed of the Driving Pipeline is given by the following equation:

$$vehicle\ speed = \frac{D_s}{T_p} \tag{7}$$

where

$D_s$ = *scanning distance*

These equations (Equation 4 - 7) describing the vehicle speeds explain the following vehicle behavior patterns, which demonstrate the adaptive control capabilities of the Driving Pipeline:

- The most time consuming processing step limits the highest vehicle speed. The Driving Pipeline is capable of adjusting the vehicle speed to be as high as the processing times will allow.
- Longer driving unit intervals produce a higher vehicle speed. If the robot vehicle drives in easy driving conditions such as a broad, flat, straight road, then the Prediction step may define driving units with large intervals. The vehicle speed will then be adjusted to be higher.
- Likewise, shorter scanning distances produces a slower vehicle speed. If the Perception step has to look at objects from a closer distance, the vehicle slows down. This behavior is similar to a human driver looking around carefully.

These behaviors need not be explicitly programmed into the system. They arise naturally as a result of the operation of the Driving Pipeline and the calculation of each driving unit interval based on the geometry of the road, the vehicle, and the sensor field of view.

Although Equations 4~ 7 assume that each processing step always requires a constant execution time, the actual requirements may vary from time to time and place to place. In such a case, the Driving Pipeline calculates the vehicle speed with the following equation, which is a modified version of Equation 7:

$$vehicle\ speed = \frac{D_r}{T_r} \tag{8}$$

$D_r$ = *remaining distance of local path plan*

$T_r$ = *remaining job time*

In this equation, $D_r$ is the distance from the current vehicle position to the end of the path plan in the *current* driving unit, and $T_r$ is an estimate of the total remaining execution time for the Prediction, Perception, Modeling, and Local Path Planning steps working on the *next* driving unit. The initial value of $T_r$ is a predicted execution time for these processing steps. Whenever these processing steps finish processing a driving unit, $T_r$ and $D_r$ are recalculated and the vehicle speed is updated. This allows the vehicle speed to adaptively respond to the changing requirements for its own computation time.

## The Driving Pipeline In Action: Experimental Results

### Implementing the Driving Pipeline

We have developed and tested the Driving Pipeline through building several experimental mobile robot systems, called *Sidewalk System 2*, *Sidewalk System 3*, and the *Park System*, [5] [6] [10]. Sidewalk System 2 and Sidewalk System 3 drive an experimental vehicle called the Terregator on the network of sidewalks on the campus of Carnegie Mellon University. The Park System drives the NAVLAB, a computer-controlled van, on a road in Schenley Park adjacent to Carnegie Mellon. Figure 22 shows these vehicles, which are both equipped with color TV cameras and a laser range scanner made by ERIM. While the Terregator is linked to several SUN-3 workstations in the laboratory with radio

**Figure 22:** Terregator and Navlab

communication and cables, the NAVLAB carries four SUN-3s on board. In the remainder of this chapter, we will describe primarily Sidewalk System 3 because it demonstrates the Driving Pipeline most clearly.

Figure 23 shows the module structure of Sidewalk System 3. The processing steps are implemented as individual programs and are linked through the CODGER distributed database, a system-building tool written at Carnegie Mellon to support large-grain parallelism for mobile robot navigation [9]. CODGER makes it relatively easy to build the Driving Pipeline because of its capability to support parallel processing among multiple computers. All of the systems mentioned above use CODGER in this way.

## Processing Steps and Driving Units

Figure 24 shows a diagram of the primitive processing steps working on one driving unit in approaching an intersection. Figure 24(a) shows the driving unit placed by the Prediction step. In Figure 24(b), the trapezoid is the sensor view frame aimed by the Perception step to cover the driving unit. Figure 24(c) shows the vehicle position estimated by the Modeling step. The Vehicle Control step drove the vehicle as illustrated in Figure 24(d).

## Pipeline Execution and Parallelism

Figure 25 is a recorded timing diagram of the processing steps. The bars in the figure indicate the time during which each step is processing a driving unit. The driving unit number appears next to the bar. Because Sidewalk System 3 has a complete pre-stored map database, the Prediction step does not need to wait for the Perception step to place a new driving unit and the consecutive pipeline executions overlap

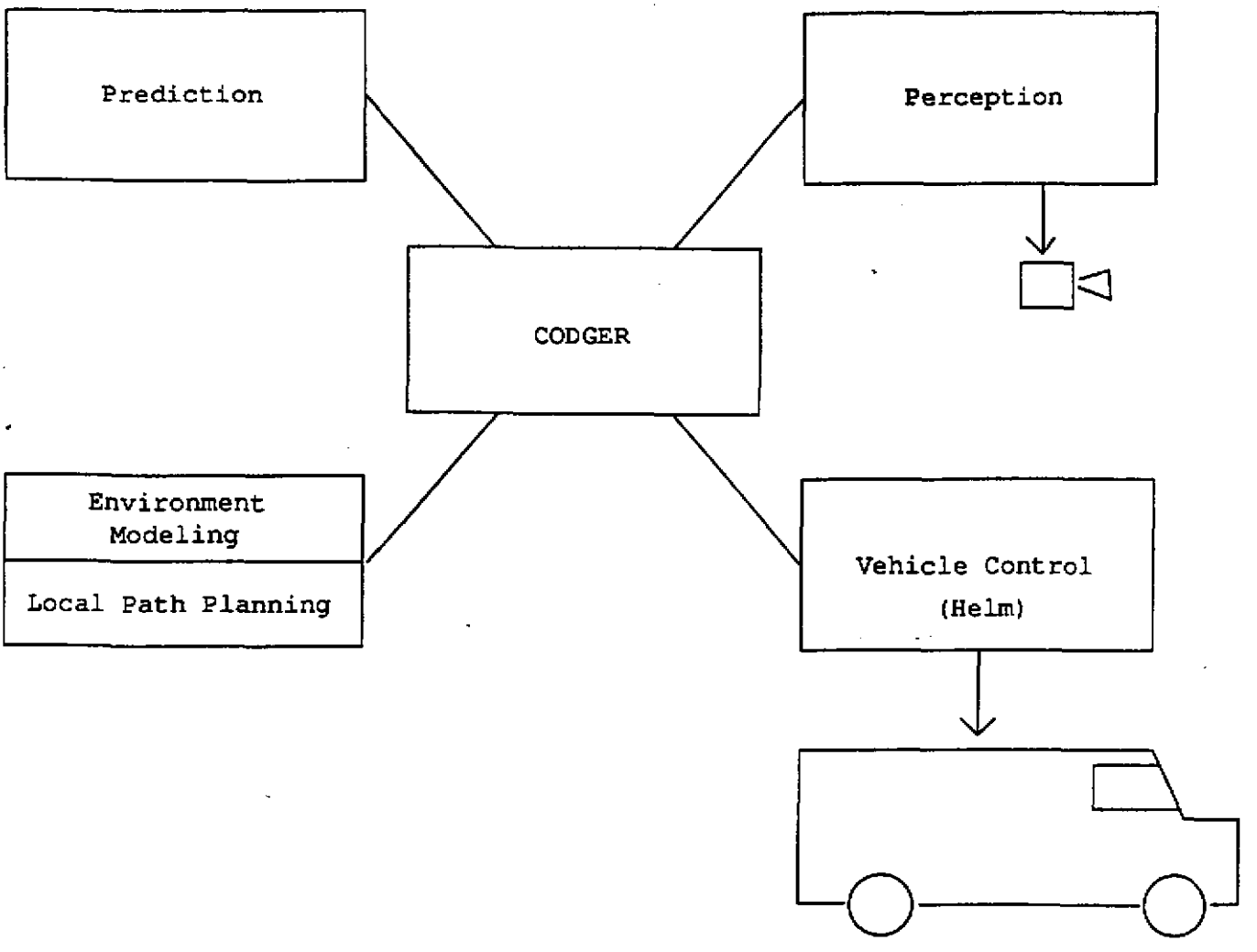



**Figure 23:** Module Structure

completely. This is the "map navigation" mode described above. Because the scanning position and the vehicle speed were adjusted as described above, the most time consuming step (Perception) was the limiting factor in the cycle time of the system.

## Execution Intervals

Because turning at intersections requires more accurate vehicle position estimation than following sidewalks, and because the Terregator vehicle makes larger dead reckoning errors in turning than in straight motion, the Prediction step uses a shorter driving unit interval while the vehicle is turning. Figure 26 shows the driving unit intervals around the intersection and the straight sidewalks. On the other hand, Sidewalk System 2 used constant driving unit intervals and had unstable turning because of the large dead reckoning error. Sidewalk System 3, however, did not have such unstable motion thanks to the adjustment of the driving unit intervals.

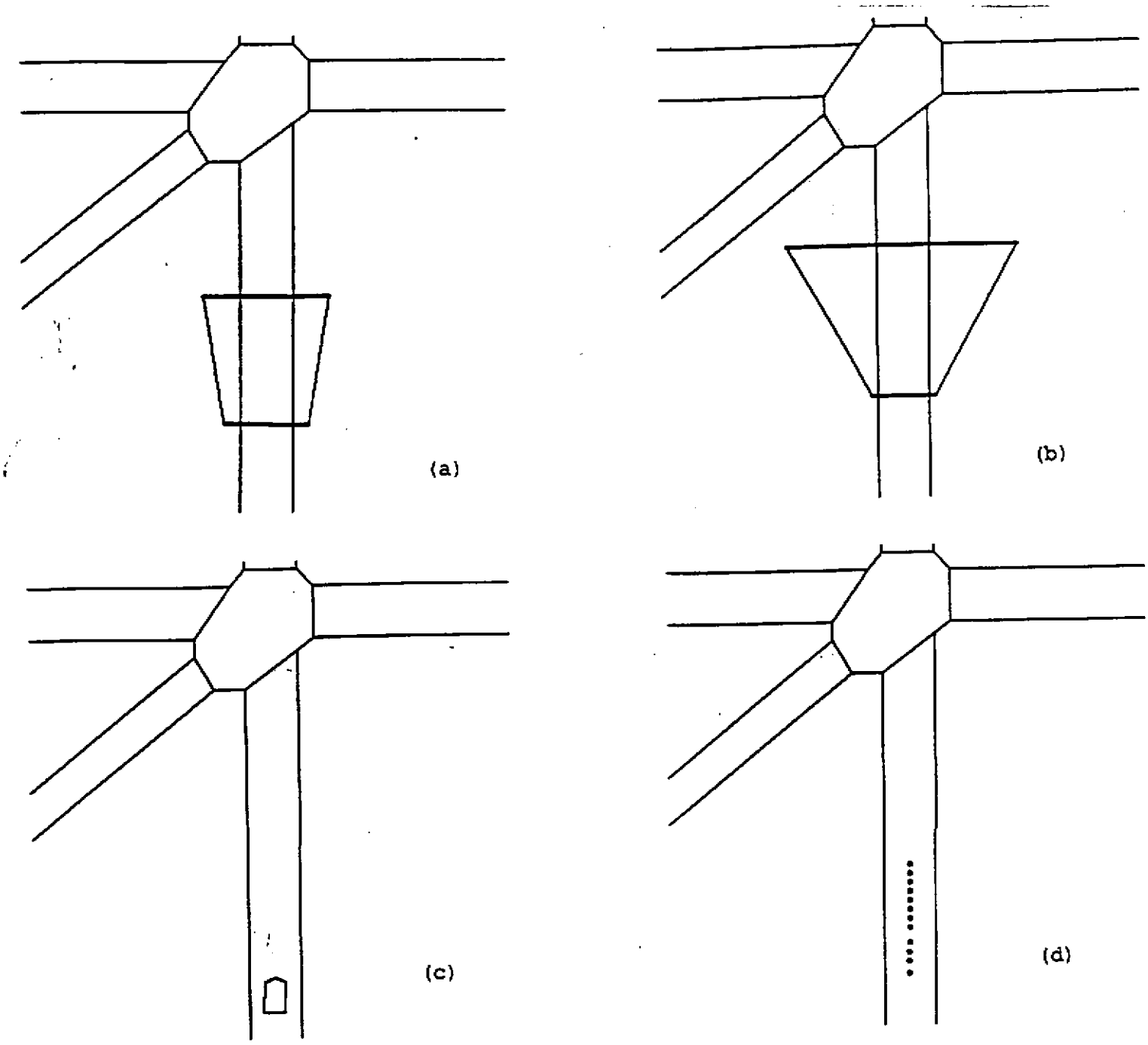



**Figure 24:** Processing Steps

## Vehicle Speed

Figure 27 shows a recorded vehicle speed that was adjusted according to Equation 8. The vehicle speed was recalculated whenever the processing steps were done. The vehicle slowed down around the intersection where the driving unit intervals were shorter and went back to a high speed on the straight road where the driving unit intervals were longer. Because of the hardware limitations of the Terregator vehicle, the vehicle speed could not be changed frequently; this is the reason that the recorded vehicle speed is not smooth.

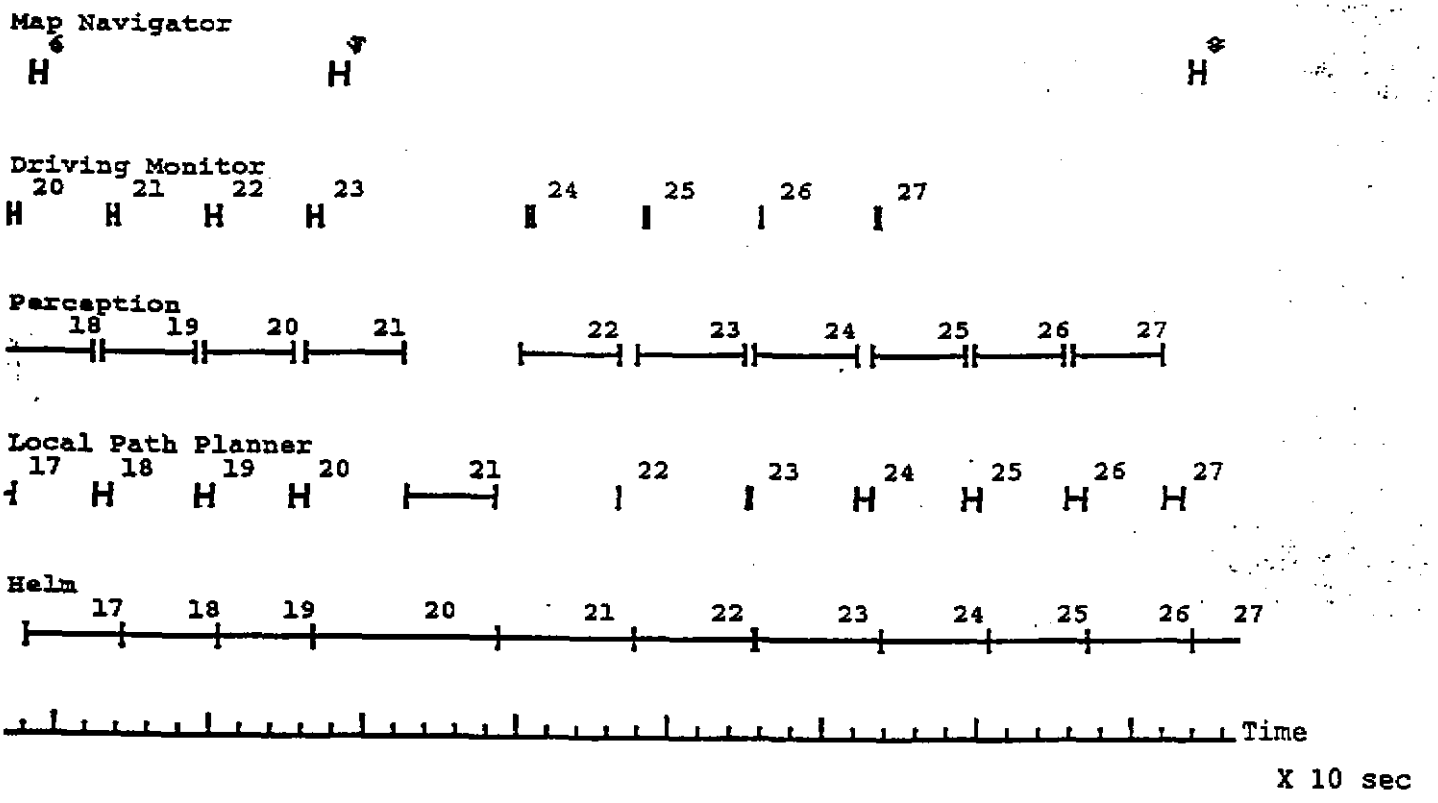



**Figure 25:** Timing Diagram of the Processing Steps

## Sensor Aiming

Our experiments on the Carnegie Mellon campus test site showed the necessity for adaptive sensor aiming. The fixed sensor view frame created a problem in turning at the intersections, because the vehicle had to turn through a large angle and the fixed sensor view frame could not cover the destination sidewalk while the vehicle was turning. To remedy this problem, the sensor view frame has to be aimed so that it covers the vehicle's destination. In addition, the scanning distance must be different in following straight sidewalks and in turning through intersections. In turning through an intersection, the vehicle position estimation must be accurate in both the vehicle's heading direction and the direction perpendicular to the vehicle's heading. Therefore, the scanning distance must be short. During straight travel, however, the vehicle position estimation along the vehicle's heading direction does not need to be so accurate and the scanning distance may be longer.

Figure 28 shows the sensor view frames and the scanning positions. The scanning positions were calculated using Equation 1 and the local path plan that was produced in the previous execution cycle. The scanning distance varied at the intersection and on the sidewalks.

To aim the TV camera into the predicted driving units, *pan* and *tilt* mechanisms are needed. This can present a very challenging timing problem if mechanical pan and tilt mechanisms are used. To avoid this, the Terregator vehicle was equipped with two cameras and switched between them instead of using a mechanical pan. The TV cameras had wide angle lenses and covered broad areas. The Perception step processed the desired rows of the image in place of a mechanical tilt. This "software pan/tilt" is very fast

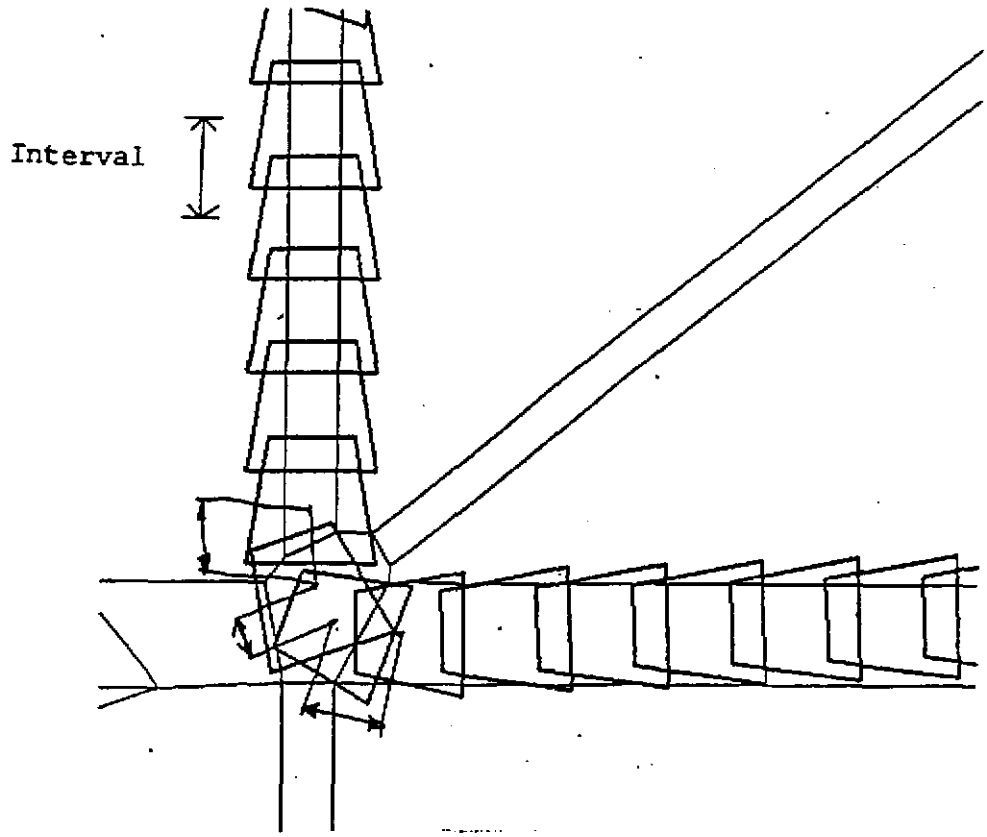



**Figure 26:** Driving Unit Intervals

and simple to program, as opposed to a mechanical pan/tilt which is relatively slow and difficult to control optimally. However, the software pan/tilt requires duplicated sensor hardware.

Our experiments have demonstrated the basic operation of the driving pipeline and dynamic adjustment of the execution interval, vehicle speed, and aim of the sensor. We have shown that the speed of the vehicle must be reduced and the driving unit shortened in situations involving uncertainty in the map or large vehicle control error (e.g., driving in intersections). Likewise, we have shown that the vehicle can drive quickly using large driving units on well-mapped straightaways. At both extremes and across the range we have demonstrated how the scanning distance can be adjusted to maximize parallelism.

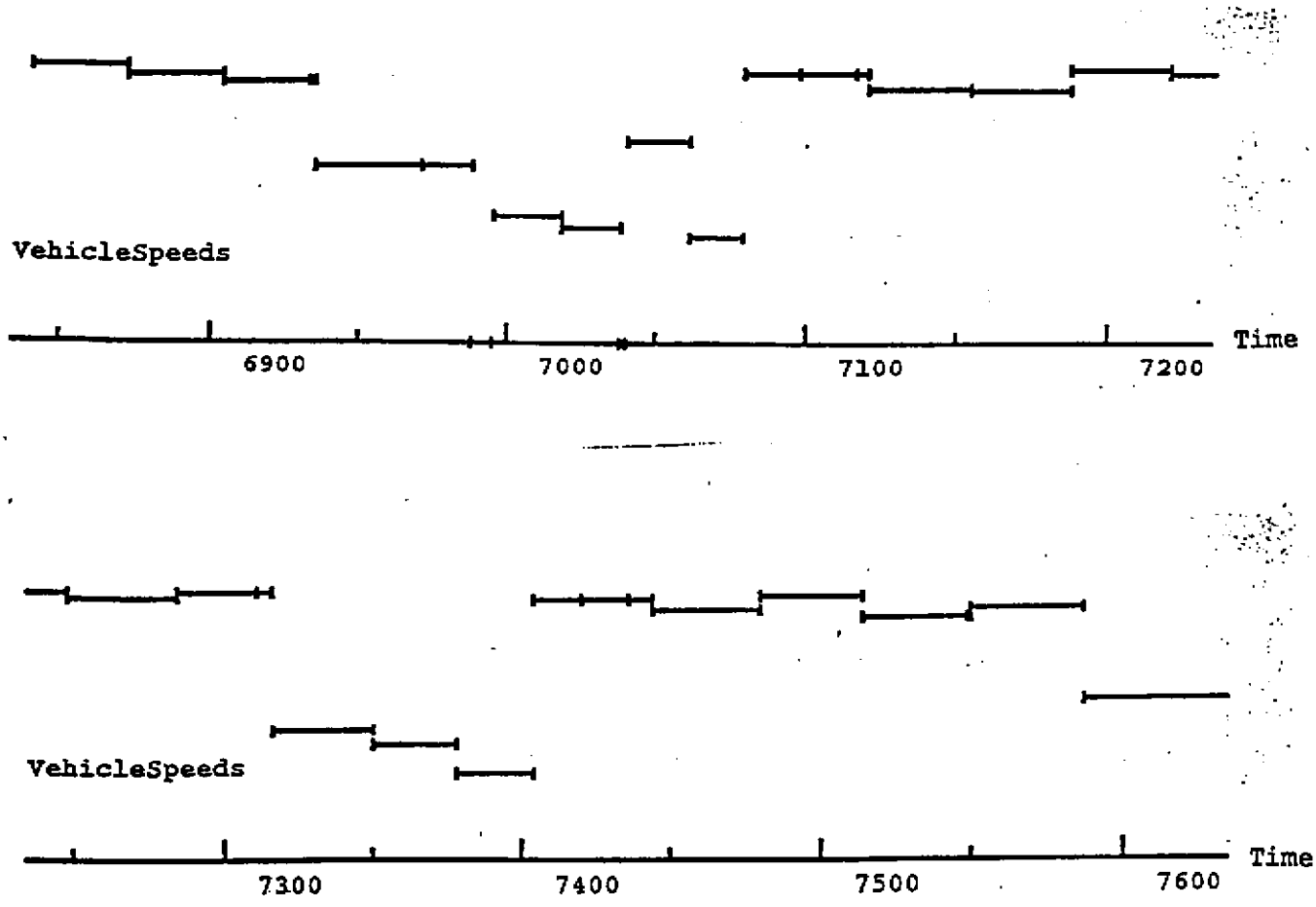



**Figure 27:** Control of the Vehicle Speed

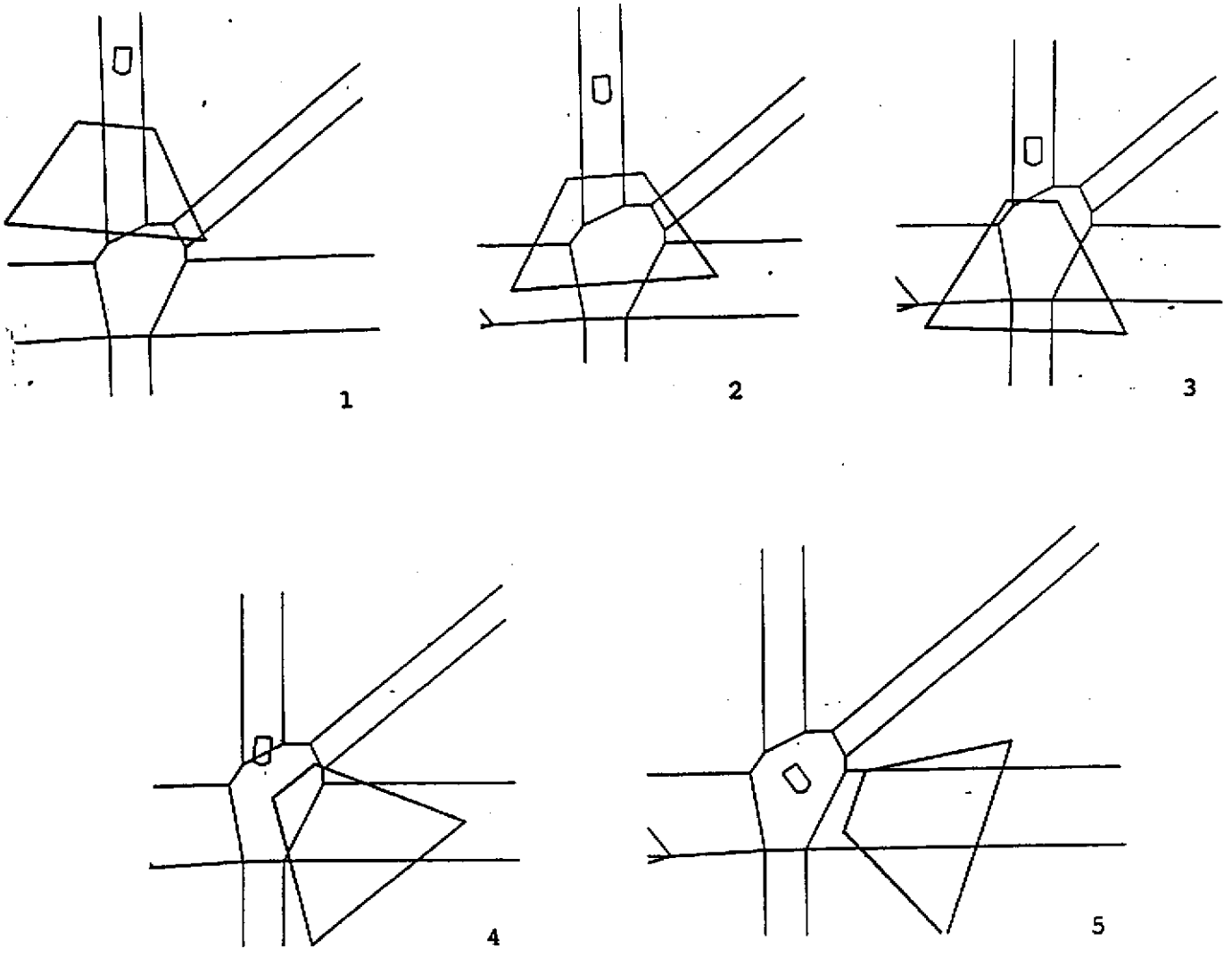



**Figure 28:** Sensor View Frames

# Section V

# Conclusions

## Evolution of the NAVLAB Vehicle

The NAVLAB vehicle has been a successful platform for mobile robot research, logging over 900 hours of experimental time. At this point, we are pushing up against the various physical limitations of the vehicle: electrical power, air conditioning, internal volume, and weight capacity. This limits the total computational power of the NAVLAB and its suite of sensors. Thus, future improvements must optimize the quality and use of these resources rather than simply adding on more and more equipment.

We have also found in this research that the low-level vehicle control must incorporate many different subsystems, each of which may implement a simple control scheme, rather than doing everything in a single computational loop. For example, we needed to build an analog control system to provide constant engine speed, so that power would not fall off during uphill runs. This would have been very difficult to implement by adding more code to centralized controller software.

In addition, there is a constant demand for more and more powerful sensors and control systems. The reason is that the robot has certain needs, such as knowing its position and the 3D description of the environment. These needs can either be met by adding appropriate complex hardware, or by clever software with simple hardware. The software is sometimes theoretically possible to write, but developing it is major research in itself, and it may or may not work. Therefore, for actually building a vehicle, the best solution is almost always to buy the best available hardware. This means a very large capital outlay is needed to obtain the equipment necessary to sustain the most productive research. Otherwise, the researchers spend all their time trying to compensate for the poor quality equipment. In the NAVLAB, this shows up most clearly in the need for a high-quality GPS and INS for vehicle position determination, and the need for the WARP and other massive computational power.

## Evolution of the CODGER Blackboard

The CODGER blackboard system has reached a certain level of maturity in its current form, CODGER II. CODGER II includes many facilities for map data representation, map revision, and vehicle position estimation, that distinguish it qualitatively from other mobile robot systems. Although not all of the uncertainty-modeling facilities have yet been implemented, the system has already proven to be very useful in simple map-updating experiments. Such experiments are among the most challenging mobile robot tasks, because they require perception of an unknown environment as well as integration of information with existing map data.

To accomplish this, the CODGER system centralizes the task of managing geometric and other map data. This is an example of the "database" approach in which each module talks to a central database of map information, as opposed to the "message-passing" approach in which each pair of communicating modules do so as needed. The database approach is more powerful than the message-passing approach because it allows anonymous storing and fetching of data; thus, it is more conducive to

supporting robot development research. CODGER uses this central database to implement centralized facilities for storing and retrieving geometric data.

The data representation facilities of CODGER include primitive geometric objects, organized into a complete 2D geometric modeling network with local coordinate systems, and time-varying transformations among the objects. We have developed the concepts of *frame generators* and *affixment groups* as ways to manage the complexity and ambiguity of representing time-varying relationships. With this battery of tools, the NAVLAB uses CODGER to implement real performance of map updating missions.

## Experiments With the Driving Pipeline

The Driving Pipeline is a driving control scheme to control a robot vehicle maneuvering in the physical world. By organizing and managing the primitive processing steps, the Driving Pipeline provides the following capabilities:

- **Continuous Vehicle Motion:** The Driving Pipeline drives the vehicle continuously by adjusting the vehicle speed and executing the Vehicle Control step in parallel with other processing steps.
- **Parallel Execution:** The Driving Pipeline executes the primitive processing steps in parallel and maintains a high degree of parallelism. Thanks to the pipelined execution, the Driving Pipeline achieves the highest possible vehicle speed.
- **Adaptive Control:** The Driving Pipeline is capable of adapting sensor aiming, vehicle speed, and execution intervals to the driving conditions.

These capabilities of the Driving Pipeline are made possible by the two key ideas of the Driving Pipeline, the driving unit and the pipelined execution of the processing steps. By using driving units, the data to be processed is divided into a sequence of driving units that can be processed separately by the processing steps. The steps themselves are designed to work in a fixed order on each driving unit. Because of the pipelined execution, the computation for these processing steps can be overlapped on successive driving units. These pipelines in both the processing steps and the data enable the pipelined execution, giving rise to parallel computation and continuous vehicle motion. The driving units also enable adaptive control. By adjusting the location, size, and interval of each driving unit, the Driving Pipeline adapts the processing to the driving situation. The pipeline execution thus enables the adaptive control in the continuous vehicle motion.

The Driving Pipeline clearly describes the driving control scheme in four aspects: primitive processing steps, organization of these processing steps, execution scheduling, and control parameters. In the case of stop-and-go motion, the last three aspects of the driving control scheme are implicit and do not need to be well defined. However, to achieve our goals -- continuous motion, parallel execution, and adaptive control -- we have developed the Driving Pipeline based on an explicit understanding of all of these aspects. This is why the Driving Pipeline is capable of controlling both geometry, such as the sensor view frames, and time, such as execution timing. Adjusting the vehicle speeds demonstrates these capabilities of the Driving Pipeline.

Although the Driving Pipeline supports continuous vehicle motion, the primitive processing steps involved in the Driving Pipeline employ only *static* algorithms. The Perception step, for example, analyzes the sensor data without taking into account the vehicle motion. Similarly, the Local Path Planning step

determines the trajectory path plan as if the vehicle were not moving while the Local Path Planning step is processing. By introducing the driving units, the Driving Pipeline converts dynamic problems into a set of static problems for each driving unit. By employing the pipelined execution, the Driving Pipeline overlaps the static processing steps to perform dynamic vehicle motion. This feature of the Driving Pipeline gives two advantages. First, the Driving Pipeline makes it easier to build mobile robot systems by integrating relatively well developed processing algorithms for perception and path planning. Second, the Driving Pipeline provides a test bed for studying these primitive algorithms using real mobile robot systems.

Future research will center on expanding the concept of the driving unit and pipelined execution to accomodate multiple sensors, uncertainty in the map database, and off-road travel. Multiple sensors with different view frame sizes introduce additional synchronization points into the pipeline, thus affecting the execution flow. Uncertainty about the positions of objects in the map affects the aiming of the sensors and vehicle speed. For example, in the presence of little uncertainty, the vehicle can look far ahead and drive quickly. Off-road travel provides a new set of Prediction, Perception, and Planning steps to be incorporated with on-road travel in a single pipeline to permit multiple modes of navigation. Algorithms are needed to dynamically determine the parameters of the pipeline in these scenarios while maximizing parallelism.

## Technology Transfer From This Research

The NAVLAB has a fairly unique status as a robot vehicle whose architecture is suited for research in both integrated robot systems and invidual component technologies (path planning, map navigation, and perception). Thus, the NAVLAB fills an important role in the research community as a focal point for technology transfer operations.

One level of technology transfer involving the NAVLAB has been the exchange of software and concepts for perception and planning. In 1987, researchers from the University of Massachusetts obtained data from the NAVLAB for use in their visual motion research under DARPA's SCVision program. They sent CMU their code, which was evaluated at CMU in terms of its suitability for tasks such as visual navigation for the NAVLAB. Additionally, Hughes Corp. developed an x-y-θ path planner, which appears to be very valuable for mobile robot navigation. CMU is now undertaking to improve and enhance this path planner by incorporating vehicle kinematics models, uncertainty modeling, and computational speedups. The resulting module promises to be a key ingredient in future versions of the NAVLAB software system.

In addition, the NAVLAB hardware and software developed under this contract has been exported to other sites or used as the basis for research in other robot vehicles. The CODGER blackboard database has been sent to Martin-Marietta, where it has controlled the ALV, and to other ALV contractors including ADS and FMC. It has also been sent to several non-DARPA sites, including NASA-Goddard, DEC, and Florida Atlantic University (for use in underwater robot design). On the low-level side of the system, the controller has been adapted for building the Locomotion Emulator (LE), a platform for the emulation and study of various schemes for wheeled robot locomotion. The LE controller consists of two Intel 80286 processor boards that run code developed for the NAVLAB; the user interface code was expanded to be more user-friendly since the LE's application involves more human interaction than the NAVLAB. The development of the LE controller occurred at the same time that the NAVLAB was switching from Galil motion control boards to the newer Creonics boards. The LE was used for development and testing of

the new Creonics device driver, to reduce the downtime of the NAVLAB vehicle.

The NAVLAB controller architecture, along with much of the perception software, will be the basis for corresponding components of the Mars Rover being developed at CMU under NASA sponsorship. As in the NAVLAB, a multitasking, priority-based real-time operating system is used to implement asynchronous I/O and coordination of robot motions. The Creonics motion control cards, found to be very effective for the NAVLAB vehicle, are now being adapted for the Mars Rover. In addition, the path planning and terrain perception capabilities of the NAVLAB are being used as the basis for the Mars Rover software. In addition, the FASTNAV project under sponsorship from Caterpillar Corp. used the NAVLAB as the basis for studying high-speed autonomous traversal of known roadways.

## Future Directions

We have identified several problems and issues as likely directions for our research in the next year:

- We need to develop a new generation of the low-level controller system that provides a high-performance UNIX-like environment.
- The vehicle path tracking is not as predictable as we would like. We have begun to develop a new path tracking method based on continuous replanning of quintic arcs to provide more precise vehicle control.
- We will continue development of the x-y-θ path planner based on the Hughes path planner, and add to it uncertainty management and representation.
- The Driving Pipeline concept has been very serviceable, but it has some key limitations. In particular, the need for all subsystems to operate in a pipeline means that computational and sensing resources are not operated at maximum efficiency. The new path planner may provide a good alternative scheduling mechanism for perception and other activities.

# References


[1] R. A. Brooks.
A Robot Layered Control System For A Mobile Robot.
*IEEE J. Robotics and Automation* RA-2:14-23, March, 1986.

[2] J. L. Crowley.
Navigation for an Intelligent Mobile Robot.
*IEEE J. Robotics and Automation* RA-1:31-41, March, 1985.

[3] E.D. Dickmanns, A. Zapp.
A Curvature-based Scheme for Improving Road Vehicle Guidance by Computer Vision.
In *SPIE Symposium on Optical and Optoelectronic Engineering*. October, 1986.

[4] Georges Giralt, Raja Chatila, and Marc Vaisset.
An Integrated Navigation and Motion Control System for Autonomous Multisensory Mobile Robots.
In *The First International Symposium on Robotics Research*, pages 191-214. 1986.

[5] Y.Goto and A. Stentz.
CMU Sidewalk Navigation System .
In *Proc. of Fall Joint Computer Conference*, pages 105-113. November, 1986.

[6] Y. Goto and A. Stentz.
The CMU System for Mobile Robot Navigation.
In *Proc. of IEEE International Conference on Robotics and Automation*, pages 99-105. March, 1987.

[7] H. P. Moravec.
The Stanford Cart and the CMU Rover.
*Proc. of the IEEE* 71:872-884, July, 1983.

[8] S. Shafer and W. Whittaker.
*June 1987 Annual Report: Development of an Integrated Mobile Robot System at Carnegie Mellon.*
CMU-RI-TR 88-10, CMU Robotics Institute, July, 1987.

[9] S. Shafer, A.Stentz, C. Thorpe .
An Architecture for Sensor Fusion in a Mobile Robot .
In *Proc. of IEEE International Conference on Robotics and Automation*, pages 2002-2011. April, 1986.

[10] C. Thorpe, S.Shafer, T.Kanade .
Vision and Navigation for the Carnegie Mellon Navlab .
In *Proc. of Image Understanding Workshop*, pages 143-152. Defense Advanced Research Project Agency Information Science and Technology Office, February, 1987 .

[11] S. Tsuji.
Monitoring of a building environment by a mobile robot.
In *Robotics Research 2*. 1985 .

[12] A. M. Waxman, J.J. LeMoigne, L.S. Davis, T.Siddalingalah.
A Visual Navigation System for Autonomous Land Vehicle.
*IEEE J. Robotics and Automation* RA-3:124-141, April, 1987.

# Appendix I

## Experimentation Issues for Mobile Robot Systems

The following document is incorporated into the annual report. It chronicles the role that the research under this contract has played in aiding DARPA's formulation of a research agenda in mobile robots and real-world machine perception.

A workshop was held in Vail, Colorado, in April 1988, for the purpose of planning the ongoing research in the ALV (Autonomous Land Vehicle) and SCVision (Strategic Computing Vision) programs. One of the key issues addresed at this workshop was how the basic research community might benefit from the continued availability of working, integrated robot systems such as the ALV, and what are the limitations of such integrated systems for supporting basic research. The PIs on this (NAVLAB) contract made a presentation to outline a number of possible research paradigms, and also to indicate what we have learned from the NAVLAB about the limitations of using an integrated system to support basic research.

After the workshop, we were asked by DARPA to prepare a document to summarize these issues. The following is that document. It has been used by DARPA internally and in conjunction with other research contractors in the ALV/SCVision community, as an aid to identifying the best strategies for continued research in this area.

Although the document refers specifically to the ALV at Martin-Marietta, the broad issues apply generally to big-system robotics research and may therefore be of interest to all readers. For that reason, we include the document in this annual report.

# EXPERIMENTATION ON THE ALV: TEMPLATES FOR EXPERIMENTS IN 1988 AND BEYOND

**by Steve Shafer, CMU**

18 April 1988

Submitted to DARPA and the ALV Experiment Steering Group.

## Abstract

At this point in the ALV and SCVision programs, there exists a highly capable and instrumented vehicle and accompanying software system at Martin-Marietta, along with an engineering and development staff. At the same time, a number of the Technology Development Contractors (TDCs) in these programs have developed research paradigms and software with varying degrees of maturity. To further the development of research in vision and navigation, plans are now needed for interaction to provide the TDCs with the data and system facilities they need from the ALV to promote basic research, while at the same time providing Martin-Marietta with access to the most mature software to add to the repertoire of the ALV system.

Two relevant facts have become clear through the research to date: First, the notion of building a single "integrated" system by somehow applying Super-Glue to all the component technology research is neither practical nor desirable at the present time; and Second, the disparate properties of the various technology research efforts demand many different plans for interaction with Martin-Marietta.

This document presents a brief discussion of the nature of system integration and how it differs from experimentation. At present, it is experimentation rather than integration that will serve as the best model for joint effort between Martin-Marietta and the TDCs. A number of possible modes of experimentation will then be outlined that may be suitable for various types of technology research with varying degrees of maturity. It is hoped that this outline will form a basis for closer cooperation and joint activity between the Technology Development Contractors and Martin-Marietta in the near future.

# INTEGRATION AND EXPERIMENTATION

It is tempting to believe that perhaps the ALV software system comprises a framework into which component research results can be inserted, like electrical plugs into sockets, forming a harmonious working system with interchangeable parts.

Unfortunately, the state of robotics research is not sufficiently advanced to support this model. In order to create working mobile robot systems, typically numerous software modules must be made to work together in harmony, each consisting of tens of thousands of lines of code, and each performing a highly complex function. We attempt to define the interfaces between these modules as specifically as we can, but these descriptions fall far short of being complete characterizations of such complex software. The most obvious aspects of a module that we typically describe as interface specifications include some abstract task description and perhaps the programming conventions for communication with the other modules; yet equally important are the programming language and operating system assumptions made by the module, the amount of time it is allocated for execution, the amount and nature of the vehicle motion between successive invocations, the nature of the sensors, the resolution of the input and output data, the nature of the test data used to develop the module, et cetera, et cetera, et cetera!

**All these factors must be compatible with the other modules in the system in order for the integrated system to succeed.**

Suppose for a moment that we desire to create a high-performance integrated system using pieces from more than one development site. Not only must these factors be described in detail in the interface specifications, but each contractor must build this complex research software in conformance with these elaborate descriptions. This would not be a recipe for successful research -- it would be a demanding development effort suitable only for mature software -- the antithesis of creative and wide-ranging research.

To develop integrated systems at the current state of the art demands an extraordinarily high bandwidth of communication among the module developers over an extended period of time, so that each module can be conceived and matured within a shared model of the context for execution of every module in the system. For this reason, multi-site integration has not been the methodology utilized by the successful system-building efforts at Martin-Marietta, CMU, Hughes, and other ALV/SCVision sites. Rather than that, these sites have relied on a methodology of intensive in-house system development, with the smallest possible bandwidth of interface to software developed elsewhere. This has been a successful approach so far, and should continue to be so in the future.

However, this model does little to contribute to the development of component technology research, which is essential for us to push the state-of-the-art most rapidly. For this research, it is not reasonable to demand that preliminary conceptual development should produce polished "modules" that will instantly fit into someone else's highly evolved system. A more appropriate view is that the existing ALV system can contribute in various ways to the maturation of the concepts and software being developed at the various sites. This can take place through a number of forms of experimentation that take forms other than the integrated system model described above.

This document briefly sets forth descriptions of a number of possible templates of experimentation that appear to be promising models for productive collaboration between Martin-Marietta and the Technology

Development Contractors to promote research in both systems and component technologies for navigation and vision.

## EXPERIMENTATION TEMPLATE FOR SYSTEM DEVELOPMENT RESEARCH

To date, several systems have been developed for outdoor navigation of sophisticated vehicles within the ALV and SCVision programs. These systems have all had several features in common:

- a *perception* subsystem
- a *planning* subsystem
- a *virtual vehicle* to follow elementary path descriptions
- a *software framework* to bind together these elements

In some cases, all of these elements have been developed at a single site, such as Martin-Marietta or CMU. Such efforts have been quite successful, but fall outside the scope of this document.

There have also been successful experiments involving ALV support for other contractors' system development efforts, and these establish a template for future efforts as follows:

**TEMPLATE A: VIRTUAL VEHICLE SUPPORT FOR SYSTEMS EXPERIMENTS**

Description: When a contractor has developed a complete system, there may be many reasons for testing it on the ALV:

- To perform live testing when the developer does not posess a vehicle.
- To test the system in conditions not available at the development site.
- To take advantage of hardware, software, or expertise not available at the development site.
- To test the system in a standardized scenario for comparative purposes.

*In this case, the usual desire is to preserve the integrity of the system as much as* possible, using only the smallest bandwidth interface to the ALV. This involves using the hardware and Virtual Vehicle of the ALV, with all of the other software elements being provided as part of the imported system.

Suitability: This model of experimentation is appropriate for a complete system developed outside of Martin-Marietta, which runs in real-time on hardware and operating systems available at the ALV site.

TDC Preparation: Preparation by the the TDC includes ensuring that the system conforms to the interface requirements of the ALV virtual vehicle, and ensuring that the system will run with the sensors, computing hardware, and system software at Martin-Marietta.

*M-M Preparation:* *Martin-Marietta is responsible for the vehicle and sensor hardware, the virtual vehicle,* and the basic computing hardware and system software.

One result of each experiment in this model is the potential for Martin-Marietta to accumulate these complete working systems as tools to support the other experiments described below. Of course, Martin-Marietta cannot be expected to provide substantial manpower for the maintenance of such systems over time.

Like all experiments involving the ALV, the TDC must expect to send one or more people to Martin-Marietta for some period of time to accomplish the experiment. Because of the intensive resource and personnel requirements on the part of both Martin-Marietta and the TDC, it may be appropriate for

concrete planning to be undertaken well in advance of the experiment and for DARPA to provide explicit funds for undertaking the experiment. Joint research proposals to DARPA appear to be a good mechanism for achieving both of these goals.

## EXPERIMENTATION TEMPLATES FOR COMPONENT TECHNOLOGY RESEARCH

The previous model is aimed at systems experiments, but does not address the role of the ALV in supporting basic research. There has been some concern on DARPA's part that perhaps the ALV is not a useful tool for promoting basic navigation and vision research, but this appears to be groundless. The lack of support in the past can be attributed to the startup and system-building effort at Martin-Marietta, and the progressive maturation of the technology development efforts throughout this period. At present, there appears to be a clamor for access to the ALV on the part of the TDCs, because of the richness of the data and the system context it can provide.

Each module or algorithm being investigated as a component technology needs to undergo a potentially lengthy process of maturation in terms of quality of results, robustness, speed of execution, and mating to the ALV system, before it can be fully integrated as a module in the system. At each stage of evolution, a somewhat different type of experiment with the ALV may be appropriate. Accordingly, several models of experimentation are outlined here, in order of increasing evolution of the module. It is not suggested that every module utilize all of these types of experimentation, nor that this list is exhaustive; this is simply a menu of several options that currently appear to be of general interest. Some of these have already been performed in the past, or are currently in progress.

### TEMPLATE B: GENERIC DATA COLLECTION -- THE SIMPLEST CASE

Description: The simplest use of the ALV to support basic researchers is in generic data collection. This involves the use of the hardware and sensors of the ALV, along with calibration and/or ground truth data. The input is a specification of the data needed, which must conform fairly closely with the capabilities of the ALV system itself. The output is the data set and accompanying descriptive data.

Suitability: This mode of operation is suitable for modules in early stages of development, with little or no compatibility with the ALV system.

TDC Preparation: The TDC must provide a complete specification of the data to be collected. In addition, the TDC should expect to send someone to Martin-Marietta to participate in the data collection process.

M-M Preparation: Control of the hardware and software, and provision of the accompnying data.

### TEMPLATE C: CUSTOMIZED DATA COLLECTION WITH DIRECT VEHICLE CONTROL

Description: There are some reasons why generic data collection may not be adequate for a particular basic research effort:

- need for unconventional sensors or configurations
- need for data collection patterns not compatible with normal ALV operation

In these cases, a more appropriate form of experiment would be for the TDC to mount the desired sensors (if other than the usual ALV sensors) on the vehicle, and to perform data collection while directly providing instructions to the virtual vehicle to cause the ALV to move in the desired way. In this case, the ALV is being utilized

simply as a platform for moving sensors outdoors, along with the accompanying data recording equipment and instrumentation.

Suitability: This mode of experiment is also suitable for modules that are not particularly compatible with the current ALV system, and would provide much more detailed control over the data collection than Type B experiments.

TDC Preparation: The TDC must ensure that the necessary sensors will be available to the ALV, and must prepare software to give the desired commands to the virtual vehicle of the ALV software (unless manual control is to be used).

M-M Preparation: Martin-Marietta must ensure that the virtual vehicle is working and may need to provide a nicer interface for the TDC to utilize. It may be desirable, for example, to provide a small repertoire of LISP functions that the TDC software can call to cause the vehicle to move in simple ways. In addition, Martin-Marietta must ensure that the TDC will have software access to the sensors and the data recording media for the experiment. If the sensors include controls such as pan/tilt, zoom, or focus, some hardware and software interface must also be provided.

**TEMPLATE D: CUSTOMIZED DATA COLLECTION WITH THE ALV SYSTEM**

Description: A variation of the above plan is to perform data collection while the ALV moves along the path it would normally follow (such as a roadway), but using unconventional sensors or movement increments. In this case, the TDC needs to meet all the requirements above, but the intention is to utilize the entire ALV system to move the vehicle to successive data collection points, rather than interfacing directly to the virtual vehicle. It is also possible that the experimenter will need access to some internal data from the ALV such as the vehicle attitude.

Suitability: This is better suited than Type C experiments when it is important for the vehicle to travel as it will in an actual demonstration run. For example, this may be desirable for an object recognition module looking at an object at the side of the road as the ALV travels along the road -- in this case, the TDC does not desire to control the vehicle path, but may need to control the distance of travel between image collection points. The vehicle path must be controlled by the ALV system, which must therefore utilize all the normal perception and planning elements of the full ALV system.

TDC Preparation: Similar to Type C. In addition, if internal ALV system data is needed, the TDC will need to utilize the software provided by Martin-Marietta to make it available.

M-M Preparation: Supporting a Type D experiment requires a substantially more sophisticated interface to the ALV system than the Type C model above. The mode of operation would still be essentially stop-and-go, but the full ALV system will be running essentially in parallel with the TDC software. Again, Martin-Marietta will probably need to provide a small repertoire of commands to be used by the TDC software to invoke vehicle motion; however, only the distance of motion would be adjustable by the TDC. If the module needs internal data from the ALV system, Martin-Marietta will have to provide a software mechanism to make it available.

**TEMPLATE E: OPEN-LOOP PIGGY-BACK EXPERIMENTATION**

Description: The data collection models presented above should allow for much more flexible and sophisticated data collection than has occurred in the past. However, the amount of data that can be pragmatically collected and transmitted to the TDC by these means is still somewhat limited. When a module has been tested on such stored data and has matured to the point that it runs in reasonably realistic time, it is possible to expose the algorithm to a much larger amount of data by actually running it in a "piggy-back" mode, in parallel with the ALV system but only loosely connected to the basic ALV software. This can be viewed as an extension or evolution of the Type D data collection experiment; but rather than store the data at each point, the data

would be actually run through the experimental module on-line. The output of the module could be stored for later analysis, or could be displayed for direct on-line evaluation, debugging, and error analysis. This can put the researchers into fairly intimate contact with the performance of their algorithm under real operating conditions. In fact, if the output is fairly closely related to some internally generated data of the ALV system, Martin-Marietta might provide a way for the TDC software to get a copy of the internal data for comparison purposes. In this case, a direct comparison can take place on-line -- though the ALV will, of course, be actually controlled by its own internal data rather than by the potentially flaky results of the experimental module. If the module is really running at real-time speeds, then the ALV might be able to undertake continuous motion while running the piggy-backed experimental module in parallel.

Suitability: Before a module can realistically be run in this mode, it must be tested on real, canned ALV data, producing reasonably good results, and it must run in a reasonable amount of time. Processing time per frame of sensor or path data ought not to exceed, say, several minutes or perhaps a fraction of an hour; otherwise, the motion of the vehicle will be only a few frames per hour, which can be taxing on the ALV vision system due to environmental changes, and would be in any case a colossally inefficient waste of research time and money. In the best case, processing time per frame ought to be between a few seconds and a minute. Of course, this might require considerable engineering of the module such as recoding for the WARP or Connection Machine. However, if the module is promising under Type D experimentation, then there is a strong motivation to do the necessary enginering to take it to the stage of this Type E on-line testing.

TDC Preparation: The TDC must be willing to prepare the module to meet the criteria above, and to integrate the vehicle control commands into the module to allow it to be run in conjunction with the ALV software. If special sensors are needed, these must of course be provided and configured by the TDC.

M-M Preparation: Similar to the Type D scenario, in the simplest case. If the vehicle is to run in continuous motion, or if the TDC needs access to internal data from the vehicle to compare with its output, then more work will be required by Martin-Marietta.

**TEMPLATE F: CLOSED-LOOP PIGGY-BACK EXPERIMENTATION (Module Replacement)**

Description: The Type E "open-loop" model provides for ALV input to the experimental module, but the output of the module is not fed back into the ALV system. That model is therefore useful for early on-line testing. Once a module has been run successfully in that mode, if the output is useful to the ALV system, then it may be desirable to hook up the output to feed back into the ALV system. In this way, a functional replacement can be made for a part of the ALV software, or a new source of information can be made available to it. This allows on-line testing of the module as a system component, which is a qualitatively different concern than the previous experimental models that test the module as an entity on its own. At this stage, Martin-Marietta may decide to incorporate the module into its baseline ALV system. This constitutes technology transfer of actual code, which can be seen as an evolutionary step that must follow a substantial preliminary process of module development and testing as outlined in the previous templates.

Suitability: Only a mature module, running in realistic time on the ALV and known to provide high-quality output in a form compatible with the ALV system, is a suitable candidate for this mode of experimentation.

TDC Preparation: Successful Type E experimentation is probably a pre-requisite for closed-loop testing. In addition, the instigator of the experiment (the TDC or Martin-Marietta) must be prepared to modify the module to produce its output in a form suitable for direct utilization by the ALV system.

M-M Preparation: The ALV system itself must be modified to accept the data produced by the module.

## PROGNOSIS FOR RESEARCH PROGRESS USING THE ALV

At the present time, there appears to be a substantial demand on the part of the technology and systems researchers to have access to the ALV as a data collection platform, experimental system context, and virtual vehicle. While the interactions between Martin-Marietta and the technology development sites have been limited in the past, the current degree of maturity of the various efforts is creating an increasing need for data that can only be provided by the ALV or by prohibitively elaborate laboratory facilities. Hopefully, this document will promote future interactions by providing some common models of experimentation between Martin-Marietta and the various Technology Development Contractors.

# PUBLICATIONS

Selected publications by members of our research group, supported by or directly related to this contract:


1. *Color Vision for Road Following.* Crisman, J. and Thorpe, C. Presented at *SPIE Conference on Mobile Robots*, November 1988.
2. *The Driving Pipeline: A Driving Control Scheme for Mobile Robots.* Goto, Y., Shafer, S.A., and Stentz, A. *International Journal of Robotics and Automation*, to appear. Also appeared as technical report CMU-RI-TR-88-8.
3. *Perception for Rugged Terrain.* Kweon, I., Hebert, M., and Kanade, T. Presented at *SPIE Conference on Mobile Robots*, November 1988.
4. *Experimentation on the ALV: Templates for Experiments in 1988 and Beyond.* Shafer, S. Presented to DARPA for use as a planning document within the ALV and SCVision Programs.
5. *1987 Year End Report for Road Following at Carnegie Mellon.* Thorpe, C. and Kanade, T. Technical Report CMU-RI-TR-88-4, 1988.