The Driving Pipeline:
A Driving Control Scheme for Mobile Robots

Yoshimasa Goto, Steven A. Shafer, Anthony Stentz

CMU-RI-TR-88-8

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

June 1988

This research was sponsored by the Defense Advanced Research Projects Agency, DoD, through ARPA order No. 5682 and monitored by the U.S. Army Engineer Topographic Laboratories under contract DACA76-86-C-0G19. 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 Research Projects Agency or of the U»S, Government

Table of Contents

List of Figures

Abstract

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 paper presents a new 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.

1. Introduction

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 1-1, 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.

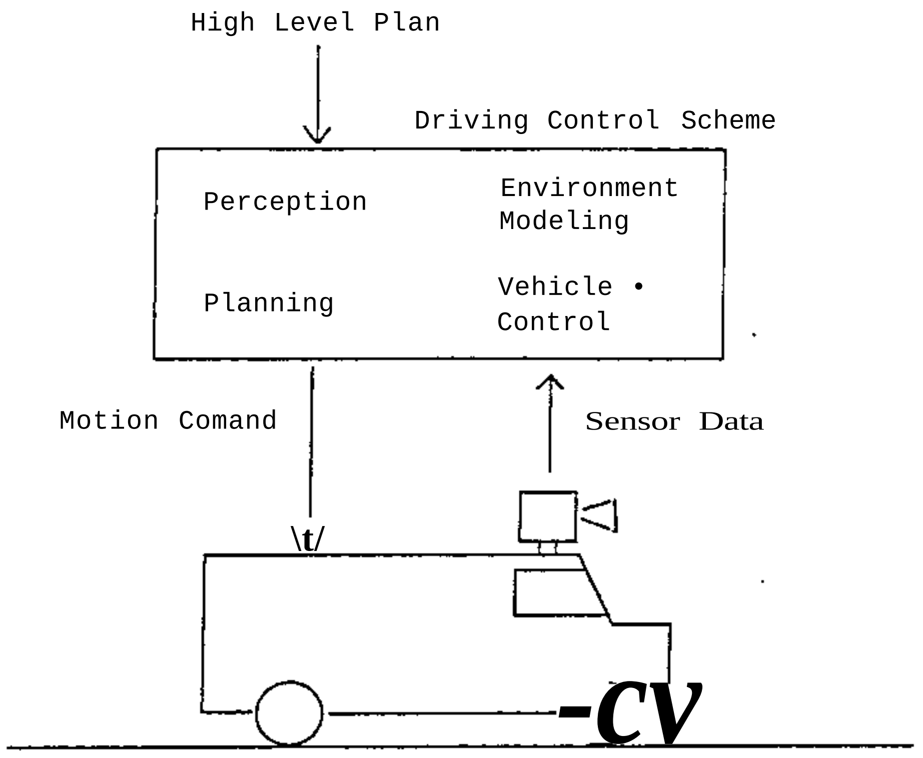


Figure 1-1: Driving Control Scheme

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

compulation and maintain effective coordination among numerous perception and planning processes.

• Parallel Execution: For reai-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 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 die senses- view frames* execution intervals, and vehicle speed.

A&bcnigh several mobile robot systems have been built in the past, they did not address driving control scbeooe very deeply. Stop-and-go motion, although it does incorporate all of the primitive processing steps, deliberately avoids the problem of continuous motion control [2,4,7,10]. Waxman et al. mentioned the necessity for vehicle ^eed adjustment using knowledge, but didn't show any method for doing so [11]. Brooks developed a kyeei control stnKto« thai drives a vehicle continuously [1]. However, it does not have the ability to adapt the control to n»e£ the dimping needs of perception. Dickmarms and Zapp develped a system for high-speed navigation on the German Autobahn PI* TMs system tracks simple visual features (e.g., white lines bordering the road) and cannoc bo easiy cxioicicd to handle more difficult percepfsiai scenarios.

To solve these pottons, we hive developed the concept of the Driving Pipeline and verified it in two ajxrimettai mobile robot syacms: the Terregator awl the NAVLAB. This paper describes the Driving Pipeline* including the component concepts of the Driving Unit and the Execution Pipeline, and describes our experiments wife these vehicles.

2. Processing Steps and Driving Unit

We iviic ihe *mmpxtMm* necessary for driving control into the following primitive processing steps:

• TJ» Prtilctioa step plans the wet that the vehicle wii move into next

• The Perctpttea step detects avigablt *mm* boundaries and obstacles using sensor data,

• The lUrrlnxuMBt Modeling siep makes a description of the vehicle environment and updates the estimate of the vehicle position*

• The Local Pith Ranting *step* pitas the vehicle trajectory*

• The Vthkke Contra! &tp drives the vehicle mechtasm*

Ttics* *ite^i* must each execute &n uun to process each area of terrain that the vehicle will traverse.

We detetoped the *zmttpi* of the *driving ana* to indicate the area that each primitive step will process ORC« JI **each execution** cycle. The vehicle's eaare route is divided into driving units which arc passed, one at t time, © sac ft **of the prim**it:vc processtug step, la Uus My, planning and perception arc synclbronizeci to provide driving ccw-5-

2.1. Prediction and the Driving Unit

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 2-1).

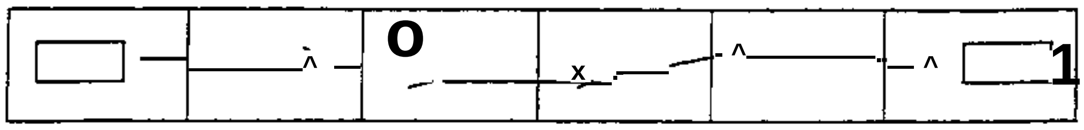


Figure 2-1: Sequence of Driving Units

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 eonsequtive 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.

7JL Perception and 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 infoimation. 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.

23. 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 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.

• 2.4. 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 2-2, 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.

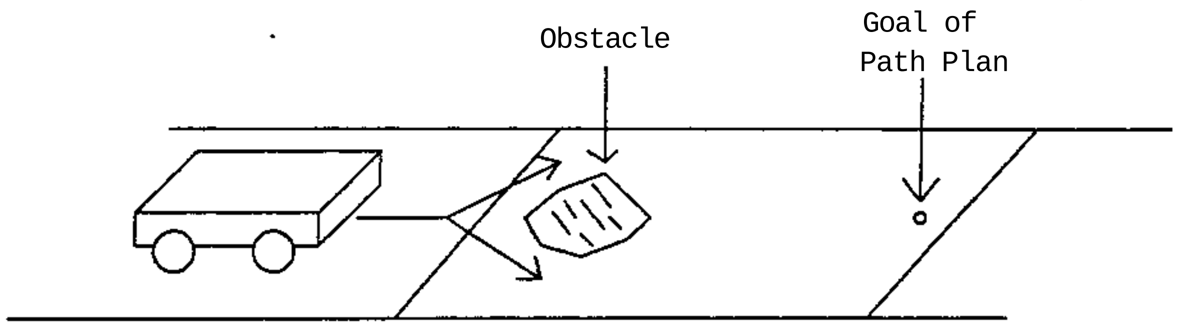


Figure 2-2: Driving Unit Size for Vehicle Maneuvering

The Driving Pipeline includes two levels of path planning: the driving passage from the Prediction step 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 *basal* on perception results *fmm* tte previous driving units. This is the reason why we include planning the vehicle passage in the the Driving Pipeline level of the system rather than in a higher level.

2.5. 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 in Section 3.4.

3. 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.

3.1. 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 morion needs a Driving Pipeline to process multiple driving units in parallel.

The Driving Pipeline supports continuous vehicle motion by using *pipelined execution.* As described in Section 2, 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 3-1 illustrates the pipeline execution of the Driving Pipeline as follows:

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

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 feanires 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.**

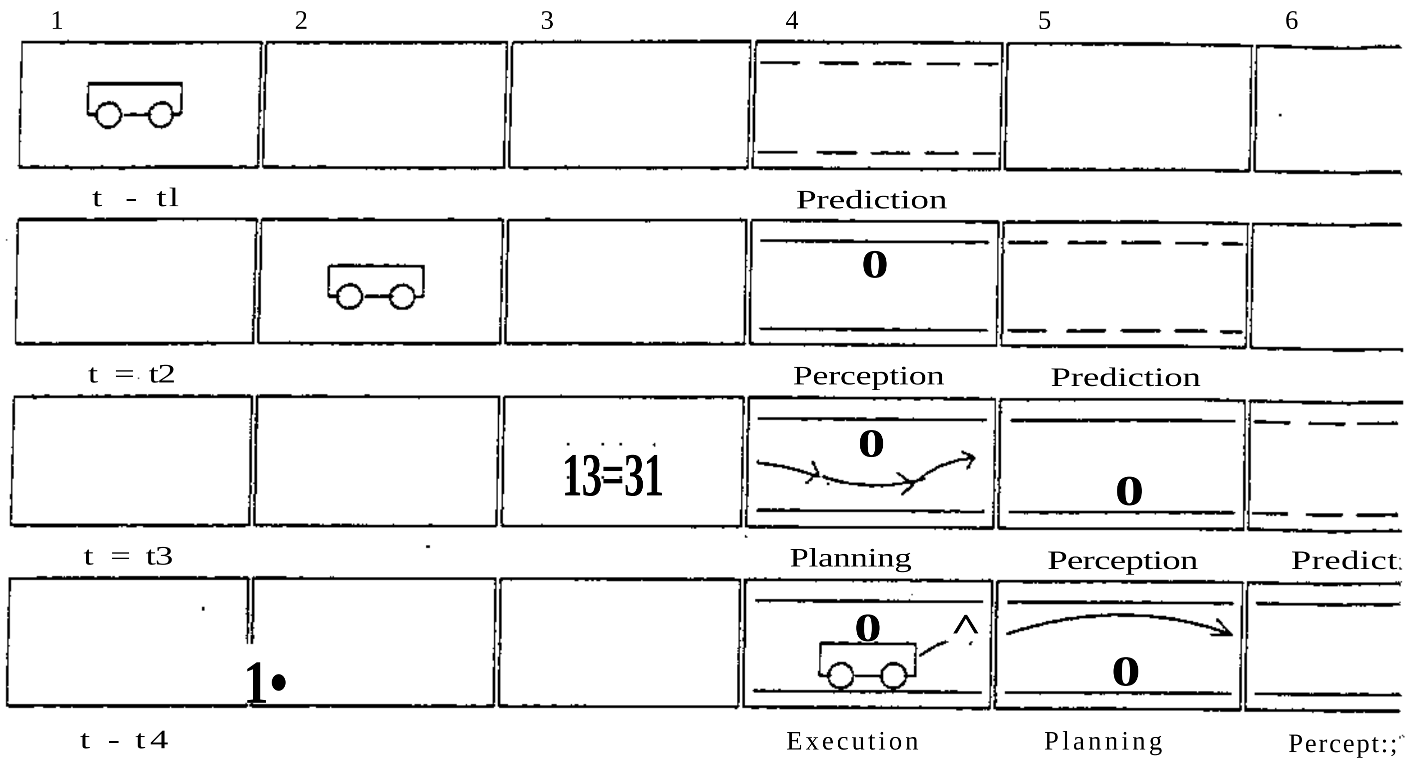


Figure 3-1: Pipelined Execution of the Driving Pipeline

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 pennits 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[11] 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.

3.2. 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.

As mentioned in Section 2.4, 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.

33. 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 3-2 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 *st&p.*

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

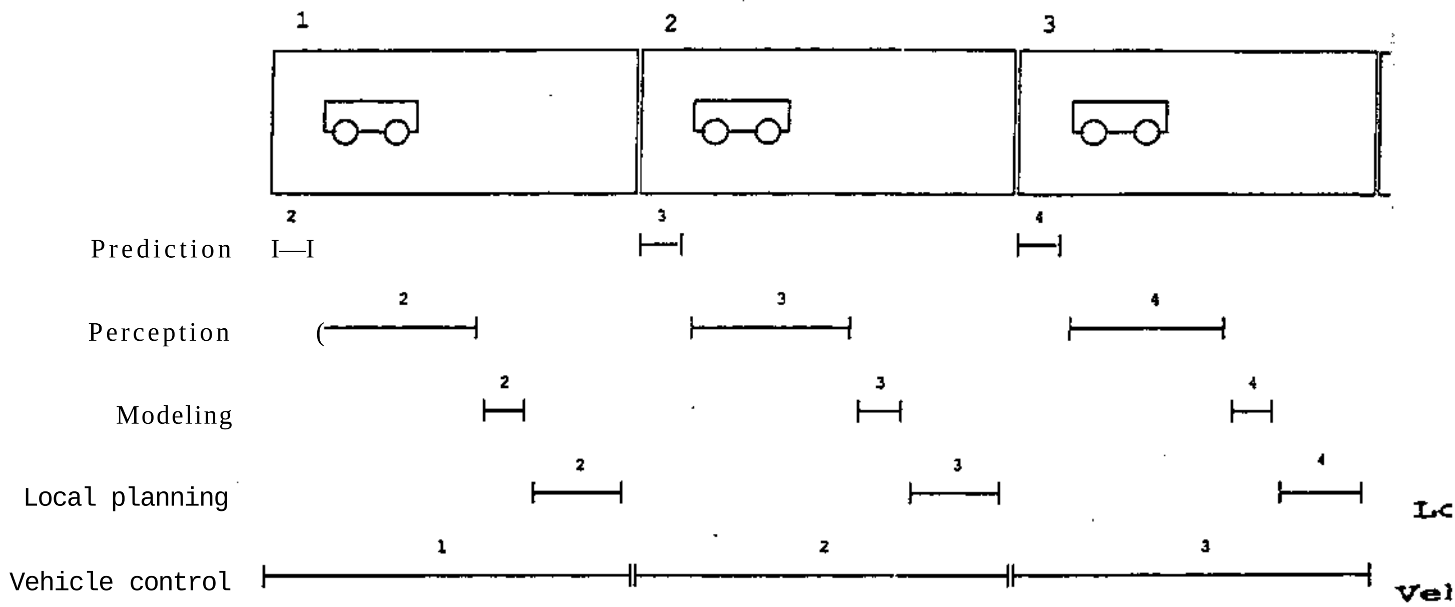


Figure 3-2: Badly-Balanced Execution of the Driving Pipeline

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 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 3-3. In this case, since all processing steps are ready to woik on the next driving unit just after finishing the current one, complete pipelined execution is achieved

In the map building mode, the map database does not have enough information about the unscanned areas, so the Prediction step *netds 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 uniL The resulting execution pattern is illustrated in Figure 3-4. 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 is able to produce the 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 allowmg large-driving units.

In both execution modes, the scanning position is a key factor in maintaining these parallel execution patterns because Jt regulates the execution patterns. The Environment Modeling step, the Local Path Plan step, and tite

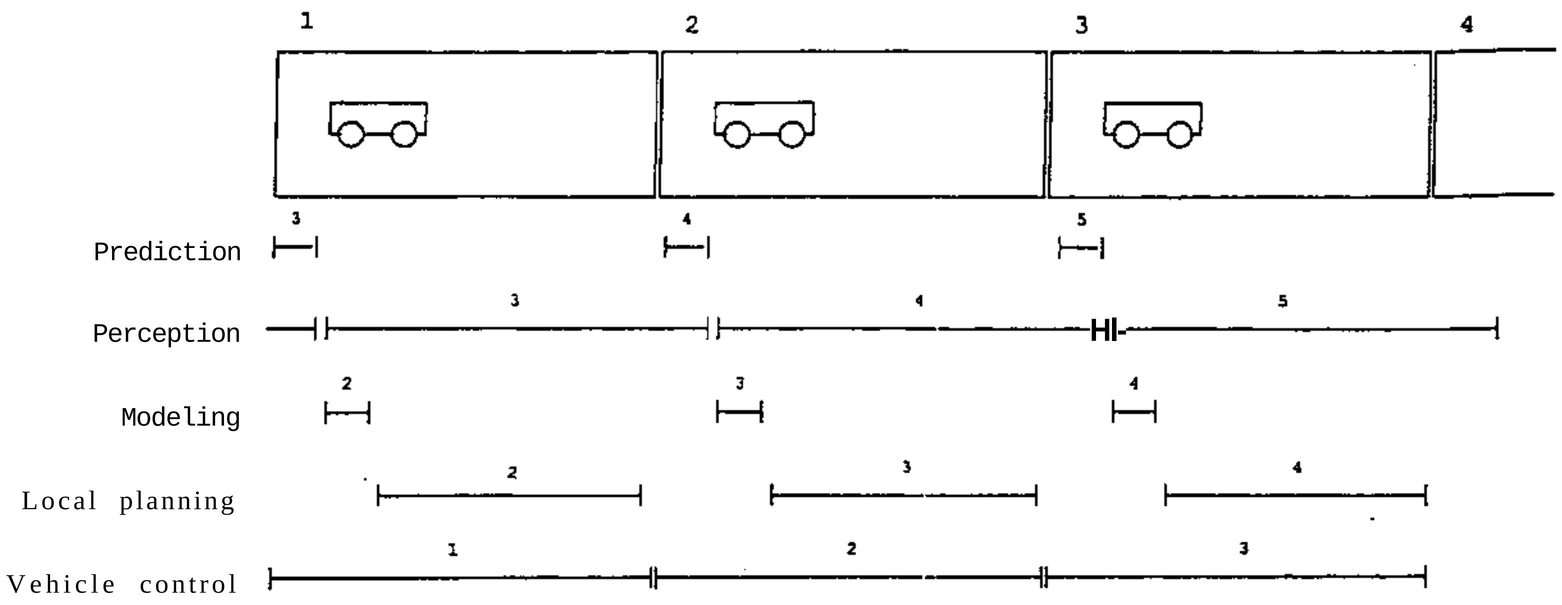


Figure 3-3: Parallel Execution Pattern in the Map Navigation Mode

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 3-3 and 3-4, are given by the following equation:

$$\textit{scanning distance} = \frac{T_p}{T_c} \cdot L_i \qquad (1)$$

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 lo travel one driving unit. In the map navigation mode, the cycle time is determined as:

$$T_c = T_m \qquad (2)$$

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

$$T_c = Max(T_n, T_t) \qquad (3)$$

where

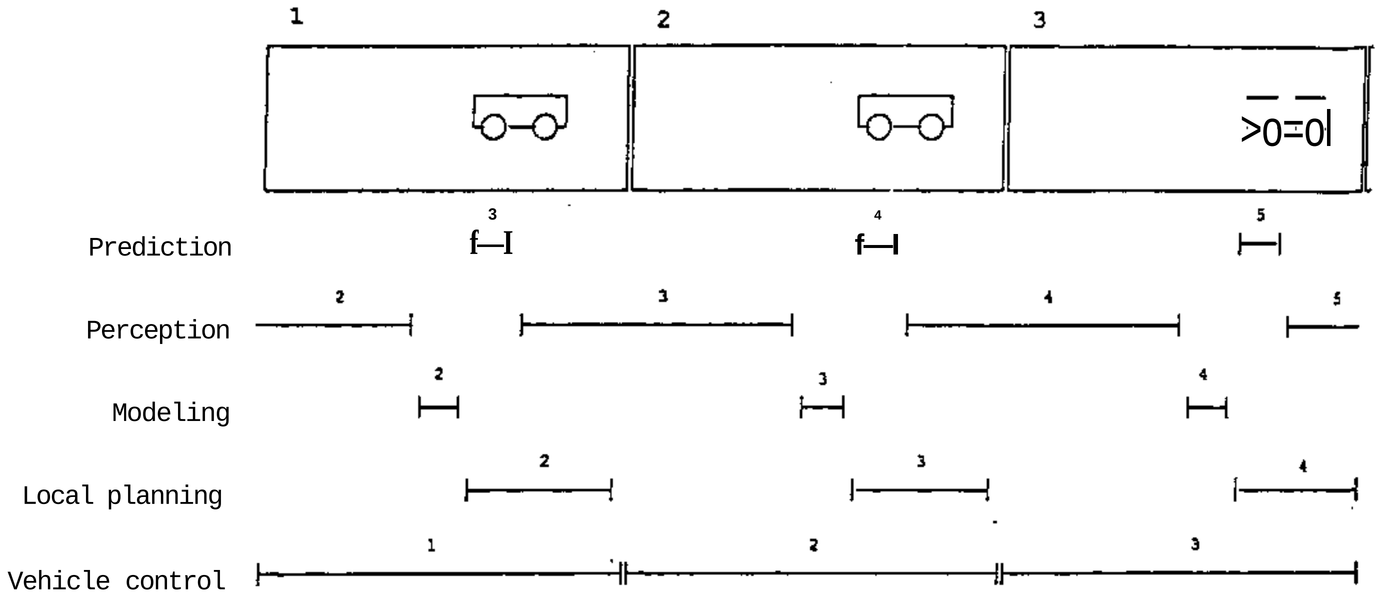


Figure 3-4: Parallel Execution Pattern in the Map Building Mode

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 rime," 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 w«h the processing of this sequence for consecutive driving units. Therefore, this execution sequence behaves Hcs one individual processing step. That is the reason for the above equation for the map building mode.

3A Vehicle Speed and Driving Pipeline

The Vehicle Control step must take into account the execution time of aU the processing steps in order to achiew 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 so achieve it.

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

$$\textit{vehicle speed} < \frac{L_i}{T_c} \qquad [illegible]$$

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 in Section 33, the cycle time is given by Equations 2 and 3. Then the above equation can be rewritten as follows:

in the map navigation mode,

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

and in the map building mode,

$$\textit{vehicle speed} = \frac{}{Max(T_m,T_t)} \tag{6}$$

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 in Section 2.2. 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:

$$\textit{vehicle speed} = \frac{D_s}{T_P} \tag{7}$$

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:

$$\textit{vehicle speed} = \frac{D_r}{T_r} \qquad (8)$$

D_r» *remaining distance of iacai path plan*

T_r *9 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 sseps. Whenever these processing *steps* finish processing a driving unit, T_r and D_f 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.

4. The Driving Pipeline in Action: Experimental Results

4.1. Implementing the Driving Pipeline

We have developed and tested *the* Driving Pipeline through building several experimental mobile robot system* called *Sidewalk System 2, Sidewalk System 3,* and the *Park System,* [5] [6] [9]. The Sidewalk 2 and the Sidewalk 3 systems -drive an experimental vehicle called the Teiregator on the network of sidewalks on the campus of Canwgfe J4elto University. Hw Park fyssera drives the NAVLAB, a computer-controlled van, on a road in Scfaentey Rat adjacent to Cantegie Mellon. Figure 4-1 shows these vehicles, which are both equipped with color TV cameras and a kstf *rmgt* scaacr made by ERIM. WMk the Tenegator is linked to several SUN-3 workstations is te iabofai&xy with iBdto communicatiofi and cables, the NAVLAB carries four SUN-3s on board- In the remainder of *MM* chapter, we will dtacrite prim«iiy SkJewalk System 3 because it demonstrates the Driving Pipeline matt ckariy.

Figure 4-2 stiwi ifae module s&uctve of Sidewalk System 3. The processing steps axe implemented « individual jrajpus and *m* Uakad ffrosfii lie COIXJER distributed database, a system-boilding tool wrtoi *m Cmmgm* Mdla to support torgc-pwii pmBdism fior n»blc rdx>t navigation [8]. CODGER mal^ *it nbtinty* any *m* teald te Dilwqi Pipeline; tea^ of its a ^ i i i ^ to aipjxw parallel processing among multiple compatts. At! of tte lynems memtoned above ina CODGER in this way.

42, Processing Steps and Driving Units

Fq^it 4-3 *ihc*$* i dsagasn cf ftc primUive pocttsfeg acps woMng on one driving unit in appraehiftg *M* .r^rsKi&n. Figure 4-3-1,- >^C"*s sis driving »tt placed by tic Prediction step. In Figure 4-3<b)» Ac tnpezttil il tte senior vie^ &ar-s ^r^: b> ±*t* Pcitcpwi ^p so COTOT the driving unit. Figure 4-3(c) shows the *nMck* **position estim**^ by tfce '•' **Modeling st**,p, The Vcbkk Council soep tove the vehicle as iHustrated in Figwe 4-3{#^

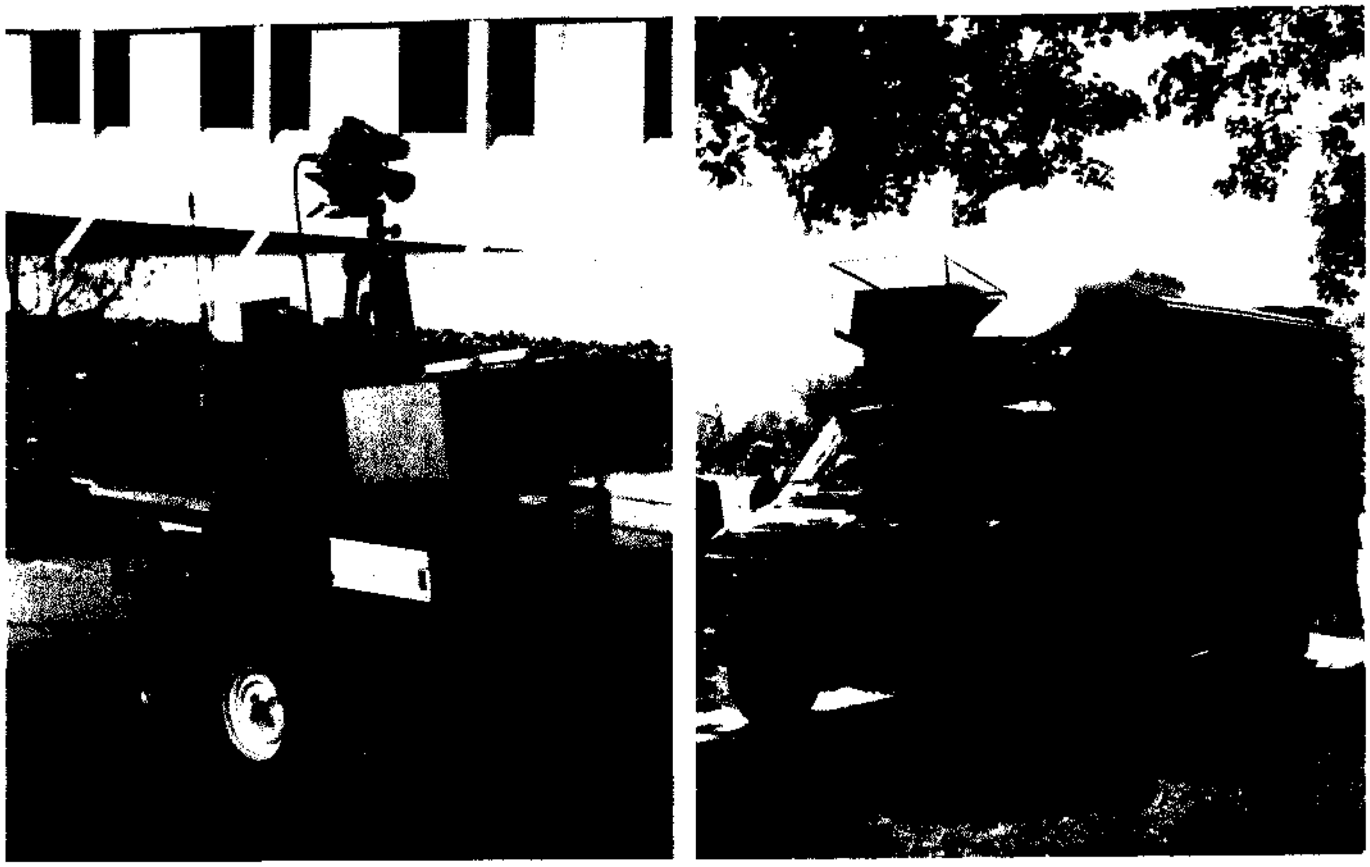


Figure 4-1: Terregator and Navlab

43. Pipeline Execution and Parallelism

Figure 4-4 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 for placing a new driving unit and the consecutive pipeline executions overlap completely. This *is* the "map navigation" mode described above. Because the scanning position and the vehicle speed were adjusted as described in Section 3.3, the most time consuming step (Perception) was the limiting factor in the cycle time of the system.

4A Execution Intervals

Because turning at intersections requires more accurate vehicle position estimation than following sidewalks, and because the Tenregator 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 4-5 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.

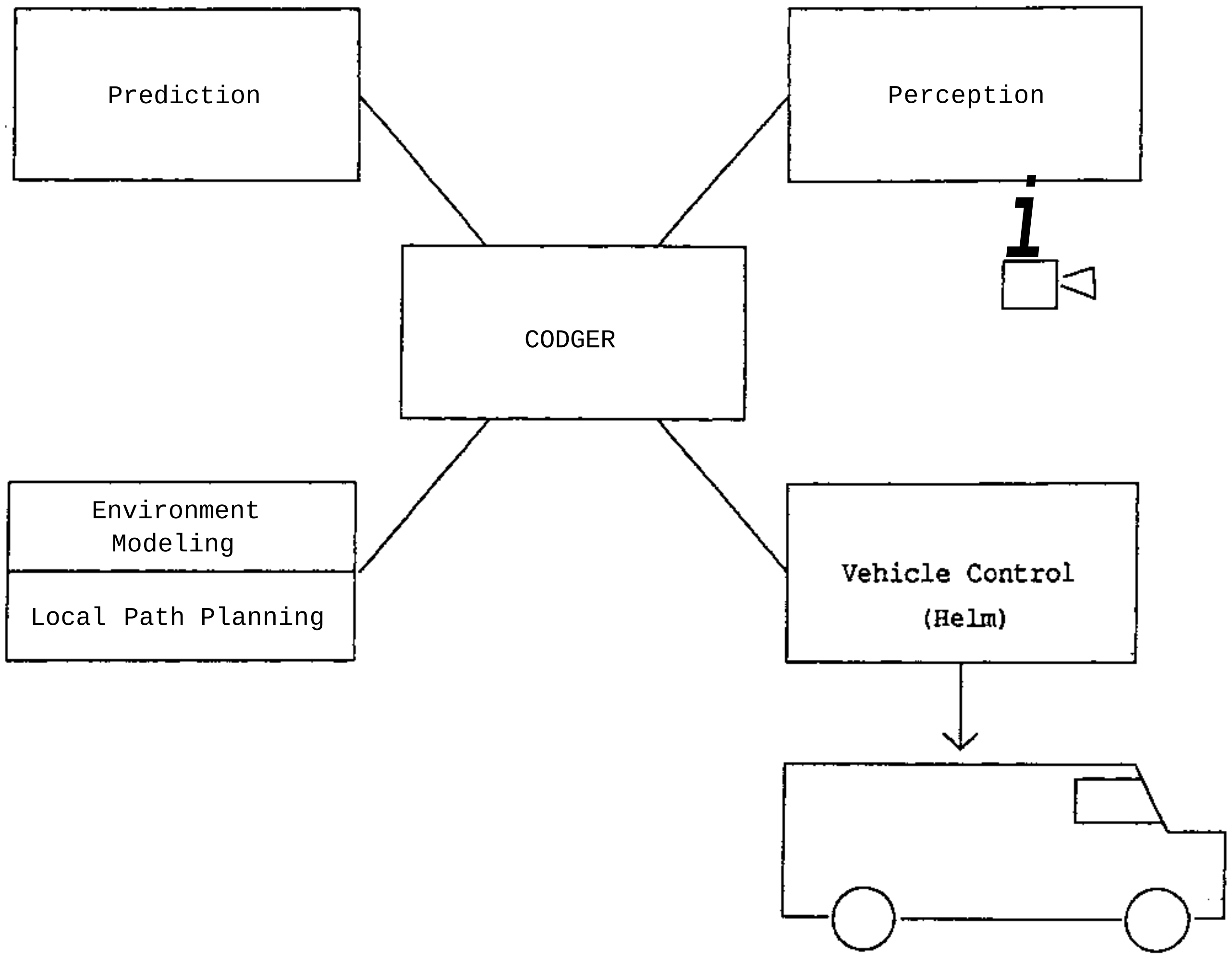


Figure 4-2: Module Structure

4.5. Vehicle Speed

Figure 4-6 shows a recorded vehicle speed that was adjusted according to Equation g. The vehicle speed was recalculated whenever the processlag steps were *dorm** The vehicle slowed down around the Intersection where the driving null Intervals were shorter and went tack to a high speed on *c suaight road where the driving unit intervals were longer. Because of the hardware limitations of the Tcucgaior vehicle, the vehicle speed could not be changed frequently; this is the reason that the recorded vehicle speed is not smooth.

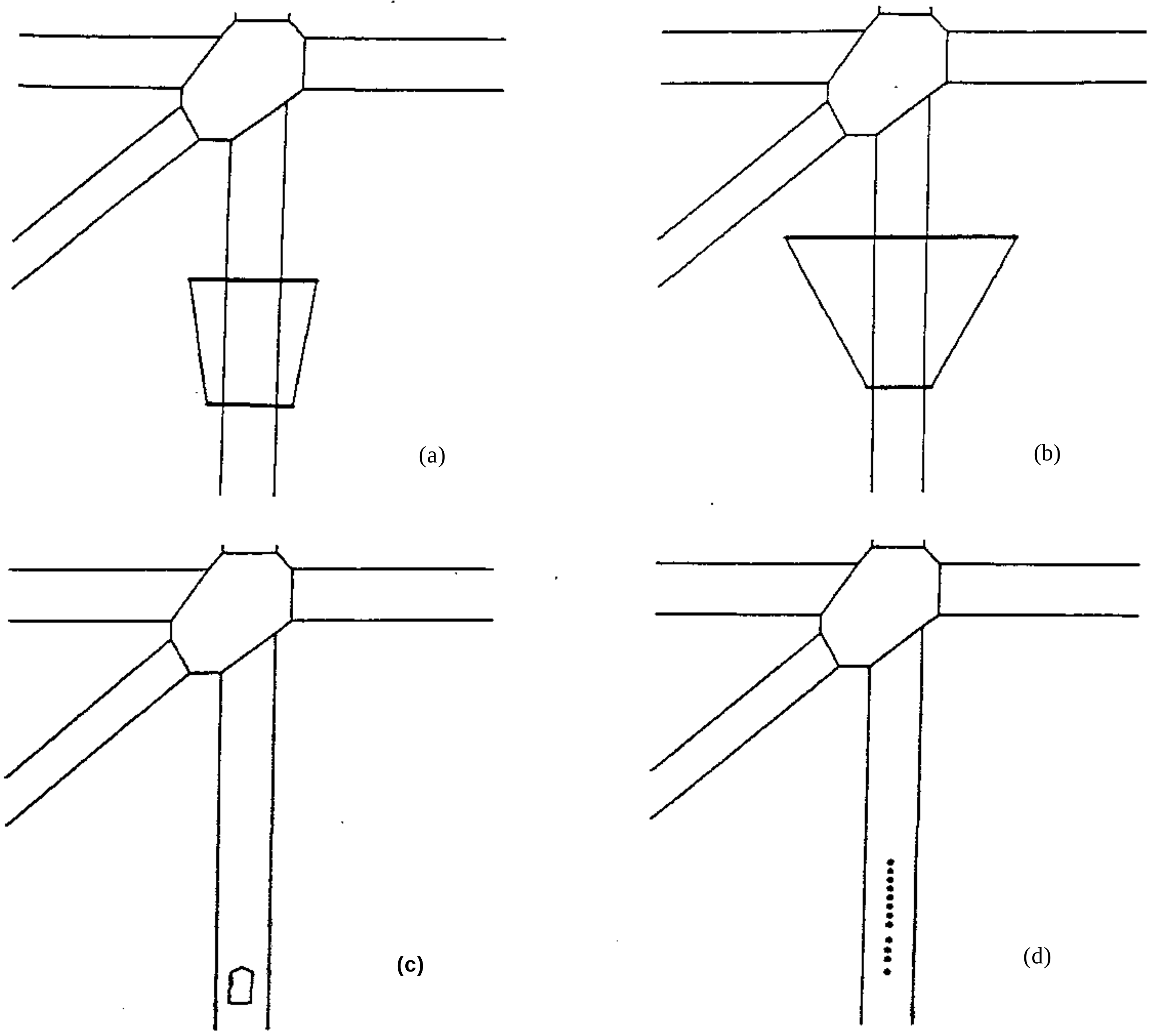


Figure 4-3; Processing Steps

4.6. Sensor Aiming

Our experiments on the Camegie Mellon campus test site showed the necessity for adaptive sensor aiming. The fixed sensor view frame citated a problem in tumiBg at the intersections, because the vehicle had to torn 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 ^w frame has to b© aimed so that it covers the vehicle's destination. In addition, the scanning distance must be different in following straight sidewalks an! in turning through intersections. In taming through an intersection, the vehicle position estimation must be accurate in both the

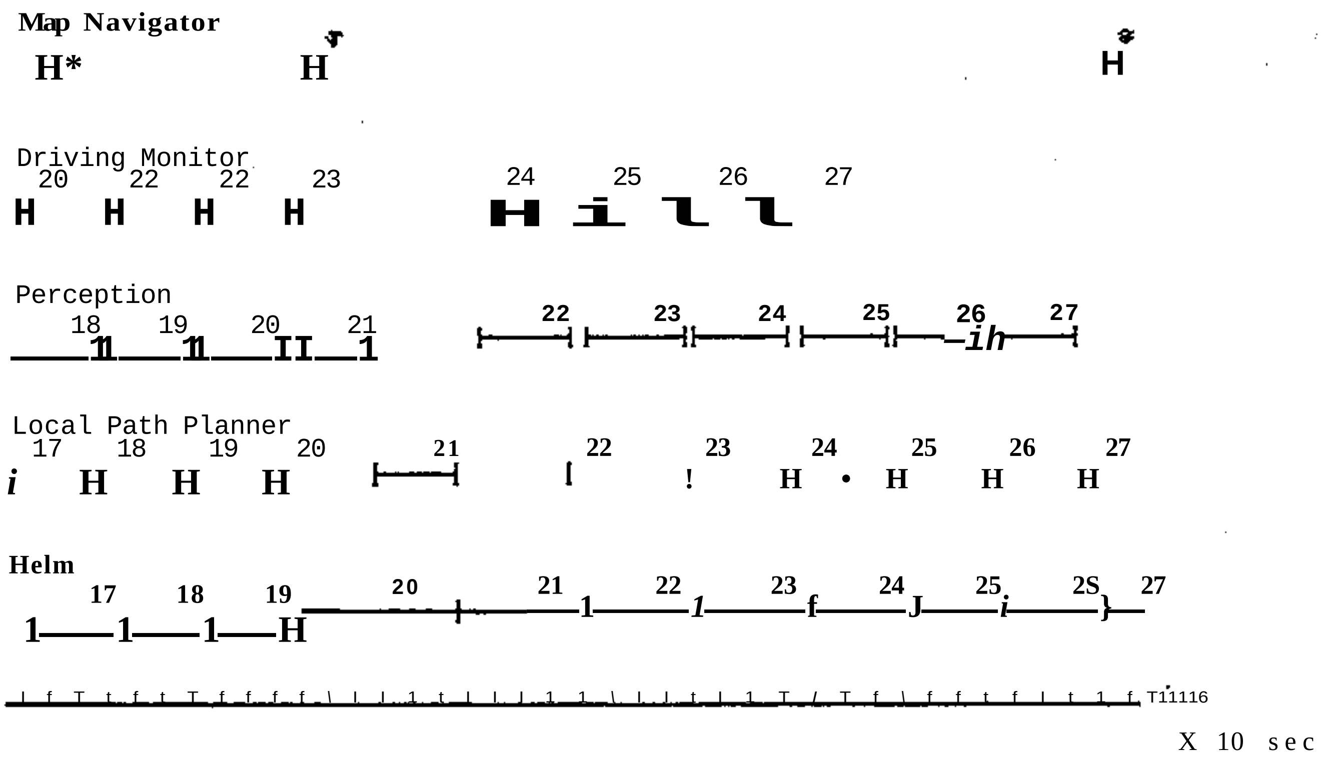


Figure 4-4: Timing Diagram of the Processing Steps

vehicle's healing direction and the direction perpendicular to the vehicle's heading. Therefore, the scanning distance must be short During straight navel, 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 4-7 shows the sensor view frames and the scanning positions. The scanning positions were calculated using Equation I 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* medmnisms are needed This can present a very challenging timing problem if mechanical pan and tilt mechanisms arc *used** To avoid this, the Terregator vehicle was equipped with two cameras and[1] 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 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

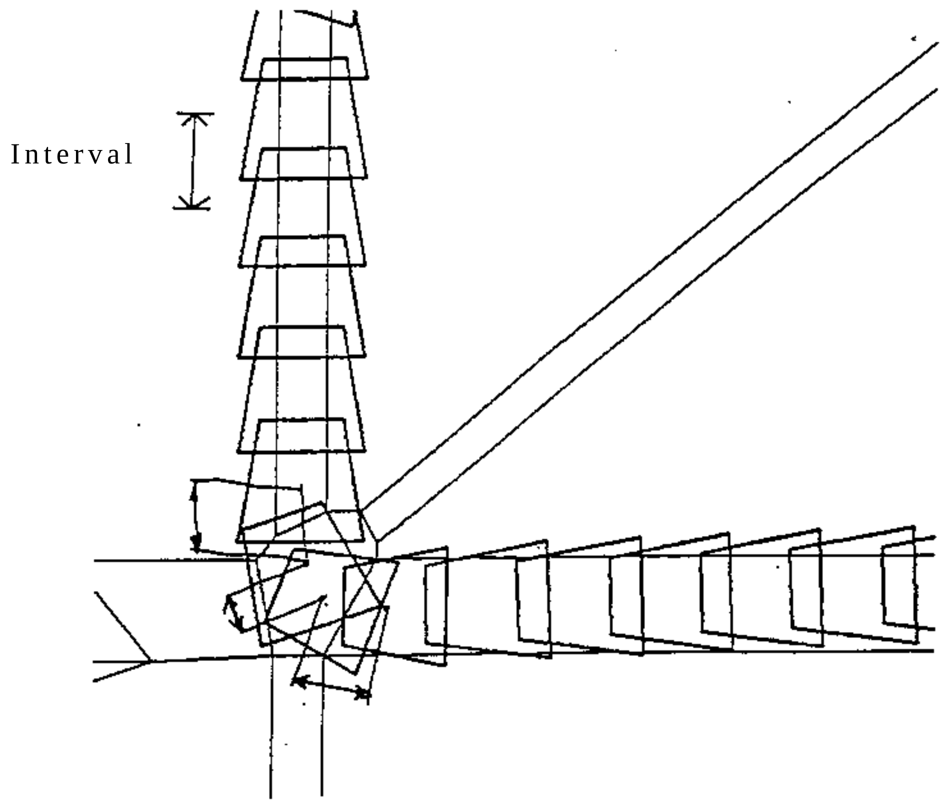


Figure 4-5: Driving Unit Intervals

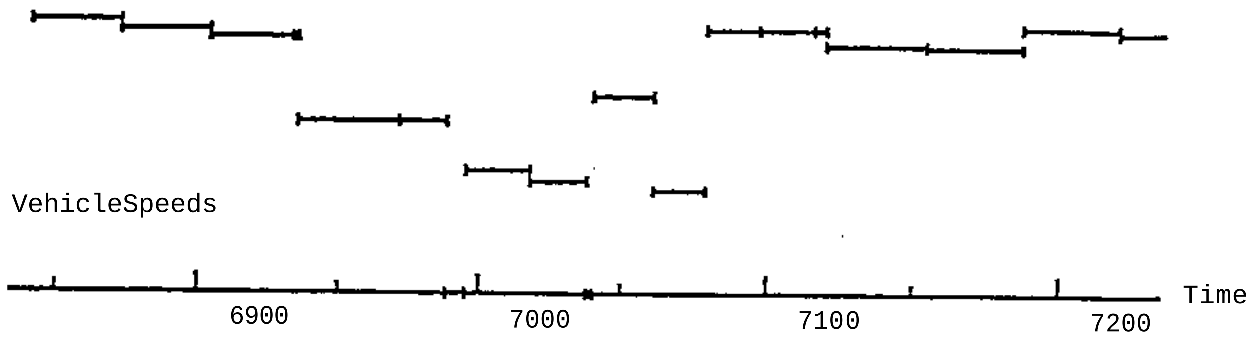
requires duplicated sensor hardware.

5. Conclusion

This paper has described the Driving Pipeline, 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. *Thmks* 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 o€ the Driving Pipeline are made possible by the two key ideas of ifae Driving Pipeline, the .driving unit and Che pipelined execution of the processing sieps. By using driving units, the data to be processed is divided kilo a sequence of driving unite thai can be processed separately by the processing steps- The sieps themselves am 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



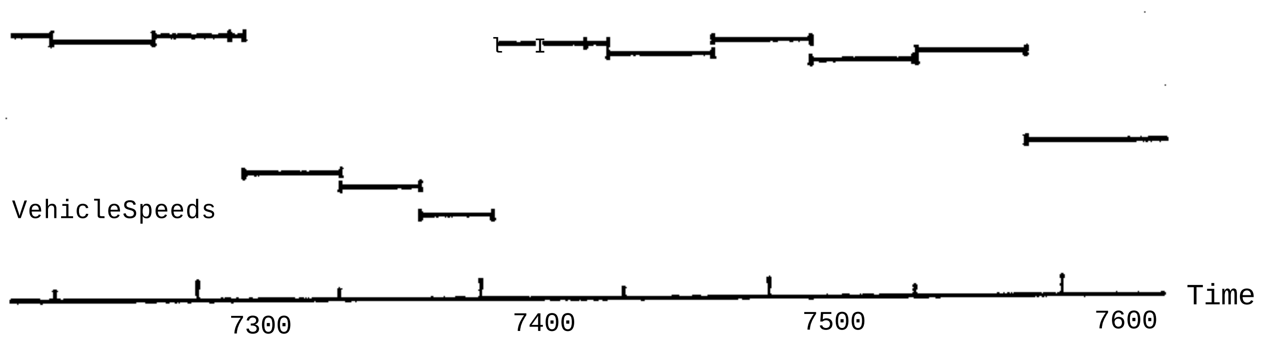


Figure 4-6: Control of the Vehicle Speed

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 (be case of stop-and-go motion, the last three aspects of the driving control scheme are impicit 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 suppom contmaous vehicle motion, the primitive processing steps involved in the

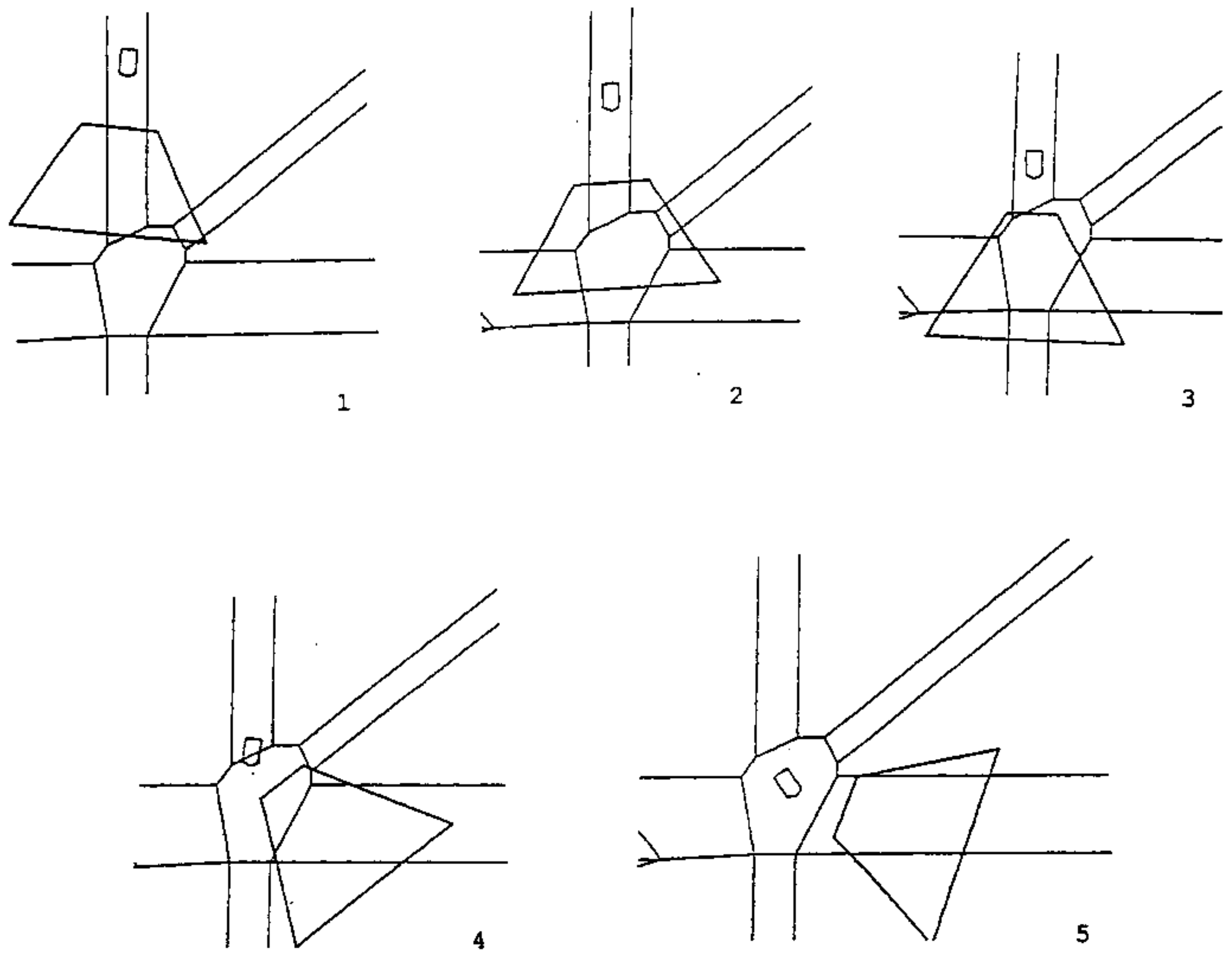


Figure 4-7: Sensor View Frames

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 *m* 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.

We envision two major directions for future research in this topic The first *is,* the further development of the

Driving Pipeline. This includes the following topics:

- Multiple sensor view frames in one driving unit When the Perception uses multiple sensors and their view frame sizes are largely different, the sensor with smaller view frame may need to scan more than twice on one driving unit In this situation, we may have to expand the concept of the driving unit.
- Uncertainty in the map database. In Section 3.3, we have discussed how a richer map database can produce higher parallelism among the processing steps and the length of the driving unit intervals. It seems plausible for humans that the greater accuracy in the map database, such as more accurate road shapes, allows a faster vehicle speed. We would like to build the theory and demonstration to account for this intuition.
- Cross-country traveL The same concept of the Driving Pipeline can be used for cross-country travel, with a different version of the Prediction, Perception, and Environment Modeling steps. By selecting the appropriate versions of these processing algorithms for each driving unit, cross-country travel may be incorporated with road-following travel in a single Driving Pipeline, producing a system that can combine on-road and off-road navigation. Cross-country travel may include greater uncertainty in vehicle motion control and more occlusions in the sensor data, thus reducing the vehicle speed through the same natural mechanisms described above for roadway travel.

The other major topic for further research is the development of a dynamic driving control scheme. The faster vehicle speed and the quicker vehicle response may need dynamic aigorithms for perception, planning, and vehicle control. For this end, we need new approaches both in the algorithms for the individual processing steps and the scheme to organize and coordinate them.

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.

[5] Y.Goto, etc. .
CMU Sidewalk Navigation System .
In *Proc. ofFall Joint Computer Conference, pages* 105-113. Nov, 1986.

[6] Y. Goto, A. Stentz.
The CMU System for Mobile Robot Navigation.
In *Proc. of IEEE International Conference onRobotics 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, AJStcatz, C. Thorpe.
An Architecture for Sensor Fusion in a Mobile Robot.
In *Proc. of IEEE International Conference on Robotics and Automation*, pages 2002-201L April, 1986.

[9] 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 .

[10] S. TsujL
Monitoring of a building environment by a mobile robot.
In *Robotics Research 2,* pages . , 1985 .

[11] A. M. Waxman, JJ. 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.