Below are three references related to the 1992 “Robot Olympics.” Reference 3, the most comprehensive one, was made available by the authors for inclusion on this CD-R; it is attached on the following pages.

Unfortunately, the Adobe Acrobat version of this document is barely readable on-screen, due to a font incompatibility. However, the document prints out on paper just fine.


1. Congdon C; Huber M; Kortenkamp D; Konolige K; Myers K; Saffiotti A; Ruspini E.H., “CarmelVersus Flakey - a Comparison of 2 Winners.” AI Magazine, 1993, V14, N1 (Spr), pp. 49-56.

2. Kortenkamp D.; Huber M.; Cohen C.; Raschke U; Bidlack C; Congdon C.B. ; Koss F; WeymouthT., “Integrated Mobile-robot Design - Winning the AAAI 92 Robot Competition.” IEEE Expert, 1993, V8, N4 (Aug), pp. 61-73.

3. B. Congdon, M. Huber, D. Kortenkamp, C. Bidlack, C. Cohen, S. Huffman F. Koss, U. Raschke, T. Weymouth, K. Konolige, K. Myers, A. Saffiotti, E. Ruspini, and D. Musto, 1994, “CARMEL vs. Flakey: A Comparison of Two Robots.” Technical Report issued by the Artificial Intelligence Laboratory, The University of Michigan and the Artificial Intelligence Center, SRI International, January 5.

# CARMEL vs. Flakey: A Comparison of Two Robots

Clare Bates Congdon  Marc Huber  David Kortenkamp*
Clint Bidlack  Charles Cohen  Scott Huffman  Frank Koss
Ulrich Raschke  Terry Weymouth

Artificial Intelligence Laboratory
The University of Michigan

and

Kurt Konolige  Karen Myers  Alessandro Saffiotti†
Enrique Ruspini  Daniela Musto‡

Artificial Intelligence Center
SRI International

Draft of January 5, 1994

†Iridia, Université Libre de Bruxelles
‡Institute for Information Processing, National Research Council of Italy, Pisa
*Now at The MITRE Corporation, Houston, TX

# Contents

# 1 Introduction

The University of Michigan's CARMEL and SRI International's Flakey were the first- and second-place finishers respectively at the AAAI Robot Competition in July, 1992. The approaches used by the two top teams are markedly different, but there are also some high-level similarities. This paper is intended to compare the two architectures, focusing on abilities exhibited in the robot competition and the underlying approaches used by the two teams. A shorter version of this paper appeared originally in [6].

## 1.1 Competition Overview

An *AI Magazine* article by competition organizers Tom Dean and Pete Bonasso [7] describes in detail the rules, scoring, and final standings of the 1992 AAAI Robot Competition. Briefly, the competition consisted of three stages. In Stage 1, each robot was required to roam through an octagonal arena roughly 21 meters in diameter, avoiding people and obstacles. In Stage 2, each robot was required to find and visit ten objects randomly placed amongst the obstacles in the ring. In Stage 3, each robot was required to visit three of the objects it had discovered in Stage 2 in an order specified by the judges immediately before the stage began; additional points were awarded to the fastest robots. Each of the three stages lasted 20 minutes, and each was scored by three judges. In all stages, the obstacles were boxes roughly 1 meter high grouped together to form different shapes. In Stages 2 and 3, the objects were tall poles three inches in diameter; the teams were allowed to tag each pole with their own distinguishing markers. Stage 1 was a qualifying round of sorts; robots that did not exhibit minimum competency in this stage were not to be allowed in Stages 2 and 3. In the actual competition, all robots passed Stage 1, and a separate award was given for the high-scoring robots in this stage of the competition. The scores from Stages 2 and 3 and were combined to determine the robots' rankings in the competition.

In Stage 1, Flakey finished second and CARMEL finished third (the first place finisher, TJ2 from IBM, competed in a smaller ring for little robots). Flakey impressed the judges with its flawless object avoidance, while CARMEL was impressive for its smooth travel at high speeds. In Stage 2 CARMEL finished in first place, having found and visited all ten objects in less than ten minutes. Flakey finished in third place (behind Buzz of Georgia Tech) having found and visited eight of the ten objects, using the full 20 minutes allotted. In Stage 3 CARMEL again finished in first place, with a time of just over three minutes, beating TJ2 of IBM by 30 seconds (TJ again competed in the small ring). Flakey finished fourth in Stage 3 with a time of around 11 minutes. When the Stage 2 and Stage 3 scores were combined, CARMEL finished first in the final ranking and Flakey finished second. While CARMEL excelled in both Stage 2 and Stage 3, Flakey's high final score was the result of their consistent performance in both stages. Other teams (such as Buzz and TJ2) finished very high in one stage and very low the other.

## 1.2 Physical description of the two robots

There are many physical similarities between the two robots. Both robots are fairly large, heavy, and roughly cylindrical; both are wheel driven and use dead reckoning (described below); and both are equipped with multiple computers and sensors. This section describes the two robots in detail.

### 1.2.1 Physical overview: CARMEL

CARMEL (which is an acronym for Computer Aided Robotics for Maintenance, Emergency, and Life-support) is based on a commercially available Cybermotion K2A mobile robot platform. CARMEL is a cylindrical robot about a meter in diameter, standing a bit less than a meter high when equipped with a large hollow shell (for holding electronics and other equipment) on top (see Figure 1). CARMEL moves using three synchronously driven wheels; it has a top speed of approximately 780 mm/sec and a turning speed of 120 deg/sec. The hexagonal top is decoupled from the wheels, so that when the robot itself turns, the orientation of the top is unchanged. Wheel encoders calculate the robot's displacement from a homed position; this calculation is called *dead-reckoning*. Errors accumulate in the dead-reckoned position due to wheel slippage; dealing with these errors is a major concern of both teams.

CARMEL is equipped with a ring of 24 ultrasonic sonar sensors evenly distributed around the robot's torso, each with a two-meter range and sensing cone of about 30 degrees. A grayscale CCD (charge-coupled device) camera was added to CARMEL to give it visual capabilities. The camera is mounted on a rotating tower, allowing it to turn 360 degrees, independent of the robot's orientation.

CARMEL has three computers working cooperatively while the robot is running. A motor control processor (Z80) receives motion and steering commands from the top-level computer and controls the robot's wheel speed and direction. This processor also maintains the robot's dead-reckoning information. An IBM PC XT clone is dedicated to the sonar ring, controlling the firing sequence and filtering sonar crosstalk and external noise from the sensor data. Finally, an IBM PC clone running a 33 MHz, 80486-based processor performs the top-level functions of the system. Image processing, planning, absolute positioning, etc. are all done on this computer. This computer communicates with the sonar- and motor-control processors via RS-232 links (9600 baud). CARMEL was also given a voice synthesizer, primarily to communicate with the judges and programmers.

### 1.2.2 Physical overview: Flakey

Flakey is a mature (some might say old) mobile robot, fully functional in 1985 (see Figure 2). Since 1985, the hardware has remained stable with relatively minor additions to the sensing and communications capabilities. In form, it is a custom-built mobile robot platform approximately one meter high and .6 meter in diameter. There are two independently-driven wheels, one on each side, giving a maximum linear velocity of about 500 mm/sec and turning

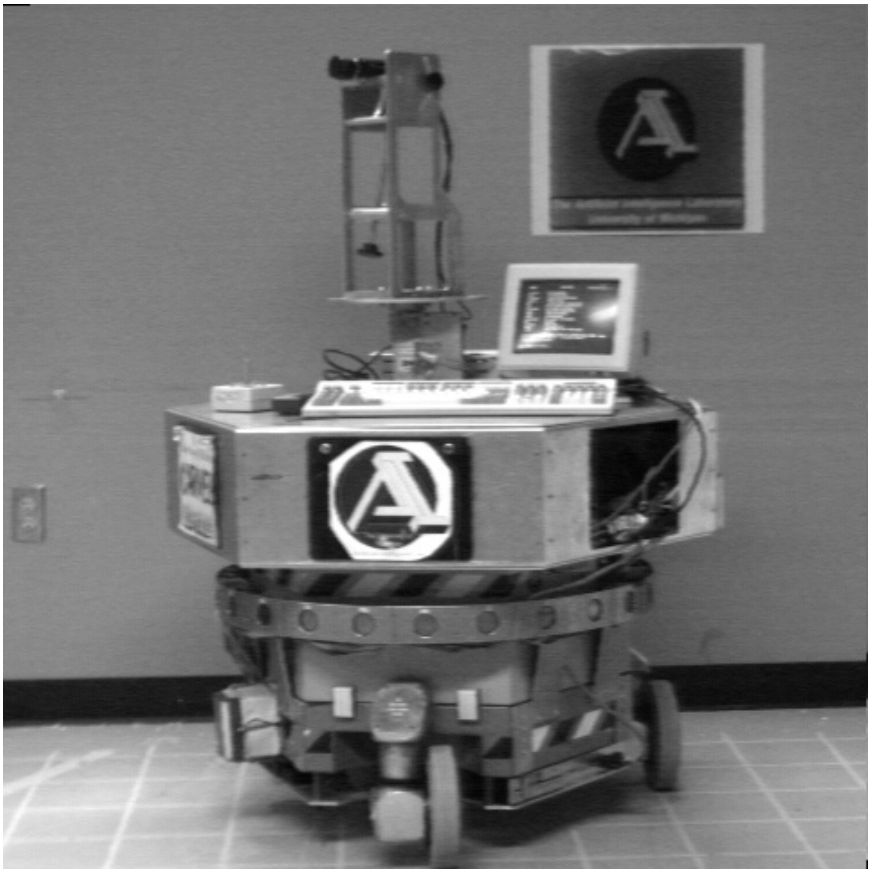


Figure 1: CARMEL stands just under a meter tall and moves at speeds up to 780 mm/sec. A ring of 24 sonars surrounds the robot's torso; a grayscale CCD camera mounted on a rotating tower provides visual capabilities. The monitor and keyboard are used for debugging, and not used in actual competition.

---

velocity of about 100 degs/sec. Flakey is equipped with optical wheel encoders to provide the same kind of dead-reckoning information as CARMEL. Like CARMEL, Flakey has ultrasonic sonar sensors good to about two meters, but instead of a uniform ring, FLAKEY has four sensors facing front, four facing back and two facing to each side. Additionally, Flakey has eight touch-sensitive bumpers around the bottom perimeter of the robot coupled to an emergency-halt reflex. For detailed object recognition, it employs a structured light sensor, which is a combination of a light stripe and a video camera that is capable of providing a dense depth map over a small area in front of Flakey.

FLAKEY has three computers working in parallel, two onboard. A Z80 motor controller is responsible for the sonars and wheel motors. A Sun 3 computer controls the structured

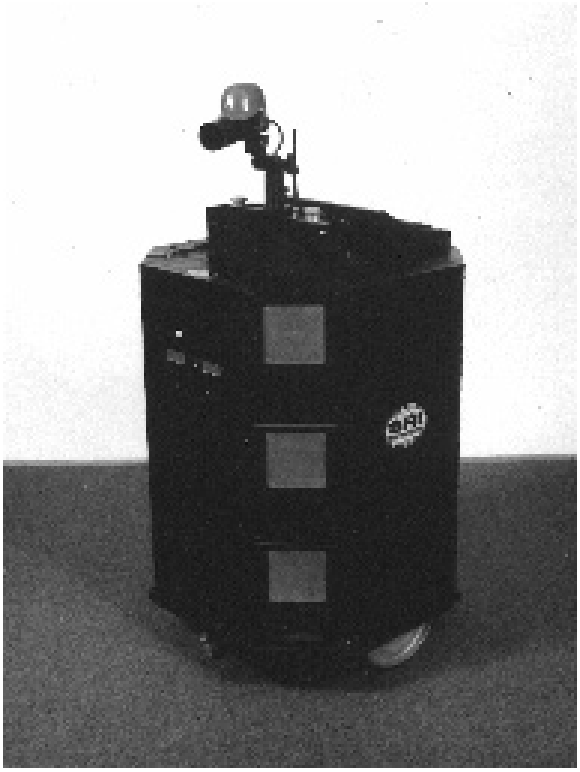


Figure 2: Flakey is about one meter high and moves at speeds up to 500 mm/sec. Twelve sonars (four in front, four in back, and 2 at each side), and eight touch-sensitive bumpers surround the robot. A structured-light sensor provides a dense depth map over a small region in front of Flakey.

---

light sensor and performs image processing and geometrical calculations necessary to produce a depth map; it also communicates with an offboard Sparcstation by a radio ethernet. The Sparcstation is responsible for all high-level interpretation and control functions: sensor integration, navigation, map registration, and so forth. Finally, Flakey has a speech synthesizer to communicate with programmers and judges.

Like many other teams in the 1992 AAAI Robot Competition, the SRI team could not get the radio link to work correctly in the noisy competition arena. Fortunately, they were able to tape a portable Sparcstation to Flakey, and make a direct ethernet connection. Also, the bright mercury-vapor lamps in the auditorium interfered with the structured light sensor, and were turned down to half power during the SRI runs in Stages 2 and 3.

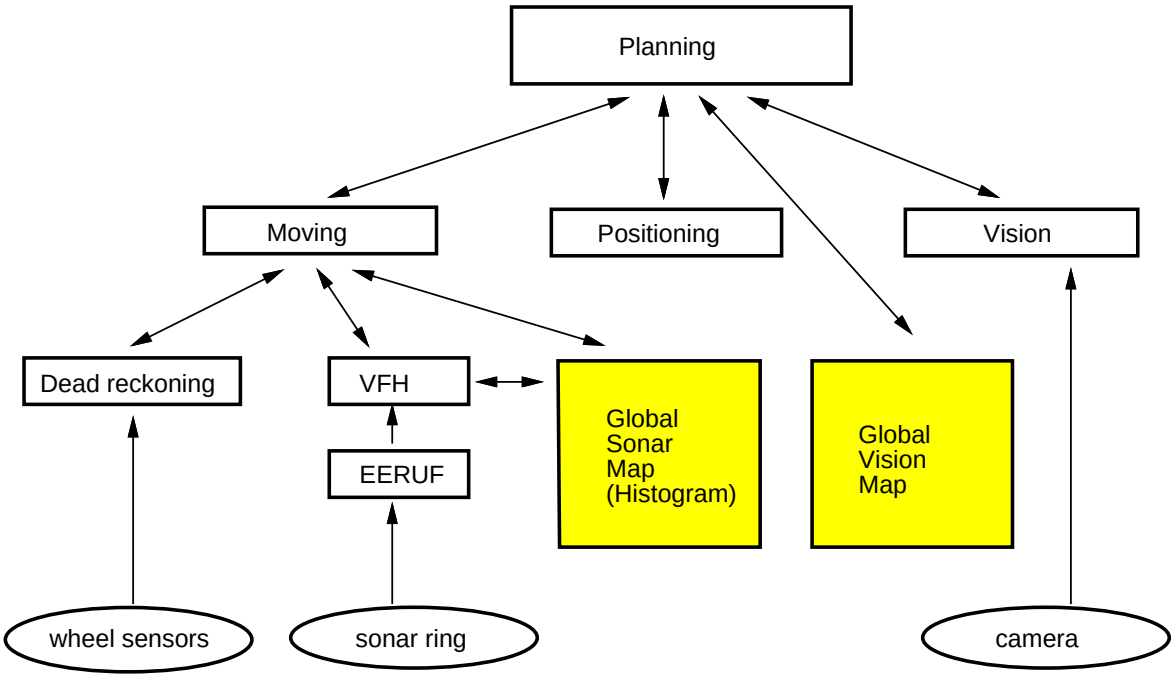



Figure 3: CARMEL's hierarchical software design consists of a high-level planning routine that calls lower-level routines for moving, correcting dead-reckoning errors, and vision. At the bottom of the illustration are the sensors.

## 1.3 Software overview of the two robots

There are a number of differences between the software designs of the two robots. These differences will be highlighted in the rest of the paper. This section provides a brief overview of each robot so as to provide context and structure to the following discussion.

### 1.3.1 Software overview: CARMEL

CARMEL has a hierarchical software structure. At the top level is a planning system that decides when to call subordinate modules for movement, vision, or recalibrating the robot's position. Each of the subordinate modules is responsible for doing low-level error handling, and must return control to the planner in a set period of time, perhaps reporting failure; the planning module will then determine whether to re-call the submodule with different parameters or to resort to another course of action. Figure 3 shows the major components of CARMEL's software architecture.

The movement module is a point-to-point, goal-directed obstacle-avoidance algorithm called VFH (described in Section 2.1.1). Using CARMEL's sonar sensors, VFH will guide

the robot from one prespecified point in a world coordinate system to another while avoiding any obstacles, stationary or moving. The vision module is a computer vision system that will locate the objects for the robot in Stages 2 and 3. The vision system uses a single camera and a one-pass algorithm to detect horizontally striped, barcode-like tags on each of the ten objects. A distance and heading to each object is returned. The recalibration module uses object locations to correct CARMEL's dead-reckoning errors, using triangulation to three known objects (this process is described in Section 4.2.1).

The software system of CARMEL was kept modular to allow for a team design, whereby small groups could work independently on one level of the design. The software system of CARMEL was also kept simple so that it could be run completely on board, allowing CARMEL to navigate at high speeds while smoothly avoiding obstacles. This is in contrast to most of the other robots in the competition that were sending sensor information to off-board processors. Some of these robots operated in a jerky, stop-and-go fashion, moving a bit, but then having to stop and wait while sensor information was sent off board, processed, and the results transmitted back to the robot. This architecture, therefore, allows CARMEL to be extremely reactive in the situations where reactivity is very important, namely, when the robot is moving about in the unknown and possibly dynamic world.[1]

### 1.3.2 Software overview: Flakey

Flakey, in contrast to CARMEL, is a distributed system. The system architecture is organized around the local perceptual space (LPS), an egocentric Cartesian plane in which all sensor information is registered, and various artificial constructs (or *artifacts*) are entered. The LPS gives Flakey an awareness of its immediate environment, and is critical in the tasks of combining sensor information, planning local movement, and integrating map information. The perceptual and control architecture makes constant reference to the local perceptual space. The diagram of Figure 4 shows the major components.

In Brooks' terms [4], the organization is partly vertical and partly horizontal. The vertical organization occurs in both perceptual (left side) and action (right side). Various perceptual routines are responsible for both adding sensor information to the LPS and processing it to produce surface information that can be used by object recognition and navigation routines. On the action side, the lowest level behaviors look mostly at occupancy information to do obstacle avoidance. The basic building blocks of behaviors are fuzzy rules, which give Flakey the ability to react gracefully to the environment by grading the strength of the reaction (e.g., turn left) according to the strength of the stimulus (e.g., distance of an obstacle on the right).

To move to desired locations, more complex behaviors are used to guide the reactive behaviors. These behaviors utilize surface information and artifacts and may also add artifacts to the LPS as control points for motion. At this level, fuzzy rules allow Flakey to

[1] In Stages 2 and 3, CARMEL did noticably pause to take and process camera images (which takes approximately 2 seconds per image), but this activity was kept to a minimum. CARMEL did not pause while processing sonar information (as teams that were sending this information off board did), which enables it to travel smoothly at high speeds.

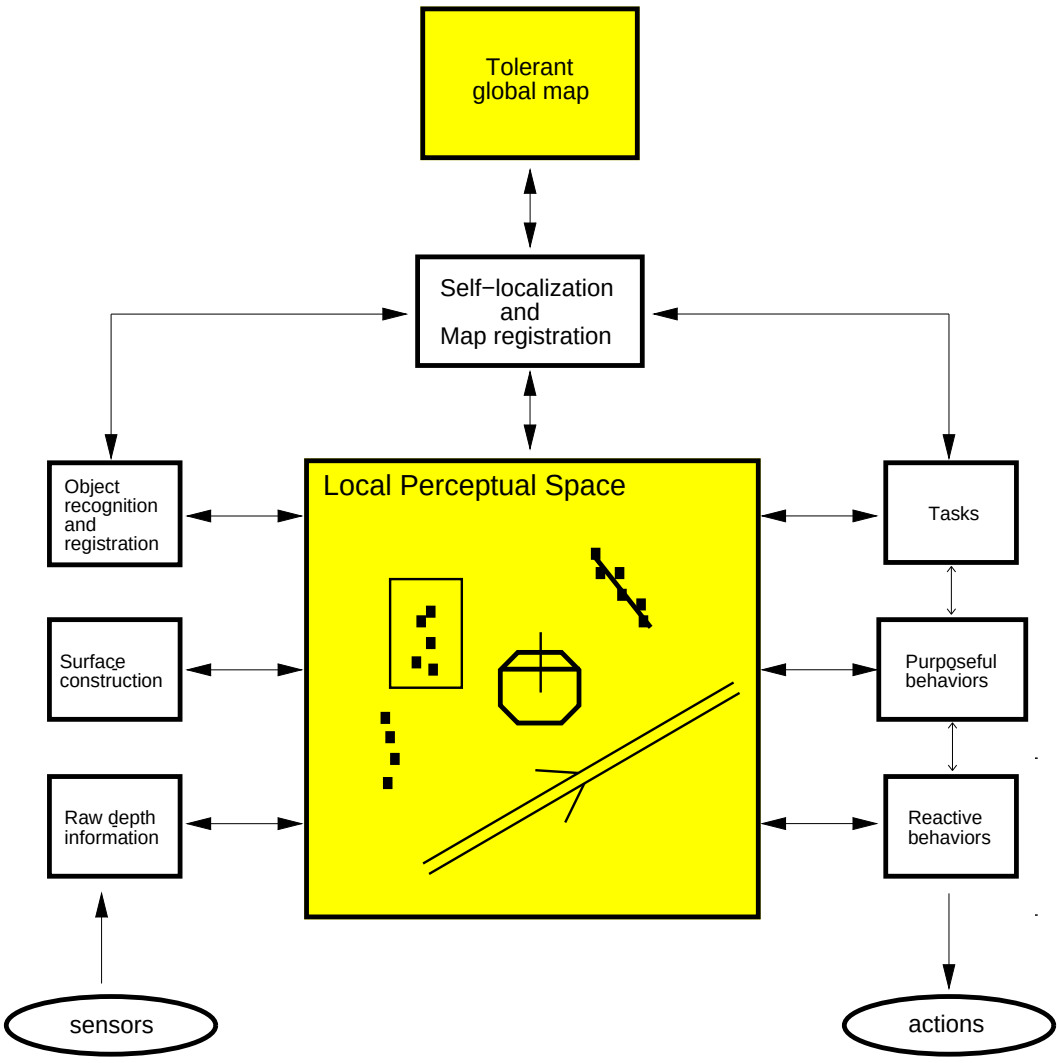



Figure 4: Flakey's system architecture has multiple routines updating and referring to the local perceptual space; these routines operate in parallel. In this illustration, perceptual routines are on the left, and action routines are on the right. The vertical dimension gives an indication of the cognitive level of processing, with high-level behaviors and perceptual routines at the top. A map-location module continuously matches local perceptual information to a stored global map, updating Flakey's global position. All modules operate independently in a distributed fashion.

---

blend possibly conflicting aims into one smooth action sequence. Finally, at the task level, complex behaviors are sequenced and their progress is monitored through events in the LPS. The horizontal organization comes about because behaviors can choose appropriate

information from the LPS. Time-critical behaviors such as obstacle avoidance rely on very simple processing of the sensors because this information is available quickly; however, these routines may also make use of other information when it is available, e.g., prior information about expected obstacles that comes from the map [13].

To navigate through extended regions, Flakey uses a *tolerant global map* that contains prior, imprecise spatial knowledge of objects in the domain. For the competition, this map initially contained the walls of the arena, along with their approximate length and angles between adjacent walls, and was completed by Flakey with the positions of the poles during Stage 2. There is no consistent global Cartesian map since building such a map can be difficult, and imprecise local connections are all that is required for successful navigation. A set of predicting and matching routines operate to localize Flakey within the map and keep the LPS consistent with it.

Flakey's software system is designed so that all processes operate in parallel with a basic cycle time of 100 milliseconds. This means that Flakey is continuously processing sensor information, recognizing objects, updating its map, and deciding what to do next, all in real time. Even though some processes, such as map registration, could take many seconds of information gathering and processing to complete, all processes were written so that they save partial results and complete within the cycle time. Thus, Flakey could keep moving even as it was looking for objects and updating its map; it was the only robot except for Scarecrow (see [7]) that never stopped moving during all stages of the competition.

The modular and distributed design of the system means that it is both flexible and extensible. The SRI team incorporated large portions of code previously written for navigation in an office environment, including most of the perceptual routines and the low-level behaviors. Work on competition-specific behaviors, tasks, pole recognition, and navigation began only one month prior to the start of the competition. The standard language for modules is Common Lisp, although a few of the lowest-level perceptual routines are written in C. A small real-time distributed operating system, written in Common Lisp in a standard Unix environment, sufficed for realtime operation on the Sparcstation, making it easy to debug and upgrade the software and hardware.

### 1.4 Overview of paper

In creating a robot for the competition, both teams had to deal with a number of significant issues. In most cases, CARMEL and Flakey used very different approaches. Of course, for both teams the decision on which approach to use was often not made purely on performance, but on constraints such as time, financial resources, and previous research efforts. However, the structure of the competition, in which a set task is performed in a fixed environment and performance is scored by objective judges, allows one to analyze various design decisions in a somewhat objective fashion based purely on performance. That is what this paper attempts to do. The four main issues that arose in the competition are identified as: Moving, Object Recognition, Mapping, and Planning. These four issues will be presented in the following four sections, each of which introduces the issues, describes each team's approach, and analyses the two approaches. The following section describes the two

teams' overall performances in the competition; this is followed by a section tha summarizes the architectural differences. Lastly, a more general comparison of the similarities in the approaches used by the two teams and differences between the two architectures is given, allowing insight into what both teams did right in order to succeed at the competition.

# 2 Issues in Moving

In Stage 1, both robots had to roam the arena, avoiding all obstacles and people. Avoiding obstacles is no trouble if you're stationary; the robots were also requested to roam the ring in either a directed or random fashion. In this case "roaming" was not defined precisely, but indicates that the robot should not be confined to a small area of the arena, e.g., turn in a small circle.

## 2.1 Obstacle Avoidance

Both robots used sonar sensors as their primary obstacle avoidance sensors. Sonar sensors are both unreliable (subject to noisy readings) and imprecise, giving only approximate information about the distance and direction of objects. To combat this, both teams used a two-part strategy for handling sonar sensors. The first part was an attempt to minimize noise and imprecision and the second part was to map out the environment in such a way that the remaining imprecision would have minimal effect. Finally, both robots had to use their map to choose a direction of motion. Each of the team's approaches is described below.

### 2.1.1 Obstacle avoidance: CARMEL

To deal with sonar sensors noise, CARMEL used a new algorithm called EERUF (Error Eliminating Rapid Ultrasonic Firing), which allows CARMEL to rapidly fire and sample the ultrasonic sensors for fast obstacle avoidance. The innovative feature of EERUF is its ability to detect and reject ultrasonic noise, including crosstalk. The sources of ultrasonic noise may be classified as either *external sources*, such as ultrasonic sensors used on another mobile robot operating in the same environment; or *internal sources*, such as stray echoes from other onboard ultrasonic sensors (see Figure 5). The latter phenomenon, known as crosstalk, is the reason for the slow firing rates in many conventional mobile robot applications: most mobile robots are designed to avoid crosstalk by waiting long enough between firing individual sensors, allowing each echo to dissipate before the next sensor is fired. EERUF, on the other hand, is able to detect and reject about 97% of all erroneous readings caused by external ultrasonic noise or crosstalk.

In general, external noise is random and can be detected simply by comparison of consecutive readings. Crosstalk, however, is mostly a systematic error, which may cause similar (albeit erroneous) results in consecutive readings. EERUF overcomes this problem by firing each sensor after individual, alternating, delays that disrupt the repetitiveness of crosstalk

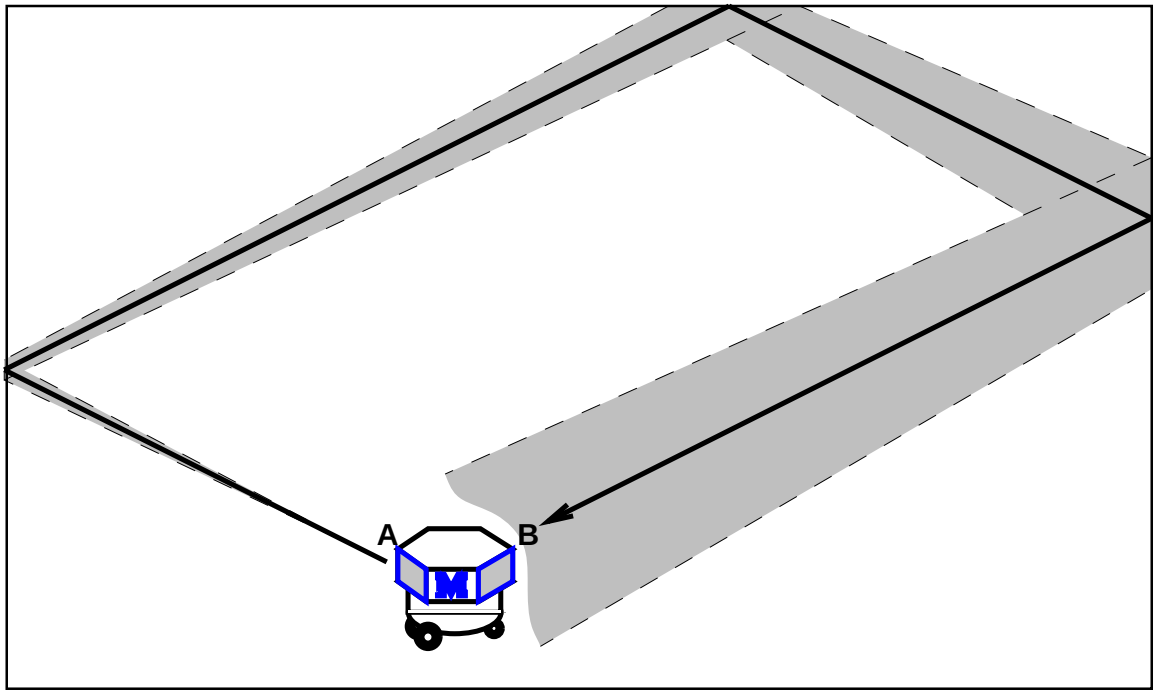
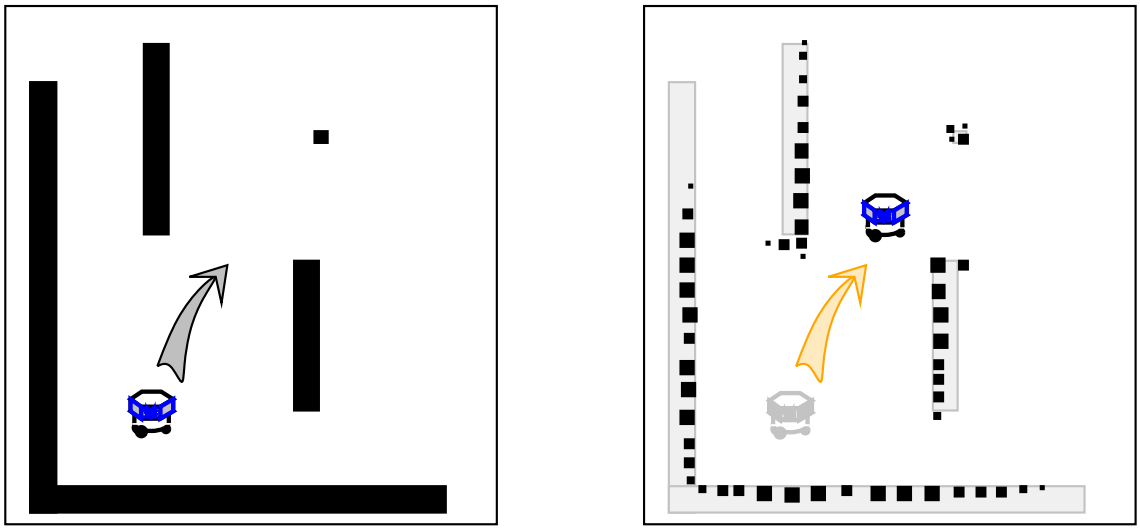



Figure 5: Crosstalk occurs when a sonar sensor picks up stray echoes from other onboard sonars. For example, when sonar A is fired, the sounds may reflect against walls; the reflected sound may be picked up by sonar B. Sonar B is unable to distinguish this from its own reflected signal.

errors, rendering these errors detectable by the method of comparison of consecutive readings [3].

Since EERUF practically eliminates the problem of crosstalk, it allows for very fast firing rates. With EERUF, a mobile robot with 24 ultrasonic sensors can fire all sensors within a period of 60 ms. On CARMEL, EERUF was only partially implemented, because CARMEL's ultrasonic sensor system was built before the development of EERUF, allowing a minimum firing period of 160 ms. Even at this rate, CARMEL's firing rate is two to five times faster than that of most conventional sonar implementations. This fast firing rate is a crucial factor that allows CARMEL to react in time to unexpected obstacles, even when traveling at high speeds.

To map the obstacles in the environment, CARMEL uses another innovative approach called a Vector Field Histogram (VFH). One of the most popular approaches to obstacle avoidance is based on the principle of potential fields. However, in the course of experimentation with this method, Koren and Borenstein [8] found that at higher speeds potential-field methods will inherently cause oscillations when travelling near obstacles or in narrow



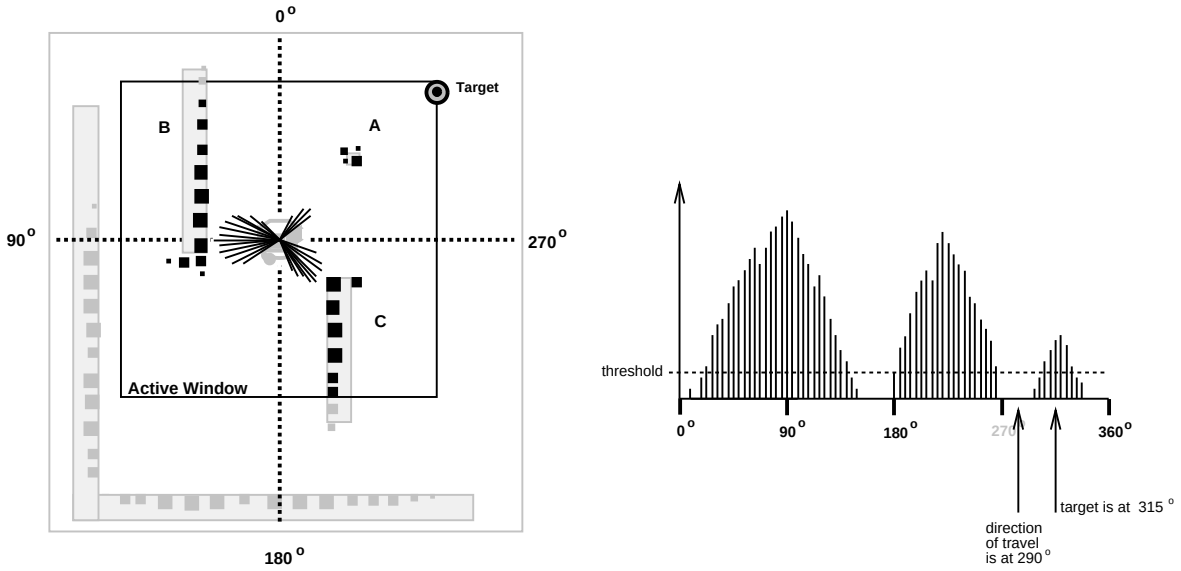
(a) (b)

Figure 6: CARMEL constructs a *histogram grid* as it moves through unfamiliar territory, to collect data from ultrasonic range sensors. In (a) above, CARMEL is at a start location, and heading roughly northeast. In (b), CARMEL has moved, collecting sonar readings in the process, and formed a histogram grid. The histogram grid is illustrated as dark squares; the larger the square, the higher the certainty of an object at that location.

---

passages. To overcome these problems, they developed VFH. The VFH method uses a two-dimensional Cartesian grid, called the *histogram grid* (illustrated in Figure 6), to represent data from ultrasonic (or other) range sensors. Each cell in the histogram grid holds a certainty value that represents the confidence of the algorithm in the existence of an obstacle at that location. This representation was derived from the certainty grid concept that was originally developed by Moravec and Elfes [11]. In the histogram grid, certainty values are incremented when the range reading from an ultrasonic sensor indicates the presence of an object at that cell.

Based on data in the histogram grid, the VFH method creates an intermediate data representation called the *polar histogram*. The purpose of the polar histogram is to reduce the amount of data that needs to be handled for real-time analysis while at the same time retaining the statistical information of the histogram grid, which compensates for the inaccuracies of the ultrasonic sensors. In this way, the VFH algorithm produces a sufficiently detailed spatial representation of the robot's environment for travel among densely cluttered obstacles, without compromising the system's real-time performance [1, 2]. The spatial representation in the polar histogram can be visualized as a mountainous panorama around




(a) (b)

Figure 7: Using the data from the histogram grid in Figure 6b, CARMEL creates a second representation of objects called the polar histogram. In (a) above, histogram cells within the active window are used to form vectors centered at CARMEL; longer vectors represent objects that are nearer and more certain. In (b), these same vectors are represented in the polar histogram, which resembles a series of mountains and valleys. The target location is at 315 degrees relative to CARMEL's current location, but CARMEL will travel at 290 degrees — the valley indicates that this is a safe direction to travel. In (a), this means that CARMEL will pass between obstacles A and C on its way to the target, rather than between obstacles A and B.

---

the robot, where the height and size of the peaks represent the proximity of obstacles, and the valleys represent possible travel directions (see Figure 7). The VFH algorithm steers the robot in the direction of one of the valleys, based on the direction of the target location.

The combination of EERUF and VFH is uniquely suited to high-speed obstacle avoidance (CARMEL travelled at speeds of up to 780 mm/sec in the competition). Manz, Liscano, and Green [10] showed that VFH fares favorably when comparted to several other obstacle avoidance methods.

### 2.1.2 Obstacle avoidance: Flakey

Flakey uses two basic techniques for obstacle avoidance. The local perceptual space integrates sensor readings from the sonars, in a manner similar to the occupancy grid of

CARMEL; but additional artifacts are introduced by subsequent perceptual processes to aid in maneuvering. On the action side, Flakey was unique among the robots at the competition in using fuzzy control rules as a programming language for behavior.

Much as CARMEL has its histogram grid, Flakey also uses the local perceptual space (LPS) to register multiple sonar readings over time. In addition, the LPS contains artifacts, either reconstructed surfaces or movement control points, that help in obstacle avoidance. Figure 8 shows a typical LPS configuration. The space represents a four-meter square of the environment in a plan view, with Flakey in the middle and oriented towards the top (the forward direction for Flakey), as shown. The LPS is egocentric: Flakey always stays at the center of the space in the same orientation, so that as Flakey moves, the world rotates about it. This representation makes it very easy to write perceptual and action routines: for example, to follow a wall on the right, a perceptual routine looks for wall objects on the right side of the LPS, and action routines try to move Flakey to keep the wall lined up vertically.

Within the LPS are various forms of sensor readings and information extracted from them by the perceptual routines. Each of the small black squares represents the center of one sonar reading. A buffer of the last 100 readings (about five seconds' worth) is kept in the LPS, and updated according to Flakey's motion. The end result is a series of readings that outline various objects, especially along the sides of Flakey during straight-line motion. There are perceptual routines that attempt to fit line segments to coherent sequences of sonar readings. One such long segment is on the right of Flakey, labeled with "W" as a possible candidate for a wall. These segments, essential for navigation (see Section 5), are also useful in filling in missing information required for obstacle avoidance. Because of the placement of the sonars to the front and sides, Flakey has large blind spots on the forward diagonals. By noting the trend of a side segment bearing towards the front, it is possible to hypothesize a blocked diagonal, even though it cannot be sensed.

The large rectangle at the top of the LPS is an area of sensitivity monitored by a perceptual process. For controlling Flakey's motion, many such areas were defined as a means of understanding what obstacles were present and deciding how to avoid them. The sensitivity area in the figure indicates that there is something a moderate distance ahead and to the right of Flakey. Since the sensitivity areas correspond to actions, they do *not* move in the LPS. For example, if Flakey were to turn left, out of the way of the obstacle, then the obstacle would rotate right in the LPS, and the area of sensitivity would no longer register the obstacle.

Once the local perceptual space is constructed, Flakey uses fuzzy control rules to control its motion and avoid obstacles. A fuzzy control rule has the form:

$$A \rightarrow c \ ,$$

where $A$ is a fuzzy expression composed by fuzzy predicates and the fuzzy connectives AND, OR and NOT, and $c$ is a control action. A typical control rule might be:

> If an obstacle is close on the left, then turn right 4 degrees.

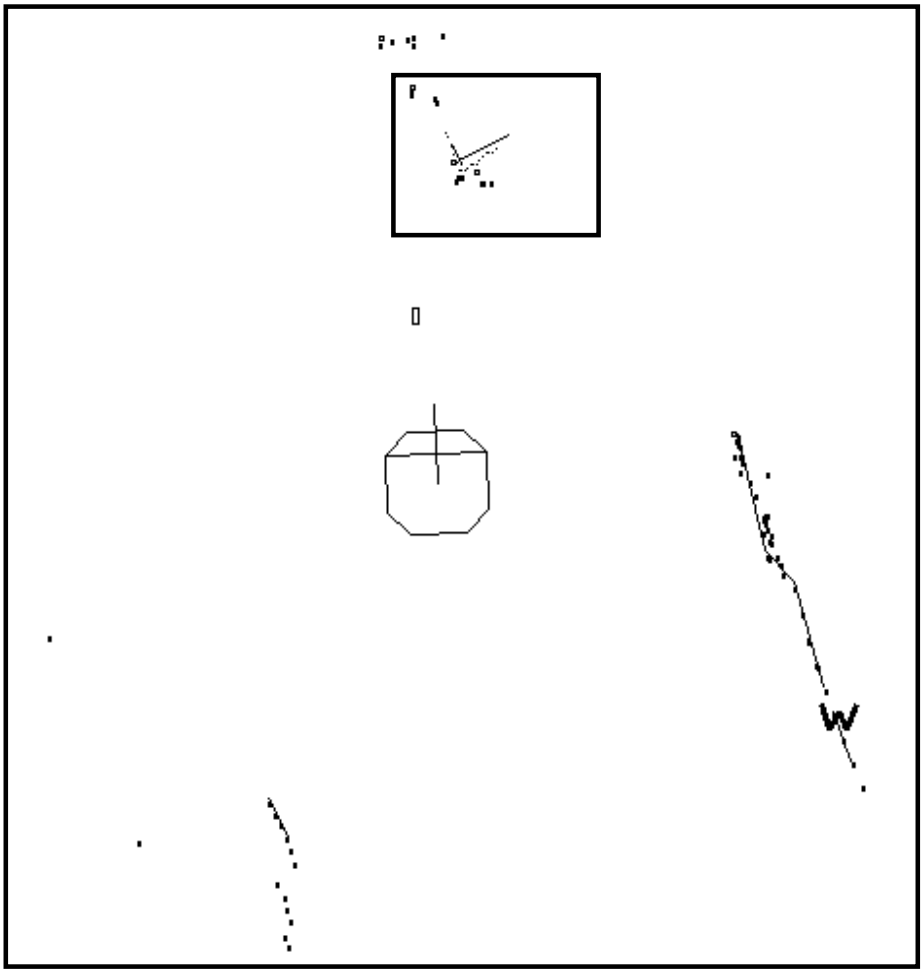



Figure 8: Flakey's local perceptual space (LPS) gives some indication of Flakey's forward speed and rotation. The arrow pointing from the center is a velocity vector; its length is proportional to the speed. Just above Flakey and slightly to the left is a direction set point: the motor controller will attempt to turn Flakey to keep the set point aligned straight ahead. The large rectangle at the top of the LPS is a sensitivity area that indicates that there is an object a moderate distance ahead and to the right of Flakey. A collection of sensor readings at the right edge of the LPS is a candidate for a wall, labelled "W".

---

A fuzzy predicate can be *partially* true in a given state of the world. Namely, a fuzzy predicate has a number in the interval $[0, 1]$ as its truth value, with 0 standing for complete falseness, 1 for complete truth, and intermediate values for partial satisfaction. E.g.,"close" has truth value 0.5 if distance is 700 mm (see Figure 9). *Max*, *min*, and complement to 1 are used to compute the truth value of disjunction, conjunction and negation, respectively.

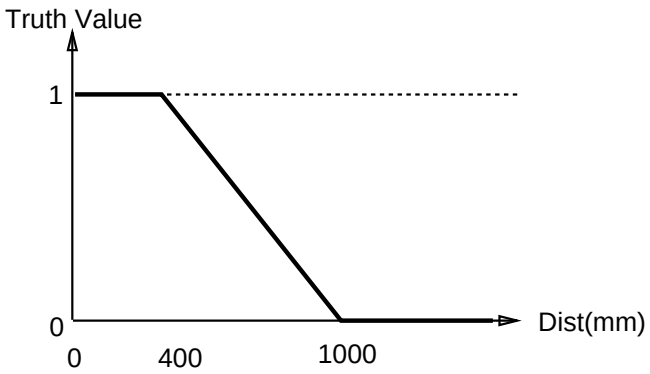



Figure 9: The fuzzy predicate "close" used by Flakey has a truth value of 1 for distances less than 400 mm, and 0 for distances greater than 1000 mm. An object that is in between 400 mm and 1000 mm away will have a truth value between 0 and 1, as illustrated above.

---

The truth value of $A$ determines the desirability of applying the control action $c$. At every cycle (i.e., every 1/10 of a sec), all the rules are checked: the truth value of their premises are evaluated, and a corresponding desirability value for the conclusion is generated. A simple "defuzzification" technique uses desirability values to merge the outputs of all the rules into one tradeoff control action. (This contrasts with typical non fuzzy approaches, where only one rule is selected at each cycle.) For instance, a desirability of 0.6 for the action "turn left 10 degrees" and a desirability of 0.3 for the action "turn right 4 degrees" are synthesized into the one command "turn left 5 degrees". Control actions typically refer to forward velocity and angular orientation, by sending the appropriate set-point to Flakey's motor controller. A detailed description of Flakey's fuzzy controller is contained in [14].

The rules are organized into sets called *behaviors* in a modular fashion. For example, one set of four rules deals with emergency maneuvers when there is an obstacle very close to Flakey (the PROTECT behavior). The modular organization has several advantages. First, a behavior can integrate a number of rule sets designed for tackling different aspects of one common goal. Second, behaviors can be composed to produce a more complex behavior that will smoothly integrate between different, competing goals, producing an tradeoff action that satisfies each of them as much as possible. The two obstacle avoidance behaviors are good examples of this technique. The PROTECT behavior is appropriate when obstacles are close to Flakey relative to its speed: it slows Flakey down and turns sharply towards open space. A longer-range avoidance procedure (the DEFLECT behavior) operates when the immediate space is clear. It does not slow Flakey down, but turns it more gently for smoother deflection from obstacles. The AVOID OBSTACLES behavior is a combination of these two.

The most critical part in the implementation of the fuzzy controller was the tuning of fuzzy predicates. This was mainly based on experimentation. Emphasis was placed on the reliablity of obstacle avoidance; this was made more difficult by blind spots on the diagonal.

### 2.1.3 Obstacle avoidance: Analysis

Obstacle avoidance in the competition meant, first and foremost, that the robot should not run into stationary or moving objects, especially the judges. In this respect both robots did remarkably well, despite attempts by the judges to surprise and contain them. Flakey's fuzzy control resulted in impressive reliability and smoothness of movement. The two-part obstacle avoidance rules worked as required, forcing emergency stops and maneuvers when a close object was detected, and moving more smoothly in relatively open space. The overall impression is best summarized in some of the judges' comments: "Very smooth control. Excellent movement among obstacles. Only robot felt I could sit or lie down in front of." (which he actually did!) Flakey moved at a relatively slow 200 mm/sec, not touching any obstacles or judges. Blind spots on the diagonal were responsible for the slowness: objects in these positions had to be relatively close before they would be seen by the sonars.[2]

Like TJ2 from IBM, the winner of this stage of the competition, CARMEL was distinguished by a "zippy" motion around obstacles in open terrain, very pleasant to watch. It moved at a speed of 300 mm/sec in this stage of the competition, noticeably faster than Flakey. However, under prodding from the judges, CARMEL touched two obstacles and grazed a judge. This was partly due to the fact that many variables in CARMEL's obstacle avoidance code need to be tuned for the environment in which it is running. The Michigan team had assumed an environment with dynamic but benign obstacles. Both teams noticed that behavior could be improved markedly by tuning the parameters of the avoidance routines.

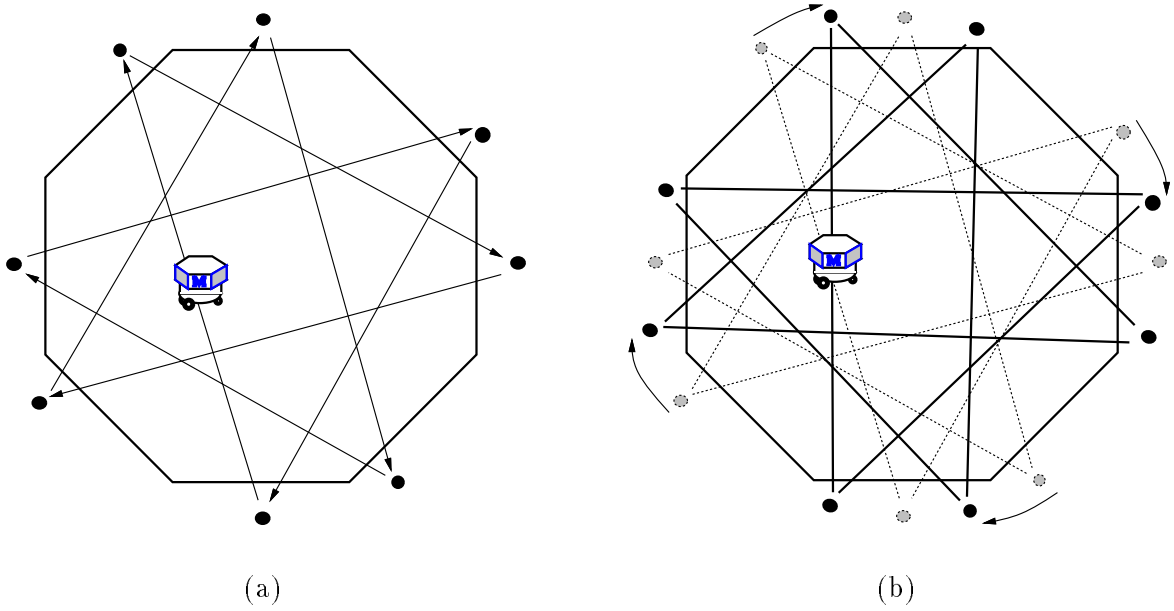
## 2.2 Roaming

In Stage 1 of the competition, the robots were expected to "roam" the arena, moving to different areas. The two teams used different approaches to achieve this behavior. CARMEL used a point-to-point travel strategy that (along with its high-speed movement) ensured good coverage. Flakey was provided with a wandering behavior composed of lower-level "go-forward" and "avoid-obstacles" behaviors.

### 2.2.1 Roaming strategy: CARMEL

Since the VFH obstacle avoidance algorithm requires a goal location to move towards, the Michigan team needed a way to give CARMEL a series of goal locations that ensured that CARMEL would cover a substantial portion of the arena. Consequently, CARMEL's Stage 1 planner is simply a hardcoded sequence of goal locations, such as that shown in Figure 10;

[2]Flakey has since added sonars to look on the diagonal, and obstacle avoidance now works reliably at 400 mm/sec.



(a) (b)

Figure 10: To ensure coverage of the arena, CARMEL was programmed to visit a series of eight points represented in (a) above. As CARMEL moved through the eight-point cycle, dead-reckoning errors caused the star to rotate relative to the arena, as illustrated in (b). This leads CARMEL to visit even more regions in the arena.

---

the robot's goal is to reach successive locations in the "star". Navigating between goal locations requires CARMEL to cross approximately through the center of the ring, relying upon the VFH-based obstacle-avoidance algorithms to get there. Visiting all of the goal locations indicated in the figure, then, requires several passes across the ring, each pass taking CARMEL into new territory at the periphery. Once all of the goal locations have been visited, CARMEL starts over again. This approach provides CARMEL with a roaming behavior, somewhat tailored to the environment, but for an otherwise goal-less task.[3]

As dead-reckoning errors accumulate, the location of the multi-pointed star moves about relative to the actual arena, getting rotated and translated relative to the actual coordinate system, as shown in Figure 10. The star pattern is robust to rotation errors (the ring is still fully covered if the star rotates, as goal points rotate towards positions vacated by other goal points). Placing the points of the star outside of the ring allows the star pattern to overcome small translation errors as the ring is still covered if the star translates by small amounts. Since the goal points are located outside of the arena's boundary, CARMEL is never able to actually reach the points. VFH prevents it from going through walls, and

[3] Although the goal points are hardcoded in this implementation, they could also be computed on the fly.

instead the robot times out and then moves on to the next goal point.[4] CARMEL is also able to recognize situations in which it is trapped in a concave formation of obstacles. The map used by VFH allows detection of concave formations so that recovery is possible.

Experimentation showed that using the star worked quite well, even for extended runs. Part of this success may be due to the rotation of the goal points from dead-reckoning errors. As CARMEL cycled through the star pattern multiple times, this rotation provided enough variation of the path taken between the points that CARMEL was able to more thoroughly cover the entire arena. As the points rotated, CARMEL would head into areas of the periphery that were not initially designated as goal points. The path variance would also aid in avoidance of trap situations encountered during previous cycles. The shifted path would eventually change enough that CARMEL would approach the concave formation of obstacles at a different angle or bypass it altogether.

A wall-following algorithm was also considered to reorient and reposition CARMEL at intervals (similar to SRI's registration process, described in Section 3.2). This would have been necessary if CARMEL would have had to wander the arena for days instead of minutes (to account for the significantly large translation errors), but the idea was rejected because it wasn't needed for a 20-minute run. In fact, it was never apparent that there was any significant translation error in any of the runs that CARMEL made.

### 2.2.2 Roaming strategy: Flakey

The SRI team noted in simulation and experimentation that a very simple strategy would satisfy the requirement of roaming. This strategy is to simply go forward until an obstacle is encountered, use the obstacle avoidance routines to deflect the forward motion, and then continue on in the new direction. Given the relatively open environment of the arena, there was little chance of getting stuck in one particular place.

The team implemented an elementary WANDER behavior as a composition of AVOID OBSTACLES and GO FORWARD behaviors. As mentioned, AVOID OBSTACLES guides Flakey away from occupied areas, and keeps it from bumping into close objects. GO-FORWARD just keeps Flakey going at a fixed velocity, given as a parameter. The relative predominance of one component over the other in the resulting WANDER behavior is determined by the fuzzy state variable "approaching-obstacle": AVOID OBSTACLES is most active when there is an obstacle in the way, otherwise it is quiescent, leaving control to GO FORWARD. The visual result for an external observer is that Flakey "follows its nose", while turning away from obstacles as it approaches them. Maximum speed was set at 200 mm/sec in open areas, and half that near obstacles.

One further feature of the AVOID OBSTACLES behavior is its ability to "unstick" Flakey if it got into a local minimum. This situation is recognizable by a continuous inability to go forward: the GO FORWARD behavior is frustrated. In this case, a 90-degree rotation is performed to give Flakey a new heading.

[4] Upon initiation of navigating to a goal point, CARMEL estimates a maximal amount of time to allot for the movement. Surpassing this alloted time halts the robot, which at this time may perform further planning to recover from the failed move, or, as in this case, ignore the failure.

It is easy to implement more elaborate roaming strategies for Flakey, given the ability of fuzzy control rules to accommodate multiple goals. For example, a strategy similar to that of forays (see Section 5) would define arbitrary lanes within the arena; the borders of the lanes are fuzzy, so that Flakey could go out of the lane to avoid an obstacle, then return when the coast is clear. However, the simpler WANDER strategy sufficed for the competition.

### 2.2.3 Roaming strategy: Analysis

Both CARMEL and Flakey have the ability to avoid local minima in roaming the arena, the basic requirement for not getting stuck in one place. In terms of covering more ground in the arena, it is likely that CARMEL would have done better. Although covering a large percentage of the ring was mentioned as a scoring criterion in the rules, it was not a strong criterion to the judges, as long as it was demonstrable that the robots would be able to roam freely under varying obstacle configurations.

Flakey placed ahead of CARMEL in this stage of the competition, and was only 1 point behind the winning entry, TJ2 from IBM. Part of the reason why CARMEL did not do as well was because CARMEL's wandering strategy is constrained by the fact that the VFH obstacle-avoidance algorithm is goal-driven, i.e, it must move *to a point*, rather than move *in a direction*.[5] This purposeful motion towards a goal point led to CARMEL's lower score in this round; the judges expected to be able to "herd" the robots in a particular direction, and CARMEL was not programmed to respond to such prodding. Instead, it doggedly kept trying to head towards its goal location. In retrospect, it might have been better if CARMEL had been programmed to "give up" on a goal point much sooner. The "soft-fail" approach of fuzzy control rules and Flakey's simple wandering strategy worked better in this regard.

# 3 Issues in Object Recognition

Object recognition is an essential component of Stages 2 and 3 of the competition. The eight-foot high PVC poles that represented objects in the competition must be recognized. Also, recognition of boxes and walls could prove to be advantageous depending on the navigational methods and strategies employed. The variations in sensing capabilities and underlying design philosophies for the University of Michigan and SRI robots lead to widely differing approaches to object recognition.

## 3.1 Object recognition: CARMEL

The ability to accurately detect and identify object poles was important for earning the maximum number of points, as well as for keeping position and orientation errors within

[5]Of course, a distant goal point could be generated (one that CARMEL would be guaranteed to never reach because it was outside of the arena), and this would have the same effect as telling CARMEL to head in a particular direction.

tolerable limits (see Section 4.2.1), consequently, CARMEL's system was designed from the onset to be as reliable, as accurate, and as fast as possible. Various object identification schemes were considered, but a vision-based system had an important advantage in its potential for long-range sensing. A major concern was the inherently heavy computation generally required for image processing. However, by intelligently designing the object pole tags, the computation was greatly reduced.

### 3.1.1 Object pole tags

The object pole tag design used for CARMEL consists of a black and white stripe pattern placed upon PVC tubing with a four-inch diameter, allowing the tags to be slipped over the three-inch-diameter object poles. Example object tags are shown in Figure 11. The basic stripe pattern is six evenly spaced horizontal black bands of 50 mm width, with the top of the top band and the bottom of the bottom band spaced 1000 mm apart. The white gaps between the black bands correspond to the bit positions in a five-bit word. A white space between two bands corresponds to an "off" bit, while filling the space with black corresponds to an "on" bit. The five bits between the six bands can then represent 32 unique objects. One of the most significant aspects of the striped PVC tags was that they look the same from every direction (barring lighting changes). This had a great impact upon the exploration algorithm used, as the robot did not have to approach object poles from a particular direction. Rather, CARMEL had only to get within visual range of an object pole in order to identify it.

### 3.1.2 The object recognition algorithm

The commitment to computer vision for object identification and localization introduced many issues, such as reliability, robustness, accuracy speed and range. Although the University of Michigan team experimented with pre-processing methods, the final vision system had no image preprocessing. This would have added complexity to the system and slowed down the vision process and, hence, the entire process of exploration. Color image processing methods were also explored. The red and green components of a color camera were used to segment bright orange bands from background noise. However, the additional overhead of processing two images instead of one was inefficient, and a gray-scale vision system was preferred because of reduced processing and simplicity. The algorithm had experimentally shown to be immune to most background noise and color vision methods were not necessary.

As mentioned above, object identification and localization is performed with a single-pass, gray-scale, vision algorithm. About two seconds of processing time on the 486-based computer is required per image. The structure of the algorithm is a simple finite-state machine. The algorithm goes down each column of the image looking for a white-to-black transition that would mark the start of a potential object. When a white-to-black transition is detected the algorithm switches to a state that looks for a black-to-white transition. When such a transition is found, the algorithm calls it a band. The band is measured and this measurement is used to detect future bands in the same column (i.e., all future bands should

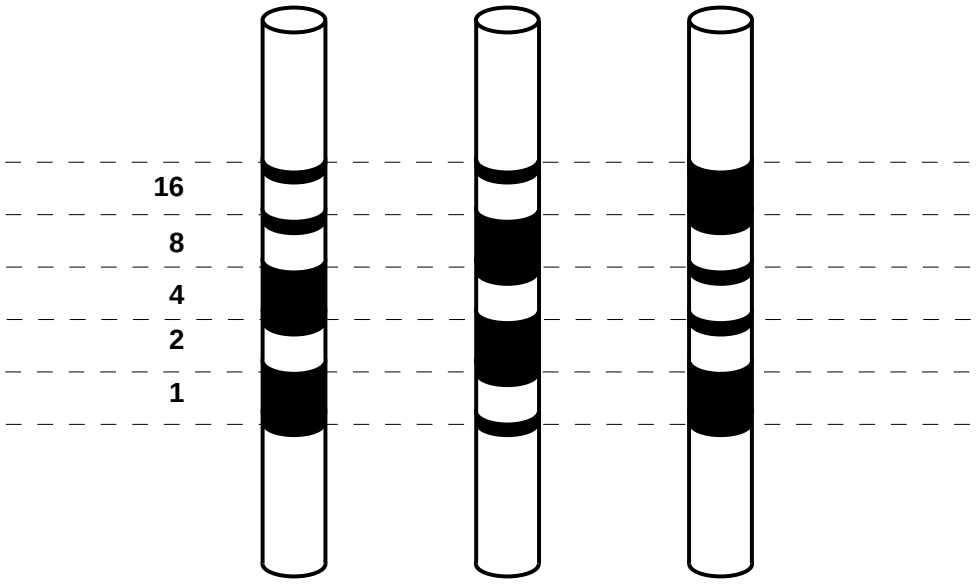



Figure 11: Object recognition for CARMEL was done using a five-bit pattern on PVC poles. The poles above illustrate the bit patterns for objects numbered 5, 10, and 17.

---

be of the same size or three times that size). The algorithm then resumes looking for a white-to-black transition. After finding enough bands to comprise a tag the algorithm stores the tag id and length. Once a column is complete, the eligible objects are heuristically merged with objects found in previous columns. Objects are slowly "grown" in this fashion until an object's edge is found and no more columns are merged into it. Once the entire image is processed, another heuristic merging process is invoked that merges multiple segments of an object that happened to slip through the initial merging algorithm. The distance between the top of the top band and the bottom of the bottom band, in terms of the number of pixels in the image, is then used to estimate the actual distance from the camera to the object. The location of the object on the image plane is then used to calculate the orientation of the tube from the robot.

One unexpected problem was that moving objects in the scene tend to generate many false positive object sightings, resulting in significant added computation. The interlaced scanning of the CCD cameras electronics require 1/60th of a second between even and odd scan lines, and the boundaries of moving objects create an interlacing of bright and dark bands one pixel wide. These readings are initially flagged as potential objects that require additional heuristic processing. While the algorithm removes most of the false readings using heuristics, the additional computations are a burden that can be eliminated.

To avoid this extra processing, a band-width constraint was added, requiring at least two pixel widths on the image plane before acceptance. Theoretically this would decrease the

effective distance of identifiable objects. In practice however, this is almost never the case because it is very rare for all of the black bands to be discernable when they are only one pixel wide on the image plane. Usually only a few bands are noticeable at such distances, in which case the object cannot be identified anyway. Experimentally, the minimal band width for effective recognition was found to be two to three pixels.

## 3.2 Object recognition: Flakey

In the recognition process, it is important to distinguish between object *types* and *individuals*. Type recognition occurs when the process identifies the object as belonging to a class, e.g., a chair or a box or a person. Individual recognition occurs when the object is identified as a unique individual, differentiated from others in the class. Obviously, the latter is a stronger form of recognition and conveys more information.

The basic design philosophy of the SRI team is to leave the environment as it is, rather than engineering it to make the task easier. Thus, Flakey's recognition capabilities are based solely on the naturally occurring objects in the arena: boxes, object poles and walls. Because these objects bear no marks that Flakey's perceptual system could use to individualize them, type rather than individual recognition is performed. Flakey uses external information to promote type recognition to individual recognition, e.g., the location of an object pole serves to differentiate it from all other object poles. This process is called *registration*, since it primarily involves registering a perceptually-typed object with stored information about an individual of that type.

Flakey's type recognition routines are able to recognize all the objects in the arena (boxes, walls, and object poles) with varying degrees of certainty, depending on the situation and the sensor. This section contains a description of type recognition for walls and object poles, the two most important objects for navigation. Subsequent sections on map design and planning deal with the problem of registration.

### 3.2.1 Walls

Flakey recognizes walls on the basis of coherent sets of its side sonar readings. Low-level perceptual routines segment the readings to form piecewise linear surfaces. Since walls are the longest straight objects in the environment, a simple length filter is sufficient for identifying candidates. Candidates are then either accepted of rejected as real wall signatures by the registration routines described in Section 4.2.2 below. Figure 8 shows a typical wall section identified by the perceptual routines.

Flakey also has the potential to integrate surfaces constructed from the structured light sensor with those found by the sonar routines. While this method would lead to more robust recognition in general, it did not produce enough of a performance gain in the competition domain to warrant its use.

### 3.2.2 Object poles

Flakey's pole-recognition capabilities are based solely on physical properties of unenhanced object poles (namely, their size and roundedness) and *a priori* environmental knowledge made available to all participants. The key environmental features are that non-pole objects are significantly bigger than object poles and are at least one meter from all other objects (specifically walls, boxes and object poles). Flakey and Huey (from Brown University) were the only entries that attempted recognition of unenhanced object poles.

Flakey's method of object pole-recognition employs a two-tier approach. Objects that might possibly be object poles (referred to as *candidates*) are detected from sonar input. A verification routine based on structured-light data is used to categorize objects as pole objects or non-pole objects.

For verification, a shape-interpretation routine uses the structured-light depth map to find any pole objects in its field of view. This routine extracts from the data those rounded objects whose sizes are close to that of object poles, filtering out objects that fail the constraint of non-proximity to other objects. Figure 12 illustrates the quality of the pole signatures produced by the structured light routines; extracting rounded objects from this data involved simple tests on the Laplacian of the one-dimensional input. Recognition of object poles using structured light in this manner is highly accurate; however, Flakey's vision is nearsighted: the range of the structured-light system is restricted to a conical area one to four feet in front, within a 30-degree angle.

It would have been impossible to find many object poles within the alloted 20-minute period using the structured light verification routine, given the large size of the competition ring. For this reason, Flakey monitors its side sonar readings while in transit for the presence of *candidate* object poles. The sonars cover a much broader area than does the structured light, extending to approximately two meters on either side of the robot. Moving in a straight line, Flakey sweeps out an area in front and to the sides of approximately four meters. Once a candidate has been detected, Flakey temporarily deviates from its current path in order to get the candidate in the field of view of the structured light, and verify or disqualify it.

Sonar information is used to identify candidate rather than full-fledged object poles due to the difficulties involved in interpreting sonar data. Fragments of boxes and walls can sometimes have sonar signatures that look very similar to the signatures of object poles. For example, boxes in sonar range whose corners pointed directly at Flakey often produce sonar signatures that are indistinguishable from those of object poles. The difficulties involved in differentiating object poles from fragments of other objects are most problematic when Flakey is turning towards or away from such objects, due to difficulties that arise in registering successive sonar points accurately with respect to each other.

Candidate object poles are derived from clusters of sonar points that satisfied conditions of coherence, total area and nonproximity to other objects. Some pruning of candidates is done based on historical information: areas where object poles could not possibly exist were marked in the local coordinate system. For example, object poles can not be situated in areas close to long surfaces (from either walls or boxes) nor near failed candidate object

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Raw Data* : | 0 | 0 | 0 | 78 | 80 | 81 | 81 | 81 | 81 | 80 | 79 | 77 | 0 | 0 0 |
| *Laplacian* : | * | * | −2967 | 3062 | 3346 | 653 | −31 | −27 | −44 | 674 | 3384 | 308 | −2984 | * * |

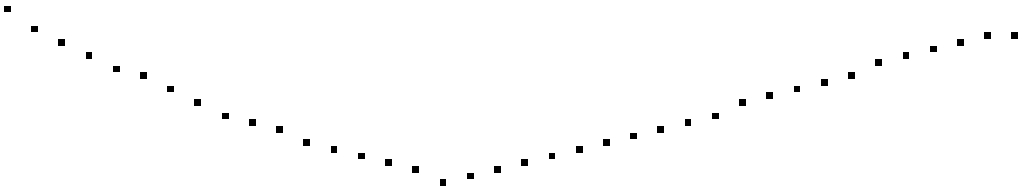


Figure 12: *Top:* A numerical and pictorial display of the structured-light depth signature of a pole located approximately one meter in front of Flakey. The points are the reconstructed spatial position of the pole surface. The data readings are taken directly from the camera; lower readings indicate the reflection is from further away. A value of 0 indicates that no surface was detected within the depth limits of the structured light system for that position. The semi-circular shape of the pole is easily detected from the smoothed Laplacian of the one-dimensional depth information: a pole appears as a sequence of strong positive readings followed by two to five negative readings, then another sequence of strong positive readings. Intuitively, such a pattern in the Laplacian corresponds to two zero-crossings of the second derivative, one for each side of the pole.
*Bottom:* The structured-light signature of a box corner viewed from approximately the same distance.

poles. In addition, classification of a location as having a candidate pole is delayed until the candidacy test was passed several times. This last filtering is essential in eliminating candidates that were not object poles. It also leads to a seemingly theatrical behavior on Flakey's part: often, a candidate would not be declared until after Flakey had passed one or two meters beyond it. At that point, Flakey would turn around and proceed back to the candidate for verification using the structured light.

Flakey's pole-recognition routines worked extremely well in the competition. The structured light verification routines were completely accurate, recognizing all object poles that occurred within the structure light field and not misclassifying any non-pole objects as object poles. Flakey's use of sonars to detect candidate object poles proved invaluable in the competition. Five of the eight object poles found by Flakey were originally detected using the sonar recognition. The various methods used to filter candidate object poles were

critical to Flakey's performance, given that the validation/invalidation of a candidate pole consumed valuable exploration time (anywhere from 20–45 seconds). Only one non-pole object (a box) was labeled as a candidate pole; Flakey succesfully classified this object as a non-pole using the structured light routine. Flakey did pass by two object poles during its first circuit of the ring without having the sonars detect them as candidates. These omissions occurred because the object poles were too close (approximately 20 cm) from the side sonars of the robot. However, Flakey found the object poles on its second pass around the ring, when it passed by them at greater range.

### 3.3 Issues in Object Recognition: Analysis

The recognition components of CARMEL and Flakey performed extremely well during the competition and were instrumental to the success that both teams enjoyed. It is impossible to rank one approach as superior to the other, given the vastly different technologies and assumptions upon which they are based. Both, however, have certain characteristics that distinguish them.

Flakey's recognition routines had the virtue of not requiring modifications to the environment. Although the rules permitted doctoring object poles in order to make them more easily recognized, the SRI team demonstrated that such modifications were unnecessary. Rather, reliable object type recognition was possible using only physical characteristics of the objects and simple domain constraints (such as nonproximity to other objects).

Flakey performed recognition for *classes* of objects (both walls and object poles) based on physical characteristics of those objects. As such, particular object poles or walls were distinguishable only by using information about the individual's location. In contrast, CARMEL detected particular object poles by recognizing specialized tags that served both to facilitate perception and identify individual object poles. As such, CARMEL performed recognition of tagged individuals, and only those individuals.

CARMEL's use of long-range sensing provided a strategic advantage in Stages 2 and 3 of the competition since object poles could be found from 12 meters away (over half the diameter of the ring). In this regard, CARMEL contrasts sharply with Flakey who could recognize object poles and candidates only within its local perceptual space. This point is discussed further in the following section. Note that CARMEL's exploration strategy worked well for this competition, in which the poles were guaranteed to be visible over the obstacles, but this strategy would not be as successful in many domains.

## 4 Issues in Mapping

In order to successfully participate in Stage 3 of the competition, it was necessary for the robots to generate maps of the environment during Stage 2. These maps would include, at a minimum, the location of poles that had been discovered. In addition, maps could contain further information that teams felt would be useful, such as the position of obstacles or walls. Complementary to map construction is the problem of *self-localization*, which involves

having the robot determine where it is relative to the map. A critical issue faced by both robots in solving these problems was the inaccuracies inherent to dead-reckoning.

## 4.1 Map design

Automated map generation remains a topic of current research for the field of robotics. Michigan and SRI chose two very different approaches in the design of maps for their robots.

### 4.1.1 Map design: CARMEL

CARMEL used a single global coordinate system to keep track of object locations. One corner of the arena was labelled as (0,0). As CARMEL took an image, its vision routine would return the heading and distance of each object in the scene. That heading and distance was transformed to the global coordinate system and the object was placed in the map.

CARMEL also keeps a separate histogram grid map of the entire arena, which is used for obstacle avoidance. This map also has a single global coordinate system that is identical to the coordinate system used to record object locations. When CARMEL wished to move to an object it was simply a matter of setting the goal of the obstacle-avoidance system to the coordinate location of the object (actually, to a point slightly in front of the object). The histogram grid map was periodically cleared of readings to eliminate clutter from errors and moving objects, thus no obstacle locations were permanently maintained. CARMEL could use the histogram grid map of obstacles to plan a local path around known obstacles, however, this was only used when CARMEL was trapped and the VFH obstacle avoidance system could not extricate CARMEL from the trap.

### 4.1.2 Map design: Flakey

In contrast to CARMEL, Flakey does not try to maintain a global coordinate system map. The primary reason is that Flakey's sensing is too short range to construct a reliable map of large spaces using triangulation techniques. Consider Figure 13, in which Flakey's sensor zones are superimposed on the arena. If Flakey were given a completely accurate Cartesian map with all the objects indicated, it could navigate by going to the nearest object, reducing its dead-reckoning errors by sensing the object, move on to the next object, and so on. There are enough objects in the ring to prevent dead-reckoning errors from accumulating too much between objects. However, the competition rules stated that no prior knowledge of the location of poles or obstacles would be allowed. So Flakey has to *construct* the map, a much harder task than just staying registered. Because Flakey can only sight one object at a time, it cannot use triangulation to reduce its dead-reckoning errors, and these will accumulate during map construction. By the time it travels from one end of the arena to another, the errors are substantial, and render the map-construction process futile.

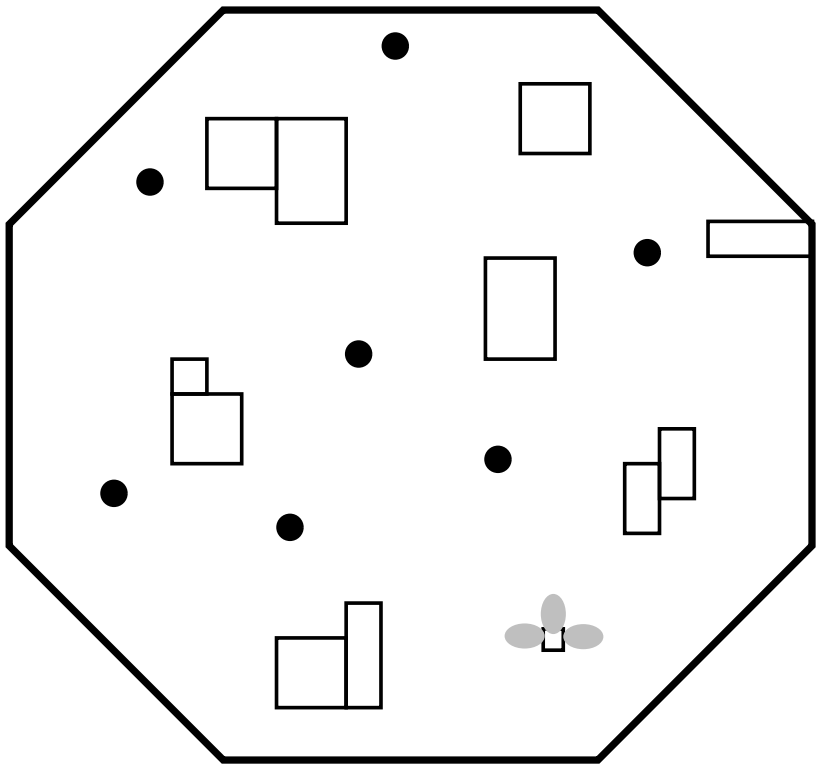


Figure 13: The relative range area of Flakey's sensors is small compared to the size of the arena and the distance between objects. The robot and its sensor zones are the small object in the lower right corner. Flakey can not see more than one object at a time, and significant distances separate objects. It is impossible to perform triangulation among the objects.

---

Instead of a global Cartesian map, Flakey relies on a representation of space called a *tolerant global map*. This map is a network of places, called *patches*, and connections between them. It is "tolerant" in the sense that the connections can contain imprecise metric information. For the competition, each wall was chosen to be a patch, and its approximate length and the angle of its neighboring patches were stored. Figure 14(a) shows the two patches for walls A and B. The metric information was derived by an imprecise survey of the arena using a tape to measure the length of the walls, and a protractor for the angles between walls. The survey took only a few minutes, with the resulting imprecision visible in Figure 14(b). In fact Flakey itself could have carried out this survey to the same degree of accuracy, which is one of the advantages of the tolerant global map.

Each patch contains several elements:

- Landmarks. These are detectable sensory features that can be used to keep Flakey

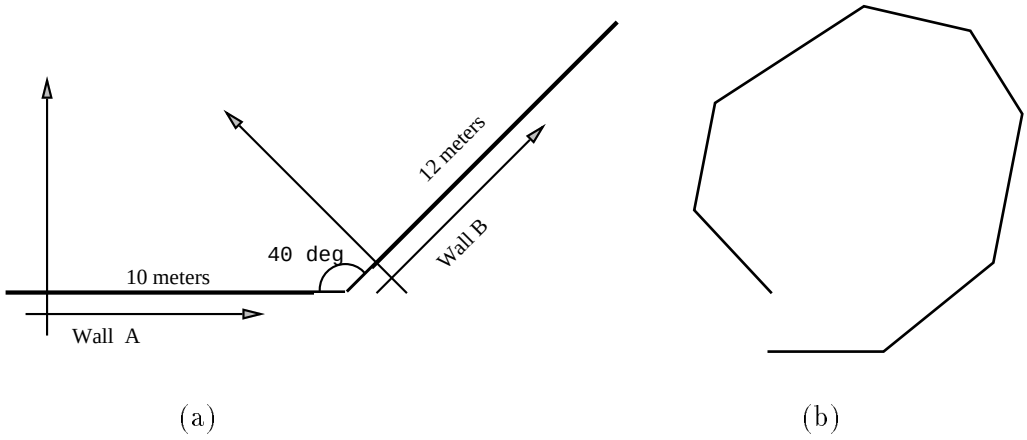



Figure 14: Flakey's representation of space is called a *tolerant global map*, comprised of a network of local *patches*. In (a), two wall patches are shown with their approximate lengths and the angle between them. The crossed axes are local Cartesian coordinate systems that are used for local navigation within each patch. In (b), Flakey's *a priori* map of the arena walls is laid out in Cartesian coordinates. The map was constructed by an imprecise survey of the arena using a tape to measure the length of the walls, and a protractor for the angles between walls. Note the large accumulated error in traversing all the segments of the ring.

---

registered with respect to the patch. Note that landmarks do not have to be points; in the competition, both wall junctions and entire wall surfaces were used as landmarks.

- Local coordinate system. Each patch has its own local Cartesian coordinate system for dead-reckoning movement.
- Behavioral information. Patches are "behavior-centered" in that movement to the borders of the patch is specified in terms of behaviors.
- Connections to other patches.

The "patchwork" representation of space lends itself to two different modes of navigation. Over small distances, dead-reckoning with respect to the local coordinate system is possible. The landmarks of the patch are used to keep Flakey registered with respect to this local system. Dead-reckoning movement is useful for various activities; in the competition, these included moving to a candidate pole to verify it with the structured light sensor, revisiting targeted poles and making forays into the interior of the arena (Section 4.2.2).

The second mode of navigation is behavior-based, and does not require precise dead-reckoning capability. For the wall patches, the obvious means to get from one end of the wall to another was the FOLLOW-WALL behavior. Behavioral movement is tolerant of

imprecision in the geometry of the patch. Even if the representation of walls as straight segments is inaccurate, the wall-following behavior can follow walls with substantial degrees of curvature, reliably achieving the goal of getting to the end of the patch.

#### 4.1.3 World modelling: Analysis

As mentioned above, global Cartesian maps are generally inappropriate for large spaces because of the accumulation of dead-reckoning errors. CARMEL's dead-reckoning capabilities and long-range sensing were sufficiently good to enable the use of a single global Cartesian map for the competition ring. Because the SRI team used only walls and their junctures as landmarks (rather than adding unique tags to the poles) and because Flakey's sensing was limited in range, the SRI team opted instead for a tesselated view of space, a series of local maps linked together.

The use of a global map combined with its long-range sensing capabilities gave CARMEL a marked advantage over Flakey in the competition. The global coordinate system made it easy both to find objects and to determine trajectories for navigating among them. Had the arena been much larger, managing a single global coordinate system map would have been difficult and a system of linked local maps, like that used by Flakey, might have been necessary.

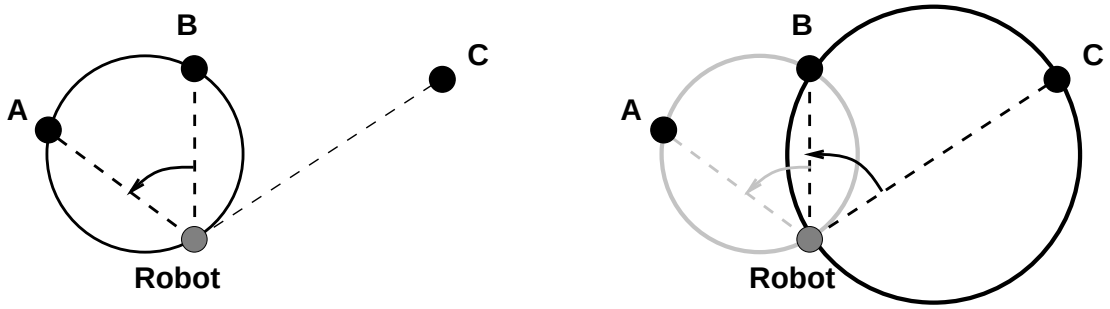
Flakey's tolerant global map led to a more involved process for finding and visiting objects, however, this approach was essential to Flakey's good performance. Division of the ring into local spaces of moderate size with well-defined landmarks (walls and their junctions) made it possible to limit the accumulation of dead-reckoning error, whereas operating within a single Cartesian coordinate space would have caused Flakey to become lost quite rapidly as dead-reckoning errors accumulated. Flakey's ability to rerecognize poles that it had discovered previously demonstrated that the registration of poles with respect to local coordinate systems was a success. Although Flakey encountered some difficulties in the use of the tolerant global map (as discussed further in Section 5), the SRI team feels it was indispensible to Flakey's performance.

### 4.2 Position Correction

Both CARMEL and Flakey used dead reckoning to keep track of their position and orientation in Stages 2 and 3. Because dead reckoning grows increasingly inaccurate over time (due to wheel slippage, etc., particularly in the orientation of the robot), some means of reorienting the robot at intervals was required. Both teams employed landmarks to adjust the robot's beliefs about its position, although different types of landmarks were used by each.

#### 4.2.1 Position correction: CARMEL

There are many different approaches to position determination, including mounting beacons or returning to a specific home position. These approaches require engineering the




(a) (b)

Figure 15: Three-object triangulation can be used to recalibrate the robot's position when dead-reckoning errors have accumulated. To use this method, the locations of three objects A, B, and C are known, and the directions to the three objects from the perspective of the robot are known; the robot's location is unknown. In (a), objects A and B, plus the angle between them (relative to the robot) are used to calculate a circle; the robot is somewhere on the perimeter of this circle, but its exact location is not yet known. In (b), a second circle is similarly calculated, using objects B and C. The two circles intersect at two points: the location of object B and the location of the robot.

---

environment and/or repeatedly returning to certain locations, activities that are generally not desirable, and possibly not feasible. A better solution for absolute positioning is one that uses objects already found in the environment (i.e. landmarks), in this case the object poles.

The method used on CARMEL for triangulation is based on circle intersection (see [5] for an overview of three-object absolute-positioning algorithms). Given objects A, B, and C, a circle is formed with objects A, B, and the angle between them as viewed from the robot, as shown in Figure 15. Because this angle is a relative measure, the known x and y coordinates of the robot are not required. A second circle is formed similarily with objects B, C, and the angle between them. The two circles intersect at two points: object B's location and the robot's location. Therefore, the robot's location and orientation can be determined.

As in most triangulation methods (see [5]), this method is affected by errors in an object's location and angle measurement. Error in the angular separation between objects was less than 1 degree; object distance was similarly quite accurate, typically within 2% of the actual distance. Also, the robot's position, with respect to the objects, determines how sensitive the absolute positioning solution is to these errors. It was discovered that the accuracy of the triangulation method was highly dependent on whether or not the robot was inside a triangle formed by three viewed objects. When inside such a triangle, the error in absolute positioning is very small (on the order of a few tens of centimeters). However,

small errors in object position cause large triangulation errors when the robot is outside a triangle of objects. A heuristic search was employed to find the best combination of three objects, in the most recent visual sweep, with which to perform the triangulation. This might fail, however, if there were an insufficient number of objects in the most recent visual sweep, or if the objects seen were too poorly localized. CARMEL would then have to move to a point where it is inside a triangle formed by three objects whose locations are known precisely enough to be useful, perhaps taking a new visual sweep to improve the accuracy of the triangulation. Note, however, that this approach assumes that CARMEL is never terribly lost — the robot must be able to find its way into a triangle of three objects with certainty. Otherwise, more drastic recovery mechanisms are called for. In practice runs, CARMEL never became lost to such a large extent.

Unfortunately, although the algorithm was tested extensively in simulation and on CARMEL, last-minute changes at the competition prevented its incorporation. CARMEL was able to compensate for dead-reckoning errors using other means (see Section 5). However, the success of a global coordinate system map relies on an absolute positioning algorithm like the above.

#### 4.2.2 Position correction: Flakey

Because the SRI team did not add any identifying artifacts to the domain, it would have been extremely difficult for Flakey to recover in the event that it became lost. For this reason, it was critical that Flakey maintain a reasonably accurate estimate of its position at all times. Given the patchwork nature of the tolerant global map, Flakey's position is defined in terms of both a current patch and a position within the local coordinate system of the patch. In order to acquire and maintain this knowledge, Flakey exploits the most stable features of its environment, the walls, as its chief navigational aid while in transit. Essentially, Flakey "feels" its way around the environment by keeping sonar contact with the walls, in much the same way that a blind person might use touch. Forays into the interior of the arena can be made to explore or to visit previously discovered poles but Flakey will always return to the current wall in order to reregister its position. Making a foray is a risky venture: during a foray, Flakey must rely solely on dead reckoning. The risk arises that Flakey's expectations about the position of the wall are sufficiently inaccurate to make it impossible to locate the wall at the end of the foray.

Navigating to adjacent walls is achieved through the behavior FOLLOW-PERIMETER: this behavior executes a sequence of FOLLOW-WALL behaviors over the successive walls of the arena. When the end of the current wall approaches, Flakey switches to the patch of the upcoming wall. Using prior knowledge about wall lengths and angles, Flakey can generate a rough estimate of its own position. However, Flakey will revise this estimate once it has perceived the new wall. Flakey can compute the coordinates of the intersection of this and the previous wall. The intersection marks the origin of the local coordinate system of the current patch, while the patch wall determines the orientation of the $x$-axis of the system. As such, the vertices of the octagonal arena are Flakey's main landmarks for orienting itself in a new patch. Any pole discovered while in this patch is registered with

respect to this landmark: its coordinates relative to the vertex are stored in the tolerant global map, together with the patch name.

Figure 16 shows a snapshot of Flakey's navigation along the perimeter. The double line is a *wall artifact*, representing Flakey's belief about the position of the current wall. The wall artifact is created initially based on Flakey's prior knowledge about the shape of the ring. Flakey's progress along the wall is measured by the double arrow on the wall artifact, which is updated based on dead reckoning.

Dead reckoning errors will eventually cause the registration of a wall artifact to deteriorate as Flakey proceeds along the wall, thus Flakey uses sensing to update the correspondence of the wall artifact to the real wall, and hence, also to update its own position within a patch. This updating process is called *registration*. Registration proceeds in three steps: 1) Find candidate wall segments; 2) Filter the segments to find true wall segments; and 3) Update the position of the wall artifact to the true wall. Wall candidates are found using straight-line segments extracted from the side sonar data. In Figure 16, a strong wall candidate has been found and marked with a "W." The filtering process [12] evaluates candidates for the likelihood of them being an actual wall. It uses several sources of information: intrinsic properties of the candidate such as its length, the expected position and possible deviation of the wall, geometric constraints (no candidates should ever be found *behind* a known wall), and the nature of the false alarms that could occur (straight-line segments from box obstacles). In the figure, the wall candidate is relatively long and corresponds well with the estimated wall position; hence, it is accepted as being part of the true wall.

To update the wall artifact, the orientation and endpoints of the wall are shifted to correspond to the accepted candidate. In the case of Figure 16, the shift will be small, giving a slight correction to the orientation of the wall artifact. Over a distance of ten meters, however, the dead reckoning errors can be significant, making the constant registration provided by sensing essential. Recall that updating the position of the wall with respect to Flakey corresponds to updating the coordinates of Flakey with respect to this patch's local coordinate system.

The wall registration procedure, especially filtering, requires *a priori* knowledge of the location of walls. This knowledge need not be very precise; for the competition, crude estimates of wall lengths and angles of intersection were made by hand and input to Flakey. Figure 14b displays just how approximate those measurements were: the picture was obtained by drawing the successive clockwise walls from a fixed start wall using *a priori* knowledge given to Flakey. Throughout the competition, Flakey would constantly engage the walls with its sonars, making only limited forays (about 7 meters deep) into the arena interior when necessary. If Flakey were started at a known point along a wall, then it should keep registered throughout its travels.

There is always the danger that the mismatch between wall artifacts and actual walls will become too great for the registration routines to handle. This could happen in a number of ways: if the walls were obscured for too long a stretch by obstacles, or an interior foray lasted too long, or an obstacle was mistakenly identified as a wall. In such cases, substantial mismatches would be detected by failed expectations: Flakey would place the wall artifact

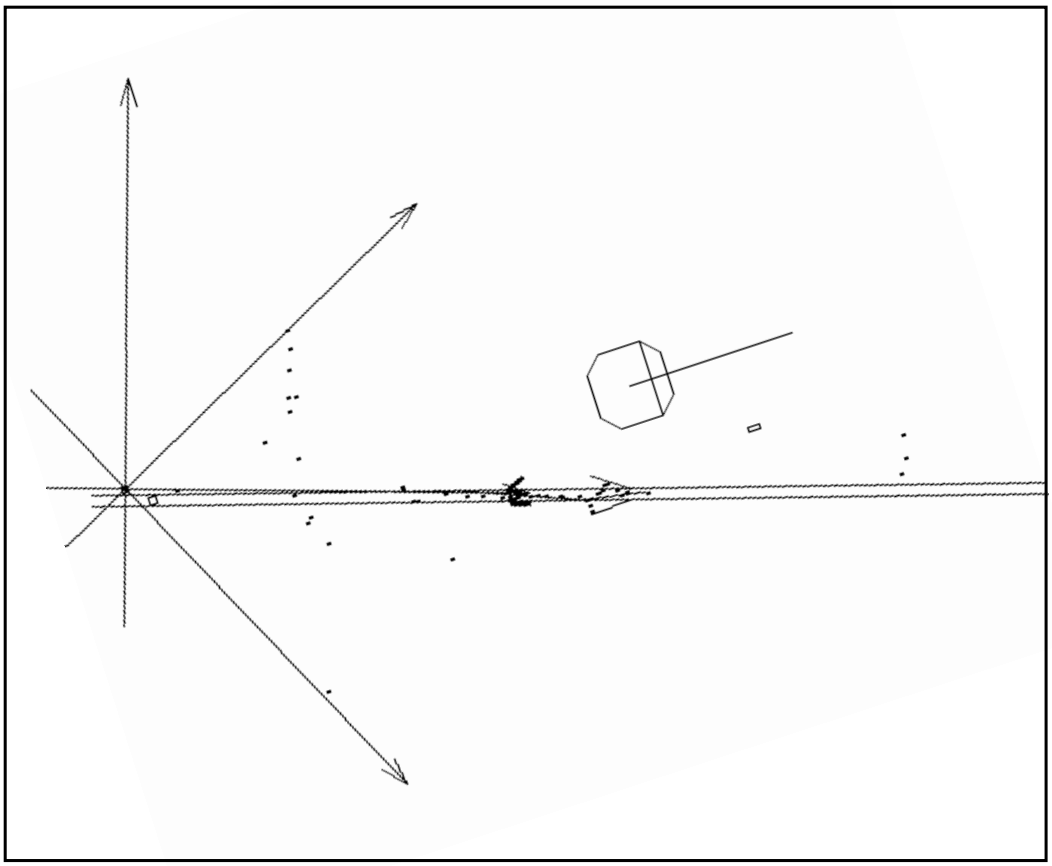


Figure 16: Flakey navigates by referring to wall patches. The long double line is a wall artifact: a straight-line segment of fixed length that serves as the x-axis of the patch. The origin of the wall depends on the direction of travel of Flakey; in this case, it is at the left-hand end, and is indicated by the vertical coordinate system arrow at that end. Note that there is a second patch coordinate system superimposed here, leftover from the termination of the previously traversed wall. The relative orientation of the patches indicates the approximate angle for Flakey to turn upon arriving at the new wall. (The next wall, the wall to the right of the current one, is too far ahead to be visible in this picture.)

---

in a region in which no reasonable candidate could be found when the artifact was not obscured.

Flakey will not necessarily become lost in the event that it loses track of a wall. Flakey's estimate of the position of the next wall will remain reasonably accurate, even when the

previous wall is lost. Since the vertex-finding routines are more tolerant than their wall-registration counterparts, the new wall can often be registered and Flakey's estimate of its position adjusted. The vertex position will still be inaccurate, as it depends on the estimated position of the previous wall. After registering the current wall, however, the next vertex can be reliably positioned. Thus, if Flakey can recover two consecutive walls, it will be completely registered again. On the other hand, should Flakey become very lost, it must plan to *reacquire* its position estimate. General reacquisition of position can be a difficult and time-consuming task, since the robot must look for and recognize landmarks, and decide if a particular configuration of landmarks corresponds to a unique location within the map. The SRI group is currently working on the reacquisition problem, but was not able to give Flakey this ability for the competition.

### 4.2.3 Position correction: Analysis

CARMEL's vision-based triangulation algorithm was not used for the competition because of last minute changes to the code (which produced incompatibilities with the position-correction code). This did not cause any run-time problems for CARMEL because its planning code was designed to be robust to dead-reckoning errors; furthermore the speed with which CARMEL completed the stages did not allow large positional errors to accumulate.

In contrast, Flakey relied extensively on its position correction mechanism to track its position in the arena. Flakey's ability to recover walls and rerecognize poles that it had discovered previously jointly demonstrated that the use of relative landmarks (here, walls and wall junctions) for position correction was successful. (Flakey did encounter some positional difficulties near the end of its Stage 2 run, an issue that is further explored in the next section.) The cost of this approach, however, was the need to stick close to the walls of the perimeter. This requirement greatly slowed down Flakey's exploration and pole-visiting activities.

The different approaches taken by the two teams towards position correction illustrate a fundamental trade-off between generality and robustness. As noted above, the lack of absolute reference points in the environment meant that Flakey would have been unable to correct its position should its estimate of its position have become extremely inaccurate. Even with absolute references, Flakey's lack of long-range sensing could have made position adjustment very time consuming. The Michigan team's approach based on long-range sensing of absolute reference points is certainly better in domains where it can be applied.

## 5 Issues in Planning

There were two major planning tasks in Stages 2 and 3. The first was to explore the arena and visit and map objects (Stage 2). This meant ensuring that the robot "looked" everywhere for objects and that it picked a visition order. The second planning task was to go directly to three of the objects found in Stage 2 and return home (Stage 3). These two tasks will be studied in this section.

## 5.1 Exploration Strategies

To be able to find all of the objects, the robots would need to explore the entire ring, perhaps redundantly. There was no prior information about object locations, requiring a general and thorough exploration methodology. There was a 20-minute time limit on finding all ten objects.

### 5.1.1 Exploration strategies: CARMEL

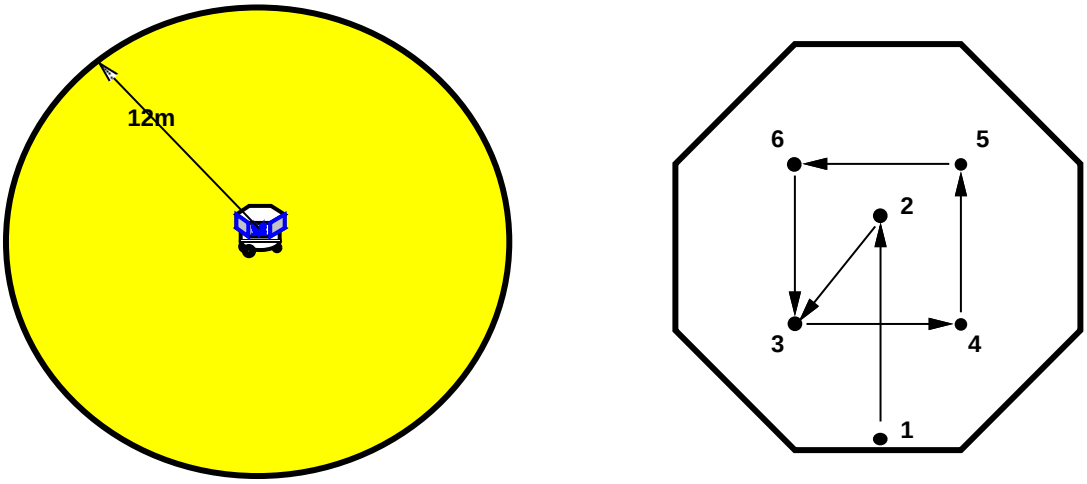
CARMEL's vision system could identify objects from a 12-meter distance, and experimentation showed that it was very robust and accurate in determining objects' locations. Such a large visual range greatly reduced the amount of motion around the ring required to cover the area visually. From a single location near the center of the ring, it was possible to see *all ten* of the objects, although it was possible for some objects to occlude others. Therefore, CARMEL was programmed to simply take a 180-degree vision sweep at the start of the run, and move across the center of the ring to a point slightly past the center, and take a full 360-degree sweep. In the event that these two vision sweeps did not see all the objects, four additional vision locations were defined to form a square roughly 8 meters on a side, centered within the ring. At each of these locations, another 360-degree vision sweep could be done, if needed. In actual competition, only the first two vision sensing locations were needed. The vision locations used for Stage 2 are illustrated in Figure 17

By taking two visual sweeps of the entire ring relatively early in Stage 2, the object map created was reasonably accurate and was known to not contain any large dead reckoning errors. This would help in visiting the objects in Stage 2 and in Stage 3, and in dealing with dead reckoning errors that developed through the course of the exploration. The absolute positioning algorithm compares the objects in the map against the objects from a camera sweep and recalculates CARMEL's location in the ring, so it is important that the map be as accurate as possible (as was stated in Section 4.2.1 , the absolute position algorithm was not used during the actual competition).

Once seen, objects were visited using a simple point-to-point strategy — the robot simply went to the next closest unvisited object. Using a planner to determine an optimal visitation order would have been time consuming and unnecessarily complex (the travelling salesperson problem), since any two points were never terribly far away.

When visiting each object, CARMEL moved to a point slightly in front of the object and then took another image to update the object's position and heading relative to the robot. This relative distance and heading were used, rather than the global map, to calculate a final approach vector. This ensured that CARMEL approached the object within the required two-robot-diameter distance.

The planning system had to field a number of potential errors from subsystems in Stage 2. There are two major causes of errors that the planner has to deal with: 1) Errors in movement, whereby the robot could not move to the requested position; and 2) Errors in object location, whereby the vision system could not reacquire an object once it had approached it. There are two types of movement errors: 1) Traps (concave formations




(a) (b)

Figure 17: CARMEL has a vision range of approximately 12 meters, as illustrated in (a), which enables the robot to see the entire ring (approximately 21 meters in diameter) from a central location. Because one pole may block CARMEL's view of another pole, one view is not necessarily sufficient to see all ten poles, and a contingency plan of multiple locations is called for. CARMEL was given a series of six vision locations, illustrated in (b). Due to dead-reckoning errors that accumulate as CARMEL moves, the first vision location contains the most accurate information; however, it covers less than half of the ring. In the actual competition, only the first two vision locations were needed.

---

of obstacles) which are detected automatically by VFH; and 2) Failure to attain the goal location, because, for example, the goal is surrounded by obstacles. Traps are recovered from by using a global path planner to plot a set of goal points that will lead the robot out of the trap and to the goal location. Failure to attain the goal location errors are handled differently depending on the situation. In some cases, CARMEL may be close enough that it doesn't matter if it attained the goal location. If CARMEL is not close enough, the planning system may decide to to try again or may choose a new goal location.

Recovery from errors in object location are handled in several ways. First, CARMEL can rotate its camera left and right in an ever-widening arc, searching for the object. If this fails, the planner assumes that CARMEL is too close to the object to detect it (the vision system has a minimum range of one meter) and so it tells CARMEL to back up. If this fails, the object is assumed to be occluded and the planner chooses a new goal location on the opposite side of the object and starts the verification process over again.

At the end of the run, a map of the objects was saved to a file to be used in Stage 3. The current map was also saved at intervals during the run in the event of some unforeseen problem or in the event that CARMEL was unable to complete Stage 2 in the 20-minute

time period.

### 5.1.2 Exploration strategies: Flakey

Given the limited range of Flakey's sensors, the search for objects in a large-scale space requires a physical traversal of the space to ensure complete coverage. For the competition, it was necessary that Flakey traverse the entire area of the arena, modulo its 4-meter wide perceptual field (cf. Figure 13). To this end, Flakey employed *purposeful behaviors* built out of the low-level reactive behaviors, and *tasks* built out of purposeful and reactive behaviors. The architecture shown in Figure 4 illustrates Flakey's hierarchy.

Reactive behaviors, such as AVOID-OBSTACLES, provide a reliable basis for safe movement. *Purposeful behaviors* provide more directed activity by taking explicit goals into account. While reactive behaviors respond to perceptual features, purposeful behaviors repond to abstract features in the local percpetual space called *artifacts*. Typically, an artifact indicates a goal to achieve or maintain a certain relation between itself and the robot. Artifacts used for the competition include *control-points*, for achieving $(x, y)$ positions with a given heading and velocity; *lanes*, for following corridor-like strips of space with fuzzy markings; and *objects* such as poles to be visited. Once an artifact is introduced in the local perceptual space, its position with respect to Flakey is continuously updated as Flakey moves. A purposeful behavior is built by writing a fuzzy rule set whose goal is to keep Flakey in a certain relation with the artifact. This rule set is typically combined with other rule sets (or sub-behaviors) that take care of other, concurrent goals (e.g., avoid obstacles along the way). Combination of concurrent goals, or *behavior blending*, is performed using the methods of fuzzy logic [14]. Figure 18 shows the result of blending a wall-following behavior with an obstacle-avoidance behavior (here, KEEP-OFF). Context-dependent blending of different behaviors allows Flakey to exhibit reliable operation in unstructured dynamic environments.

Coordinating purposeful activities to accomplish complicated goals requires yet another level of control. *Tasks* provide a simple means of combining purposeful behaviors with perceptual events (e.g., a landmark has been reached, or a pole candidate recognized). In essence, a task is a finite-state machine whose transitions are conditional on perceptual events. A state transition can activate a behavior or another task as a side effect. Hence, tasks provide a simple robot programming language that uses both reactive and purposeful behaviors as primitive operators and state transitions as the main control mechanism. Tasks are organized in a stack where the topmost task is in control; when a task completes, it pops itself and new topmost task assumes control.

Flakey's exploration strategy for the competition is defined by the task EXPLORE shown in Figure 19. Flakey is started along a wall and given a rough estimate of its original position. To begin, Flakey activates the FOLLOW-PERIMETER behavior to commence an initial trip around the ring and immediately enters the *1st-round* state of the task. The purpose of this circuit is to allow Flakey to detect poles located near the perimeter of the ring. There were three main reasons for chosing a strategy based on perimeter following: first, this allows Flakey to effectively use the walls of the arena as primary registration cues; second,

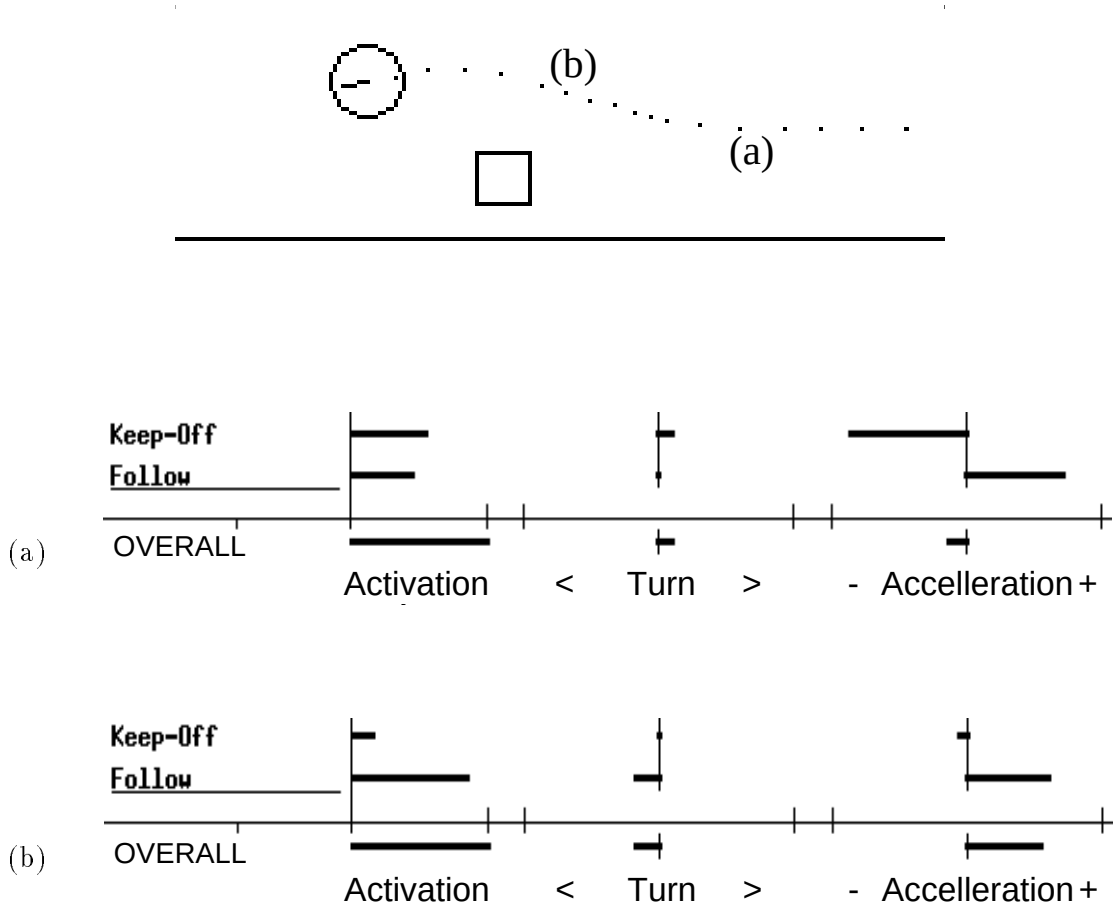



Figure 18: Flakey uses fuzzy rules to blend reactivity with purposeful action. In the top figure, Flakey moves from right to left. At location (a), an obstacle has been detected, and the preferences of KEEP-OFF are dominating; later, at location (b), the path is clear, and the goal-oriented preferences expressed by FOLLOW re-gain importance. The bars show the level of activation and the preferred controls for each behavior, and the result of the blending.

---

the available prior information about the shape of the ring could be integrated; third, this provides a cheap indicator of the current state of coverage of the arena. Flakey continuously monitors the distance to a wall to ensure safe registration during the FOLLOW-PERIMETER behavior: if it finds itself too far from the wall (e.g., after having checked a pole far away, or avoided a large cluster of obstacles), it immediately pushes a RECOVER-WALL task onto the task stack, making it the currently active task.

While proceeding around the perimeter, Flakey monitors its sonar input for pole can-

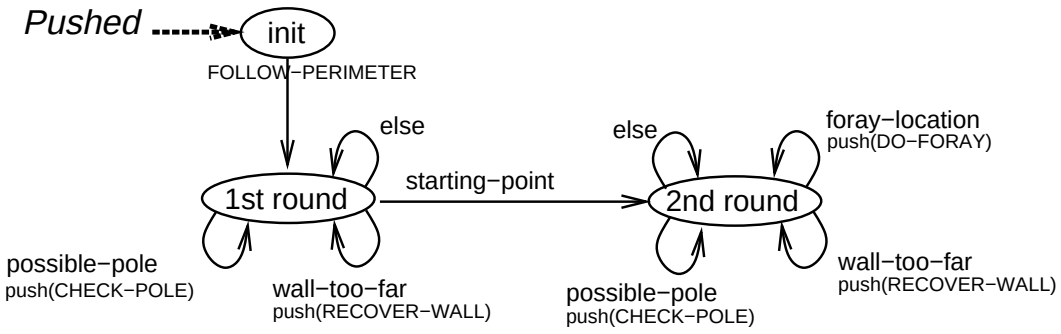



Figure 19: The tasks used by Flakey are finite-state machines. The EXPLORE task, is illustrated here; state transitions are labelled by the triggering perceptual event (lower case), and by the behavior to be activated or task to be pushed (upper case).

---

didates. When a candidate is found, Flakey pushes a CHECK-POLE task to execute the sequence of actions involved in verifying a pole candidate using the structured-light sensor: i) turn to face the pole candidate; ii) move close enough for the laser to hit the pole; iii) turn slightly to ensure that the pole is in the visual field; and iv) resume previous task. Upon verification of a pole using structured light data, the pole is registered in the tolerant global map, using the local coordinates of the pole in the current patch.

After completing an initial circuit of the ring, Flakey enters the *2nd-round* state of the EXPLORE task. Flakey proceeds with a second circuit but this time makes forays into the interior from three predetermined walls. A foray consists of three connected corridors called lanes. The width and length of these lanes are fixed (about 3 by 7 meters), but their confinement is elastic: by using its fuzzy rules, Flakey tries to stay within the lanes but will deviate from them to avoid obstacles.

Figure 20 depicts the route traced by Flakey during a simulated run of Stage 2. Here, Flakey travelled in the counter-clockwise direction, beginning along the top wall heading left. The effects of various CHECK-POLE maneuvers are visible near several poles that were classified as pole candidates by the sonar routines. After having traversed the perimeter once, Flakey executed a foray from the top wall, during which it discovered two new poles. The actual foray was much longer than the one originally planned (the corridor marked by the dashed lines) due to the large cluster of boxes encountered during the phase of the foray running parallel to the wall. The simulator includes noise in both movement and sensing comparable to that experienced by the real robot.

### 5.1.3 Exploration strategies: Analysis

It is difficult to compare exploration strategies on their own, because they are closely tied to the sensing capabilities of the robots. With its excellent long-range object recognition

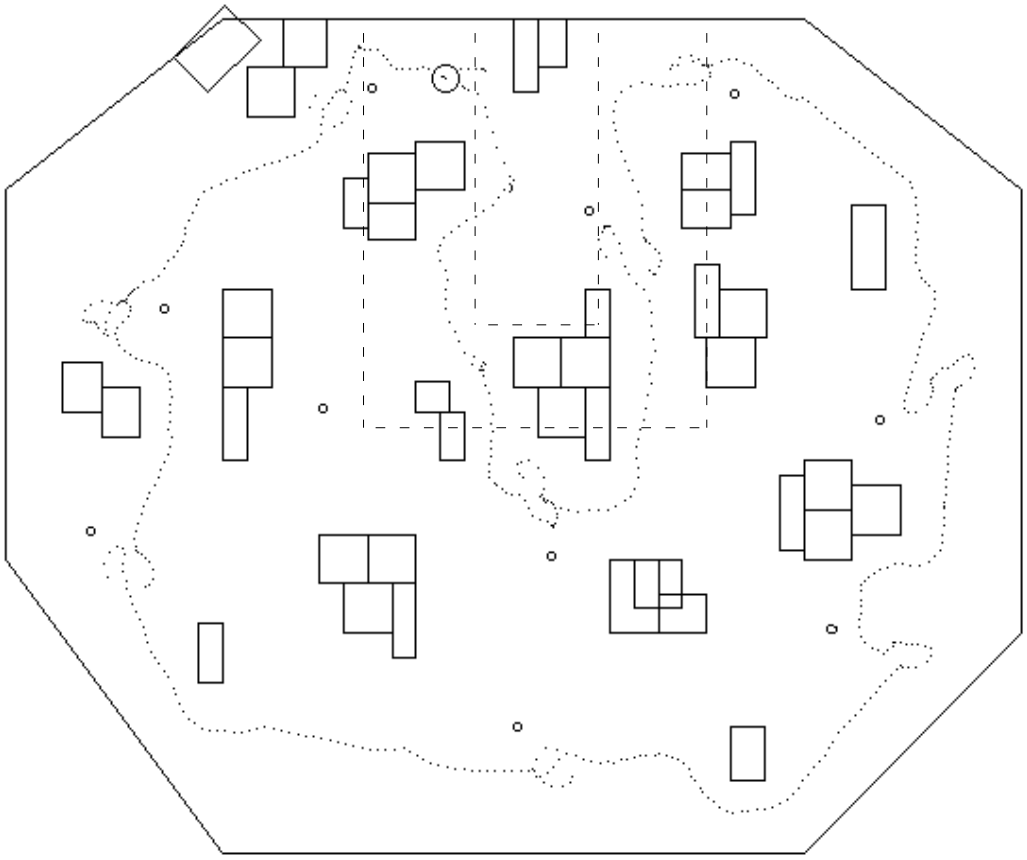


Figure 20: The dotted path traces the movements of Flakey during a partial run of Stage 2 for a simulated competition arena. Flakey started from the middle of the top wall heading left. Here, it has finished searching the perimeter of the arena and has just returned from its first foray (the dashed rectangle in the picture). CHECK-POLE maneuvers are visible near poles.

---

capabilities, CARMEL did not have to use a very elaborate exploration strategy. Flakey, on the other hand, was burdened with short-range sensing and so it had to do more exploration.

Flakey's major exploration problems arose while it made its final foray from a wall to check a pole. During this foray, Flakey became confused about its position, causing it to misidentify previously found poles as new poles. And although Flakey did manage to find a new pole, the pole was mapped to a fairly inaccurate position. The localization problems arose because Flakey began its foray before registering the wall. Flakey's expectation of where the wall should be located (based on a priori knowledge) was fairly inaccurate; this discrepancy would have been corrected when the wall was registered. Allowing forays to be made before registration of the wall was a tactical mistake in the SRI design, and points to

the need for developing more general self-location strategies.

The Michigan team's use of long-range sensing easily enabled CARMEL to find all ten poles within the allotted 20-minute search period. In contrast, the physical exploration executed by Flakey was very time-consuming. In the end, Flakey found and correctly registered only eight of the ten poles before time expired. Certainly, the Michigan approach was superior given the conditions of the competition environment. In particular, Michigan took full advantage of the fact that objects would be visible above all obstacles. Since Flakey's method was not based on any such assumptions, it was less efficient; however, Flakey's exploration method could be employed in more realistic environments, where objects may be occluded.

## 5.2 Directed Search Strategies

In Stage 3 of the competition the robots were given three poles that they needed to visit in order and then return to a home position. This stage was timed, with the robots receiving points based on their time with respect to the other robots.

### 5.2.1 Directed search strategies: CARMEL

Since CARMEL had saved a map of the objects from the Stage 2 run, Stage 3 was straightforward. CARMEL executed a simple loop, moving to each object in turn, and verifying and identifying it as in Stage 2. Speed was not an issue here: CARMEL would probably be able to move faster than the other robots in the competition. The primary concern was again dead-reckoning errors; Dead-reckoning errors from Stage 2 would result in an imperfect map, while dead-reckoning errors in Stage 3 would result in moving to the wrong location.

However, in practice, neither of these caused much problem. Since objects were identified and placed in the map very early in Stage 2, dead-reckoning errors had little effect on the map. Errors accumulated while moving in Stage 3 were taken care of by the same object verification techniques used in Stage 2. That is, if CARMEL's dead-reckoning had accumulated enough error so that its location relative to an object was off when it went to visit the object, CARMEL would fall back upon contingency planning in an effort to locate the object. These mechanisms were included in the code, but in the actual competition, CARMEL's Stage 3 run was so fast (3 minutes) that dead-reckoning errors never accumulated to any large extent. The recovery behavior was observed by the crowd, however, on one of CARMEL's two false starts. In the false starts, a wrong initial orientation was entered at the beginning of the run. Therefore, CARMEL headed about 37 degrees away from the location of the first object, and fell into recovery mode to try to locate it, which it did. The judges subsequently gave CARMEL another chance when the error was discovered.

### 5.2.2 Directed search strategies: Flakey

The SRI strategy for visiting poles necessarily relied on perimeter-following, given that Flakey registered poles with respect to local coordinate systems based on perimeter walls. Visiting an object registered with respect to a wall $W$ involved three steps: 1) determine the direction of the shortest perimeter path to $W$ (either clockwise or counter-clockwise); 2) follow the perimeter in that direction until $W$ is encountered; and 3) use dead-reckoning within the coordinate system of $W$ to move close enough to the pole for structured-light verification.

After verification, Flakey would recover the wall of the coordinate system and repeat the process for the next pole. The starting location was registered in the coordinate system of the initial wall as an object named *home*, thus allowing Flakey to employ the same basic strategy for returning home after having visited the third pole.

### 5.2.3 Directed search strategies: Analysis

Analysis of the directed search strategies cannot be decoupled from the mapping approach used by each robot. Flakey's strategy of registering objects and itself to walls meant that it had to traverse the perimeter of the arena to reach each object. Though this was an expensive price to pay in terms of execution time, the outcome was the reliable registration of the poles in an environment far larger than Flakey's perceptual capabilities. In fact, Flakey's locative abilities were accurate enough to position it right in front of the given pole before verifying it with the structured light. CARMEL, with its global map, could quickly move across the arena, directly to the target objects.

One difference in directed search strategies was in the final, homing move. Like many other teams, the Michigan team placed an eleventh pole at the home position. CARMEL used this pole to allow for correction of any dead-reckoning errors that may have accumulated during the run. Flakey needed no such modification to the environment, but was able to treat the home location in the same manner as other positions of interest (such as foray positions or pole locations) without having to add a pole to mark the position.

# 6 Stage-by-Stage comparison of performance

Previous discussion has focussed on general issues of robot navigation that needed to be addressed by each team. This section looks at how design decisions of the teams related to the actual performances of both CARMEL and Flakey in each of the three stages of the competition.

## 6.1 Stage 1: Roaming in a Semi-Structured Environment

Flakey finished second in Stage 1 and CARMEL finished third. Flakey's success was attributed to the use of fuzzy rules that resulted in pleasing smoothness of movement. Even

given Flakey's blind spots in its sonar arrangement, its reactivity was completely reliable in avoiding obstacles and judges. In Stage 1 it moved at a top speed of about 200 mm/sec.

Using VFH, CARMEL had even smoother movement than Flakey and moved at much higher speeds. Where CARMEL lost points was its difficulty coping with being trapped and cornered by the judges. In order to ensure adequate coverage of the arena, the running speed of CARMEL had been set at 300 mm/sec, a good speed for avoiding most moving obstacles, but a difficult speed in which to react to quickly converging judges who were intent on trapping CARMEL. Thus, CARMEL brushed two obstacles and one judge.

### 6.2 Stage 2: Exploring a Semi-Structured Environment

CARMEL finished first in Stage 2 finding and visiting all ten objects in less than ten minutes. The second place finisher in Stage 2 was Buzz from Georgia Tech, which found nine of the ten objects in 20 minutes. Flakey finished third finding and visiting eight of the ten objects in the allotted 20 minutes. In addition, the SRI team earned kudos from the judges and audience for not attaching tags to their poles as most other teams, including Michigan, had done.

CARMEL's impressive performance (it was the only robot that didn't use the full 20 minutes) was due to its long-range sensing abilities and high-speed obstacle avoidance. This goes to show that good sensing can allow for simple exploration strategies, as long as the exploration strategy is designed to take advantage of the sensing capabilities of the robot.

Flakey was able to visit only eight of the ten objects in Stage 2, and also incorrectly labelled a previously visited pole as a new pole. It should be noted that neither of these problems indicates a fault of Flakey's perceptual routines, which worked perfectly. The sonar sensors classified only one non-pole object (a box corner) as a candidate pole. This candidate pole was then eliminated from the LPS by the results of the structured light procedure. The two missed poles were not found because of Flakey's exploration routines did not account for objects that were too close to be sensed. Flakey was in the neighborhood of these objects, but never realized it. The dually labelled pole was a result of the decision to use multiple local maps rather than one global map; the pole was placed in two separate maps which were never compared.

### 6.3 Stage 3: Directed Search

CARMEL easily won Stage 3, finding all three objects and returning home in about three minutes, beating second place TJ from IBM by about 30 seconds (it should be noted that TJ competed in a half-size ring for smaller robots). Odysseus from Carnegie Mellon University finished third (Odysseus also competed in the smaller ring) and SRI finished fourth in Stage 3, visiting all three objects and returning home in about 11 minutes.

CARMEL's success in Stage 3 was a combination of its accurate global map produced in Stage 2 and its high-speed obstacle-avoidance abilities. CARMEL's simple visitation strategy worked well. Because the map was relatively accurate and because CARMEL moved directly to the objects and dead-reckoning errors did not have a chance to accumulate,

the contingency plans were invoked only once. When CARMEL approached the second object, the pole was not where CARMEL expected. CARMEL looked to the right and was able to identify the pole.

While CARMEL was able to move directly to each pole, Flakey had to move about the perimeter of the ring to register each pole to the specific wall in the appropriate patch, which was a much more time-consuming process. In fact, in the actual competition, this problem was magnified because boxes had been moved to the walls in between Stage 2 and Stage 3. Additionally, visiting the sequence of designated poles required traversal of almost half of the perimeter in each case. Because of Flakey's reliance on perimeter-following, the wide separation of poles affected the robot's travel time more so than for robots that navigated directly between poles. Flakey's local maps from Stage 2 were also very accurate, so that once Flakey located the correct patch by perimeter following, it was able to move directly to the pole. Flakey's contingency routines, to find poles that were not where expected, were never invoked.

# 7 Architectural Differences

The features described in the previous sections are summarized in Table 1. It is interesting to try to compare the two system architectures, which arose out of different research programs. In less than a month, the SRI team was able to write and debug half a dozen complex movement routines that integrated perception and action in the service of multiple simultaneous goals, building on existing research in office navigation. CARMEL's program was designed specifically for the competition, making use of the existing low-level VFH and EERUF routines. All other modules (e.g. planning, vision, and triangulation) were written specifically with the competition in mind; the team spent approximately five months on this.

In terms of overall design, it is difficult to compare the relative merit of the two architectures because the approaches to solving the problem were so different. Flakey's distributed control scheme allows various modules to run in parallel, so that (for example) self localization with respect to landmarks occurs continuously as Flakey moves towards a goal location or searches for poles. However, the distributed design leads to behavior that is more difficult to predict and debug than that of CARMEL's top-down approach, in which all the perception and goal actions are under sequential, hierarchical control.

While Michigan was the winner of the competition, it is not clear that their system can be easily extended to other domains. Certainly, the obstacle-avoidance routines are necessary in any domain and are widely applicable, but CARMEL's reliance on a global coordinate system and tagged objects restricts it to engineered environments that can be accurately surveyed. Also, CARMEL's simple exploration strategies would be naive in an environment were objects can be occluded. CARMEL's keys to victory were fast, graceful obstacle avoidance and fast, accurate vision algorithms, not cognitive smarts.

Flakey, on the other hand, while moving more slowly and possessing less accurate and more local sensing, had to rely on a smart exploration strategy and constant position cor-

| Feature | CARMEL | Flakey |
|---|---|---|
| Sensors | 24 sonars around perimeter<br><br>grayscale CCD camera | 12 sonars at front, back, sides<br>8 touch-sensitive bumpers<br>structured-light sensor |
| Software Structure | hierarchical modules | parallel behaviors |
| Moving<br>obstacle avoidance<br>roaming | <br>EERUF and VFH<br>8-point star | <br>fuzzy rules and LPS<br>composite behaviors:<br>wander, avoid-obstacles,<br>go-forward |
| Object recognition | tagged poles (bar codes)<br><br>long-range vision | type recognition of poles<br>(no tags added)<br>two-part identification:<br>sonar identifies candidates<br>structured-light sensor<br>recognizes poles |
| Mapping<br>map design<br>position correction | <br>global Cartesian map<br>three-object triangulation | <br>patches and tolerant global map<br>registration to walls |
| Planning<br>exploration<br><br>directed search | <br>6 predefined vision locations<br><br>proceed to location<br>indicated on global map | <br>traversal of perimeter and<br>forays into center of arena<br>follow walls till appropriate patch is<br>reached; make foray into patch |

Table 1: CARMEL and Flakey differ on a number of factors mentioned in previous sections of this paper. These features are listed above for comparison.

rection. One of the key research ideas behind Flakey is that natural (i.e., non-engineered) landmarks are sufficient if the right map representation is used, and it was gratifying to see this work in a new environment. Still, Flakey could be more efficient in navigating open spaces if it would incoporate more global geometric information.

The fact that CARMEL, which is sensing-rich and cognitively-poor and Flakey, which is sensing-poor and cognitively-rich, came in as the top two robots in the competition clearly shows that fundamental tradeoffs can be made in engineering mobile robots. Complex sensors can allow for simple planning; simple sensing requires complex planning. In no sense is either robot more complex than the other, it is just that the complexity lies in different places.

# 8 Similarities in Approach

Both CARMEL and Flakey must be considered unqualified successes, having bested a dozen or so other entries in a nationwide competition. Even though the two approaches had many differences, there were significant similarities between the two robots that led both of them to be successful.

## 8.1 On-board processing

Both teams did all of their processing on-board the robot, without resorting to radio links. Michigan did this by design; SRI did this after discovering their radio link would not work properly in the arena. Almost every team that used a radio or video link encountered enormous problems in interference. These teams also ran notably slower than did CARMEL and SRI due to time delays in transmitting information and the extra time required to clean up the transmitted signal. Both CARMEL and SRI showed that keeping processing close to sensing is a winning strategy.

## 8.2 Work on the basics

For any mobile robot, the basics are being able to move around in the world and not bump into anything. Both Michigan and SRI invested substantial effort into obstacle avoidance and it showed. Almost no other competitors could move as quickly or as smoothly as CARMEL and Flakey, while still avoiding both stationary and moving obstacles. A mobile robot that cannot effectively move without hitting things is not a mobile robot.

## 8.3 Simulations can help

Both Michigan and SRI developed simulations of the competition and of their robot to help in developing and debugging code. For Michigan an X-Windows planning simulator allowed for development of higher levels of code to proceed while other levels, such as obstacle avoidance and vision were still being perfected. The planning system could be tested under many different scenarios without taking up valuable time on the real robot.

SRI also had an emulator that connected to their high-level computers in place of Flakey. SRI had the capability to make "movies" of Flakey's behavior and then play them back for debugging. A movie includes all of the sensory information and motor actions that took place over a span of time, saved in a file as a sequence of packets. This file can be replayed as if it were the actual robot — similar to virtual reality. Unlike virtual reality, however, the motion of the robot during movie playback cannot be influenced; but the SRI team has found the movie technique very useful for debugging perceptual routines and actions, since they can move quickly to a point at which there is a problem, and single-step from there. Of course, in no case can a simulator replace an actual robot in an actual environment for final testing and debugging.

### 8.4 Experienced robots

Both CARMEL and Flakey are relatively old, both being around more than five years. There has been plenty of time for researchers at Michigan and SRI to develop stable libraries of communication and control routines as well as learn the ideosyncracies of their robots. It can take many years for a mobile robot to become an "everyday" research resource on which more complicated tasks can be implemented.

### 8.5 Geometric path-planning

It is surprising that neither team used any geometric planning for navigation around obstacles to a goal point, although this is a large area of robotics research [9]. Instead, they relied on the simple strategy of heading towards the goal and using reactive behavior to avoid obstacles, with very simple methods for getting out of cul-de-sac situations. Geometric planning requires some sophistication in perception and mapping of obstacles, and can be difficult to perform in real time. The large openings around obstacles in the competition made it easy to pursue simpler strategies, and we speculate that in other domains geometric planning will also play a minor role in navigation.

## 9 Conclusion

The AAAI Robot Competition allowed for direct comparison between competing techniques in accomplishing the same task in the same environment. This paper is meant to show how two winning robots approached the tasks and dealt with the issues that arose. There were many other robots with many other approaches and the experience of a head-to-head competition was invaluable. While substantial differences existed between robots, those that won had many characteristics in common.

## Acknowledgements

The authors wish to thank the other members of the CARMEL team, including Dr. Johann Borenstein, Doug Baker, Chris Conley, Roy Feague, LiQuang Feng, Rob Giles, Kevin Mangis, Alex Woolf, Annie Wu, and Cigdem Yasar. Support for the CARMEL team was provided by The University of Michigan College of Engineering, The University of Michigan Rackham School of Graduate Studies, the American Association for Artificial Intelligence, ABB Robotics Inc. and ABB Graco Robotics Inc. Marcus Huber and Frank Koss are supported by DARPA grant no. DAAE-07-92-C-R012. Support for David Kortenkamp is through Department of Energy grant no. DE-FG0286NE37969, which also provided the funding to purchase CARMEL.

Support for Kurt Konolige and Karen Myers came partially from ONR Contract No. N00014–89–C–0095. Research performed by Enrique Ruspini was supported by the U.S. Air Force Office of Scientific Research under Contract No. F49620–91–C–0060. Daniela Musto and

Alessandro Saffiotti were supported by a grant from the National Research Council of Italy. Additional support for the SRI team was provided by SRI International.

The views, opinions and/or conclusions contained in this note are those of the authors and should not be interpreted as representative of the official positions, decisions, or policies, either express or implied, of the Air Force Office of Scientific Research, the Department of the Army, or the United States Government.

## References

[1] Johann Borenstein and Yoram Koren. Histogramic in-motion mapping for mobile robot obstacle avoidance. *IEEE Journal of Robotics and Automation*, 7(4), 1991.

[2] Johann Borenstein and Yoram Koren. The Vector Field Histogram for fast obstacle-avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3), 1991.

[3] Johann Borenstein and Yoram Koren. Noise rejection for ultrasonic sensors in mobile robot applications. In *The Proceedings of the IEEE Conference on Robotics and Automation*, 1992.

[4] Rodney A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2(1), 1986.

[5] Charles Cohen and Frank Koss. A comprehensive study of three-object triangulation. In *SPIE Mobile Robots VII*, 1992.

[6] Clare Congdon, Marcus Huber, David Kortenkamp, Kurt Konolige, Karen Myers, Alessandro Saffiotti, and Enrique H. Ruspini. Carmel vs. flakey: A comparison of two winners. *AI Magazine*, 14(1), Spring, 1993.

[7] Thomas Dean and R. Peter Bonasso. 1992 AAAI robot exhibition and competition. *AI Magazine*, 14(1), Spring, 1993.

[8] Yoram Koren and Johann Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *The Proceedings of the IEEE Conference on Robotics and Automation*, 1991.

[9] J. C. Latombe, A. Lazanas, and S. Shekhar. Robot motion planning with uncertainty in control and sensing. *Artificial Intelligence*, 52(1), 1991.

[10] Allan Manz, Ramiro Liscano, and David Green. A comparison of realtime obstacle avoidance methods for mobile robots. In *Second International Symposium on Experimental Robots*, 1991.

[11] Hans P. Moravec and Alberto Elfes. High resolution maps from wide angle sonar. In *Proceedings IEEE Conference on Robotics and Automation*, 1985.

[12] Daniela Musto. Intelligent perception. Technical Report B-04, National Research Council of Italy - Institute for Information Processing, IEI, Pisa, Italy, 1993.

[13] Daniela Musto and Alessandro Saffiotti. Some notes on perception and action in situated agents. Technical Report B-09, National Research Council of Italy - Institute for Information Processing, IEI, Pisa, Italy, 1993.

[14] Alessandro Saffiotti and E. H. Ruspini. Blending reactivity and goal-directedness in a fuzzy controller. In *Proceedings IEEE International Conference on Fuzzy Systems*, 1993.