Mobile Robotics 1994

Graduate Students: O. Fuentes, J. Karlsson, W. Meira, R. Rao
T. Riopka, J. Rosca, R. Sarukkai
M. Van Wie, M. Zaki
Staff: T. Becker, R. Frank, B. Miller
Prof. C. M. Brown

The University of Rochester
Computer Science Department
Rochester, New York 14627

Technical Report 588

June 1995

Abstract

On 7 December 1994, four student-built autonomous robots demonstrated various strategic, tactical, and mechanical approaches to a delivery task. That event was preceded by approximately two years of history and two days of frenzied preparation. Our robotics efforts were based on materials from MIT's well-known 6.270 course. This report summarizes our experiences, from pedagogical goals and organizational matters through mechanical and electronic techniques. Our intended audience is future robot-builders, and organizers of robot-building courses. We assume familiarity with material in Jones and Flynn's *Mobile Robotics* text, and with the various materials available from MIT over the internet.

Keywords: Mobile Robotics, LEGO, Interactive C, MIT 6.270, Sensors.

This material is based on work generously supported by the Department of Computer Science (Tom LeBlanc, Chair) and NSF IIP grant CDA-94-01142. Wagner Meira Jr. is supported by CNPq-Brazil Grant 200.862/93-6

Contents

1 Prehistory

In 1992 – 1993, Lambert Wixson, Jonas Karlsson (Grad students in thesis mode) and Ray Frank ordered some "miniboard" kits through connections discovered on the `comp.robotics` newsgroup. Miniboards are scaled-down versions of the microcontroller boards ultimately used in the robotics work described here. Wixson and Karlsson produce a successful autonomous LEGO corridor-running robot, and the miniboard was used later in real-time applications (sensing for digital closed-loop control).

Thanks as usual to student enthusiasm and agitation, Brown volunteered in the Spring of 1994 to be the statutory organizer and titular head of a robot-building seminar to be held in the Fall semester of 1994. One constraint was that if the necessary parts could not be ordered in kit (one call does all) form, it was not deemed practical to pursue the idea. For the more hardy, parts lists are available in the very helpful documentation available at MIT in support of their 6.270 course. Luckily, Randy Sargent of MIT had gone into the business of selling these kits, and so we proceeded. Over the summer an organizing committee (Brown, Karlsson, Frank, Brad Miller and Tim Becker) dealt with the kit-ordering process, wrote a proposal for an Undergraduate version of the course, made up lists of tools and other support material deemed needed to get started (some details are in Section 2).

The organizers undertook to assemble robot during the summer and assess the difficulties, and to design a graduate-level seminar based on the robot-building and -competition conception (details in Section C).

2 Getting Started

A good place to get general information is:
`http://www.cs.cmu.edu:8001/afs/cs.cmu.edu/project/ai-repository/ai/html/faqs/ai/robotics/top.html`

Of course the newsgroup `comp.robotics` is still the standard place read and keep current, or to post specific inquiries.

The "consumables" in our work were almost entirely the electronics parts to assemble the microcontroller computer. Appendix A gives parts lists and two different alternatives for robot kits. The LEGO kits used for mechanical robot construction can be recycled. The $500/group cost comes to $167 per student. Students might decide to buy their robot or microcontroller once assembled, we might charge a lab fee of $180/student. As of now we don't have a fixed plan for how to finance this significant ongoing cost.

Aside from the above consumable items, ideally the rest of the course equipment is "permanent", meaning that students take responsibility for returning it with only reasonable wear and tear.

3 The Competition

The stars of this report are the four robots that were built during the course. The robot descriptions follow, along with algorithms, assumptions, associated team goals, successful and less successful ideas, and so on. It is interesting to note how the individual mechanical and software designs interact to produce coherent component capabilities and integrated systems.

First, however, to understand some of the design decisions, it helps to know the history of the course assignments and the rules governing the final competition. More course details are presented in Appendix C. The course had three phases: construction, robot olympics and compulsory exercises,

and the competition. To provide context for the reports on individual robots, the competition rules are presented here.

On 7 December 1994 a "robot Federal Express" competition was held with four contestants. There was an enthusiastic turnout of locals; the press was not notified until after the event. We had a re-run for the press a week later, attended by two television stations, the local paper, and a radio station. The resulting coverage was responsible and good-natured and more than likely good for business. Following are the rules and a diagram of the arena as it was set up for the competition. In fact one team (MAP-MAKER) did use a 3rd IR beacon for ego-location.

- The spirit of the competition is to investigate trades-off between the following capabilities – line following – beacon following – obstacle avoidance – memory (learning) and geometric representations – other ingenious ideas invented by contestents
- Competition takes place on the 8' by 8' white sheet, with walls, markings, and obstacles added. Walls are sturdy and 6" high, made of 1/2" or 5/8" plywood. Interior of walls painted white. There will up to a maximum of eight physical obstacles, 3-4 inches high and a foot square, at unannounced positions on a 1-foot square grid. The sides of obstacles will be painted grey. See Fig. 1.
- Robot must not be tethered to receive either power or digital information; it must be completely autonomous.
- Robot begins or arrives at a Pickup Area, marked by an IR beacon at height of 12 inches above floor. Robot requests (via its beeper) and within 3 seconds is given, in its ball-carrying hopper, a ping-pong ball.
- Obstacles and Pickup and Delivery areas are one foot square, placed on one foot grid squares. These areas will be in a fixed position on the arena, KNOWN in advance, to be fixed and painted black soon.
- Between the pickup and delivery areas will be a path marked in eletrical tape. This path is UNKNOWN in advance.
- There will be not more than eight obstacles in the arena, whose location, like the path, is variable and UNKNOWN in advance.
- Over the pickup area will be 100Hz IR beacon at the height of 12".
- Over the delivery area will be 125Hz IR beacon at the height of 12".
- A team can design and deploy more beacons:
  - prior to the competition a single IR or visible-light beacon can be set up. These beacons' location must be communicated to the organizer (CB) before the arena is set up for the competition: their position cannot depend on the location of obstacles, for instance.
  - during the competition the robot can drop beacons or other navigational aids freely.
- Both pickup and delivery areas are black areas on floor. Pickup and Delivery areas may or may not be against walls. We shall use black electrical tape to darken these areas (as soon as we determine that it will work). THUS if you are over a big black area (bigger than the tape) and you can sense an IR beacon in the direction you've been heading. there's a good chance you're in the desired area.

- Robot is to deliver the package to the delivery area: take it there and dump it. There will be separate style points and perhaps a separate award for the elegance or distance or accuracy of ejecting the pingpong ball but as long as it gets "off the truck" somehow while the robot is positioned (at least partly) over the drop-off area, full credit results.
- The robot makes its way back to pickup area and repeats (only we hope with improved performance).
- There will be a tape path, guaranteed unobstructed, between the pickup and delivery areas. It is guaranteed not to be the shortest path. It will have no radius of curvature less than one foot. We'll try to be nice and not put minimum clearances near maxima of curvature. We guarantee 6" of clearance on either side of the tape.
- We guarantee at least a 12"-wide path between any separated obstacles.
- There will be partial scores for partial performance, to be determined. But the main score is how many packages can be delivered in four minutes. If robot functions perfectly, four minutes should be enough to show it off. If it malfunctions, four minutes is an eternity.
- A human may intervene in the contest to rotate (not translate) or to repair (replace bits that have fallen off) the robot. The clock runs during interventions and each separate intervention costs one ping-pong ball. Any number of repairs, but only one rotation, are allowed per intervention. The intent here is to minimize robot and team and audience frustration caused by simple mechanical or logical failures.
- In the spirit of minimally changing the rules, if robots A and B have each delivered N balls at the end of four minutes, then that robot farthest along its way to delivering N+1 balls is the winner. This criterion should apply pairwise to all robots, and should order the set. The judge's (or judging panel's) opinion will be final. Videotape replays may be needed but we have the technology.

4 Map-Maker (Sarukkai, Meira, Zaki)

With this section we commence descriptions of individual robots.

Map-maker is an ambitious robot with complex internal representations and sensing capabilities. The MAP-Making robot is uniquely distinguished by the ability to plan paths, determine present location, and accomodate sensor errors during position determination. Given a grid model of the world, positional information of the robot is sufficient to build the world representation. For instance, if an obstacle is detected by the robots front switches, then the position determining routine is executed, followed by marking the appropriate location of the world as an obstacle.

4.1 Map-maker: General description

We can divide our robot into the following functional parts:

Traction: the traction includes the two back wheels and the gear train (including one motor) associated with them. We used the "differential" provided with the lego set, distributing the power between the wheels.

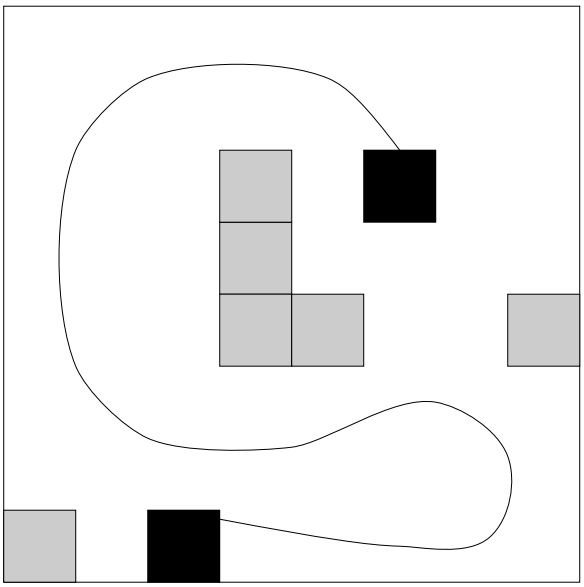


Figure 1: The eight-foot-square arena. Black line marks black tape path. IR beacons of different frequencies hang 1 foot over start and finish squares and also over the south-east corner. Shaded squares show obstacles.

Steering: Our steering is provided by one wheel that is controlled by a motorized gear train. There are two lateral sensors that prevent excessive steering. To determine if the robot is going straight, we used an optical solution with photodiodes: we attached the emitter to the robot body and the receiver to the steering wheel in such a way that, when they are aligned (i.e. the receiver is receiving the emitter's signal), the wheel is in a straight direction.

Bumper system: We have four bumpers, two frontal and one in each side. There is a digital switch associated with each of them.

Localization System: The mechanics involved in the localization system consists of the servo motor that drives three IR receivers and is mounted on a tower.

Ball holder: The ball holder consists of lego-built box with a motor included. The motor has a stick attached to its axle. When the motor is activated for 0.5 seconds, the stick throws out the ball.

4.2 Beacons and Triangulation

In order to detect our exact position in the board, we needed three referencial points. It is known that the IR receiving task is CPU intensive, so in order to take advantage of the special sensor electronics of the board, we decided to use only the two standard frequencies; thus two of the three transmitters were constrained to use the same frequency.

This caused two problems:

Bimodality detection: Detecting two emitters of the same frequency causes two peaks at the end of the scanning information. Sorting them out and dealing with them required context-dependent information such as knowledge about the minimum angle between beacons (see later).

Proximity effect: if we are really near one beacon, usually we are not able to detect the other, not only because of the intensity of the nearest that is bigger, but also because the minimum angle is not effective in this case.

4.3 Localization Tower Design

One of the problems we realized when using the servo motor is that it does not cover the nominal range of 180^o. We guess that it does not reach the final 5^o in each side. In order to be able to perform the localization, we attached three IR receivers with the same angular distance between consecutive ones (120^o).

We had various problems with interference due to reflections of the IR signals transmited from the reference points. To avoid such problems, we used aluminum foil around the receiver "collector". Another detail is that the "collector" is taller than it is wide, being able to detect transmitters sited at various heights without horizontal accuracy loss.

4.4 Photoresistor Mounting

The photoresistors were used to implement line following. Each of them was mounted in a black paper cylinder. The purpose of this tube is to avoid external light interference. These tubes are mounted in the front wheel structure, with their entrance standing 2mm over the floor.

The calibration was done manually, checking the readings for black and white areas. Despite this, we found that light variations (like a person's shadow) are enough alter the thresholds that distinguish the regions.

4.5 Beacon Tracking

We used our servo motor to perform beacon tracking. We attached three IR receivers to the servo motor in order to cover all the 360^o around the robot. We detect the position of a beacon by positioning the servo motor in different angles and checking the IR receivers.

The scanning procedure has two steps: coarse scan and fine scan.

In the coarse scan we check angles spaced by 5 degrees to detect the regions where the beacons could be. After this procedure we perform the fine scan, where we check a 10 degrees angle with increments of 1 degree.

4.6 Line Following

Our line following strategy is simple but efficient.

We attached two photoresistors to our front wheel in such way that if we are following the line, the photoresistors will sense the black region. If one of them loses the line then we know that we are not only going out of the desired direction but also to which side of the line, allowing us to take the correct recovery action.

4.7 Steering

We used the motors supplied with the kits to implement our steering mechanism. These are high speed motors, and we needed a gear train with a reduction factor of about 3600:1 that is attached to the front wheel.

To avoid damage to the steering and robot, we put two touch sensors in the front of the robot that deactivate the steering motor when it tries to turn more than allowed.

The main problem with this approach is the inaccuracy of the turns. Gor reactive tasks like line following it works satisfactorily, but things become really hard when we talk about spacially exact actions like the ones required in a mapped environment. It is hard to measure progress by timing the activation/deactivation of motors since their performance is influenced by the (variable) state of battery charge.

Another problem is that the gear train is not accurate enough to avoid some slippage when the motor direction is reversed. Another precision - related problem is the speed: reducing it to allow adequate control had the effect of slowing down steering appreciably.

4.8 Map-Maker: Gearing

Map-maker has two gear trains, one for steering and other for traction. Both use the same motor and have reductions of 3600:1 and 243:1 respectively. The gear train building should be done carefully, not only to avoid friction in the axles (that can cause the train to block), but also to save space in the robot.

4.9 Map-Maker: Lego Experience

The main lesson from our lego adventures is that we should be conservative. We tried to build some structures and had a lot of problems:

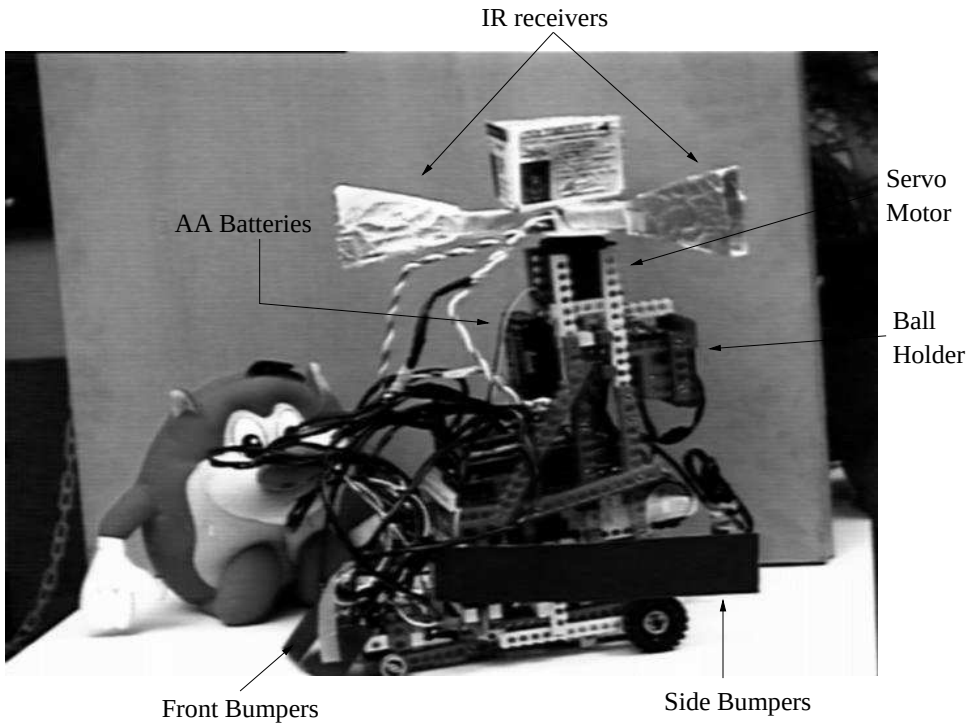


Figure 2: The MAP-Maker

Steering: one of the big problems with our steering mechanism is gear slippage. Due to the fact that the plastic gears have sloppy tolerances, high precision constructions like the steering have low accuracy.

Skirt: We designed a bumper system that uses a skirt around the robot. The main feature of the skirt is free movement in the horizontal plane. Properly attached to four touch sensors (front, left, right and back), it allows easy handling of all possible bumping situations.

Grasp and Lift: We tried to devise a pretty complicated "grasp and lift" device for the robot. The idea is using only one motor, perform two tasks. The motor power is shared via a differential. We start grasping, when this operation is complete, the differential transmits the power to the lift mechanism. The reverse operation is similar, as the force to unlift is bigger than the one to ungrasp, we first put the device down and then ungrasp. The problem in this case was gear slippage also; they are not strong enough to hold while the differential is transferring power to the other task.

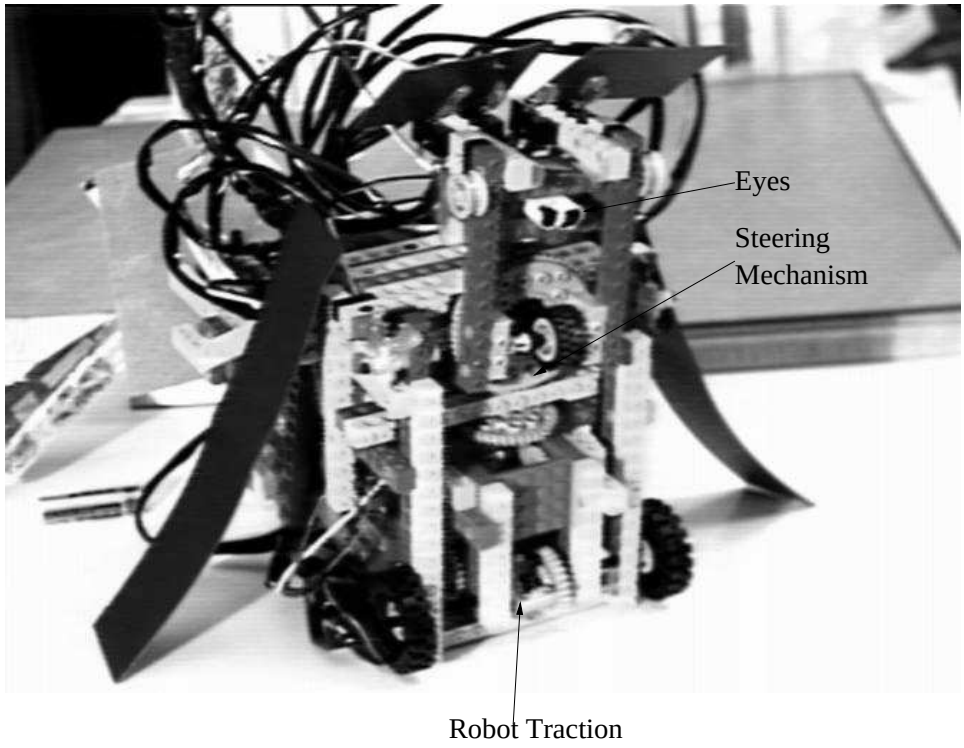


Figure 3: The MAP-Maker

Algorithm MAIN
1. Start at Starting point START.
2. Determine ORIENTATION.
3. Take Commands returned by function FindActions();
4. If Obstacle,
then SCAN to find present LOCATION.
Find ORIENTATION.
Mark MAP with OBSTACLE.
Recompute PATH to goal.
5. goto step 3 until GOAL REACHED.
6. Throw BALL.

The main algorithm is pretty straightforward. The starting point for the task is fixed. Our position determining mechanism is based on the angular locations of three IR transmitters that had to be placed in an appropriate geometric configuration, so as to enable unique position detection, and is detailed later. Based on the "quadrant locations" of the IR transmitters, the head-on direction or *orientation* of the robot with respect to the three IR transmitters, and thus the board can be determined robustly.

The action sequence is constrained to one of the following: STRAIGHT, RIGHT, LEFT, BACK. The actual path planning algorithm (shown in a later section) is basically a dynamic programming search from the goal location to the start location of the map *using the available action sequences*. In our preliminary experiments, the map grid locations were filled with distance (to the goal) values, and actions required to go from the START position to the GOAL position were then determined. However, this need not be the best possible *action* sequence: an alternate would be to use a dynamic programming approach and accumulating the penalties of each *reverse action* from the GOAL state to the START state. During a final run, due to the very low 16K (including system stack) onboard memory, the actual exhaustive dynamic programming search had to be abandoned in place of a local greedy action selection mechanism. Thus, the procedure was to determine the present location, find path and take the appropriate action. If an unexpected event such as the detection of an obstacle occurs, mark appropriate cell in the map as an obstacle, and recompute an alternative path.

The idea behind fast, robust determination of the location of the robot was twofold: use a table-lookup (angular information from the real world worked better than geometrically computed values for some "ill-posed" locations) to match sensor angles with stored angles to get the *k nearest neighbours* (locations); the important second aspect was to cross-validate the *K-NN* locations with the *neighbourhood of the expected location*. At locations more than a square away from the IR transmitters, the cross-validation scheme enabled recovery from positional errors accumulated due to friction, wheel slippage etc. Finding the angles accurately could be expensive; however, we reduced the cost of scanning by doing two passes. The first pass is a coarse scan, and the second pass is of fine resolution in a 10 degree range of the maxima detected during the first pass.

4.10 Map Algorithms

Algorithm FindPath
/** The map is represented by a 8 x 8 table. ***/
1. Clear MAP at all points, leaving the obstacle locations unchanged.
2. Mark START and END locations.
3. Starting at the END location, apply recursively the following step:
- Depth at END =0;
- Let current location be x, y, and depth of recursion=DEPTH.
- Let x', y' be a neighbor of x, y.
- MAP[x', y']= min(MAP[x', y'], DEPTH+1);

4.11 Actions algorithm

Algorithm FindActions
1. Find (using the 3 IR beacons) the initial ORIENTATION of the robot.
2. Assign ORIENTATION.
3. X= STARTx ; Y = STARTy;
4. For each of the following moves determine the value entered in the MAP, and choose the action wth the minimum value.
LEFT : RIGHT : BACK : STRAIGHT : STRAIGHT2MOVES
5. Assign X=ActionX; Y=ActionY; ORIENTATION=ActionORIENTATION;
6. If not reached END then goto step 4.

```
EXAMPLE


Initial configuration of MAP
==========
|........|
|....*...|  S= START
|.*S.*...|
|........|  E= END
|........|
|....*...|  *= Obstacle
|.*......|
|..*..E..|
==========

Filling the MAP with path information by using  dynamic programming

Algorithm  FindPath applied.
Note that the path is given by following the sequence abc...S in reverse.

==========
|lkjihghi|
|kjih*fgh|  a= 1 step away from END. b= 2steps and so on...
|j*Sg*efg|
|ihgfedef|
|hgfedcde|
|gfed*bcd|
|h*dcbabc|
|ij*baEab|
==========

Finding Action Sequences after applying Algorithm FindActions.
Initial ORIENTATION is to the RIGHT ---->

Action Sequence generated:
RIGHT STRAIGHT2 STRAIGHT LEFT STRAIGHT

Position of Robot after each ACTION:

1. INITIAL
```

```
==========
|........|
|....*...|
|.*@.*...|
|........|  @ position of  ROBOT
|........|  ORIENTATION Facing Right
|....*...|
|.*......|
|..*..E..|
==========
```

2. RIGHT

```
==========
|........|
|....*...|
|.*S.*...|
|...@....|   ORIENTATION Facing down
|........|
|....*...|
|.*......|
|..*..E..|
==========
```

3. STRAIGHT2

```
==========
|........|
|....*...|
|.*S.*...|
|........|   ORIENTATION Facing down
|........|
|...@*...|
|.*......|
|..*..E..|
==========
```

4. STRAIGHT

```
==========
|........|
|....*...|
|.*S.*...|
|........|   ORIENTATION Facing down
|........|
|....*...|
|.*.@....|
|..*..E..|
==========
```

5. LEFT

```
==========
|........|
|....*...|
```

```
|.*S.*...|
|........|   ORIENTATION Facing Right
|........|
|....*...|
|.*......|
|..*.@E..|
==========

6. STRAIGHT
==========
|........|
|....*...|
|.*S.*...|
|........|   ORIENTATION Facing Right
|........|
|....*...|
|.*......|
|..*..@..|
==========

REACHED GOAL
```

4.12 Position Finding

We use three infra-red transmitters to determine the robot position anywhere on the grid uniquely.

Refer to figure 4. We use the servo motor to scan and get the three angles A, B and C. We next find the centers of the three circles passing through the points : ROBOT, GOAL, THIRD IR; ROBOT, START, THIRD-IR; and ROBOT, START, GOAL. These centers are denoted as P, Q, and S respectively. Let Rx, Ry denote the x and y co-ordinates of the Robot. Let a similar notation denote the co-ordinates of the three centers. We can now find the Robot position by the following set of equations:-

$$(Rx - Px)^2 + (Ry - Py)^2 = Pr^2 \quad (1)$$

$$(Rx - Qx)^2 + (Ry - Qy)^2 = Qr^2 \quad (2)$$

$$(Rx - Sx)^2 + (Ry - Sy)^2 = Sr^2 \quad (3)$$

Now from equation 1 and 2 we get the following:-

$$2Rx(Qx - Px) + 2Ry(Qy - Py) = (Pr^2 - Qr^2) + (Qx^2 - Px^2) + (Qy^2 - Py^2) \quad (4)$$

And similarly from equation 2 and 3 we have the following:-

$$2Rx(Sx - Qx) + 2Ry(Sy - Qy) = (Qr^2 - Sr^2) + (Sx^2 - Qx^2) + (Sy^2 - Qy^2) \quad (5)$$

From these two equations we can easily compute the two unknowns, namely Rx and Ry. However there is still the matter of computing the radius and the centers co-ordinates of the three circles. We'll show how to do this for the circle passing through ROBOT, START and THIRD-IR, i.e., the one centered at Q.

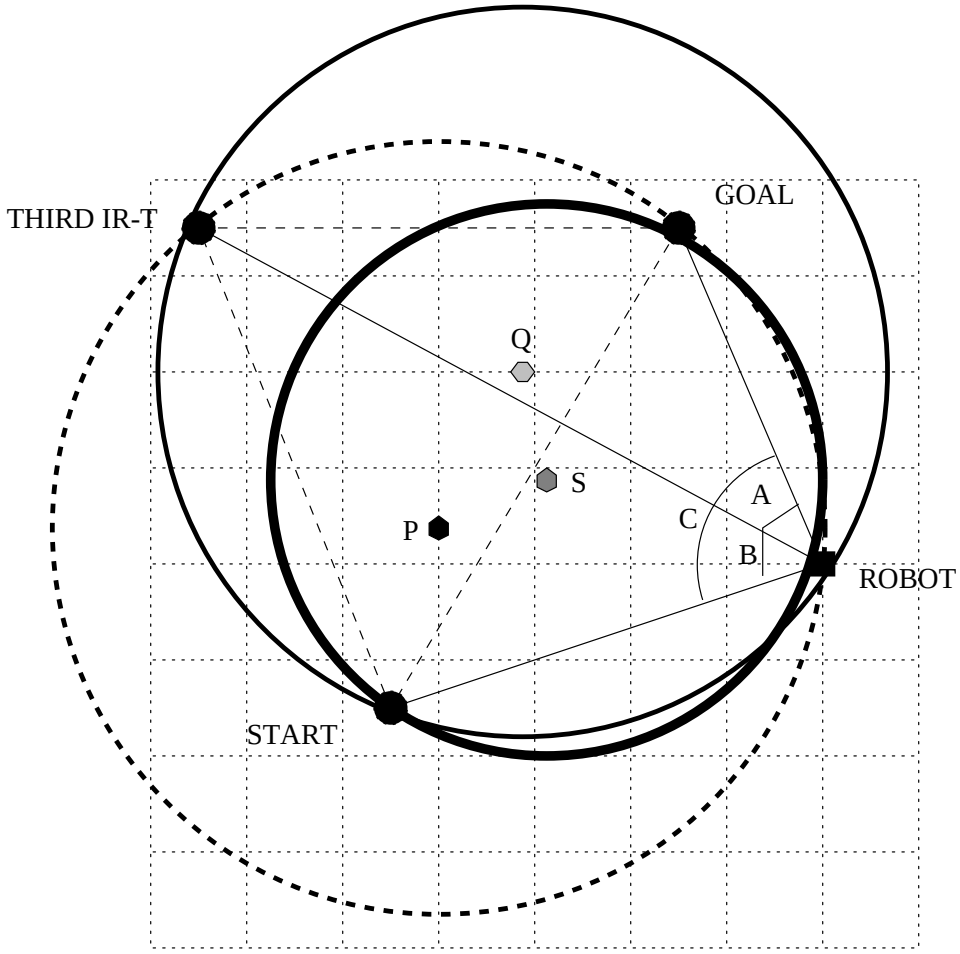


Figure 4: Position Finding

There are two cases to be dealt with here. In the first case the angle between the THIRD IR, ROBOT and START, namely the angle B, is less then 90 degrees, which is shown in figure 5. Recall the fact that the angle made by the chord THIRD IR—START at the center of the circle is twice B. Therefore, when we drop a perpendicular from the center onto the chord we get two right triangles with B as one of the angles. We already know the length of the chord, lets call it AB, and we know the angle D. We first find the radius od the circle, Qr, as follows:-

$$Qr = AB/2sin(B) \tag{6}$$

Now that we know Qr, and know the angle D+90-B, we know the hypotenuse and one acute angle of the right triangle with thick lines. We can easily find Qx and Qy relative to the co-ordinates of the THIRD IR, say Tx and Ty. We can then find the real co-ordinates of the center by adding or subtracting Tx and Ty appropriately. The following equations achieve that:

$$Qx = Qr * Sin(D + 90 - B) \tag{7}$$

$$Qy = \sqrt{Qr^2 - Qx^2} \tag{8}$$

The second case is shown in figure 6, where the angle B is greater than 90 degrees. In this case we use the angle 180-B which must be less then 90 degrees. Then the angle made by the chord

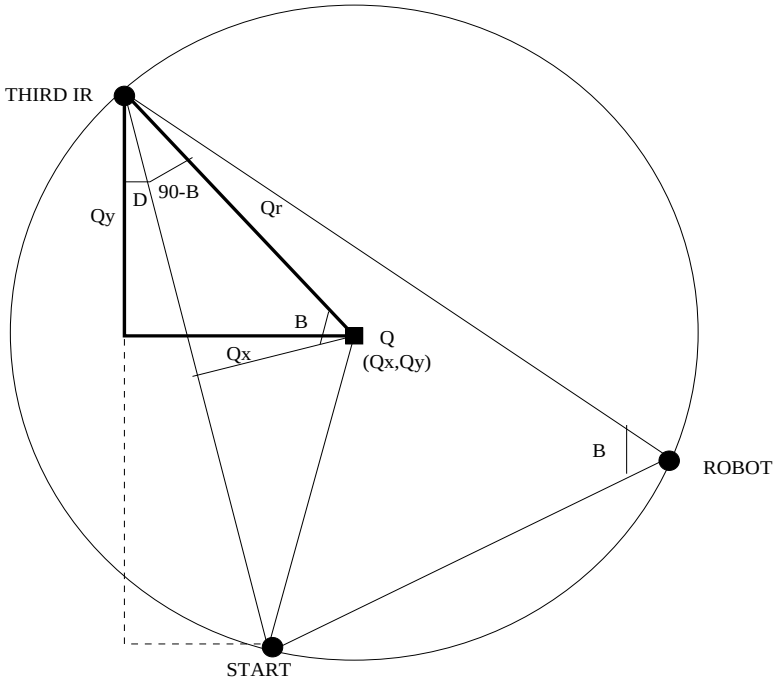


Figure 5: Case I :B < 90.0

at the center of the circle is twice 180-B, and dropping the perpendicular from the center onto the chord we get a right triangle with 180-B as one of the angles. From this information we can easily get the angle B-90-D of the thick right triangle. The radius Qr is calculated as before and we can easily obtain Qx and Qy as follows:

$$Qx = Qr * Sin(B - 90 - D) \quad (9)$$

$$Qy = \sqrt{Qr^2 - Qx^2} \quad (10)$$

In the beginning we were planning to do this at run time, but we soon ran into memory problems. We then decided to make a table of positions along with the angles at that position. We decided to keep a minimal sized tables with 64 entries corresponding to the positions of the centers of the squares in the 8x8 grid. To find our position the robot would scan the grid and obtain the three angles. We could then simply search through the table and output the closest match. However, there was one additional problem with this approach. The IR transmitters at the GOAL and THIRD IR-T both transmitted at 100Hz, while the START was at 125Hz. When the robot is close to the GOAL, and especially in the top right squares, it becomes difficult to distinguish the two transmitters, since the GOAL IR obscures the THIRD IR-T. We got around this problem by measuring the angles returned by the servo at the trouble spots, and storing these values in the table instead of the actual values we should have obtained.

This difficulty could be solved if the THIRD IR-T were at a different frequency. But we didn't have time to hack the assembly code to achieve this. A good feature of our approach is that due to the palcement of the transmitters we could uniqely identify which region we were in, and that helped us in calculating the exact position of the robot. There are four regions.

Having found the robot position correctly, we can also find the robot orientation. So far we had been using the angle differences, i.e., we actually measure the absolute angles to the three transmitters, but our position finding algorithm uses the differences between these. We can get our

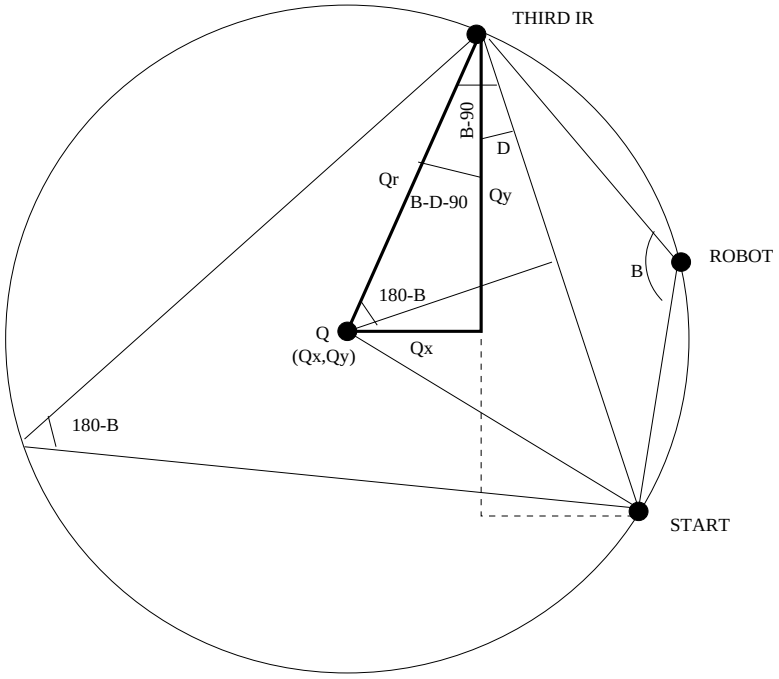


Figure 6: Case II :B > 90.0

orientation by taking the absolute angle we get, and comparing it to a fixed reference point. For example we can say that facing the THIRD-IR and GOAL line means that we are North. So if the absolute angle differs by 45 degrees then we can say that we are facing north-west.

5 Line-Follower (Miller, Becker)

Line-Follower pursued the strategy of following the black line on the floor from start to finish, dropping the ball, doing an open-loop 180 degree turn, and returning to the start. Five reflectance sensors across the front gave the control a startling robustness, and the final robot was very small and compact. It used a gear-head drive motor and thus was very quiet. It was rather slow, but its no-nonsense attitude let it deliver three balls in the alloted four minutes. This robot's design is an example of retrograde evolution, starting from a highly original design that involved new sensors and the goal of measuring absolute 2-D location.

One of the ways to solve the problem in maze-walking was to develop an absolute positioning system. Unfortunately, monies not being available for a geostationary sattelite, or even for a gyroscope (inertial guidance), and having no expertise in vision, Miller leveraged his prior work on IR communication for the minibot with some knowledge of history. (This team had given some thought to how two robots might use the IR detectors to communicate information between themselves).

At one point in history, navigation was done by the stars, specifically, the angle of the north star was used to calculate the latitude, while a compass and the position of the star with respect to the time of day was used to fix the longitude. Unfortunately, electronic compasses being off budget expenditures, Miller investigated the notion of using a pair of stars for calculating absolute position, with the possibility of using dead reckoning and tracking of a single star to update it as the robot moved.

To do this, the team built a primitive astrolabe out of lego. A servo moter was used to adjust the azimuth which would give the distance to the star. A normal motor and break-beam sensor was used, suitably geared, to adjust the angle of the astrolabe (henceforth referred to as θ) to the mounting. Two Sharp IR sensors were used, each with a different length brass tube attached. These were used to limit the angle of sensitivity of the sensors to the emitters (stars). Each star was modulated at a different frequency, and primitive PLLs (new sensor hardware developed by Miller and Ray Frank for this project) were used to register the frequency being detected by the wide-angle sensor.

The basic idea was that with the robot stationary, the astrolabe would be swept until the wide angle sensor found the appropriate signal to be in view (thresholded to eliminate reflections), and then the narrow-angle sensor would hillclimb until the star position was fixed. From the azimuth we then knew the distance to the beacon, as well as the angle to the robot. Now we fix on the other star, and simple geometry gives us an absolute position, given the fixed nature of the stars (one was the starting beacon, the other was a separate beacon we erected at a sufficient distance off the floor to give reliable azimuth information, as well as reduce the possibility that both might be seen by the 'bot at the same θ in a portion of the maze).

Once the position was fixed, the robot could then use sensors on its motors to judge the delta in a particular angle, and errors tracking the star could be correlated into a guess of a new position. From this we hoped (with our knowledge of the basic map layout), we could map the maze.

While preliminary results were encouraging, the following problems ensued: The line following strategy of the robot was very robust, but caused a large amount of angle change to occur as it tracked the line, far greater than the slew rate of the θ moter on the astrolabe. To increase the slew rate, however, would drastically decrease its accuracy (currently to 1 degree, but another 2 degree error from the narrow aperture sensor meant we were pretty much at the limits of sensitivity when far from the star). To compensate for this we should have to refix our position on the two stars more frequently, and this was seen as slowing down line following enough that unless the maze was particularly tilted against simple obstacle avoidance strategies (unfortunately, it wasn't), the cost of absolute positioning swamped performance. Additionally, it was unclear the software needed to reason about position while tracking the star, in addition to the line following code and basic beacon and astrolabe code would all fit into the limited memory of the robot.

Additionally, the feedback from the motors needed the two ports for break-beam sensors, yet, the astrolabe itself used one. The board we were using had no provision for counting a third break-beam, so dead reckoning would have been difficult without additional sensors.

There are a number of possible solutions to this dilemma given funds:

1. Fix the astrolabe position, put the star on the robot, and relay information (using the IR transmission scheme perhaps) between the fixed astrolabe and the robot. This would have avoided slew rate and tracking problems, but would have violated tournament rules since the robot would not have been autonomous, and we would have needed an additional cpu board.

2. Add a compass to the robot. This would have meant only one star would have been needed to fix position. This would not have solved the problem of dead reckoning, we still would need a third break-beam port.

3. Use inertial guidance for dead reckoning. Presumably this can be done without a break-beam or equivalent. Since we don't know initial θ, but do know the angle of the start position to our star, we could dispense with the compass (since the inertial system should give us any $\delta\theta$ as well as offset from our starting point). An accurate enough system might be able to dispense with the external reference point as well, e.g. that used for cruise missiles.

4. Use a different approach, e.g. embedding wires into the arena and checking flux changes as the robot passes over them. Modulating each wire with different current, or amplitude changes, means we can either read absolute position more or less directly, or at least can distinguish horizontal from vertical grid lines and count as in a very accurate dead reckoning system.

6 Cyclops (Riopka, Rosca)

Cyclops is a LEGO mobile mini-robot that is capable of exploring a planar world by combining several high-level behaviors such as line following, beacon following, horizon scanning, obstacle detection and avoidance, wall following and map construction. Cyclops can be taught a set of micro-skills which can be used in writing its higher-level behaviors. Cyclops has the capability of building a map of a grid-world. Such a map can be used as a starting basis in implementing a simple task planner for task optimization.

The high level design decisions that guided the implementation of Cyclops were:

1. Support a software architecture that can implement well the tradeoff between reactive and reasoning capabilities
2. Learn a map of the competition environment by creating, updating and using the necessary information opportunistically
3. Keep track of the vehicle orientation or reinitialize it and be able to realize if the orientation is lost.

We intensively experimented with various design decisions influencing the first major goal above. The second and third goals are inherently very difficult to implement. Although we made advances in this direction, any map construction facilities were infeasible given the specific delivery task and the amount of time allowed to compete. However, we experimented and equipped our robot with various sensors that support the creation and interpretation of a higher level representation of the environment. We outline here that competition rules, and most importantly the given world configuration and the time allowed for each robot may rule out capabilities that make a robot more intelligent. This would be the case with a very simple world which implicitly favors very simple architectures. Our robot has been designed to cope with a more complex world.

6.1 Cyclops functionality

Cyclops is equipped with a variety of sensors and actuators. Each of them will be shortly described at two abstraction levels. First we refer to the devices used. Secondly we mention the desired functionality which is reflected in the software.

The desired functionality influences every aspect of the device: the mechanical, and electrical component choices, the sensor or actuator mechanical assembly, the way sensors are interfaced to a microprocessor board and the driver routines used.

We start our description with the desired functionality and the goals pursued. Then we describe the sensors and actuators used and the way they are interfaced to the controller board.

Given the objectives mentioned before, we defined a set of necessary functions our design has to support. Most of these impose constraints on the sensors and actuators that will endow our robot. The discussion on sensors and actuators will follow these more detailed goals.

- Drive forwards and backwards.

- Perform turns with a very low steering radius.
- Control the steering angle with a very high precision.
- Estimate the distance traversed by the vehicle regardless of speed or steering, in any given time interval.
- Detect obstacles bumped into when driving forwards or backwards. Detect if an obstacle is positioned to the left, right, front, or rear of the vehicle.
- Detect obstacles touched by the sides of the vehicle
- Detect obstacles that appear at a given distance in front of the vehicle before bumping into them.
- Detect a nearby infrared beacon emitting a modulated IR signal of a given frequency.
- Detect side obstacles without touching them.
- Follow a black winding stripe on a lightly colored floor (e.g white floor) regardless of illumination conditions.

6.2 Cyclops sensors

Cyclops is equipped with the following sensors:

1. Four bumping sensors. All are simple microswitches attached to the vehicle in appropriate mountings.
   (a) Two front sensors. They are mounted on a common bumper. They can be switched on either independently or together if the vehicle bumps frontally into an obstacle.
   (b) Two rear sensors made of roller microswitches. They are mounted at the rear left and right corners of the vehicle.
2. Two touch sensors or side bumpers made of ALPS switches. They are mounted on the left and right sides respectively, close to the front corners of the vehicle, and carry side wings of the length of the vehicle, so that whenever a side of the vehicle is pressed against an obstacle or a wall, at any angle, the sensor becomes switched on.
3. IR beacon detector. It is made out of four Sharp detector packages GP1U52X which can detect IR light of 100 or 125 MHz and are mounted at 0, 60, 180 and 300 degrees. 0 degrees corresponds to the direction of the vehicle axis and a forward orientation. We have experimented, with equally good results, with a simpler design in which only two IR detectors are used. They are mounted at ± 15 degrees.
4. Two IR side reflectance sensors. Each is a Sharp detector package similar to the ones used in the IR beacon detector.
5. Four IR reflectance sensors. They are made out of pairs of an IR LED and an ultrasensitive IR phototransistor (Sec. B.1).
6. One breakbeam sensor. It is mounted above a LEGO pulley wheel attached to the direction axle carrying the front wheels of the vehicle.

7. One photoresistor. It is mounted in a black-covered drinking straw tilted forwards in order to offer an analog signal on white/grey/black objects a given distance away in front of the vehicle. The photoresistor is part of an *artificial eye* system used for obstacle avoidance. The entire system will be described in more detail below.

Interfacing sensors to the microprocessor board

1. Bumping sensors. The left front and rear microswitches are mounted in parallel and connected to an analog port. However they are used as digital devices (on/off). The one corresponding to the direction of movement (front for forward movement and rear for backward movement respectively) offers information about obstacles the vehicle has bumped into.
2. Touch sensors are also connected to an analog port, although they have a digital function.
3. IR beacon detector. Each detector package is connected to a high digital port. The Interactive C software only supports ports 4-7 for reading the modulated IR light signal.
4. IR side reflectance sensors. Each is connected to a high digital port.
5. IR reflectance sensors. Each is connected to an analog port.
6. Breakbeam sensor. It is connected to a low digital port. This makes possible to use the fast, assembly language shaft-encoder routines from Interactive C. Although this alternative offers more accurate information in case high sampling speeds are needed (determined by high rotation rates of the pulley wheel), our robot did not travel at high speeds.
7. The photoresistor is connected to an analog port. It offers an analog signal that is decoded into information on the grey level of the point viewed.

6.3 The Cyclops Artificial Eye System

Linear Scanner

A line scan "camera" was constructed using a single photoresistor mounted on a servo. A single photoresistor was used in order to simplify calibration and photosensitivity problems that might arise using multiple detectors. Early measurements showed that the sensitivity of any one photoresistor not only varied considerably from one to the next, but was also non-linear over its range. Using only one photoresistor alleviated many of these potential difficulties.

The detector for the linear scanner consisted of a photoresistor mounted in one end of a 10cm long straw. The straw was covered with electrical tape with a black "cone" on the end opposite the photoresistor to insure no reflections from the straw opening. The tube was then encased in a lego structure which was centered on an axle whose axis was perpendicular to the tube but parallel (and connected) to the shaft of the servo.

The entire scanner was tilted at an angle of approximately 29 degrees to the horizontal resulting in a "view" approximately 38 cm or 1 1/4 feet in front of the vehicle. A fixed angle provided a way for the robot to measure object distance and also simplified the vision algorithm (see below). In the final implementation, 16 sample points were used, each point separated from the next by an angle of 5 degrees resulting in a scan angle of 75 degrees centered about the robot and looking forward. Experiments with a visual test display indicated that the field of view (i.e. the angle subtended by the area from which light was detected by the photoresistor through the tube) was approximately 7 degrees. This provided significant directionality and sufficient resolution to enable accurate enough edge detection for object avoidance. The location of the object edge was used as a parameter to

the general "skill" learned by the robot for object avoidance (see Skill Learning). Scan time was approximately 8 seconds per line acquisition requiring 200ms rise time for the photoresistor, 200 ms per sample to reduce scanner vibration, and 1-2 seconds for servo-switching. This was considered to be a reasonable time, but one which did not encourage indiscriminate scanner use.

Scanner Calibration and Thresholds

One of the main advantages of using a single photoresistor for line acquisition was the very simple calibration procedure. To calibrate the line scanner, two readings were required: one with the photoresistor tube aimed at an area that corresponded to the darkest scene possible under ambient light and one with the tube aimed at the lightest scene possible. The difference between these two readings is the dynamic range of the photoresistor. Because the analog value was small for "light" scenes and large for "black" scenes, the following formula was used to obtain a "graylevel" value: $graylevel = 255(1 - \frac{measuredvalue}{dynamicrange})$.

Graylevel measurements after calibration were very reasonable. The single photoresistor was calibrated on the objects for "black" and on the floor for "white". This calibration resulted in a dynamic range (as measured by the analog output of the photoresistor) of up to 60 levels (or 1.2 volts assuming a full 5 volts for logic) in the ambient light of the playing field. This enabled the scanner to tell reliably the difference between an object, the floor and even the walls of the playing field, throughout the playing field. However, shadows cast by the objects themselves resulted in similar readings for both the walls of the playing field and the floor (in certain shadowed areas). A similar observation was made in regards to the colors on top of the objects. That is, although the scanner could theoretically tell the difference between the floor, walls and even colors on top of the objects, shadows prevented the robot from taking full advantage of this potential capability.

After several experiments, it was determined that a proper gradient threshold (or more accurately, the difference in gradient value required between two points such that an object/floor transition can be considered to be very likely) could be obtained by subtracting 10 from the dynamic range. An automated vision calibration could have been developed by using the loading area for black and the floor for white. Since the starting position of the robot was known, the entire procedure could have been automated. In our implementation, the calibration constants were determined ahead of time and hard-coded into the application. (Note that this did result in problems when the television crew lights were on to film the arena, but in general, under ambient light conditions, the calibration was fairly stable.)

The Vision Algorithm

The vision "behavior" was implemented as follows. During regular robot movement, the scanner is active (i.e. the photoresistor output is monitored) and the scanner is positioned directly forward. The angle for the scanner is such that a black object is detected (i.e. light detected falls below a specific threshold) at an approximate distance of about 1 "grid" square (or about 30 cm) in front of the robot. The threshold for this initial object detection was selected empirically in such a way as to ensure that a horizontal scan line at that point intersects the approximate center of the object. Object detection causes the robot to stop and initiate a single line scan as described above. The robot then uses the edge information from the line scan to parameterize an object avoidance manoever implemented as a skill (described below).

In all the scenarios, the general strategy was to determine where the "floor" is and to move the robot in that direction, avoiding object edges. Because of the constraints on object location, shape and size, only three image scenarios are possible:

1. No floor/object transitions - only objects in front. This suggests that the robot is oriented almost perpendicular to a set of at least two and probably three adjacent objects. The reaction in this case is to initiate either a "left turn" or "right turn" skill depending on higher level information.

2. One object transition - two cases are possible, object to the left, floor to the right OR floor to the left, object to the right. In either case the robot is again almost perpendicular to an object wall. The location of the single edge is used to determine whether an "avoid object - go left" skill or an "avoid object - go right" skill is initiated. In either case, the exact location of the edge is used to parameterize the skill by varying how much the robot needs to shift its position to the left or to the right to avoid the object.

3. Two object transitions - two cases are possible, floor on either side of an object OR floor between two objects. In these two cases, the orientation of the robot is no longer clear (although due to the constraints on the robot world, it could probably be calculated). The goal of the image analysis here was to determine where a large enough floor area for the robot was in relation to the robot, and to determine which of the "avoid object - go left" or "avoid object - go right" skills was appropriate and lastly, parameterize the appropriate skill using the relevant object edge.

Image Analysis

One dimensional zero crossings were used to localize the position of the edges to eliminate edge ambiguities and to simplify the vision algorithm. Although a simple thresholding approach with an ad hoc search algorithm could have been implemented, the zero-crossing approach seemed more elegant algorithmically and resulted in a straightforward algorithm implementation. The algorithm proceeds as follows. First, first derivative (gradient) and second derivative images temporary images are formed from "padded" versions of the original input. Zero crossings in the second derivative image are used to locate possible edges. Gradient values at these locations are used to determine the validity of the edge transition using an empirically determined gradient threshold. A thresholded version of the original input as well as a holding array for the object edge locations are then generated and used by the knowledge source associated with robot avoidance strategies to implement the appropriate robot action (as described above).

Vision as a Behavior

The general idea behind the vision behavior was to treat it as a "reactive behavior" on an analogous level with lower level reactive behaviors but with the following qualifications. Because of the method used to trigger the vision capability, i.e. real-time monitoring of the center pixel for object detection, the time required for scanning can be considered to be irrelevant, as long as all other behaviors are temporarily "suspended" during line acquisition and analysis. (The vision system could easily be replaced by a real time system capable of visual analysis in a time frame compatible with the time frame of other reactive behaviors.) The execution of an "avoid object" skill can be viewed in two ways. First, one could argue that the *proper* way to perform a reactive behavior to avoid an object would be to "visually servo" the object edge as the robot moves around it, similar to the strategy used by truck drivers manoevering their vehicle in reverse. On the other hand, truck drivers are also capable of performing rather complicated turns without visually servoing, by noting the location of an obstacle and relying on experience to enable them to manoever around it. And, analogously, just as the poor execution of a truck turn can be interrupted by a vehicle collision, so can skill execution be interrupted by lower level higher priority behaviors such as bump sensors. It is this capability that we intended to capture in our "skills" implementation (as described in Section 6.4).

6.4 Skill Learning

Skill Acquisition

The development of control software for actuators is often a time consuming, trial and error process due to the inherent error between specified actions and their real world execution. Unforseen externalities and robot hardware deficiencies often conspire to hinder accurate execution of preprogrammed actions.

To facilitate "skill" learning and alleviate these difficulties, we incorporated a joystick interface into our robot and implemented a method of actuator learning similar to that used for programming some industrial robots. The joystick control enabled us to move the robot interactively. "Teaching" the robot a skill involved moving the robot interactively to execute the desired action while at the same time storing the associated motor and steering commands and odometric readings between consecutively different commands. By using the operator in the feedback loop, a skill can be precisely learned and reproduced each and every time, since the actuator commands stored are exactly those executed during operator control. The accuracy of the reproduction is limited only by the sampling error associated with the encoder used to measure distance and of course, relies on consistent contact between the odometric wheel and the floor. To execute such an action in normal operation, each set of actuator commands in the stored sequence is simply "replayed" for a duration determined by the corresponding odometric reading, in essence, executing a specific trajectory fragment. The execution of trajectory fragments continues until the entire set is either completed or interrupted as a result of a change another part of the system. For example, one such action could be a precise "left turn". A lower level, higher priority process monitoring wheel rotations might determine that the robot has stalled and consequently will never complete the specified action. Such a fact would be communicated through common data structures causing the execution of the skill to be suspended, perhaps only temporarily, enabling another, lower level, reactive behavior to take over to try and correct the impasse and possibly resume the skill execution.

Time was not considered in the implementation of skills for a number of reasons. First, the type of skills we considered did not have a dynamic element to them, that is, the speed of the skill implementation was not a relevant factor. This may or may not be true in general and hence our particular representation may not be optimal for all applications (but was for ours). In our case, the most important requirement was that the robot move in a particular trajectory for given distance. Second, using time to control the duration of motor and steer commands had negative practical ramifications for accurate replay of learned skills. This was primarily due to the fact that the instantaneous power level of the robot directly affected the speed of the robot at any given command setting for motor control, and hence the time spent in that state. For example, assume that during learning, a motor command is stored while the robot power level is high. Later, the command is replayed when the power level of the robot is low. The time stored during learning will be shorter because the robot speed will have been greater and will be too short when the robot "replays" the command later on. The use of odometry to monitor the progress of the action therefore ensures that power fluctuations and furthermore, mechanical irregularities do not impede action execution, since physical movement of the robot is required to complete it.

Skills are implemented using one array to store encoded motor and steer commands (both commands are stored using a single integer) and another array storing the distance traversed between changes in commands. The possible commands are also discretized and referred to by array index to reduce the number of command permutations, and hence reduce the size of the command array. Distance is measured using a single wheel encoder on the front wheel. Advantages of using distance traversed over time passed are:

1. "Duration" of commands does NOT depend on power level since the robot must physically move to complete a command.

2. Repeatibility during manoever reversal (see below) is improved.

3. The algorithm for manoever reversal is easy because both steering and motor commands are available at each step.

4. Skill array simplification is more straightforward because distances in the array can simply be added for identical steering commands. For example, if over some distance interval the same steering command is used, but the speed varied for some reason during learning, the distances traversed can be added and all commands with the same steering in the interval compressed into one steering command at a some fixed speed. This simplification not only reduces the size of the arrays needed to store the manoevers but allows efficient, automatic array simplification.

Primitive manoevers left-turn(), right-turn(), straight(), avoid-go-left() (a jog to the left) and avoid-go-right() (a jog to the right) were all "learned" and stored. The manoevers avoid-go-right() and avoid-go-left() were parameterized to allow edge location from visual input to affect the amount that the robot shifted to the right or to the left.

Manoever Reversal

Manoever reversal refers to replaying a set of trajectory fragments backwards in order to reset the robot to its starting position. This was found to be useful for small reversals but results varied in accuracy for more complicated trajectories. More experimentation with this type of manoever might prove very interesting. The idea of being able to "take back" a given robot move precisely might be extremely useful in many situations. For example, if a robot were to enter a trap and recognize it as such, manoever reversal might be able to take it out. A "left turn" into an object could simply be aborted and the robot returned to its original location, giving it a chance to try something new.

Photoresistor Response Time

A set of tests was made to measure the approximate rise time of the photoresistors we had available. Photoresistor rise time had ramifications for the speed of the linear scan and hence the time required for "image" acquisition. Each photoresistor was tested mounted inside a scanner tube as described in the section on the linear scanner. Before testing, the photoresistor was calibrated using black and white fields under the same lighting conditions as the test. Note that a linear response was assumed even though there was evidence that the response of some (or all) of the photoresistors was not. A simple subroutine was written to monitor the derivative of the calibrated graylevel change over time. A timer was started as soon as a "sudden" rate of change was detected. The time taken from this point to the time that the derivative became zero (i.e. no further change in graylevel) was considered to be the time to steady state. This method was used for two reasons. First, using a "sudden" rate of change in the derivative (where value of the derivative corresponding to "sudden" was determined experimentally) to start the timer allowed the measurement to be started simply by turning on (or off) a light. Second, going from total dark to bright light (under calibrated conditions of course) resulted in the largest possible dynamic range, presumably giving an upper bound on the rise time measurement. Measurements indicated a maximum rate of change ranging between 1.0 and 6.5 calibrated graylevels per ms (or if a full 5 volts can be assumed for logic, 0.02 to 0.13 volts per ms) with a rise time ranging between 24 and 197 ms. Various combinations of lighting and dynamic range were experimented with to obtain a full range of values for most possible field conditions.

Servo-multiplexing Circuit

Ray Frank designed a servo-multiplexing circuit to enable us to use the same circuit for controlling two servos, one for the steering and one for the linear scanner. In normal, free-floating mode, both

servos are active, i.e. a signal to any one servo causes both to respond. A logical one on either control line suppresses the operation of the corresponding servo. Both control lines for the servos were connected to the output of one of the motor ports. A signal corresponding to a motor command instructing the motor to turn one direction selects one servo while a command to turn the other way selects the other servo. This also resulted in a different color LED being turned on whenever one servo was active. Capacitors were added in various locations to reduce noise in the circuit, since interference from motor operation caused both servos to "twitch" periodically. A circuit diagram is shown in Fig. 7.

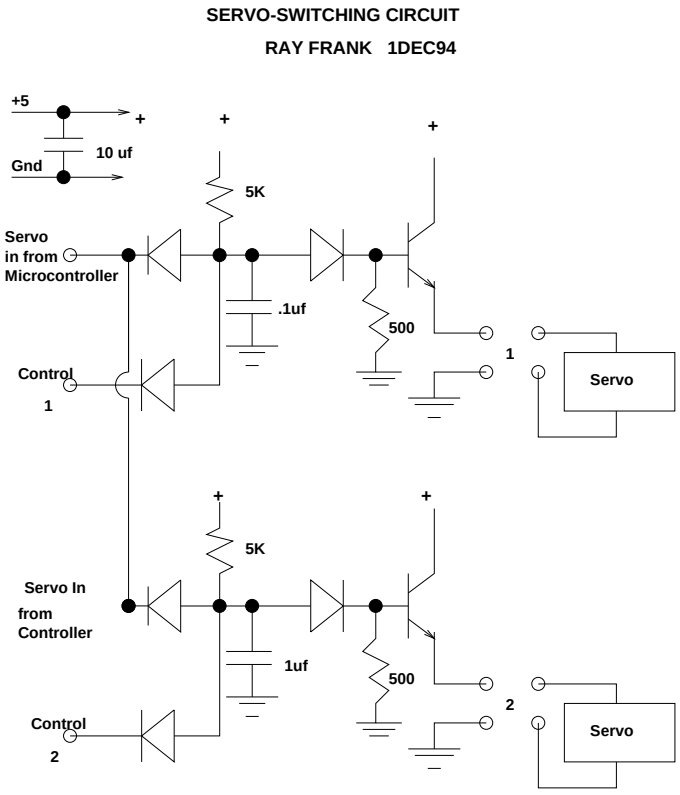


Figure 7: Servo Multiplexing Circuit.

Behavior as a Hierarchy of Goals

Goal-driven actions and "behaviors" are usually considered separately. From one perspective, one could think of a behavior as differing from the execution of some high level goal only in the scope of the resulting action. For example, in line-following, the robot generates steering and motor commands whenever a line transition alarm is indicated. Due to the swiftness of the action and the nature of the trigger, this type of action is typically called a "reactive behavior". The commands generated compete in priority with other commands to finally execute a command which essentially results in a circular trajectory of an arc-length determined by the time required to move the robot to a position straddling the opposite side of the line. One could interpret this trajectory as being the fulfillment of a minor "goal" of the system, that is, to move the robot from one side of the line to the other. This "micro-goal" is simply triggered by sensor input instead of a higher reasoning faculty.

Now, take as a second example, the action initiated by our robot upon seeing an object in its path. The robot stops, does a linear scan to acquire the location of the object edges, and then generates a move to avoid the object. In this case, the assocated logic uses the object edge location to derive motor and steer commands from the stored skill (see Skill Learning). The action corresponds again to a trajectory, albeit a more complicated one, and can also be interpreted as the fulfillment of a goal, in this case, to avoid the object. As before, this set of commands also competes with other commands for control. For example, actions initiated by "bump" sensors may temporarily interrupt execution of the skill. In this sense, there is a layering of control similar to the layering postulated by K.L. Doty, based on temporal considerations. However, the use of execution time to discriminate between levels of control does not seem very intuitive since the time taken to execute various actions depends more on hardware and the computational resources of the robot than on absolute necessity. The time taken to execute a particular action is very different from the *timeliness* of a particular action. The way in which we ensure that a timely scheduling of the lower level task processes is done is by statically describing the "lazy" or "eager" type of process to be created. This is similar in function to the types of sensed feature refreshing in [3]. One might be able to postulate a distinction between layers that is based on scale of action, that is, the resolution of the goal. In the case of line following or bump behaviors, the robot response (or goal) is rather subtle. In the case of skills, the robot response typically has greater scope, involving complicated movement trajectories.

An even more ambitious goal such as map building for improved navigation could be incorporated by treating exploratory behavior as a low priority process. That is, when no other higher priority processes are active, an exploratory behavior initiates skills such as "turn left" or "turn right" to examine an area of interest. The exact nature of the exploratory behavior could be determined by a combination of sensor information (e.g., the current location of the beacon) and memory (e.g. the expected location of known objects). The result of the exploration would then be used to update the map.

6.5 Cyclops actuators

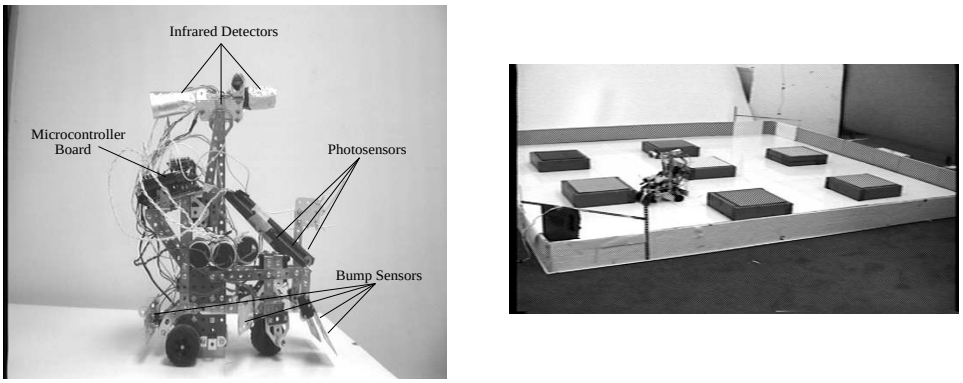
1. Driving motor. It transmits power to the rear wheels through an 81:1 gear reduced box.

2. Direction servomotor. It drives directly the steering axle that carries the two front wheels. The overall steering capability is 174 degrees. The front wheel module is mounted so that steering is straight for a servo control value half way between the maximum and minimum possible values. The steering values practically used are bounded by the ± 70 degrees representing minimum/maximum values.

3. Scanning servomotor. The commands to this servomotor are obtained by multiplexing the drive signal taken from the servomotor output of the controller board to both servomotors using a control signal taken from a free DC motor. It orients the artificial eye to a set of fixed front directions in order to scan a horizontal set of equidistant points. The reflection information obtained during a scan is used to generate high level information about the existence of objects (obstacles) in front of the vehicle.

4. Ball eject motor. This DC motor is used to drive the ball eject mechanism used in the delivery competition.

5. IR emitter. It is activated in order to detect side obstacles close to the vehicle. It can emit light on one of two frequencies 100 or 125 MHz. The IR side reflectance sensors can be tested to detect if a side obstacle reflects IR light.



(a) (b)

Figure 8: (a) The "Meccano Monster" mobile robot (b) The robot in action.

7 Meccano Monster (Fuentes, Rao, Van Wie)

The winner of the competition incorporated physical and algorithmic robustness. Physical strength came from Meccano construction, which though less suited for rapid prototyping has definite advantages of durability and dependability in the field. This robot was also extremely fast. The strategy employed by this team was to use the IR beacons for guidance and to head toward them, with reactive obstacle-avoidance to get around barriers. The unit had no memory of where the obstacles were.

The Meccano Monster (Figure 8) is a purely reactive mobile robot built with parts from the MIT 6.270 kit and two store-bought Meccano erector sets. It rests on a three wheel base, with a servo motor controlling the angle of the front (steering) wheel. In its present incarnation, its sensors include four IR sensors wrapped in tinfoil for directionality; two front, two side and one back bump sensor; three forward-looking light detectors; and two downward-looking light detectors. At the time of the UR minibot competition, a mechanical arm was attached in the place of the forward-looking light sensors.

7.1 Why Meccano?

Originally, we wanted to build a walking robot and do research into techniques for learning to walk. Thinking that it would be easier to build a bipedal body from erector set pieces than from LEGO, we volunteered to give up our LEGO pieces and instead buy a Meccano kit from the local toy store. Though we eventually gave up on the walking robot idea, using Meccano for the robot body turned out to be a good idea: in the end our robot was faster and more solid than comparable LEGO-built robots.

Meccano motors seem more easily adapted to running a robot than LEGO gears. Our motors were slower but more powerful, which meant our gear train could be relatively simple. Instead of a long 243-1 gear train, we built a simple two-gear chain that sufficed for our drive motor.

With a stronger chassis, we could affix more sensors to the robot than other teams. Having the extra sensors helped our line-following and wall-following behaviors, and improved the fault-tolerance of our algorithms.

Unfortunately, working with Meccano was not always pleasant. Our original design was a long, thin robot that would never have navigated the course we saw in the competition, and we spent many hours reworking the design to shorten and strengthen our robot.

One would think that with all that metal in the chassis, and all those sensitive electronic parts on the board, we might have problems with short-circuits. And indeed, at one point shorts became a problem.

7.2 Disaster!

We almost had a major disaster when we tried to turn the board on after mounting the IR detectors - first, the board didn't enter download mode, then, some components near the switch began getting hot and pretty soon we even saw a swirl of smoke rising majestically from one of the components.

This of course meant a short somewhere in our circuit and since things were fine until IR detectors came into the picture, we unplugged these, and it was one tense moment when we turned the board on again: much to our relief, a miracle seemed to have occured and, lo and behold - the board signaled its willingness to accept code again.

The problem turned out to be a short in one of the IR detectors, caused by the aluminum foil that had been wrapped around it for directionality. We fixed the problem by insulating the detectors with masking tape and carefully re-wrapping the foil covers.

The problem of shorts is indeed a serious hazard to the board and some of the places these are most likely to occur include:

- Connectors plugged into the board that are close to each other may have excess solder sticking to the pins and these may come in contact. Removing excess solder/plugging sensors into the board at alternating ports/using tape for insulation alleviates this problem a good deal.
- All sensor connections should be thoroughly checked and properly insulated using tape.
- The underside of the board may cause problems if not shielded (we used a cardboard sheet).
- Battery/sensor connections may be reversed.

7.3 Building the Board

A greater part of the first week was spent in reading relevant parts of the manual and reviving faded memories of diodes, transistors, resistor packs and inductances (chapter 2 was very useful). While some teams decided to sort out the various components before beginning assembly, we decided on a more direct approach considering the fact that we had three team members: two team members worked on soldering boards in parallel while one was in charge of selecting and directing the "where and what" of soldering. This gameplan speeded up the process considerably and we were able to get all boards soldered within six hours including a 45 minute break.

Lessons learned:

1. The instructions may not always be right - for example, when assembling the infrared transmitter board, if you try to solder the LED's after you've soldered the resistor packs (as instructed in the manual), then good luck! We ended up breaking on the leads of a high-power red LED. It pays to use common sense and think about the end result before jumping in to assemble things.
2. Another case in point: the cable wiring instructions for the download link - we faithfully followed the instructions step by step until we realized that we had cut the cable in vain - the phone cable could directly be plugged into the connector with no need for soldering. The instructions turned out to be outdated. Luckily, the damage was undone without much effort.

3. Piggybacking two chips was tricky - our problem was that there was a little too much solder on the connectors, making it hard for the chip on the bottom to slide into the socket. A little desoldering did the trick.

4. It helps to have a multimeter at your side to check the values of resistors, check connectivity, polarity etc. just to make sure things are going where and how they should be. Chapter 2 has some helpful diagrams for visual identification of parts.

5. Some components may have gotten damaged in transit or during soldering. One of the connectors on one of our photoresistors was broken when examined. The battery plug connector stopped functioning after soldering (probably due to overheating) - luckily, there was another one.

Apart from testing individual parts before soldering, the main assembly consisting of the microprocessor board, the expansion board and the LCD display were tested as given in the instructions. Everything went fine until we came to the part where we needed to check if the download mode was working correctly. Upon pressing the escape button and turning on the board, we were supposed to see a flickering of the yellow LED after which it would turn off. However, in our case, it still stayed on. The required behavior was however obtained when pressing the choose button. We decided to proceed with tests anyway, and it turned out that everything worked as described - we were able to download the pcode, use IC, load the program testbrd.c which checked various board components. The escape and choose buttons functioned normally. While we are not totally sure why the choose button and not escape button worked to get the board into downloading mode, one guess is that the results of pressing the two were flipped in our case due to the initial state.

7.4 Designing the Body with Meccano

Meccano design was an iterative process: we kept finding new and better ways of shortening our wheel-base, attaching sensors, and affixing the steering mechanism to the body. At least three times we completely disassembled our robot, each time cutting the length of the body significantly. Our guiding principle was to keep the robot as compact as possible, and in the end it had just enough room for its own sensors, effectors, batteries and CPU.

For steering, we mounted a single wheel on an assembly that was mounted on our servo. The CPU signaled the servo with the absolute angle it wanted (45 degrees for straight ahead) and the servo responded by turning the assembly to that angle.

Because the assembly occasionally slipped while the robot was in motion, we had to recalibrate the facing of the front wheel continually. To recalibrate, we set the servo angle to 45 degrees and manually rotated the steering assembly until it was pointed as close as possible to straight ahead.

Meccano motors are slower and more powerful than LEGO motors, which meant that our gear train could be simpler and more efficient that comparable LEGO models. In our case, a worm gear on the motor output shaft directly turned the gear on the drive-axle. Note that, while the lego-robot manual claims that using the worm-gear as part of a gear train is inefficient, we found that it worked quite well.

Sensors for the robot included five bump sensors, four IR detectors, and two light sensors.

The bump sensors were constructed from microswitches, to which we attached long, flexible meccano sheets as feelers. We tied the microswitches to the robot with twists of wire.

The IR detectors were attached to a sensor tower that rose over the center of the robot. Neither the tower nor the sensors were mobile; however, the IR detectors were omni-directional enough that even with fixed placements they covered the robot's entire range of vision. In fact, unshielded IR

detectors turned out to be too omni-directional – when approaching the beacon, all of them could sense IR, regardless of the direction they were pointed! We solved this problem by wrapping the IR detectors in aluminum foil (taking care to insulate them with tape first – see above).

Originally, we planned to use an electromagnet-driven gate to drop the ball at the destination square. Unfortunately, our experiments with electromagnets showed that anything with enough power to open a metal gate would draw far too much power from our board.

Our first attempt to build an electromagnet was the naive method of wrapping a wire around a iron nail - a test of the mechanism showed that it indeed opened the "door" for the ball to slide through, except for the problem that the electromagnet was drawing current close to 6 Amps, enough to burn the board many times over.

Later attempts proved somewhat more successful, but eventually we abandoned that design for something more striking: the mechanical arm. The arm was a catapult-like ball-throwing device, designed to throw the ball straight down, like a football player spiking the ball after scoring a touchdown or a slam dunk in basketball. A simple 90 degree rotation of the whole thing enabled it to throw the ball forward. We used a second motor to control rotation of the arm, with various physical hacks to keep the arm from damaging other parts of the robot. After several more rounds of design and construction, we had a working (and crowd-pleasing) model.

7.5 Software for Compulsory Floor Exercises

Before the robot competition came the Robot Olympics Compulsory Floor Exercises , where each team had to show a working robot running three different behaviors: line following, wall following and beacon following. For the Olympics, we stripped down to the barest essentials in sensors. Which sensors we used are listed at the head of each subsection.

For line following we used four photoresistors as sensors. The photoresistors are located about an inch deep inside paper tubes that point directly to the floor. The tubes increased reliability by minimizing the amount of ambient light that the sensors received. Preliminary tests showed that such an arrangement gave analog measurements roughly below 75 for white and above 75 for black at distances of approximately one inch.

To obtain a composite of the sensory input, we used the weighted sum of the individual sensor readings. The inside sensors were weighted by a small constant, and the outside sensors by a larger constant. The sensors to the left were weighted positively and the sensors to the right were weighted negatively. The composite reading was given by:

$$c = c_o * (Left_{outside} - Right_{outside}) + c_i * (Left_{inside} - Right_{inside}) \tag{11}$$

where $c_o > c_i$.

This way c was proportional to the change in steering angle needed to drive c to zero, which corresponded to the black line being at the center.

Finally we obtained the change in steering at time t using a PD controller as follows:

$$\Delta steering(t) = k_1 c(t) + k_2(c(t) - c(t-1)) \tag{12}$$

After tweaking the constants and adding a few hacks, we obtained a reasonably good behavior. By making the robot move as slowly as possible, we reduced the possibility of crossing the line without seeing it.

Other interesting aspects of line following included: normalizations for DC (additive) illumination changes and a proportional power control which supplied more power when negotiating sharper turns.

Our idea for the wall-following exercise was quite simple: try to keep the front sensor always off but keep the (relatively longer) side sensor always on (i.e. touching the wall). We also decided to use a controller that caused the robot's steering angle to increase in proportion with the amount of time the robot was off course – the longer the robot went without touching the wall, the sharper the correction to the steering. The implementation took less than half an hour but considerable amount of time was spent fine tuning the various parameters, such as when and how much to back-up in case of collision, rate of increase in the steering angle, and so forth.

We only used three switches for wall following, which appeared to be enough for a robust wall following behavior. The robot had two switches at the front, connected in parallel. When either of them was pressed, the robot stopped, backed up for a short time (about .8 seconds), steered to the left by a small amount (about 15 degrees) and started forward again. At the time of the Olympics, we had just one side sensor, so our robot could only follow walls on one side. When the side sensor was pressed it meant that the robot was successfully following the wall (provided the front switches weren't pressed), so it continued forward in a straight line.

When none of the sensors were pressed, the robot turns to the right, increasing the steering angle by a small amount each time though the loop until it found a wall.

With this simple strategy we obtained a surprisingly robust behavior. Before the competition the robot managed to circle the sixth floor hallway several times without human assistance.

The third compulsory exercise was beacon following. Preliminary tests revealed that the sensors were quite sensitive to IR radiation even when turned in the opposite direction.

We used cones of aluminum foil to shield IR radiation from all directions except the front, and then arranged the IR sensors in a cross, each at a ninety-degree angle from the two closest to it. The readings from these were then used to turn in the direction of the strongest response.

When no IR was received by any of the detectors, the robot continued in the direction it was originally going. This of course meant that it soon ended up bumping into obstacles, so we activated the bump sensors also during beacon following, so that the robot would back up in case of collision, hopefully getting back reception of the lost IR.

7.6 The Competition

Hardware for the competition was somewhat more sophisticated than for the Olympics: we had a full complement of bump sensors, the arm, and two downward-looking light sensors to identify the destination square. We kept the beacon-sensors as they were during the beacon- following test.

Our priority in designing an algorithm for the competition was to keep our code as simple as possible. To that end, we decided on a purely reactive, subsumption-style program. When none of the bump- sensors was active, the robot followed the beacon-following behavior from above. When the robot hit an obstacle, however, beacon-following was turned off for a short time (about a second) to allow the robot to back away from the obstacle and move away from the problem spot.

Unfortunately, the final algorithm wasn't quite that simple. The fact that we had to find the destination square complicated matters, as did the introduction of several programming hacks that were supposed to avoid certain kinds of traps.

To recognize the destination square, we used the two downward-looking light sensors. They scanned the floor for dark tape and, when either of them got three consecutive "black" readings, signaled to stop the robot. The algorithm took three readings before reporting that it had found the destination square in order to avoid incorrectly identifying the tape path as the goal.

Though this technique worked well in most cases, the algorithm often still incorrectly identified the source square as the destination square. To avoid this behavior, we added a simple test: if the

robot had seen the destination beacon in its front IR detector since the last time it had touched a wall, it assumed that any black square it found was the destination. If it hadn't, it continued looking. This approach resulted in the robot occasionally passing over the destination square without stopping, but in most cases it found it on the second try.

7.7 Beyond the Minibot

After the competition, having replaced the robot arm with an array of forward-looking light sensors, we developed a streamlined version of our algorithm. Three arrays are kept mapping sensor inputs to motor outputs: one for the various combinations of bump sensor readings, one for the various combinations of light sensor readings, and one for the various combinations of beacon sensor readings. If any bump sensor is active, we use the corresponding bump motor code; otherwise, if any light sensor is active, we use the corresponding light motor code; otherwise, if any beacon sensor is active, we use the corresponding beacon motor code. The codes are translated into steering and drive-motor outputs, which are used to drive the robot. If no sensor readings are reported, the robot continues with its last action.

The streamlined algorithm was invented for a research project we started after the course. The idea was to learn the motor codes by driving around a course at random, and our attempt at solving it involved a combination of stochastic hill-climbing and reinforcement learning. The space of motor codes was searched via stochastic hill-climbing, and the resultant motor codes were scored on their fitness through a reinforcement learning type algorithm. We refer the interested reader to [1] for more details.

In addition, we have also been working on methods for perception-based navigation using self-organizing sensory-motor networks [4].

The Cyclops robot has a complex control structure that is reported in the Spring AAAI Symposium on Instantiated Agents. The Meccano Monster was re-programmed and re-instrumented to do obstacle recognition with a triad of photo-resistors, each looking down a tube. This gave a (very) low-resolution view of the area in front of the robot. The robot used these inputs and a form of reinforcement learning to learn how to avoid obstacles [1]. Later, a simulator for the robot world was built and a simulated version of the robot learned to find its way home from any position on the arena [4].

A Parts Lists and Information

The following is to set up the lab with two workstations and to get four kits, one for assembly over summer by the course organizers, three for Fall 1994 Grad course.

The first issue is board kits.

A.1 Rug Warrior: Jones and Flynn Book

One choice: Rug Warrior kits from AK Peters (publisher of [2]).

Rug Warrior is a robot board designed to support the examples in [2]. The kit supplies all the control and sensor circuitry needed to construct a custom designed, fully autonomous mobile robot. The microprocessor, memory, and interface circuits are assembled and tested at the factory so that kits arrive ready to connect to the user's computer.

The assembly manual provides detailed instructions for installing the standard sensors and support circuitry included with the kit. Alternately, users can customize the board as desired. Diagnostic software, included with the kit, helps builders discover any errors they might make as subsystems are added. Library functions simplify access to all sensors and actuators. The self test code provides examples of how to use the library functions.

Interactive C (IC) is supplied with the kit as the suggested programming environment. Both Macintosh and PC versions are available. For those who have already built Rug Warriors on their own, the new software for Rug Warrior is now available via ftp on cherupakha.media.mit.edu. Look for:

```
   /pub/interactive-c/ms-dos/ic2853rw.zip
/pub/interactive-c/macintosh/ic-2.853-RW.sea.hqx
```

Kit features include the following:

```
Motorola MC68HC11A1 microcontroller
LCD display (32 alphanumeric characters)
32K of battery backed RAM
RS-232 serial port
Three bump switches (for 6-direction collision detection)
Two photoresistors for light detection
Two infrared emitters
One Sharp GP1U52X infrared detector
Microphone for sound detection
Piezoelectric buzzer generates tones of arbitrary frequency
Motor driver chip (SN754410) for control of two DC motors
Dual shaft encoders allow velocity/position control
Four user controllable LEDs
Battery holder, hookup wire, heat shrink tubing, and extra connector sockets
Digital and analog inputs (two each) are available for user assignment
Access to MC68HC11 bus allows construction of memory mapped devices
```

The kit is available direct from the publisher, they are charging $289.95:

```
A. K. Peters, Ltd.          kpeters@math.harvard.edu
289 Linden St.              (617) 235-2210
Wellesley, MA 02181         Fax: (617) 235-2404
```

A portion of the income generated by sales of the kits will be used to provide free kits to high schools that would otherwise not be able to purchase robots. (You can thank Anita Flynn for this policy.)

A.2 From Randy Sargent at MIT

```
6270KIT -- Complete 6.270 Kit
A complete kit that includes:
        1. Robot Board kit (assembled or Unassembled)
        2. 6.270 Notes
        3. Motor Batteries (2 sets)
        4. Sharp IR Sensors
        5. Touch Sensors
        6. LEGO Resources Kit
        7. Light Sensors
        8. Serial Cable
        9. Copy of IC

Cost:  $500                (unassembled)


ACTAS -- Actuator Assortment
(1) Solenoid, (4) 3-5V Mabuchi DC Motors, (1) Futaba Servo
Cost: $55
Misc: sensors and actuators:  $45.
```

A.3 Work Area

Another issue is a work area for electronic and mechanical assembly. We felt we needed the following items:

```
drill                   30.00
drill bits              13.00
hack saw                12.95
crescent wrench          7.50
box wrenches            23.70
hex wrenches            15.00
Phillips screwdrivers   10.00
slotted screwdrivers    10.00
hammer                   4.00
champ crimper           10.40
nut drivers             50.00
tin snips               18.10
groove pliers            9.00
slip joint pliers        5.70
multimeter              20.00
DC power supplies       50.00
tool box                15.00
2 power strips          20.00
```

```
2 work benches          164.00
2 vises                  30.00

wire cutters              6.00
needle-nose pliers        6.00
wire stripper             9.00
utility knife             1.50
2 soldering iron         30.00
2 holders                 5.00
solder                   10.00
desolderer               15.00
safety goggles            3.00
(misc: nuts,bolts,glue,etc)
                         100.00
```

B Supplemental Sensors

As was mentioned in the previous sections, some of the robots had some interesting hardware that was not included in the original parts list. Since our exercise was meant to encourage independent thought and to lead on to research, the organizers were not comfortable with enforcing uniformity on the projects. Thus no reasonable request for unique hardware was refused (including the whole Meccano kit, for instance). Also, Brown ordered several useful-looking sensors during the semester, which Ray Frank then had to get working.

B.1 Parts List

We ordered and received the following:

1. 4-wire flat cable. VERY FINE wires, too small for our stripper, thus very hard to work with. Might be worth getting smaller stripper, since the wires are more supple than the round 3-wire cables.

2. 10 Slotted optical switches (2-connectors for emitter, 3 connectors for detector). This is an analog device. As far as we can tell these devices are worse than the ones that came with the kit, but still usable. The 2+2 connector parts have a 4.5V difference between on and off, the 2+3 have 1.5V difference at best. See Section B.2.

3. 10 "reflective optical sensors". These are parts with 3 wires (yellow, orange, green) and a female connector, with what looks like 2 leds in a black plastic case. These actually turned out to be just two IR LEDs and NO detector: a cataloger's error. See Section B.3.

4. 5 electret microphones. These perform reasonably, but possibly do not have much range. This is an analog device. The classic problem with robot-mounted mics is that local ambient noise from motors drowns all other sounds. Still... See Section B.4.

5. 5 Mercury switches. Good for signalling tilts. This is a digital device.

6. 6 electric motors. Seem to work...have their rotors visible.

7. 10 IR detector/transmitter bonanza packs duplicating some we already had.

8. 10 IR LEDs.
9. . 15 Super IR Leds.
10. 5 IR transmitter/receiver pairs.
11. 10 Ultrasensitive IR sensors.
12. 10 miniature optical reflectance sensors. Small, cheapies. See Section B.5.
13. 1 ultrasound emitter/detector pair for experimentation. See Section B.6.

B.2 Break Beam Sensors

The 2+2 connector original parts seem to work better. Ray had best luck wiring them as shown in Fig. 9 (Fig. 10 is more explicit). He got a 4.5V difference between broken and unbroken. Note the 100 ohm resistor in place of the 330 recommended. Likewise the 1.2K resistor works better than the 47Kohm supplied.

For the 2+3 connector parts, Ray had best luck wiring them as shown in Fig. 9 (also see Fig. 10). He got a 1.5V difference between broken and unbroken. Note the 100 ohm resistor in place of the 330 recommended. Likewise the 1.2K resistor works better than the 47Kohm supplied.

B.3 IR LED Pairs

This seems to be a very unexciting part. One (or maybe both) of two IR LEDs light(s) up depending on where power is applied. Fig. 11 is what works (if you can call it that). Perhaps the 100ohm resistor would be better than 330, as usual.

B.4 Microphone

These have only been bench-tested at this writing, but seem to put out enough voltage to be detectable. If it is too low, mention the problem to a course organizer and we'll get Ray to whip up some amplifiers. They are analog devices, wired as in Fig. 12.

B.5 Miniature IR Reflectance Sensors

These tiny devices have three leads and two open windows with silver metal inside them. Surprise, the "business" side (the emitter and detector) are on the OTHER side of the device from the windows, shielded by a daylight-filter material of glossy brown or black glassy stuff. On the bench we find these devices to be sensitive to white reflective surfaces up to an inch away, with a nice smooth response until the surface gets close enough to block the emission. We find white and black ink to be discriminable, so these devices should be good for line-following as well as really proximal proximity-sensing. They are analog devices, wired as in Fig. 13.

B.6 Ultrasound

The emitter is labeled A3S on the back, the receiver A3R.

The ultrasound emitter must be driven with a square (or sine) wave of 40KHz. The receiver synchs with the transmitter if the signal is strong enough. The transmitter can be driven at other frequencies but the receiver only synchs if the emission is pretty close to 40KHz. There are more

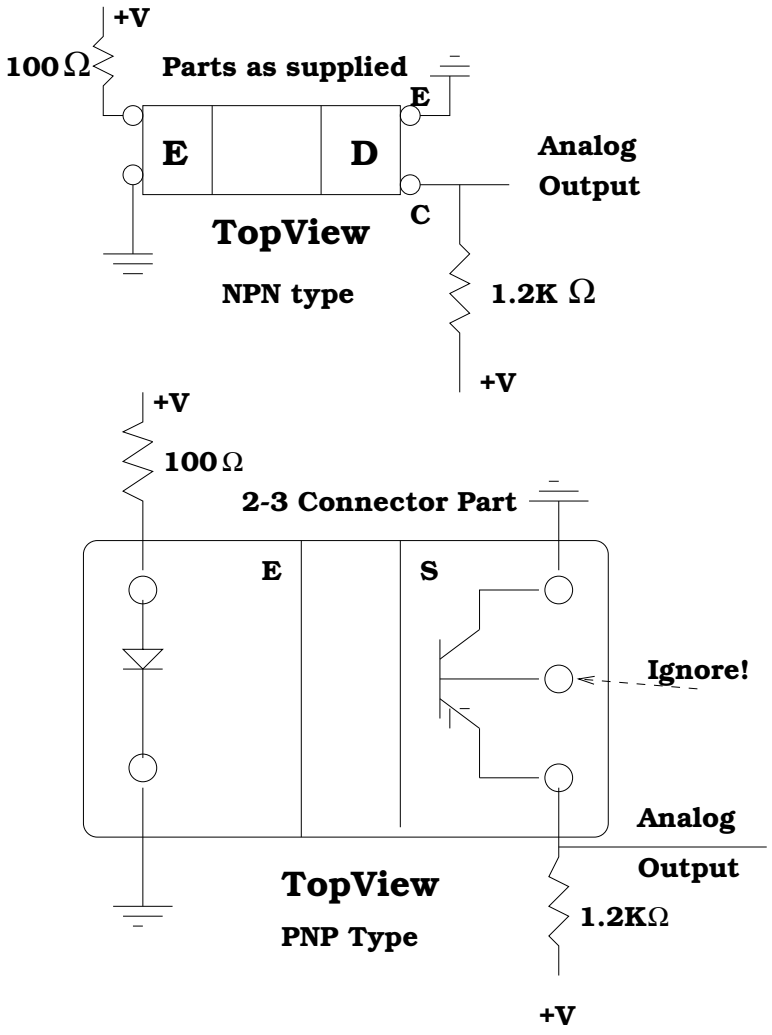


Figure 9: BreakBeam sensors (old and new) and wiring diagram.

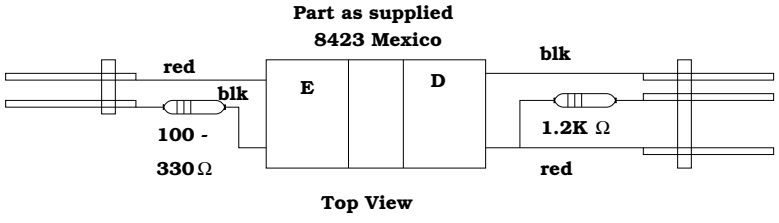


Figure 10: Physical interpretation of break-beam wiring.

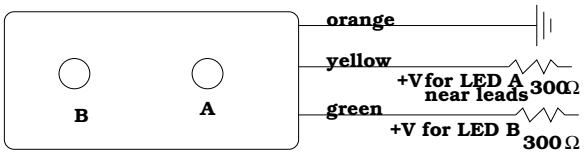


Figure 11: Two LEDs.

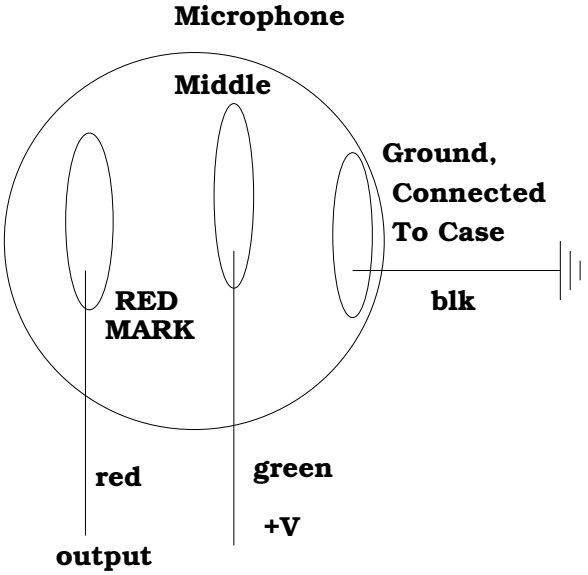


Figure 12: Electret Microphone Wiring.

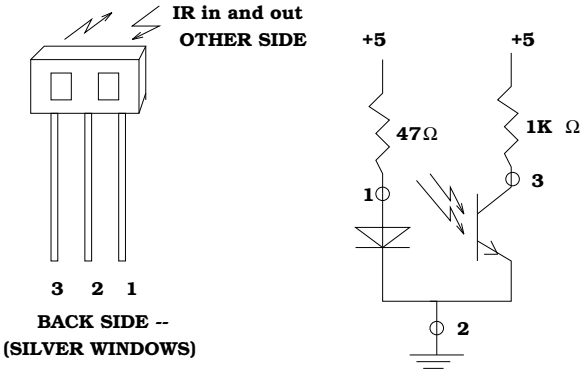


Figure 13: Miniature IR reflectance sensor wiring.

specs in the Electronic Goldmine catalog but they may not be too helpful. We were driving the emitter with up to 20 volts, at which power the range seemed to be about 3 feet. The sound bounced off surfaces, etc. At lower powers the range diminished but devices still could be useful for proximity sensing.

The IR beacon output is a 40KHz square wave modulated by 125 or 100 Hz square wave. So that should generate ultrasound pulses at 125 or 100 Hz. This hack has not been tried as of this writing. It is not hard to design a 40 Khz oscillator and there are probably enough connections on the expander board to make the ultrasound a reality.

B.7 Further Ideas

1. Colored LEDs for colored light recognition. Electronic Goldmine has.
2. Polarization filters – as in AAAI94 competition: way to recognize beacons at one of two orientations. Edmunds will have these.
3. Bending sensors – what's with ACE or ART or wherever that place is? Tim's idea of stiff whiskers inside long coil springs...
4. Hall Effect Sensors – AAAI94 again. Electronic Goldmine has.

C Course Details

C.1 Goals and Organization

577 Seminar in AI: Practical Robotics (4 hrs)

The mechanics (and electronics) of this course consist of populating a few printed circuit boards, including a powerful M68HC11 microcontroller, and building a mechanical device with sensors and effectors (by default a LEGO floor-rolling robot). Intellectual issues are the design of the hardware and the control of the final device to accomplish some task. Plans are for the class robots to take part in two competitions of some sort, with the goal of research relevance or publishable results from the second competition. The controller is a powerful device and the sophistication of applications are likely to be limited only by the difficulties of mechanical design and fabrication. Programming is in "Interactive C", cross-compiled and downloaded from a Sun or other workstation. One goal of the course is to expose students to various basic aspects of robotic construction and control, but there is plenty of time and opportunity to explore more advanced aspects of AI and robotics. Finally, the completed microcontrollers could play roles in future research projects.

Texts: *Mobile Robotics* (Jones and Flynn). (Bookstore) Motorola M68HC11 Reference Manual (Free from CS Dept) UR's version of MIT's and AAAI's 6.270 Handbook (Sold by CS Dept)

Reserve Reading: *Mobile Robotics Readings* (2 vols), various papers on control, learning, etc. as needed.

Proceedings MLC-COLT Workshop on Robot Learning June 10th 1994.

Course Committee: Brown, Karlsson, Becker, Miller, Frank, Nelson.

There will be one lecture or general session per week, which will move from being an introduction to soldering to consideration of promising research projects in the LEGO robot domain.

Otherwise student time is spent on preparing for the three robot events in the course, or in reading, writing, and thinking about mobile robot research. Students will divide into groups of 2-4 people. Each group will be given raw materials for which they will be held responsible. The

microcontroller board built during the course is a valuable piece of hardware and may be useful in future research projects.

The first robot event is a non-competitive exercise that demonstrates basic sensor and effector capabilities and LEGO engineering. The next event is a competition, which will be run twice to allow time to do further debugging and improvements.

The third event is an "individual" (really group) project that can have as much research content as practicable. Submission of papers to the next AAAI Workshop is a sample of a specific goal but will require advance planning (Oct 28 Deadline). Anything is possible here, including relaxing mechanical design constraints (e.g. ME is sponsoring a rope-climbing event in the spring – collaboration could be possible). The "autonomous" constraint can be relaxed to allow umbilicals for power and information, allowing quicker movement for pole-balancing or RHET-sized reasoning engines.

Assignments:

0. Three robot events plus one re-match.

1. Weekly written report, one per individual – mechanical or electronic construction work, or programming – tangible results from individual. – ideas contributed (used or not). – plans for next week.

2. Two written group reports: one on the competition – issues worked on by team together – competing ideas, compromises, etc. – interesting facets, solutions, wrinkles, ideas in your robot.

one on the group project – TR or conference paper style, including references.

3. Group interview – course organizers need feedback on the course: what you learned, how course could be improved, suitability for undergraduates, suitability as research launching pad.

4. Grades from group members?

———

Schedule by Week (Highly flexible – slippage is likely)

1 Lecture: Organization: course goals, choose meeting time.

At meeting time: Hand out robot kits, intro to the microprocessor board, soldering demo. LEGO robot lore on the net.

Participants: assemble and debug PC boards, get familiar with IC.

2 Lecture: Lego Engineering, sensors and effectors.

The Compulsory Floor Exercises:

test_board suite runs successfully locomotion: forward, reverse, left and right turns behaviors: wall-following with bump sensors homing on IR beacon line-following fiducial-counting and distance estimation odometry (?)

Participants: design robot to do compulsory floor exercises.

3. Lecture: Sensors and Effectors continued, Subsumption architectures.

Participants: design robot to do compulsory floor exercises.

4 Lecture: The MC68HC11 architecture: tools and opportunities.

Participants: Demonstration I: Compulsory Floor Exercises

5 Lecture: Assembly Language and interrupt-level programming (we hope). Competition I description.

Participants: Begin Competition I Design.

6 Lecture: Robot Control Architectures: Subsumption, Blackboards, Hierarchies, Reactive Systems etc. (some readings?).

Participants: Continue Competition I Design.

7 Lecture: Control – relevant aspects of open-loop, closed-loop, feed-forward control.

Participants: Continue Competition I Design.

8 Lecture: Learning – CMACs, nets, reinforcement, case-based, map-making...

Participants: Competition

9 Lecture/Discussion: State of Art in Lego Robotics, Research Opportunities.

Participants: Re-Match

10 Lecture: Discussion: What further lectures do we want/need?

Participants: Begin Serious Group Project Planning.

11 Lecture: TBA

Participants: Continue Group Project Work

12

Participants: Ditto

13 Lecture: TBA

Participants: Project Demonstrations

14 Lecture: TBA

Participants: Project Demonstrations

15 Team Interviews

————– Things that will be learned or (if not known apriori by participants):

Electrical Engineering: Basic soldering techniques, Board debugging techniques,

Sensors: Optical, Infrared, Mechanical, Gyro, GPS, Accelerometers

Mechanical Engineering: Basic concepts of mechanical advantage through levers, gears, chains. Different motor types, motor principles.

Lego Engineering: Stable structures, Lego units for gears and support beams.

Computer Science: Subsumption Architecture, Reactive Systems, Machine Learning (reinforcement learning, Q-learning, neural nets), Multi-tasking "parallel" implementations, Computer Vision (?), Research issues in the Navigation Domain, Research issues in small robotics,

C.2 Compulsory Floor Exercises

Due: 12 October

The idea here is to demonstrate your robot's ability to do basic sensory and motor tasks, without putting the individual tasks in any sort of integrated, goal-oriented context.

For a demonstration of the various capabilities, you might consider composing the various abilities and stepping through them with the escape program, as the testboard.c program in /u/minibot/ic/src does.

Note that CB bought lots of extra sensors, motors, and STUFF, outlined in the piece by cb and ray in the lab notes. Lots of it works, some is untested. Feel free to request some of these pieces if you think they'll help you.

——

You should demonstrate all the following basic capabilities:

1. testboard.c should run without problems.
2. Turn right.
3. Turn left.
4. Drive "straight ahead" open loop (of course you'll get an arc, but it should be approximately straight ahead).
5. Follow a wall as in the "rug warrior". This could use two bump-sensing switches and could subsume 1.- 3.
6. Use interrupt-level assembly-language programming to sense the output of the breakbeam sensor.
7. Use your breakbeam sensing to implement shaft-encoders. Monitor left and right wheel revolutions and use a controller to equalize them to implement "straight ahead" driving. Can you drive in a straighter line than you could open-loop?
8. Sense the IR beacon. You might want to try this two ways, using the system call and following up the hint and using a direct analysis of the digital input. The latter saves time.
9. Drive in the direction of an IR or visible beacon. Two sensors might help here, mounted left and right; as in the shaft encoder case, you might want to turn right if the left input is weaker and vice-versa. Can you drive straighter this way than in 6. or 3.? You can rig an IR beacon from one of CB's ultra-bright leds, a resistor, and one of ray's 5V power supplies.
10. Follow a black stripe on the white "robot arena" using feedback control. For this you can make your own reflectance sensor or use one of the emitter-receiver pairs CB ordered.

In addition to these basic tasks, there are plenty of fun things we'd like to try. The more capabilities we can demonstrate the better. (We shall get records of these things...video or 35mm slides or both – both the behavior and details of the mechanical wizardry involved).

So the organizers would like to encourage you to volunteer to try one or more of the following "off the wall" ideas. These are in totally random order of interest and difficulty, but that sort of thing should be obvious.

1. Detect Color. I don't think we have any color LEDs but they're easy to get and radio shack probably has some anyway. If you want to pursue this talk to CB and he'll order some if need be. There are color filters in the lab. We can get more from Edmund.
2. Detect Polarization. A la AAAI contest. I guess this takes buying some polarizing filters. Probably not a big deal...
3. First, can IR detectors detect candles? CB suspects so. Buy a small cylindrical box of Quaker Oats...throw them oats away, feed 'em to the birds, or compost them. Save the top. Cut a penny-sized, penny-shaped hole in the center of the top and put it back on. You now have a vortex cannon. Light a candle, stand a few feet away and thump the back of the box I mean fire the vortex cannon at the candle. You should be able to extinguish the flame from several feet. Now get a fine maduro cigar, preferably from Cuba (warning, illegal) or the Dominican Republic. Uppmann is a good brand. Carefully clip the end, light it, draw gently several times and let a wonderful feeling of well-being suffuse your entire body, soul, and spirit. Welcome to the wonderful world of cigar smoking. Before you finish the cigar, load the cannon with smoke. Fire it and you can see the vortices. So here's the idea. Mount the cannon on your robot and

see if you can detect and extinguish candles. You can pre-arrange candle and cannon heights and orientations so you don't have to go looking for or aiming at candles. I'm wondering just if you can detect them and kill them. I'm thinking a motor or the servo or maybe something clever with a rubber band and trigger can be used to fire the cannon repeatedly.

4. Can you fire a single ping pong or plastic practice golf ball from some computer-controlled contraption? What sort of range and accuracy is possible? How about multiple balls?

5. Can you pick up an object or somehow "collect" it and carry it? Object could be small styrofoam box, a metallic thing (maybe you can make an electromagnet?), a ball.

6. Can you detect things on the floor you might want to pick up? Can you get their direction and range? Maybe with special paint and lighting can you detect balls, white or colored styrofoam objects, etc. Would maybe an "IR designator" work, where you aim an IR emitter at what you want detected and the robot uses the reflection?

7. Can you use two directional detectors (say for IR) and do crude stereo by measuring a vergence angle? Maybe one sits on the servo and you thus can measure the angle.

8. Can you do IFF (Identify Friend or Foe?). Work with another group to see if you can use the IR detector/emitters in their usual mode: you broadcast 100 Hz and he broadcasts 125 Hz and you try to recognize/locate him and vice-versa. This is basic in all MIT contests. Can MORE information be broadcast between robots using the IR beacon? You can maybe modulate the 125 or 100 Hz square wave in different ways to send different messages...how accurately can these messages be decoded?

9. How easy or possible is it to "dribble" or simply push a ball around and keep it under control? How about propelling it on the ground in a desired direction?

10. Try out the idea of driving the ultrasound generator from the IR output port and detecting it.

11. Can you get the microphones to work? To direct robot by clapping, say?

12. Odometry....since you have shaft encoders, can you do quantitative measurement of distance travelled?

13. Can you use parabolic reflectors (perhaps from flashlights?) to send and/or receive IR messages? You'll need a long set of wires or a cooperative buddy robot. Modulate the IR emitter and detect its output (put phototransistor at focus of the receiving antenna?). Ultimately in order to communicate you'll need a transmit/receive protocol, error correction, etc. etc. Maybe one reflector houses the emitter and receiver. Should be fun.

14. Bar-code reader. Can you reliably count a number of parallel black stripes (electrical-tape width, say, separated by similar space)? Maybe the "barcode" starts with a double thickness? I'm thinking you just drive over the code and read it with reflectance sensing.

15. Try the prey-tracker posted on cs.minibot recently

16. Any other cute idea you have to extend our range of sensing and effecting capabilities.

C.3 The Robot Competition

See Section 3.

References

[1] Olac Fuentes, Rajesh P. N. Rao, and Michael Van Wie. Hierarchical learning of reactive behaviors in an autonomous mobile robot using stochastic hillclimbing. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics 1995*, Vancouver, B.C., Canada, October 1995. to appear.

[2] J. L. Jones and A. M. Flynn. *Mobile Robots: Inspiration to Implementation.* A. K. Peters, 1993.

[3] Tom M. Mitchell. Becoming increasingly reactive. In *Proceedings of the AAAI*, pages 1051–1058, 1990.

[4] Rajesh P. N. Rao and Olac Fuentes. Perceptual homing by an autonomous mobile robot using sparse self-organizing sensory-motor maps. In *Proceedings of the World Congress on Neural Networks 1995*, Washington, D.C., July 1995. to appear.