An Architecture for Sensor Fusion in a Mobile Robot

Steven A. Shafer, Anthony Stentz,
and Charles E. Thorpe

CMU-RI-TR-86-9

The Robotics Institute
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

April 1986

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

Abstract

This paper describes sensor fusion in the context of an autonomous mobile robot, the NAVLAB system, based on a commercial truck with computer controls and studded with cameras and other sensors. This paper describes the software architecture of the NAVLAB, consisting of two parts: a "whiteboard" system called CODGER that is similar to a blackboard but supports parallelism in the knowledge source modules, and an organized collection of perceptual and navigational modules tied together by the CODGER system.

Table of Contents

1. Introduction 1
1.1 The NAVLAB Vehicle 2
2. The CODGER Whiteboard System 4
2.1 Blackboards and Whiteboards 4
2.2 Data Storage and Retrieval 5
2.3 Synchronization Primitives 7
2.4 Summary of the CODGER Whiteboard System 8
3. Module Architecture of the NAVLAB System 9
3.1 The CAPTAIN and Mission Descriptions 9
3.2 The LOOKOUT 10
3.3 The MAP NAVIGATOR and Route Planning 11
3.4 The PILOT and Driving Control 12
3.5 The PERCEPTION Subsystem 14
4. Methods for Sensor Fusion 18
5. Conclusions 21
5.1 Acknowledgements 21
6. Bibliography 22

List of Figures

Figure 1-1: Sketch of the NAVLAB Vehicle 2
Figure 1-2: On-board Computing Facilities 3
Figure 2-1: Process Structure of the System 4
Figure 3-1: NAVLAB Software Block Diagram 9
Figure 3-2: Mission Description 10
Figure 3-3: The MAP NAVIGATOR Subsystem 11
Figure 3-4: The PILOT Subsystem 12
Figure 3-5: The Driving Pipeline in Action 14
Figure 3-6: The PERCEPTION Subsystem Plan 15
Figure 3-7: Future PERCEPTION Subsystem Plan 16

1. Introduction

Robot perception in real-world domains requires system architecture support not found in simpler robot systems. In the most typical mobile robots of the past, for example, the system contains one visual sensor and a simple motion encoding mechanism; planning and perceptual processing are performed off-line by a single mainframe CPU with telecommunications to the vehicle; and the control program resembles a simple aim-and-steer loop performing some sophisticated variation of visual servoing [14, 19, 20]. These architectural features cannot support perception and control in complex outdoor environments with very general task specifications.

In such more complex systems, there is a need for multiple sensors. No single system, such as a camera, a laser rangefinder, or sonar array, can provide a map of the environment with sufficient resolution and reliability to control a vehicle on a complex mission. For example, a mission might require avoiding obstacles on the roadway, which is best performed with a 3-D sensor such as a rangefinder; the same mission might require steering towards some distant landmark such as a telephone pole on the horizon, which is probably beyond the the effective range of a rangefinder yet is easily detected by a TV camera. This type of trade-off occurs at all scales of perception, and the only solution currently available is to incorporate multiple sensors on a single vehicle.

As soon as multiple sensors are employed, the system architecture requirements become very demanding. If a single processor is used, perceptual processing will swamp it and real-time control is probably out of the question. However, if multiple processors are used, the processing times may vary from one sensor to the next and so some loose, asynchronous coupling mechanism must be employed. Geometric reasoning and transformations become fundamental because sensor data from different times and vantage points must be integrated into a single coherent interpretation. Furthermore, since perception, planning, and control are all problems currently being studied, the system design may change rapidly and a very modular design methodology is needed.

At the CMU Vision Lab robot perception and control is studied in the context of a mobile robot vehicle, the *NAVLAB* [11]. The NAVLAB has a number of sensors, with several processors on board. A distributed modular software system is being built, based on a central database called *CODGER*. This paper describes the CODGER database/communication system, the NAVLAB vehicle and system block diagram, and some issues inherent in *sensor fusion* -- the integration of multiple sensors in a single system.

1.1 The NAVLAB Vehicle

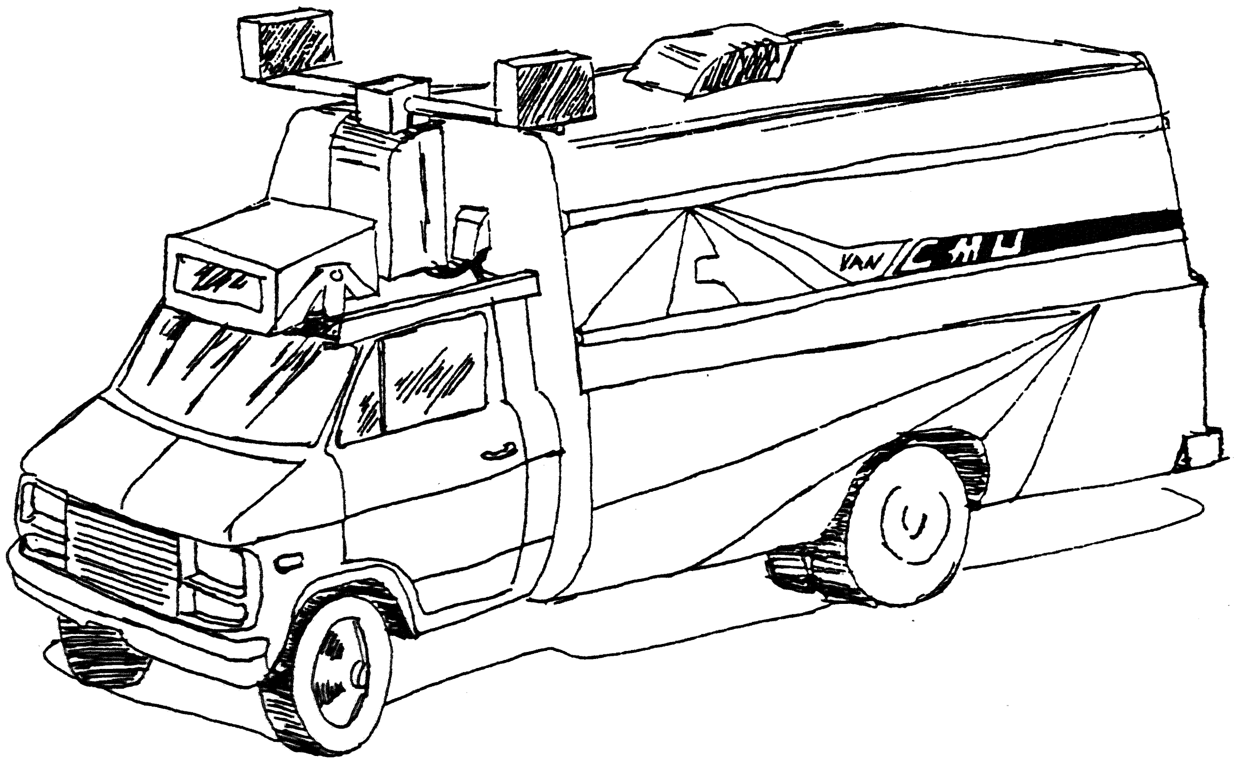


Figure 1-1: Sketch of the NAVLAB Vehicle

The NAVLAB vehicle (Figure 1-1) is a commercial truck modified by adding sensors, electronic controls, and on-board computers and power generators. It will be completely self-contained, requiring neither a cable tethering it to an off-board mainframe computer, nor an electronic telecommunications tether. Telecommunications gear is present for data recording and tele-operation, but is not required for computer control of the vehicle.

The control systems of the vehicle include computer control of a hydraulic drive/braking system, computer control of the steering wheel, and processors to monitor and control engine functions. A global system clock and a processor maintain a primitive vehicle position estimate based on dead reckoning. An inertial navigation system is also on-board.

The sensors on the vehicle include a pan/tilt head with a pair of stereo color cameras on a 6-foot baseline, an ERIM laser rangefinder on a pan/tilt mount, and sonar sensors for a "soft bumper." For special purposes, additional cameras may be mounted in various places, such as fixed cameras for identifying objects at the roadside.

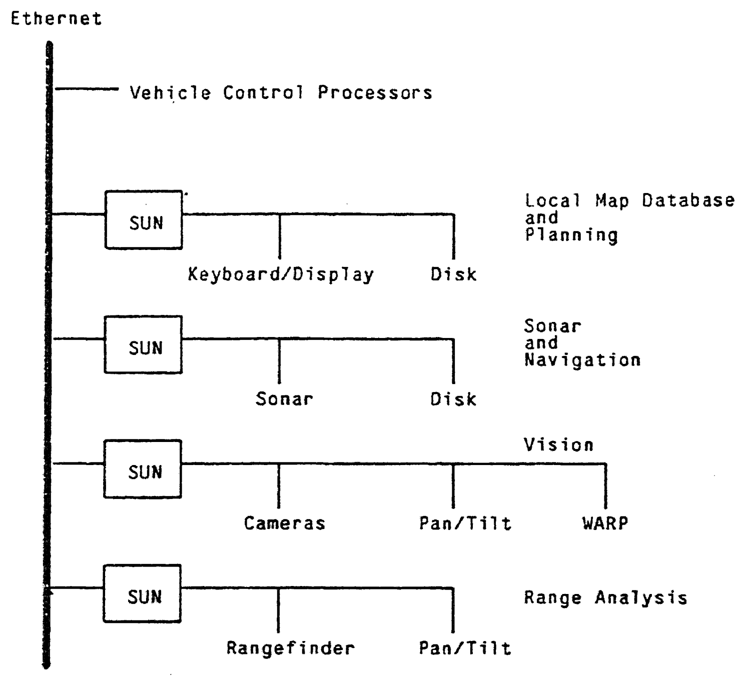


Figure 1-2: On-board Computing Facilities

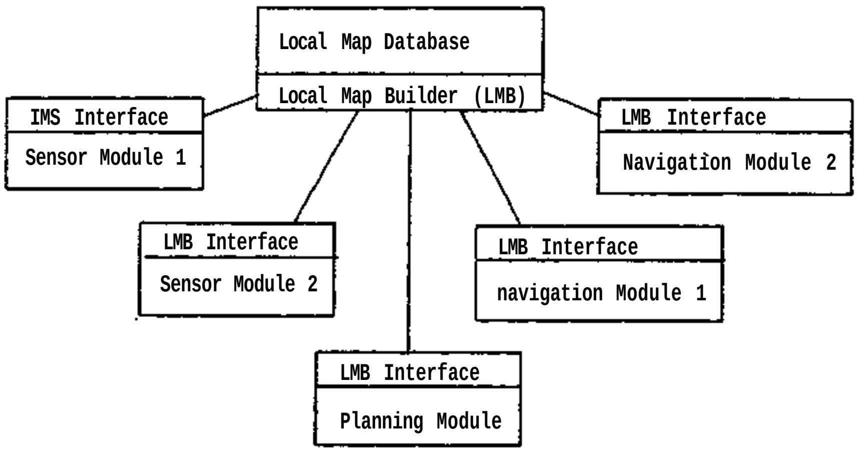
The vehicle contains on-board general-purpose computing facilities as well. There are four workstations connected by an Ethernet (Figure 1-2). Each workstation has a function dictated by the sensors to which it is connected, and sensor-independent planning tasks can be performed on any of them. There is a plan to put at least one WARP array (high-speed parallel machine) on board for vision [12]. Programming for all the workstations takes place in C or LISP with the UNIX operating system. Access to sensor data and vehicle control processors, which reside on the Ethernet, takes place through a set of subroutines called the *virtual vehicle* that are accessible to the perceptual and planning programs on the workstations.

As can be seen, the NAVLAB system provides a battery of sensors and control functions, a suite of general-purpose and special-purpose processors, and a complex processing task to be carried out in real-time, albeit slowly. The remainder of this paper describes the software architecture being used to implement this system. Since perception is by far the most demanding aspect of the system, special attention will be given to existing and planned mechanisms for supporting highly competent perception.

2. The CODGER Whiteboard System

The system will consist of several relatively large *modules,* which are independently running programs performing large pieces of the computation in the system. For example, "map navigation" and "road-edge finding by stereo vision" will be modules in the system. Relatively few, large-grain modules are used because of the overhead costs of communication and because real parallelism is limited by the number of workstations on board. Tying these modules together is a communications system and database called *CODGER* (Communications Database with GEometric Reasoning) [18]. We call CODGER a *whiteboard* because, as described below, it resembles a traditional "blackboard" but with some significant new features-parallel execution of modules and geometric reasoning.

2.1 Blackboards and Whiteboards



Figu re 2-1: Process Structure of the System

The program organization of the NAVLAB software is shown in Figure 2-1. Each of the major boxes represents a separately running program. The central database, called the *Local Map,* is managed by a program known as the Local Map Builder *(LMB).* Each module stores and retrieves information in the database through a set of subroutines called the *LMB interface* which handle all communication and synchronization with the LMB. If a module resides on a different processor than the LMB, the LK4B and LMB Interface will transparently handle the network communication. The Local Map, LMB, and LMB Interface together comprise the CODGER system.

The overall system structure-a central database, a pool of knowledge-intensive modules, and a database manager that synchronizes the modules-is characteristic of a traditional blackboard system [31. Such a system is called "heterarchical" because the knowledge is scattered among a set of modules that have access to data at all levels of the database (i.e., low-level perceptual processing ranging up to high-level mission plans) and may post their findings on any level of the database; in general, heterarchical systems impose *de facto* structuring- of the information flow among the modules of the system. In a traditional blackboard, there is a single flow of control managed by the database (or blackboard) manager. The modules are subroutines, each with a pre-determined precondition (pattern of data) that must be satisfied before that module can be executed. The manager keeps a list of which modules are ready to execute, and in its central loop it selects one module, executes it, and adds to its ready-list any new modules whose preconditions become satisfied by the currently executing module. The system is thus synchronous and the manager's function is to focus the attention of the system by selecting the "best" module from the ready-list on each cycle.

We call CODGER a *whiteboard* because it also implements a heterarchical system structure, but differs from a blackboard *in* several key respects. In CODGER, each module is a separate, continuously running program; the modules communicate by storing and retrieving data in the central database. Synchronization is achieved by primitives in the data retrieval facilities that allow, for example, for a module to request data and suspend execution until the specified data appears. When some other module stores the desired data, the first module will be re-activated and the data will be sent to it With CODGER a module programmer thus has control over the flow of execution within his module and may implement real-time loops, daemons, data flows among cooperating modules, etc. CODGER also has no pre-compiled list of data retrieval specifications; each time a module requests data, it provides a pattern for the data desired at that time. We call such a system a whiteboard--it is heterarchical like a blackboard, but each module runs in parallel with the module programmer controlling the synchronization and data retrieval requests as best suited for each module. Like other recent distributed At architectures, whiteboards are suited to execution on multiple processors **[1, 4, 6]**.

2.2 Data Storage and Retrieval

Data in the CODGER database (Local Map) is represented in *tokens* consisting of classical *attribute-value pairs.* The types of tokens are described in a *template file* that tells the name and type of each attribute in tokens of each type. The attributes themselves may be the usual scalars (integers, floating-point values, strings, enumerated types), arrays (or sets) of these types (including arrays of arrays), or geometric locations as described below. CODGER automatically maintains certain attributes for each token: the token type and id number, the "generation number" as the token is modified, the time at which the token was created and inserted into the database, and the time at which the sensor data was acquired that led to the creation of this token. The LMB Interface provides facilities for building and dissecting tokens and attributes within a module. Rapid execution is supported by mapping the module programmer's names for tokens and attributes onto globally used index values at system startup time.

A module can store a token by calling a subroutine to send it to the LMB. Tokens can be retrieved by constructing a pattern called a *specification* and calling a routine to request that the LMB send back tokens matching that specification. The specification is simply a boolean expression in which the attributes of each token may be substituted; if a token's attributes satisfy the boolean expression, then the token is sent to the module that made the request. For example, a module may specify:

tokens with **type** *equal to "intersection" and* **traffic-control** *equal to "stop-sign"*

This would retrieve all tokens whose **type** and **traffic-control** attributes satisfy the above conditions. The specification may include computations such as mathematical expressions, finding the minimum value within an array attribute, comparisons among attributes, etc. CODGER thus implements a general database. The module programmer constructs a specification with a set of subroutines in the CODGER system.

One of the key features of CODGER is the ability to manipulate geometric information. One of the attribute types provided by CODGER is the *location*, which is a 2-D- or 3-D polygon and a reference to a *coordinate frame* in which that polygon is described. Every token has a specific attribute that tells the location of that object in the Local Map, if applicable, and a specification can include geometric calculations and expressions. For example, a specification might be:

tokens with **location** *within 5 units of (45,32) [in* world *coordinates]*

or

tokens with **location** *overlapping X*

where *X* is a description of a rectangle on the ground in front of the vehicle. The geometric primitives currently provided by CODGER include calculation of centroid, area, diameter, convex hull, orientation, and minimum bounding rectangle of a location, and distance and intersection calculations between a pair of locations. We believe that this kind of geometric data retrieval capability is essential for supporting spatial reasoning in mobile robots with multiple sensors. We expect geometric specifications to be the most common type of data retrieval request used in the NAVLAB.

CODGER also provides for automatic coordinate system maintenance and transformation for these geometric operations. In the Local Map, all coordinates of location attributes are defined relative to **WORLD** or **VEHICLE** coordinates; **VEHICLE** coordinates are parameterized by time, and the LMB maintains a time-varying transformation between WORLD and VEHICLE coordinates. Whenever new information (i.e., a new VEH1CLE-U>WORLD transform) becomes available, it is added to the "history" maintained in the LMB; the LMB will interpolate to provide intermediate transformations as needed. In addition to these basic coordirutfe systems, the LMB Interface allows a module programmer to define *local coordinates* relative to the basic coordinates or relative to some other local coordinates. Location attributes defined in a local coordinate system are aut6matically converted to the appropriate basic coordinate system when a token is stored in the database. CODGER provides the module programmer with a conversion routine to convert any location to any specified coordinate system.

All of the above facilities need to work together to support asynchronous sensor fusion. For example, suppose we have a vision module A and a rangefinder module B whose results are to be merged by some module C. The following sequence of actions might occur:

1. A receives an image at time 10 and posts results on the database at time 15. Although the calculations were carried out in the camera coordinate system for time 10, the results are automatically converted to the VEHICLE system at time 10 when the token is stored in the database.

2. Meanwhile, B receives data at time 12 and posts results at time 17 in a similar way.

3. At time 18, C receives A's and B's results. As described above, each such token will be tagged with the time at which the sensor data was gathered. C decides to use the vehicle coordinate system at time 12 (B's time) for merging the data.

4. C requests that A's result, which was stored in VEHICLE time 10 coordinates, be transformed into VEHICLE time 12 coordinates. If necessary, the LMB will automatically interpolate coordinate transformation data to accomplish this. C can now merge A's and B's results since they are in the same coordinate system. At time 23, C stores results in the database, with an indication that they are stored in the coordinate system of time 12.

2.3 Synchronization Primitives

CODGER provides module synchronization through options specified for each data retrieval request. Every time a module sends a specification to the LMB to retrieve tokens, it also specifies options that tell how the LMB should respond with the matching tokens:

# *Immediate Request* The module requests all tokens currently in the database that match this specification. The module will block (i.e., the "request" subroutine in the LMB Interface will not return control) until the LMB has responded. If there are no tokens that match the specification, the action taken is determined by an option in the module's request:

o *Non-Blocking.* The LMB will answer that there are no matching tokens, and the module can then proceed. This would be used for time-critical modules such as vehicle control. Example: "Is there a stop sign?"

o *Blocking.* The LMB will record this specification and compare it against all incoming tokens. When a new token matches the specification, it will be sent to the module and the request will be satisfied. Meanwhile, the module will remain blocked until the LMB has responded with a token. This is the type of request used for setting up synchronized sets of communicating modules: each one waits for the results from the previous module to be posted to the database. Example: "Wake me up when you see a stop sign."

© *Standing Request.* The module gives a specification along with the name of a subroutine. The module then continues running; the LMB will record the specification and compare it with all incoming tokens. Whenever a token matches, it will be sent to the module. The LMB Interface will intercept the token and execute the specified subroutine, passing the token as an argument This has the effect of invoking the given subroutine whenever a token appears in the database that matches the given specification. It can be used at system startup time for a module programmer to set up "daemon" routines within the module. Example: "Execute that routine whenever you see a stop sign."

2.4 Summary of the CODGER Whiteboard System

CODGER implements a "whiteboard" with these properties:
- heterarchical system like a traditional blackboard
- parallel asynchronous execution of large-grain modules
- communication and synchronization via a central database
- transparent networking between processors
- « no pre-compilation of data retrieval specifications
- module programmer controls flow of execution through each module

In addition, CODGER provides geometric retrieval via:
- geometric objects and search specifications
- time-dependent coordinate transformations

Such a communications/database system does not, of course, solve any sensor fusion problems. What CODGER does is to give the module programmer a simple, powerful system interface to allow more attention to be devoted to the perceptual and planning research problems.

3. Module Architecture of the NAVLAB System

The NAVLAB software consists of a number of modules performing planning and perception, and communicating with each other via the CODGER whiteboard system. The modules are shown in Figure 3-1 as a system block diagram. In this diagram, each box represents one or more cooperating programs; all the lines between boxes represent information flows via the CODGER database.

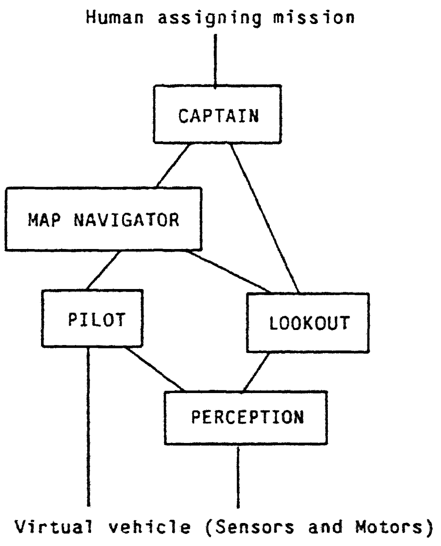


Figure 3-1: NAVLAB Software Block Diagram

It is important to bear in mind that the computational load of the NAVLAB occurs almost entirely within the PERCEPTION subsystem. All the other modules from CAPTAIN to PILOT and LOOKOUT are planning modules whose computational requirements are really negligible when compared to that of PERCEPTION. The planning portions of the system are described here because they form a rich context that constrains the perceptual task in important ways that will be discussed. The planning system itself is still quite simple; in the future we will devote effort to replacing it with a more powerful reasoning system.

To speed up PERCEPTION, we use special hardware such as the WARP; the planning system executes on general-purpose machines.

3.1 The CAPTAIN and Mission Descriptions

At the upper level of the system is the *CAPTAIN* module that receives mission instructions from the controlling person and oversees the mission. The mission description itself is a sort of program; the entire NAVLAB can be thought of as a computing engine that will evaluate and carry out the program specified in the mission.

```
step 1:
        go to destination via constraint
        at end do action
        if trigger do action
        . . .
step 2:
        . . .
```

Figure 3-2: Mission Description

Figure 3-2 illustrates how a mission might be described. The mission consists of a number of steps, and the CAPTAIN sequences through the steps. For each step, there is a *destination* that tells where to go and/or a *co^smmi* that tells how to go. For example, the instruction "go to St. John's church" gives a destination; "go /ia Road C" gives a constraint; "go to St. John's church via Road C" gives Doth. Each step of the mission also has an *action* to be performed when the destination is reached; th;$ may be to *report* success of the mission, *report failure,* or *go to another step* of the mission.

Jn add-on to the destination, which tells where to drive, each mission step may have one or more ^gger conditions that toil what to be looking for along the way. For each trigger, there is an *action* (success, 'attare, or new step) to take if the condition is satisfied. The trigger may be a perceptual cue such as "\f you see a church," or a positional condition such as "if you travel 5 miles."

For each mission step, the CAPTAIN passes the destination and constraint to the MAP NAVIGATOR, #fto» mft *i* is to drive the vehicle accordingly, and the CAPTAIN passes the trigger conditions to the LOCKOUT, which *will mue* appropriate perceptual requests and monitor the database. These form **the mission-level** control aid perception functions of the NAVLAB system.

3*2 The LOOKOUT

Nt ; *00^OUT* $ t simplt module that receives trigger condition specifications from the CAPTAIN «J ternary detowttoM from the MAP NAVIGATOR (described below); the LOOKOUT issues «p*£pr»a*e reqgetti to PERCEPTION and monitors the database for the proper conditions. **Whenever a trigger [illegible] is** sitsfied or t iandmarii is seen, the LOOKOUT will notify the CAPTAIN **or MAP NAVIGATOR, respectively.**

3.3 The MAP NAVIGATOR and Route Planning

The *MAP NAVIGATOR* receives a destination and/or constraint from the CAPTAIN for each mission step. Its job is to conduct the vehicle accordingly and report success to the CAPTAIN, or to report failure if the mission step is impossible.

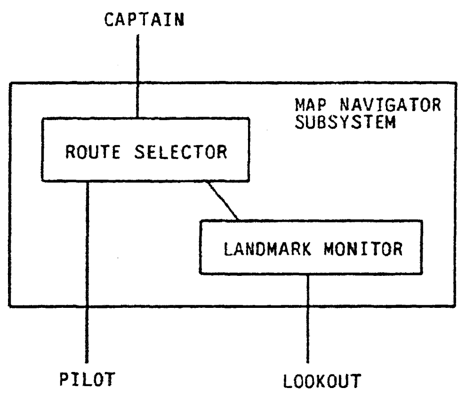


Figure 3-3: The MAP NAVIGATOR Subsystem

The MAP NAVIGATOR contains two components (Figure 3-3): a ROUTE SELECTOR and a LANDMARK MONITOR. The *ROUTE SELECTOR* consults the global map knowledge of the NAVLAB, and determines the best *route* from the vehicle's position to the destination according to the given constraints. After planning the route, the ROUTE SELECTOR breaks it up into *route segments* of a uniform driving mode (roadway, intersection, or cross-country stretch). The ROUTE SELECTOR will then monitor the progress of the vehicle along each route segment in turn, passing the route segment descriptions down to the PILOT as the vehicle travels.

Before entering each route segment, the ROUTE SELECTOR determines which landmark objects in the global map ought to be visible as the vehicle travels along that route segment. This list is passed to the *LANDMARK MONITOR*. As each landmark in the list should become visible, the LANDMARK MONITOR will pass its description to the LOOKOUT. When the LOOKOUT reports a sighting of a landmark, the LANDMARK MONITOR will use it to verify the vehicle's position estimate. Landmark sightings will be far less frequent than vehicle position estimates from dead reckoning or intertial navigation, but landmarks do not (generally!) move in space and can thus be used to correct for drift in the vehicle position estimate (the **VEHICLE-to-WORLD** transformation maintained by the LMB). When a landmark is seen, the accumulated drift error may have to be spread backwards in time across the history of **VEHICLE-to-WORLD** transformations maintained by the LMB; this will keep the history of transformations smooth rather than choppy.

3.4 The PILOT and Driving Control

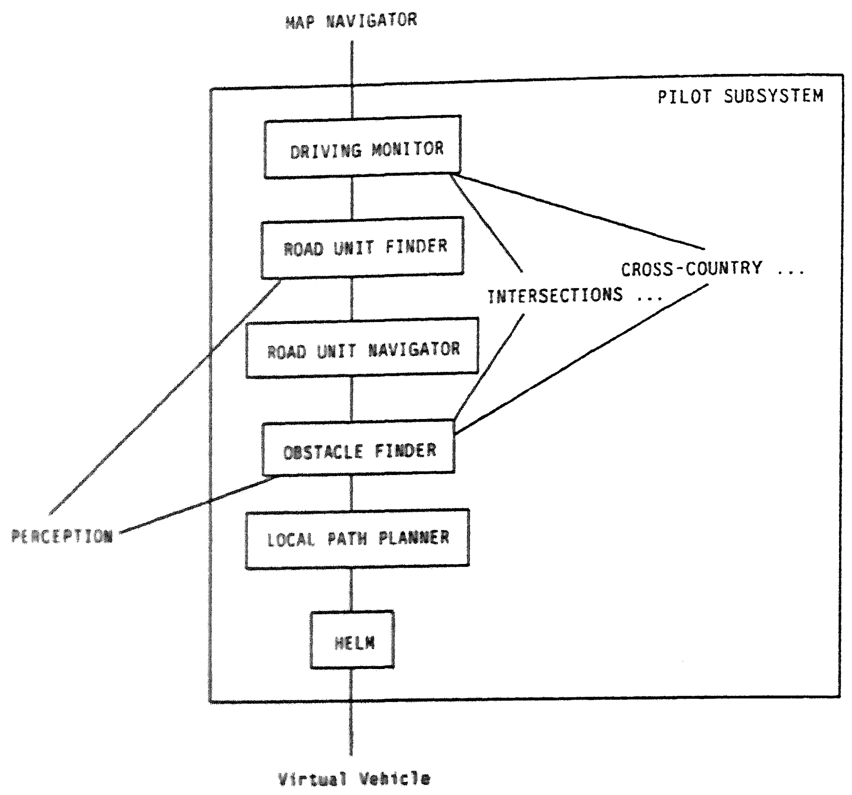


Figure 3-4: The PILOT Subsystem

The *PILOT* operates continuously to plan the vehicle's path and issue perception and control instructions. The PILOT contains several programs that form a pipeline (Figure 3-4) to process each area to be traversed. The *DRIVING MONITOR*, the top level of the PILOT, receives route segment descriptions from the MAP NAVIGATOR. The DRIVING MONITOR will keep track of the vehicle's progress on the current route segment and provide predictions for the next stage, the ROAD UNIT FINDER.

The *ROAD UNIT FINDER* breaks the route segment into perceivable pieces called *road units*. In each cycle, the ROAD UNIT FINDER asks for and receives a prediction about the upcoming roadway from the DRIVING MONITOR. This prediction includes where the road edges ought to be found, what type of edge each will be, how the road is curving or changing just ahead, etc. The ROAD UNIT FINDER issues instructions to the PERCEPTION subsystem to find the road edges that are described. When PERCEPTION has found them and reported them, the ROAD UNIT FINDER will cut the perceived road stretch into a convenient sized unit (say, 20 feet long). This will be called a road unit, and will be processed by the lower stages of the PILOT. The ROAD UNIT FINDER reports back to the

DRIVING MONITOR the size and characteristics of the road unit it found, and another prediction will be produced to begin the ROAD UNIT FINDER cycle again.

The *ROAD UNIT NAVIGATOR* is a small and simple module that watches for the road units to be posted into the Local Map database. As each one appears, the ROAD UNIT NAVIGATOR will plot *local path constraints* that define the possible driving area on that road unit. For example, a road unit may be a stretch of roadway; the local path constraints may be to stay on the right-hand side of the roadway.

The *OBSTACLE FINDER* takes each new road unit and looks for obstacles within the legal driving area defined by the local path constraints. It issues a call to the PERCEPTION subsystem to sweep the appropriate area for obstacles sticking up from the ground (or dug into it). At present we are only considering stationary obstacles. The list of obstacles will be posted to the database.

The *LOCAL PATH PLANNER* marks the end of the driving pipeline. This module looks for local path constraints and obstacle lists, and plots a local path from the vehicle's current position, through all intervening road units, and reaching to the far edge of the newly scanned road unit. The path is passed on to the HELM.

In a separate, tight real-time loop, the *HELM* module controls the vehicle and monitors its movement, updating the vehicle position estimate from dead reckoning and inertial navigation. As the vehicle exits each road unit, the HELM reports this to the DRIVING MONITOR. The HELM receives new local path information from the LOCAL PATH PLANNER whenever a new local path is computed.

The action of the driving pipeline is illustrated in Figure 3-5. In addition to the ROAD UNIT FINDER and ROAD UNIT NAVIGATOR, there are an *INTERSECTION FINDER* and *NAVIGATOR* and a *CROSS-COUNTRY FINDER* and *NAVIGATOR* that will be invoked for the corresponding types of route segments. They will, of course, find the appropriate kinds of *driving units*.

In practice, the ROAD (etc.) UNIT FINDER and OBSTACLE FINDER, which call upon PERCEPTION, will be by far the most time-consuming steps; the effective pipeline length is thus two driving units long. Taking into account the driving unit on which the vehicle is currently located, our estimates are that road edge finding will be looking about 40 to 60 feet ahead of the vehicle and obstacle scanning will take place about 20 to 40 feet ahead. If the vehicle overdrives the sensor processing, the HELM will ensure that it stops before passing the end of the current local path.

Although this PILOT subsystem is concerned with vehicle control and path planning, it embodies a

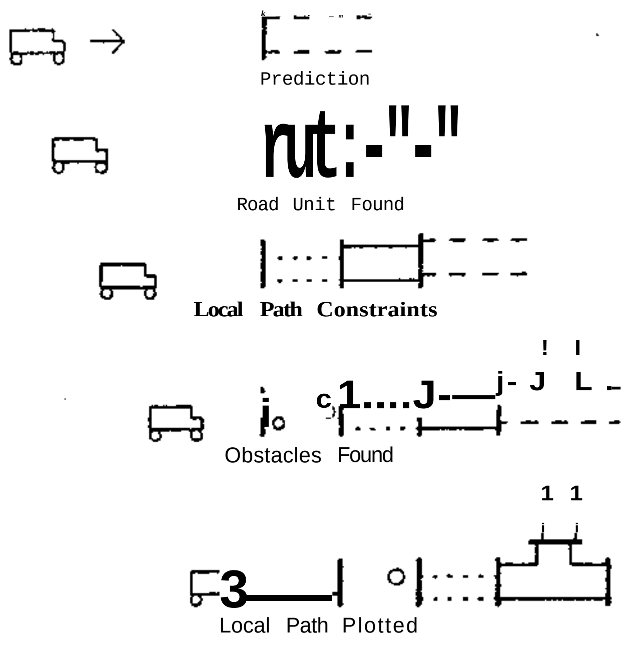


Figure 3-5: The Driving Pipeline in Action

very important aspect of sensor fusion. It would be possible, for example, to simultaneously scan the road ahead for (2-D) road edges and (3-D) obstacles. However, this would involve a rather exhaustive search over the 3-D data for obstacle detection. By finding road edges first and plotting local path constraints, the required obstacle scanning area can be cut down to a small fraction of what would otherwise be necessary. An order of magnitude speedup in obstacle detection is obtained by ordering the use of the sensors and world knowledge.

3.5 The PERCEPTION Subsystem

The *PERCEPTION* Subsystem receives perceptual requests from the planning portions of the system (the LOOKOUT and PILOT), and analyzes sensor data to compute responses. PERCEPTION will probably account for well over 90% of the total computational load of the NAVLAB system.

Every possible way to speed up perceptual processing must be utilized. One mechanism used to do this is to provide as much top-down guidance from the planning modules as possible. Our system structure facilitates this, and we (along with other cooperating sites) have adopted a small repertoire of requests that planning modules can make of the perceptual subsystem:

- Find a driving unit of a particular type (road unit intersection, or cross-country terrain patch)* In general, a predicted location and characterization of the expected driving unit will be available

- Fire! obstacles above the ground within a specific volume of space.

- Find landmarks. Initially, we will only detect specific objects at a specific location; eventually, we will deal with general object types and unknown or loosely constrained locations.

In this list, the only "open-ended" requests are to find landmarks whose identity or location are not predictable. In all other cases, the perceptual processing is being used to answer very specific queries.

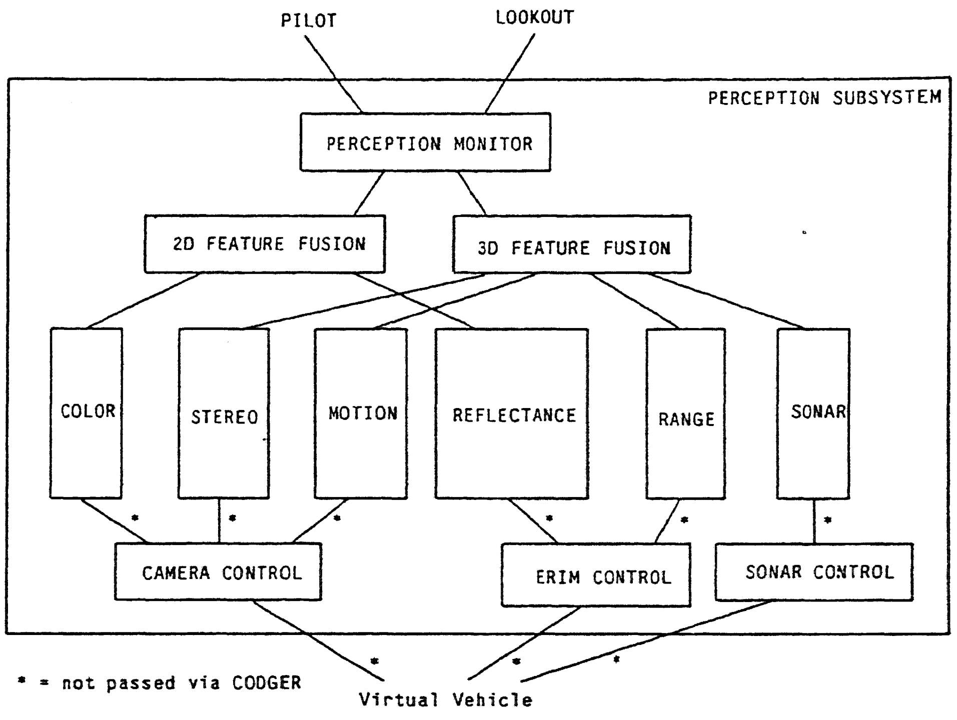


Figure 3-6: The PERCEPTION Subsystem Plan

Figure 3-6 illustrates the modules we plan for the PERCEPTION subsystem. This is not our initial system, which will be much simpler; it represents our plan for a relatively modest but general perceptual system. It includes four levels of modules.

The *PERCEPTION MONITOR* will receive the perceptual requests and call upon 2-D and 3-D feature finders to satisfy each request. Initially, our PERCEPTION MONITOR will use pre-stored strategies for perception; eventually, we would like to develop a more general language for describing such strategies so a wider variety of objects can be perceived.

The primary sensor fusion level presents two *virtual sensors*: a virtual 2-D (image) sensor and a virtual 3-D sensor. At the level of these *2-D* and *3-D FEATURE FUSION* modules, the objects being manipulated are road edges, 3-D surfaces, etc. Thus, sensor fusion will take place on fairly completely interpreted data rather than on raw or intermediate sensor data.

The workhorse modules, consuming probably well over 90% of the computational cycles of the NAVLAB, are the *SENSOR DATA PROCESSING* modules. There is one for each complete sensor modality, such as "range analysis of rangefinder data'[1] or "motion analysis of images". Each SENSOR DATA PROCESSING module will receive raw sensor data (obtained directly from sensors without shipping it to and from the database!), and is responsible for all levels of processing up to and including 2-D and 3-D features for sensor fusion. For example, a "stereo image analysis" module will include image preprocessing, edge flnding, edge linking, stereo matching, 3-D grouping, and 3-D surface fitting; only the finally produced 3-D surfaces will be passed along to the 3-D FEATURE FUSION module. In this way, we can minimize the bandwidth required of the CODGER database system and we can plug in and out experimental perceptual modalities rather easily.

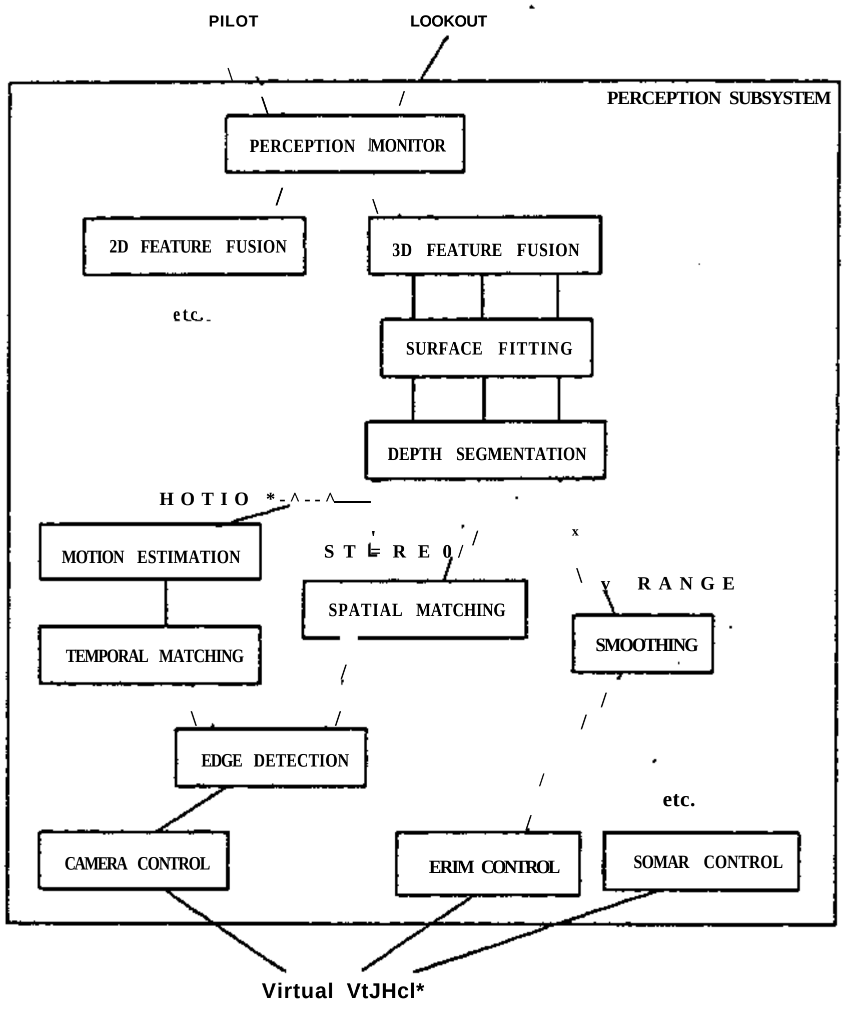


Figure 3*7: Future PERCEPTION Subsystem Plan

As we begin to better understand what specific methods of sensor data processing are desirable, we will evolve towards a perceptual subsystem as shown in Figure 3-7. In this system, each step of perceptual processing would be an independent module and each modality is simply a flow of data from raw sensor input to 2-D or 3-D features. This is beyond our current reach for both scientific reasons--such as the different spatial resolution of 3-D data from image motion and rangefinder analysis and how that difference impacts 3-D segmentation--and technical reasons--the speed and bandwidth limitations of the whiteboard system. We will explore the scientific issues as we build the NAVLAB system, and we have plans to overcome the technical limitations with new hardware, software, and CODGER implementation strategies.

The lowest level of the perceptual system is the actual *SENSOR CONTROL* code that aims the sensors, collects data, and provides appropriate coordinate system and time stamps for the data.

4. Methods for Sensor Fusion

In the course of laying out the NAVLAB system architecture, we have identified three primary methods for sensor fusion, each of which is used within the NAVLAB system:

- *Competitive.* Each sensor generates hypotheses; these hypotheses may reinforce or conflict with each other during fusion. Competitive fusion is typically used for sensors producing the same type of data, such as two sonar sensors working together with overlapping fields of view [15].

- *Complementary.* The different sensors are used in a strategy that applies each one individually. The strategy uses the relative differences of the sensors to enhance the advantages and cover the disadvantages of each. For example, in our system we use color images to find road edges, which is relatively easy, and we later scan the road area for obstacles with a 3-D sensor such as a rangefinder. We thus use the easy processing first to limit the search area required for the more sophisticated sensor.

- *Independent.* A kind of degenerate method in which one sensor or another is used independently for each particular task.

Each of these methods can be applied at any level of the system, and can be applied to actual sensors or to sensor interpretation modalities. For example, a rangefinder and color camera can be fused at a low level to produce a registered color + range image. This would be a low-level application of complementary fusion of sensors. In our system, we will fuse 3-D surfaces etc. from range data and from motion and stereo analysis of images. This is high-level fusion of sensor modalities; we can do this either competitively with competing parallel hypotheses, or complementarily with strategies to invoke each modality when its peculiar characteristics (such as spatial resolution or processing speed) make it most appropriate. Some systems have emphasized the reasoning aspect of sensor data interpretation, using competitive or complementary fusion within a general inference framework [5, 13, 16].

Another issue in sensor fusion is the strategy used to invoke sensors or sensor modalities. This can be:

- *Fixed.* A strategy embodied in special-purpose code that specifically invokes sensor processing.

- *Language-Driven.* A strategy defined in a general "perceptual language"; each object in the object database will have a description of how to recognize that object. This approach would be much more general than a fixed strategy, but developing an appropriate language for perceptual planning is a major research task [8, 9, 10].

- *Adaptive.* A language-driven system with the ability to select alternative plans based on the current status of sensor processing or the vehicle's environment. For example, an orange traffic cone has both a distinctive color and shape. Ideally, if we have already acquired a color image of an area, we might begin by looking for the orange color; but if we already have acquired or processed 3-D data for that area we might be better off to start by looking for the shape of the cone.

Both of these issues-sensor fusion method and perception strategy-are possible with either bottom-up, top-down, or a combined perception methodology. For example, in our system we provide top-down guidance to the top level of the PERCEPTION system whenever possible. This will almost certainly result in top-down queries being addressed to the 3-D FEATURE FUSION and 2-D FEATURE FUSION modules. However, we can implement those modules themselves to either pass top-down sensor data processing instructions to the actual perception modules, or to accept data from those modules in a bottom-up manner.

In the NAVLAB system there are many examples of sensor fusion:

- *Vehicle Position Determination.* The dead-reckoning system and inertial navigation will provide a continual estimate of vehicle position, but this will drift over time. From time to time, landmark sightings will be used to correct the absolute position estimate. We may also include an intermediate position estimation method using visual motion analysis. This system is an example of complementary sensor fusion with a fixed strategy.

- *The PILOT Subsystem.* As mentioned above, we will use 2-D road edge information to find navigable areas, then scan those areas with 3-D sensors (see below) to detect obstacles. This is complementary, fixed-strategy sensor fusion at a very high level.

- *The PERCEPTION Subsystem.* There are several kinds of sensor fusion within our PERCEPTION subsystem:

    o In the preliminary system, perception is performed by modules such as "road edge finding by color image analysis" or "obstacle finding by rangefinder analysis" [2,7,21]. These special-purpose modules embody independent, fixed-strategy fusion (i.e., the modules do not individually perform fusion, but the system as a whole exercises more than one sensor).

    o In the system we are developing (Figure 3-6), the recognition planning will initially be complementary and fixed-strategy, with a separate strategy for each object or roadway type; we are working on a language to allow language-directed object and roadway recognition.

    o Our virtual sensors, as embodied in the 2-D FEATURE FUSION and 3-D FEATURE FUSION modules, will probably use fixed-strategy competitive processing. However, there is some complementary processing even at this level. For example, the 3-D FEATURE FUSION module will find obstacles when called upon to do so. The rangefinder data is ideal for this purpose, since the raw data is a depth map. However, our rangefinder has relatively poor spatial resolution, and at a distance of 40 feet or more will not be able to reliably detect (6-inch) obstacles. When an uncertain obstacle hypothesis comes in from the rangefinder, we plan to have the 3-D FEATURE FUSION module invoke image motion analysis using a camera with a telephoto lens to provide a verification or denial of the hypothesis based on high-resolution images. Thus, even subtle differences between sensors (in this case, two depth-map-based modalities with different processing times and spatial resolutions) can be exploited to produce total systems with improved performance. In this case, complementary sensor fusion produces a system whose overall performance will be better than the sum of the individual parts [17].

- In the perceptual system plan outlined in Figure 3-7, fusion of sensor modalities may take place at lower levels. The same issues of fusion method and strategy still arise, and we may have virtual sensors at several levels, e.g. a virtual edge sensor, a virtual depth map sensor, and/or a virtual surface sensor (similar to our current 3-D FEATURE FUSION module). As stated earlier, there are substantial scientific issues to be addressed before such a system can be built.

5. Conclusions

We have described in this paper the NAVLAB vehicle, the CODGER whiteboard system which resembles a blackboard but with a parallel control flow, and the NAVLAB software structure which contains a simple navigation and planning system, a "driving pipeline," and top-down guidance of perception. The perceptual system itself has a central top-level interface to the rest of the system, sensor fusion of thoroughly processed 2-D and 3-D data, and several sensor modalities which are each completely self-contained modules of the system. We expect perception to consume well over 90% of the computational time in the complete system.

Sensor fusion takes place at a number of places in the NAVLAB system. We have identified these and produced a simple classification of sensor fusion methods and strategies. In particular, complementary use of different sensor modalities seems to produce systems and subsystems whose performance can be better than that of any individual modality (see also [17]).

Our plans for the future development of the CODGER system include creating a CommonLISP interface (the package is currently accessible from C) and speeding up the system immensely by geometric indexing, distributing the database itself, and using special new hardware and operating system software under development at CMU.

For the NAVLAB, our future plans call for dealing with moving objects, improving the hardware, implementing a more sophisticated reasoning and planning system, and most of all, improving the perceptual subsystem. We will be increasing the performance of perception by using new hardware and software, restructuring the system as described in this paper, and reducing our reliance on specific advance knowledge in the global map and object databases.

5.1 Acknowledgements

The NAVLAB project is a sizable effort at CMU, and contributions to the architecture have been made by the entire group. Takeo Kanade and Yoshimasa Goto, in particular, have provided input to this design effort.

We have also benefited from discussions with other sites involved in the DARPA Strategic Computing Vision program, and plan to receive some key pieces of the NAVLAB system from these sites in the future. The target NAVLAB system will be a community effort.

6, Bibliography

[1] Bisiani, R.
AGORA: An Environment for Building Problem-Solvers on Distributed Computer Systems.
In *Proc. Distributed AI Workshop.* AAAi, SearangeCA, December, 1985.
Available from CMU Computer Science Dept.

[2] Elfes, A. E.
Sonar Navigation.
In *Workshop on Robotics.* Oak Ridge National Lab, Oak Ridge TN, August, 1985.

[3] Erman, L. D., Hayes-Roth, F., Lesser, V. R., and Reddy, D. R.
The Hearsay-Il Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty.
ACM Computing Surveys 12(2):213-253, June, 1980.

[4] Erman, L., Fehiing, M., Forrest, S., and Lark, J. S.
ABE: Architectural Overview.
In *Proc. Distributed AI Workshop.* AAAI, SearangeCA, December, 1985.
Available from Tecknowledge Inc., Palo Alto CA.

[5] Garvey, T. D., Lowrance, J. D., and Flschler, M. A.
An Inference Technique for Integrating Knowledge From Disparate Sources.
In *Proc. UCAI'81,* pages 319-325. Vancouver, BC, August, 1981.

[6] Harmon, S. Y., Gage, D. W., Aviles, W. A., and Bianchini, G. L.
Coordination of intelligent Subsystems in Complex Robots.
Technical Report, Naval Ocean Systems Center, Autonomous Systems Branch, San Diego, September, 1984.

[7] Hebert, M. and Kanade, T.
Outdoor Scene Analysis Using Range Data.
Sn *Proc. 1986 IEEE Conf. on Robotics and Automation.* 1986.
Submitted.

[8] Henderson, T_M Shilcrat, E, and Hansen, C.
A Fault Tolerant Sensor Scheme.
UUCS *83-QQ3$_f$ U. Utah* Dept of Computer Science, November, 1983.

[9] Henderson, T. C.$_f$ Fai, W. S., and Hansen, C.
MKS: A Multiseosor Kernel System.
IEEE Trans, on Systems, Man, and Cybernetics SMC-14{5):784-791$_5$ September, 1984.

[10] Henderson, T.$_f$ Hansen, C., Grupen, R_M and Bhanu, B.
The Synthesis of Visual Recognition Strategies,
UUCS' 85-102, U. Utah Dept of Computer Science, October, 1985.

{11] Kanade, T_M Thorpe, C, ami Whittaker, W.
Autonomous Land Vehicle Project at CMU.
in Proc. 1988 ACM Computer Conference. Cincinnatti* February, 1986.

[12] Kung, H. T. and Webb, J. A.
Global Operations on a Systolic Array Machine.
In *IEEE Proc. of International Conf. on Computer Design: VLSI In Computers,* pages 165-171. IEEE, Port Chester NY, October, 1985.

[13] Lowrance, J. D. and Garvey, T. D.
Evidential Reasoning: Am Implementation for Multisensor Integration.
TN 307, SRI, December, 1983.

[14] Moravec, H. P.
The Stanford Cart and the CMU Rover.
Proc. IEEE 71 (7), July, 1983.

[15] Moravec, H. P. and Elfes, A. E.
High Resolution Maps from Wide Angle Sonar.
In *Proc. 1985 IEEE Intl. Conf. on Robotics and Automation,* pages 116-121. St. Louis, March, 1985.

[16] Nii, H. P., Feigenbaum, E. A., Anton, J. J., and Rockmore, A. T.
Signal-to-Symboi Transformation: HASP/SIAP Case Study.
AJ Magazine , Spring, 1982.

[17] Shneier, J., Nagalia, S., Albus, J., and Haar, R.
Visual Feedback for Robot Control.
In *IEEE Workshop on Industrial Applications of Industrial Vision,* pages 232-236. May, 1982.

[18] Stentz, A. and Shafer, S. A.
Module Programmer's Guide to Local Map Builder for NA VLAB.
Technical Report, CMU Robotics Institute, 1986.
In preparation.

[19] Thorpe, C.E.
FIDO: Vision and Navigation for a Mobile Robot.
PhD thesis, CMU Computer Science Dept, December, 1984.

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

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