FEATURES

Robots with a Vision

FEATURE ARTICLE

Bill Bailey, Jon Reese, Randy Sargent, Carl Witty, & Anne Wright

Using the Cognachrome Vision System

Having a problem mastering Sampras's serve? This robot, equipped with a color-conscious vision system, can chase down your wayward tennis balls. You'll find it's a grand slam system. Game, set, and match to M1.

Machine vision has been a challenge for AI researchers for decades. Many tasks that are simple for humans can only be accomplished by computers in carefully controlled laboratory environments, if at all.

Still, robotics is benefiting today from some simple vision strategies that are achievable with commercially available systems.

In this article, we fill you in on some of the technical details of the Cognachrome vision system and show its application to a challenging and exciting task—the 1996 International AAAI Mobile Robot Competition in Portland, Oregon.

MACHINE VISION

Vision systems typically have the architecture depicted in Figure 1a. But, this way of processing images has a problem. There's too much data in the video streams.

The NTSC video standard (used in North America, Japan, and several other parts of the world) provides about 240 lines of video at 60 frames per second. It takes a very fast CPU to do any significant processing at that rate.

The sorts of CPUs typically used in embedded systems generally can't

process video at the full 60 Hz. Ranges from 1 to 5 Hz are much more common.

The Cognachrome solves the problem by using hardware acceleration to do relatively simple vision processing (see Figure 1b). This strategy improves performance while reducing overall cost. The Cognachrome simplifies the vision task by looking for only three colors, which the system is trained to see.

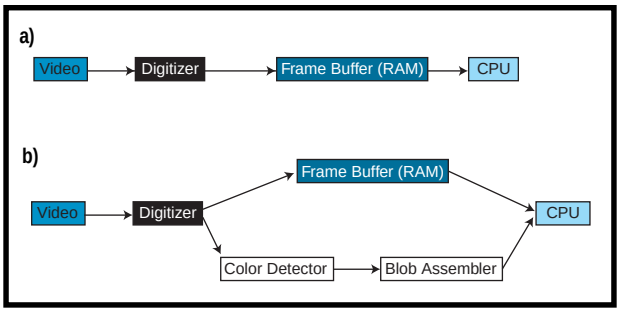


Figure 1a—*A typical machine vision system loads digitized pixels into RAM for the CPU to process in software.* ***b***—*The Cognachrome system achieves 60-Hz tracking with special hardware that detects pixels of interest and assembles them into contiguous blobs.*

During operation, the acceleration hardware compares each pixel against these colors and groups contiguous pixels of the same color into abstractions called "blobs." Client software then uses the location and size of blobs (as well as other information about them) to identify and react to its environment.

The Cognachrome uses a simple fixed-coordinate system to refer to the video image. The horizontal axis ranges from 10 to 230 (left to right), and the vertical axis ranges from about 10 to 240 (top to bottom.) The exact numbers depend on the particular camera used.

Photos 1a and b demonstrate how the Cognachrome processes a video image. The hardware presents the Cognachrome with the blobs as shown in Photo 1b. For each blob, the Cognachrome computes several interesting statistics, including:

- the *x* and *y* coordinates of the centroid (i.e., the center of gravity)
- the area (number of pixels)
- the bounding box (the *x* coordinates of the left- and rightmost pixels and the *y* coordinates of the topmost and bottommost pixels)
- the aspect ratio (how elongated the blob is). A value of 3 indicates that the blob is three times as long as it is wide.
- the orientation (only meaningful for elongated blobs; the direction of the long part of the blob)

The user can add other useful statistics.

The Cognachrome can compute these statistics at frame rates up to 60 Hz. The actual frame rate achieved depends on the number of blobs in the image, their sizes, and which statistics are computed. Aspect ratio and orientation are much more expensive to compute than centroid, area, and bounding box.

When not requesting aspect ratio and orientation, the system can handle 10–20 blobs quickly. If aspect ratio and orientation are included, it may only handle 5–7. If the system is overloaded with too many blobs, it drops to 30 Hz or less.

The Cognachrome can be used to grab frames into a frame buffer and do more traditional vision processing on them. Resolution is lower in this mode (e.g., 64×48 to 64×250). The frame rate in this mode depends on the vision processing being done, but software-only processing is unlikely to be better than 30 Hz, even for the simplest processing.

USING THE VISION SYSTEM

The Cognachrome can either be used as the main processor in an embedded system or as a peripheral to another computer.

In embedded use, the user programs the vision system by registering a callback function. The callback is invoked after every frame of video and has access to all of the blob data. Statistics are not computed for a blob until requested, avoiding unnecessary computation.

COGNACHROME HARDWARE

The vision board has NTSC video input and output jacks, which provides a lot of flexibility in the choice of cameras. We put small CCD camera boards on our mobile robots. Size isn't as much of an issue for stationary applications, so we often use camcorders for their flexibility and low cost.

The Cognachrome has a video output jack for viewing the blobs in real-time black and white, which is useful during color training. The video comes from the color-adaptive recognition phase of the hardware.

Many of the hardware resources of the 68332 are available for embedded applications, including digital I/O lines, several TPU (timer coprocessor) lines, a bus with software-definable chip selects, and one synchronous and two asynchronous serial ports.

THE CONTEST

Every year, the AAAI (American Association for Artificial Intelligence) holds robot competitions at its annual

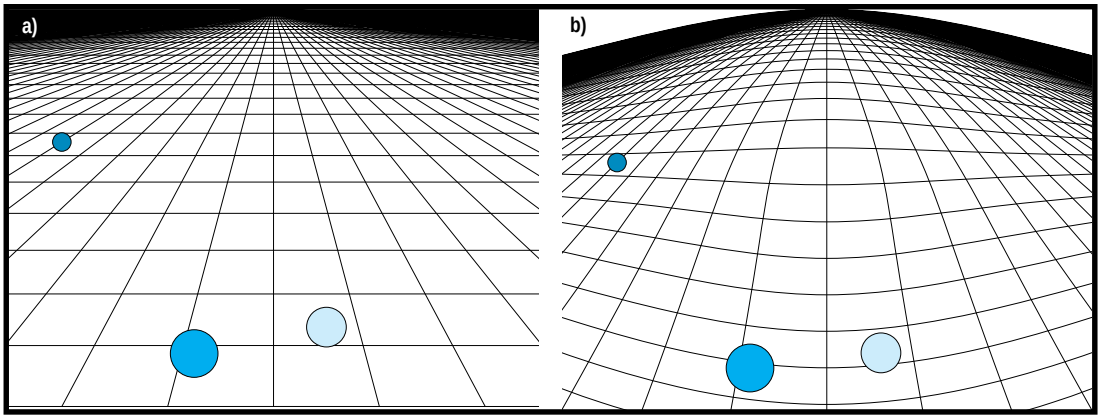


Figure 2a—*With an ideal video camera, M1 would see the world much like this.* ***b***—*The world, as seen through M1's actual camera, has fish-eye distortion, which is typical of cameras that show a wide field of view.*

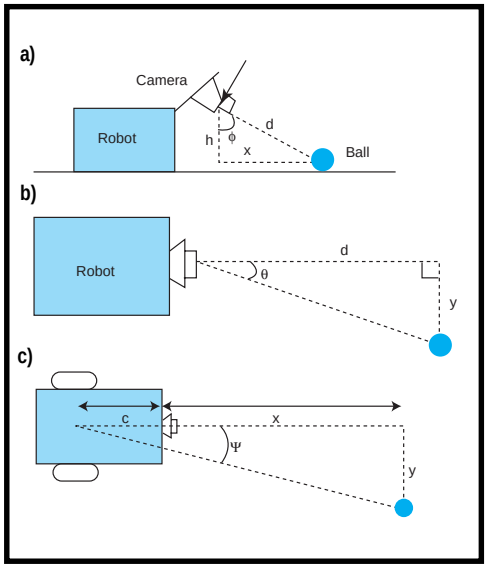


Figure 3a–c—*M1 can determine a ball's distance and angle relative to the robot from the ball's location in the camera image, assuming the ball touches the floor.*

conference. In 1996, the contest was for an autonomous robot to collect 10 tennis balls and 2 quickly and randomly moving, self-powered squiggle balls and deliver them to a holding pen within 15 min.

At the time of the conference, we had already been manufacturing the Cognachrome for a while and saw this contest as an excellent way to put our ideas (and our board) to the test. We outfitted a general-purpose robot called M1 with a Cognachrome and a gripper and wrote software for it to catch and carry tennis balls.

CONTROLLING M1

M1's base uses a two-wheel "wheelchair" drive. Connected to each wheel by a toothed belt and sprocket combination is a NEMA 23 frame stepper motor rated at 6.0 V and 1.0 A. There is also a third, unpowered caster wheel.

An SGS-Thomson L297/L298 stepper motor bipolar chopper drive powers the NEMA 23 motors with current limited to 300 mA. Steep accelerations and decelerations are possible even at this low current setting. Three NiCd batteries supply 30 V to the chopper drive, which gives the step rate an upper limit in excess of 6000 half-steps per second.

Stepper motors enable very accurate drive control, and this particular implementation appears to result in good performance and low power consumption at both low and high speeds. At 30 V, the batteries have a storage capacity of 600 mAh.

The "step now" input on each stepper motor driver is connected to a TPU line, so we can control the speed of each motor independently and precisely.

One problem with stepper motors is that they stall if you try to run them too fast or accelerate or decelerate too quickly. M1 has no stall-detection sensors, but it does have stall-recovery software. If the control software decides that no progress has been made for long enough, it will slow to a stop, which recovers from the stall.

Of course, it's much better to avoid stalls in the first place. M1 contains a software layer between the high-level control and the motors for this purpose. When the high-level control software commands a speed, this low-level software smoothly accelerates or decelerates to this speed, within the motors' safety parameters.

Internally, two sets of commands control wheel speed. One set controls the left and right wheel speeds independently. The other commands control the angular and forward velocities.

The mapping between the two command sets is simple. If *a* is angular velocity, *f* is forward velocity, and *l* and *r* are the left and right wheel velocities, respectively, then the mapping is:

$$l = jf - ka$$
$$r = jf + ka$$

where *j* and *k* are constants that depend on the units used.

GATHERING THE BALLS

M1's basic operation during the contest is to find a ball, grab it, carry it to the goal, and drop it in. And as we mentioned, there are two kinds of balls in the contest—standard tennis

Listing 1—*M1 iterates through all detected objects that are the color of the tennis balls or squiggle balls. After determining ball position, M1 can decide which to pursue.*

```
int find_targets(Target *dest, Vstate *vs, enum target_type type){
  Blob *blobs[MAX_TARGETS];
  int n_blobs;
  int i;
  int n_targets;
  /* Find largest MAX_TARGETS blobs on current channel */
  n_blobs= blobs_select_largest_n(blobs, frame_eb(vs), MAX_TARGETS);
  for (i= 0, n_targets= 0; i< n_blobs; i++){
    /* Find center of gravity of current blob */
    blob_find_cg(blobs[i]);
    /* If blob is too far left, too small, or over horizon, skip */
    if (blobs[i]->xcg < *p_track_min_col ||
        blobs[i]->area < ((*p_diam_thresh) * (*p_diam_thresh)) ||
        !camera_to_world(blobs[i]->xcg, blobs[i]->ycg,
        m1_camera_pos, &(dest[n_targets].x), &(dest[n_targets].y)))
      continue;
    dest[n_targets].area= blobs[i]->area;
    /* Set angle and distance, given robot-relative x-y coordinates */
    rect_to_polar(dest[n_targets].x, dest[n_targets].y,
                  &dest[n_targets].angle, &dest[n_targets].dist);
    /* Use perceived size and computed dist. to compute real size */
    dest[n_targets].size=int_sqrt(blobs[i]->area) *
         [dest[n_targets].dist / 1000;]
    /* If actual size is too small, ignore object */
    if (dest[n_targets].size < *p_size_thresh){
      continue;}
    dest[n_targets].type= type;
    dest[n_targets].score= 0;
    dest[n_targets].age= 0;
    n_targets++; }
  for (i= n_targets; i< MAX_TARGETS+1; i++){
    dest[i].size= 0;}
  return n_targets;}
```

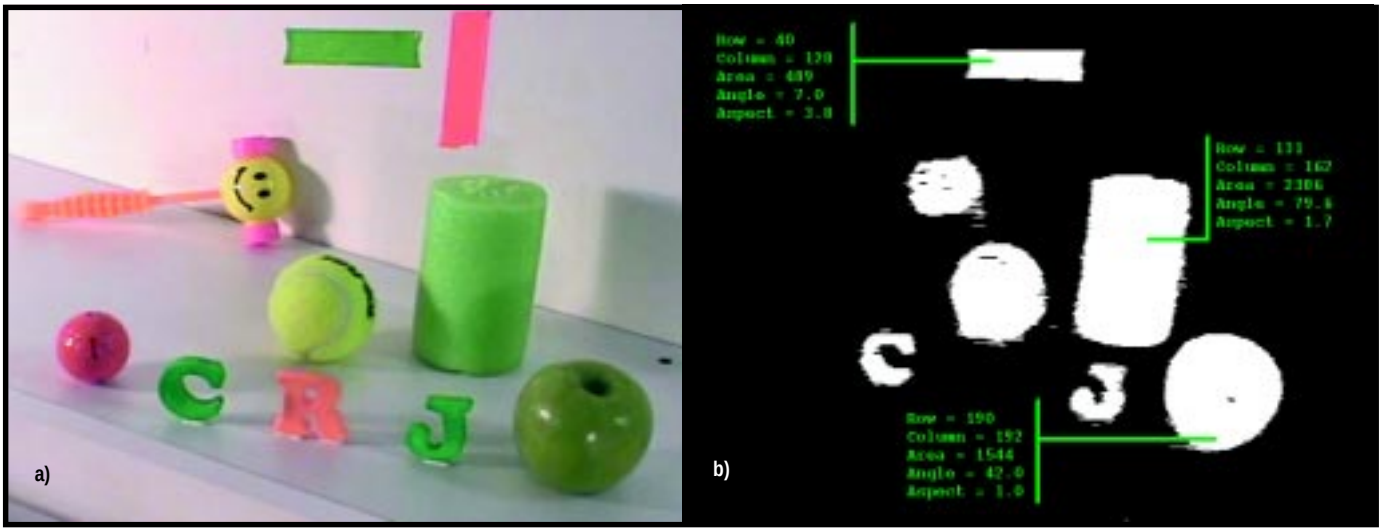


Photo 1—*There are two sides to every story, or in this case, two ways to view the same picture. These brightly colored objects (**a**) change appearance when seen by the Cognachrome vision system (**b**). The system assembles contiguous pixels of interest into blobs and calculates various statistics, such as centroid, area, elongation or aspect ratio, and direction of orientation. Notice that Cognachrome only sees colors it was trained on.*

balls and motorized, randomly-moving squiggle balls.

Of course, the real challenge is the squiggle balls. The squiggle balls are almost as big as M1's gripper and they move almost as quickly as M1, so the robot control must be very accurate to turn toward the squiggle ball and run it down. Once we can do that, it isn't hard to handle tennis balls as well.

The contest rules also require us to announce when M1 is chasing a squiggle ball. This is done via a small piezoelectric speaker that beeps when M1 sees a squiggle ball. Once the announcement is made, M1 chases the squiggle ball until it catches it or doesn't see it any more. Tennis balls are ignored to make it clear to the judge that M1 really is distinguishing tennis balls and squiggle balls.

LOCATING THE BALL

The tennis balls are greenish yellow, and the squiggle balls we use are red. We train two of the Cognachrome's three color channels on these colors.

When the Cognachrome detects the ball color, it reports the *x* and *y* coordinates of the ball's center, relative to the camera's field of view. These numbers need to be translated into a rotation angle and distance.

The control software uses the angle to decide how to turn and uses the distance to determine the ball's location relative to the gripper. The function definition is shown in Listing 1.

The translation involves some interesting math. It is handled in two stages, which we will present in reverse order.

PERSPECTIVE TRANSLATION

First, imagine that the camera gives us a nice perspective image, something like Figure 2a. From an image like this, we can compute the distance and angle to the ball straightforwardly (assuming that the ball is on the ground).

In Figure 3a, ϕ is a straightforward function of the *y* coordinate of the ball and the tilt of the camera. (M1 can tilt the camera with a stepper motor, so the ball-location routine has to compensate for this.)

We know *h*—it's the location of the camera above the ground (~8″ for M1) minus the height of the center of the ball. To find *x*, we use:

$$x = h \tan(\phi)$$

We also want to know *d*:

$$d = \sqrt{x^2 + h^2}$$

Figure 3b looks like a top view, but it's actually looking from a little bit forward of top (compare it to Figure 3c). We are looking from the direction labeled with the arrow in Figure 3a.

Here, θ is a straightforward function of the *x* coordinate of the ball, and *d* was computed above. To find *y*, use:

$$y = d \tan(\phi)$$

Now we have *x*, which is the distance from the camera forward to the ball, and *y*, the distance left or right to the ball. What we want is the angle to turn and to head toward the ball (when the turning point is centered between the two drive wheels) and the distance to the ball.

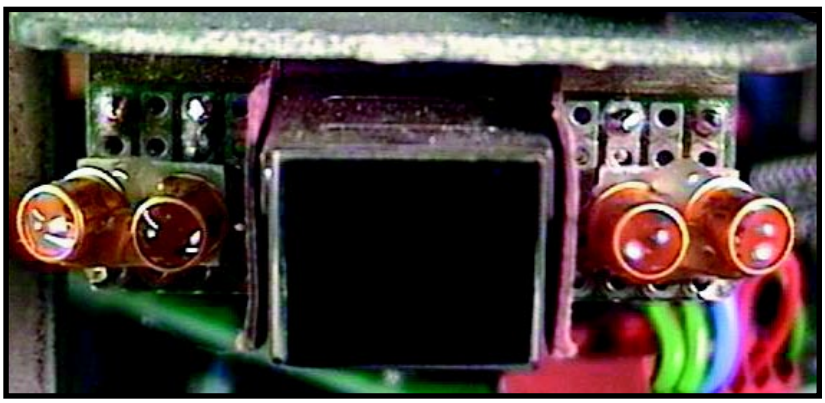


Photo 2—*The left half of M1's infrared sensor array is composed of a Sharp GP1U52X infrared detector sandwiched between four infrared LEDs.*

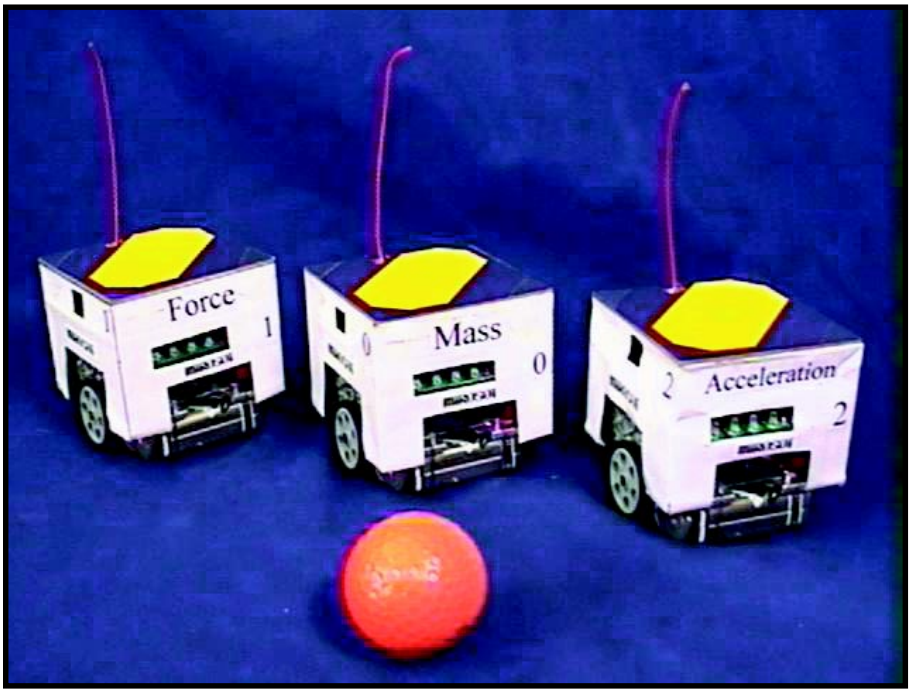


Photo 3—*Force, Mass, and Acceleration are the three members on Newton Labs' world-champion robot soccer team. (Mass is the goalie.) In the foreground is the soccer ball (actually an orange golf ball.)*

So finally, the distance to the ball is:

$$\sqrt{(x + c)^2 + y^2}$$

and the angle is:

$$\psi = \tan^{-1}\left(\frac{y}{x + c}\right)$$

FISH EYES

Unfortunately, this isn't the whole story. Remember our assumption that the camera gives a nice perspective image? It doesn't.

To get the right compromise between peripheral sensing and seeing in the distance, we use a camera with about a 100° field of view. This results is a serious fish-eye effect—the nice, straight lines in Figure 2a look curved when viewed through the camera (Figure 2b).

We need to find a mapping that undoes this fish-eye distortion before applying the above mathematics. Basically, this mapping should use the *x* and *y* coordinates from the vision data to compute the θ and ϕ angles suitable for use in the equations.

When we implemented this code, we tried to derive the correct mathematical form of the mapping. We soon decided it would be easier to approximate it. We used polynomial equations because they're easy to deal with.

A linear mapping like $\theta = axy + bx + cy + d$ is not sufficient. We need a slightly more complicated polynomial—a bivariate quadratic. We suspected this type would be adequate because the curved lines produced by the fish-eye effect look vaguely like parabolas.

However, if it had not been adequate, we had to be prepared to move on to higher-degree polynomials or find a different form of equation. Therefore, we wanted to find values for the coefficients *a*–*r* in:

$$\theta = ax^2y^2 + bx^2y + cx^2 + dxy^2 + exy + fx + gy^2 + hy + i$$

$$\phi = jx^2y^2 + kx^2y + lx^2 + mxy + nxy^2 + ox + py^2 + qy + r$$

First, we needed some experimental data. We set up a vision target as far as possible from M1 and had the robot pivot from side to side and rotate the camera up and down in a predefined grid pattern.

At each location, we recorded the *x* and *y* positions of the target according to the vision system as well as the vertical and horizontal angles, based on how far the robot pivoted and rotated its camera.

We then needed to find the values for *a*–*r* that would minimize the error between the computed θ and ϕ values and the measured values. Although this task may sound daunting, we simply plugged all the values into an Excel spreadsheet, calculated the differences for each sample, and summed the squares of the differences.

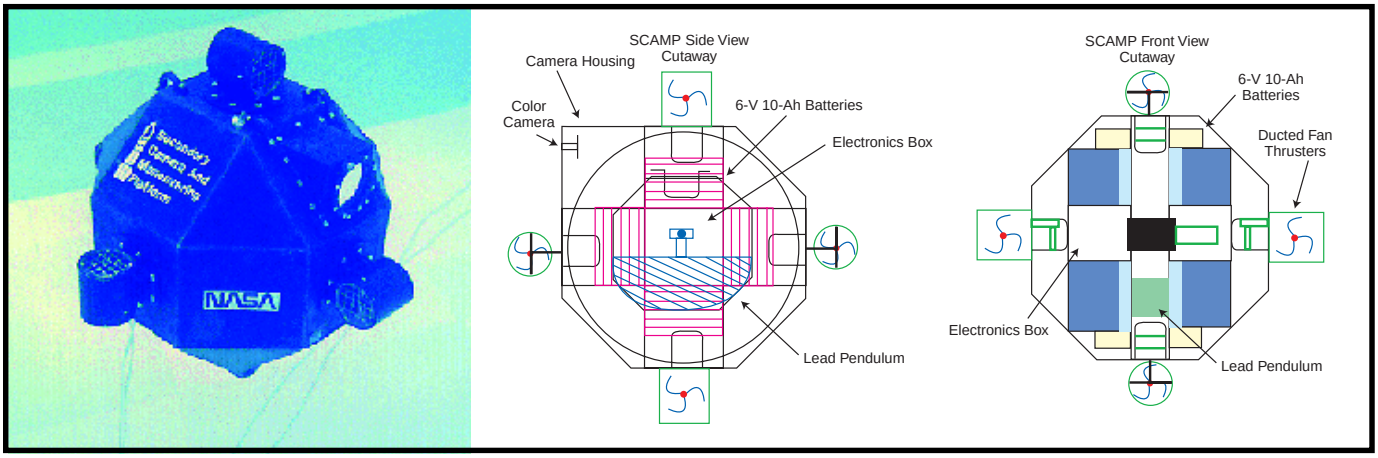
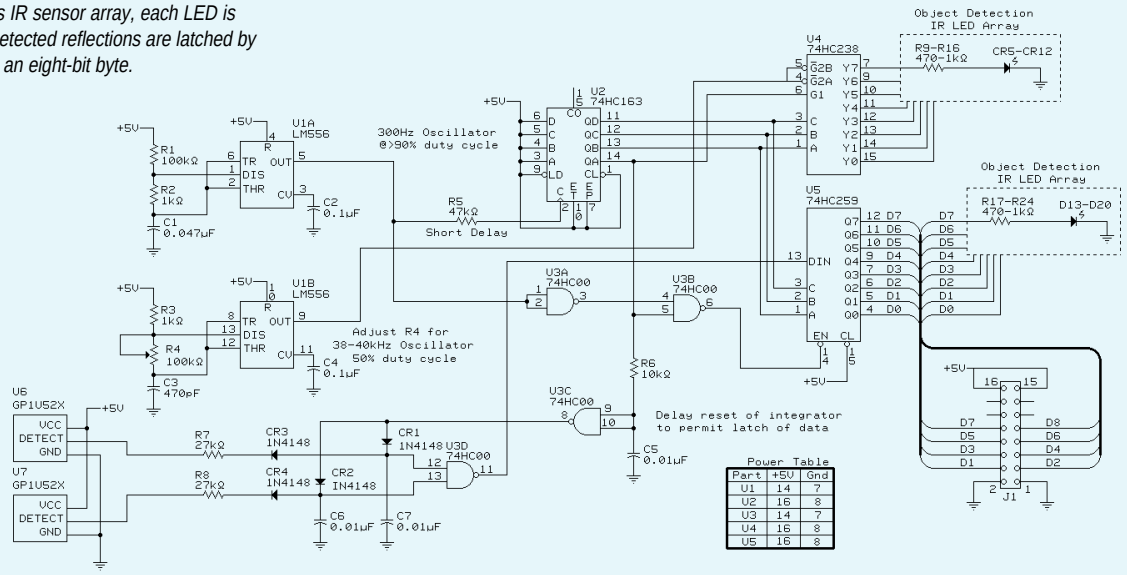


Photo 4—*The SCAMP underwater robot, created by the University of Maryland's Space Systems Laboratory, is designed to simulate zero-gravity spacecraft motion. With the use of a Cognachrome vision system, SCAMP can autonomously perform simulated docking and station-keeping maneuvers.*

Figure 4—*In M1's IR sensor array, each LED is fired in turn and detected reflections are latched by the 74HC259 into an eight-bit byte.*



We let Excel's Solver find values for *a*–*r* that minimized this error sum. (The Solver isn't installed by default, so you might need to find your installation CD to add this feature.)

GRABBING THE BALL

Thanks to all the above math, M1 now knows the distances and angles to all the balls in view. The next task is to choose a ball and chase it down, where the chase is a lot easier for a tennis ball than a squiggle ball.

We already mentioned that once M1 starts to track a squiggle ball, it continues tracking it until the ball is within reach or disappears from view. Also, once M1 starts tracking a tennis ball, it does not switch to a squiggle ball unless the squiggle ball is about half as far away as the tennis ball.

We use the following algorithm to head for a ball, given an angular offset, ψ. Here, a is the required angular velocity, e is an angular error term, and f is the required forward velocity:

$$a = k_1\psi$$
$$e = k_2\psi^2$$
$$f = sk_3(1 - e)$$

(If $e > 1$, then we set the speed to zero, rather than moving backward.)

The constant s has different values for tennis balls and squiggle balls. M1 moves as quickly as possible to chase squiggle balls. But, it's more cautious when approaching tennis balls because they have a tendency to bounce off the gripper and roll away.

We quickly found that this algorithm doesn't work all the time. If a ball is within reach but to the left or right of the gripper, M1 pivots toward the ball and the gripper then knocks the ball away. So, we use a different algorithm for this situation—M1 simply backs up.

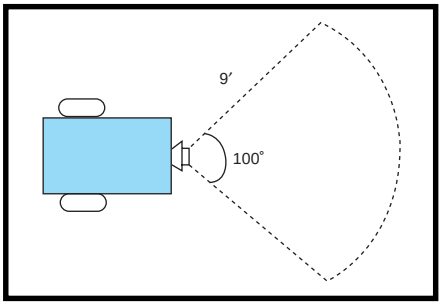


Figure 5—*M1's camera can detect balls in a pie-shaped region.*

SEARCHING FOR BALLS

If M1 cannot see any balls at the moment, it has to find some. When M1 starts looking for balls, it first spins around to try to see one. However, that doesn't always work. The repository might be in the way. Or, if the balls are too far away, M1 can't see them.

If a simple spin doesn't find any balls, M1 goes searching. It heads forward until it finds a wall (unless it finds a ball), and then it follows the wall.

M1 follows the wall using an infrared obstacle detector. The code drives two banks of four infrared LEDs one at a time, each modulated at 40 kHz.

Two standard Sharp GP1U52X infrared remote-control reception modules detect reflections. The 74HC163/74HC238 combination fires each LED in turn, and the 'HC259 latches detected reflections. This system provides reliable obstacle detection in the 8–12″ range. Figure 4 shows the schematic, and Photo 2 shows the IR sensors.

The system provides only yes/no information about obstacles in the eight directions around the front half of the robot. However, M1 can crudely estimate distance to large obstacles (e.g., walls) via patterns in the reflections. The more adjacent directions with detected reflections, the closer the obstacle probably is.

SEARCHING THE ENTIRE REGION

Unfortunately, even with M1's wide 100° field-of-view camera (illustrated in Figure 5), wall following doesn't cover the whole room. It just sees the areas depicted in Figure 6a.

So, every few seconds, M1 stops, spins 180° away from the wall, then spins back to the original direction. This sequence enables it to see into the center of the room from various points along the wall (see Figure 6b). M1 does this once every 8 s in the first 8 min. of the round, and once every 4 s in the final 2 min.

DUMPING THE BALL

Once M1 has the ball, it must dump it in the repository. Contestants can build their own ball repository, and we marked ours with a blue rectangle.

To keep the squiggle balls inside, we put a 1″ lip in the repository's gate, so the gripper has to lift the balls over this lip to deposit them. However, M1 would go after the balls in the repository if it could see inside, so we covered the gate with a black curtain and put the blue marker on the curtain.

Much like searching for a ball, M1 starts its search for the repository by spinning. If it doesn't see the blue marker, it heads for a wall and follows it around.

When it sees the blue marker, M1 heads straight for the repository. It begins to slow down and slows down even more as it nears the repository.

The size of the blue marking is used to estimate the distance. We can't use the vertical angle to the marker, like we do for the balls, because the rectangle is at roughly the same height as the camera.

Two bump sensors on the bottom of the gripper tell M1 when it reaches the lip of the gate. They also enable M1 to line up with the gate before it drops the ball.

When one bump sensor is engaged, M1 turns off the wheel on that side and turns on the wheel on the other side. This action causes M1 to line up with the gate. M1 drops the ball when both bump sensors are engaged.

Fashion collided with function when one of the spectators wore a bright blue shirt in a preliminary round. The shirt was approximately the same color as the gate marker, and the spectator stood next to the 3′ wall surrounding the playing field. When M1 picked up a ball, it often headed straight for the spectator rather than the repository. Not able to reach the repository, the robot acted quite confused.

We fixed this problem by computing the vertical angle to the gate marker (using the same algorithm, including

fish-eye correction, as for the balls) and ignoring blobs above a certain angle.

We had already compensated for a similar problem by ignoring red and yellow blobs above the horizon. Otherwise, M1 might have viewed certain spectators as huge squiggle balls.

M1's control software is surprisingly complex given its seemingly simple task. While describing the entire software system is outside the scope of this article, the state diagram in Figure 7 gives you the overall picture.

GAME DAY

We worked through the night before the contest, tweaking the algorithms. Early the morning of the contest, M1 completed three perfect runs. We called it complete then and froze the code.

To add a little extra stress, the competition was being recorded for Scientific American Frontiers with its host, Alan Alda, standing next to the arena giving commentary. M1 got off to a strong start, capturing the first tennis ball in mere seconds. It continued roaming around the arena and quickly collected almost all the tennis balls and both squiggle balls.

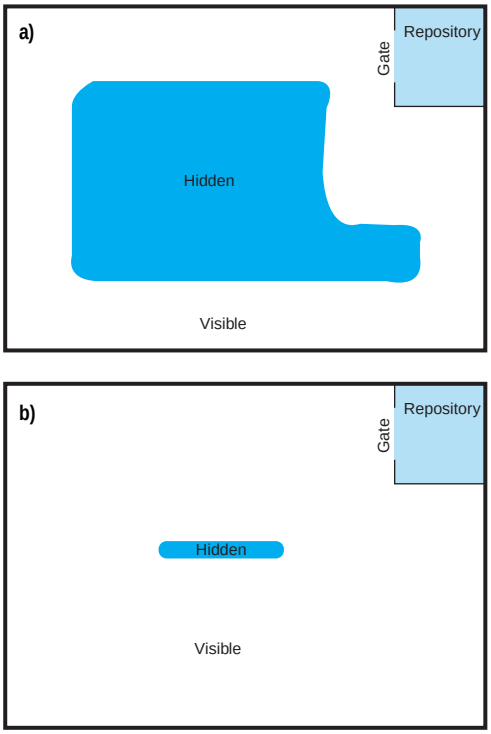


Figure 6a—*If M1 searches the arena simply by following the wall, it misses most of the middle.* ***b***—*If M1 periodically pivots to face the middle of the room while following the wall, it searches a much larger region.*

However, the final tennis ball remained elusive. It was in the exact center of the arena, and remained just slightly beyond the visual reach of M1 as it scanned the arena. Clearly, to collect this ball, M1 had to turn and look into the center of the arena from exactly the right point along the wall.

The spectators grew tense as M1 followed the wall around and around, turning and looking toward the ball but not quite seeing it. Time was running out.

Finally, on its third time around the arena, M1 looked into the center from just the right spot, collected the ball, and sped to the repository with seconds to spare, earning a perfect score. The crowd erupted into cheers and applause. And, the Newton Labs team began to breathe again.

ROBOTS SEE THE WORLD—AND BEYOND

The Cognachrome vision system serves a wide range of applications, from research uses like catching balls, autonomous spacecraft docking, and automated acquisition of cargo by helicopter, to industrial uses like sorting fruit and inspecting upholstery.

We entered (and won) the first and second International Micro Robot World Cup Soccer Tournaments (MIROSOT) held by KAIST in Taejon, Korea, in November of 1996 and June of 1997. We used the Cognachrome system to track our three robots' position and orientation, the soccer ball, and the three opposing robots. Our team is pictured in Photo 3.

Because of the robots' small size (each fits into a 7.5-cm cube), we opted for a single vision system connected to a camera over the field instead of a system in each robot. (In fact, the rules of the contest require markings on the top of the robot that encourage this. All but one of the teams used a single camera above the playing field.)

Professor Jean-Jacques Slotine and Dr. Kenneth Salisbury of MIT incorporated two Cognachrome systems into their adaptive robot arm—the WAM (whole arm manipulator). Using two-dimensional

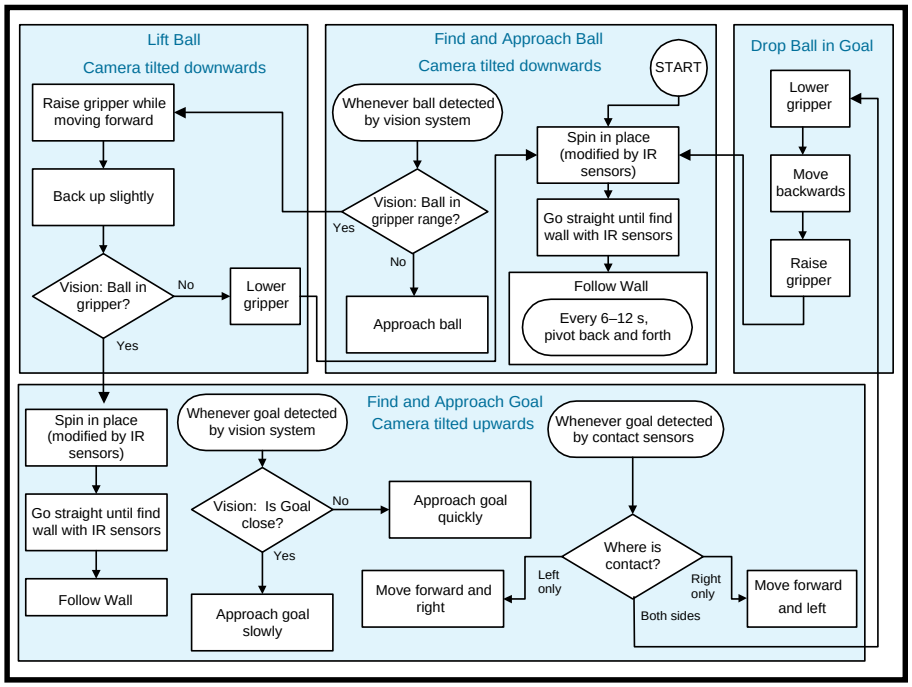


Figure 7—*This diagram gives a simplified view of M1's different behavior states and how they are activated. Not shown here are special time-out behaviors designed to get M1 unstuck if it hasn't made progress for some time.*

stereo data from a pair of Cognachrome systems, the WAM controller predicts the three-dimensional trajectory of a ball in flight and controls the arm to quickly intercept and catch the ball.

The University of Maryland Space Systems Laboratory and the Kiss Institute for Practical Robotics have simulated autonomous spacecraft docking in a neutral buoyancy tank for inclusion on UMD's Ranger space vehicle. Using a composite target of three brightly-colored objects designed by Dr. David Miller, the spacecraft (shown in Photo 4) knows its distance and orientation and can servo to arbitrary positions around the target.

So, although participating in the AAAI contest was exciting, what it really demonstrated is that robots can perform interesting tasks using a simple, fast vision system.

Bill Bailey is a design engineer at Newton Research Labs, a company that develops high-performance, low-cost machine vision hardware and software for industrial and robotic applications. The original developer of the M1 robot base, Bill has over 25 years of expertise covering analog and digital electronics, software, and mechanical design. He and the other authors may be reached via vision@newtonlabs.com.

Jon Damon Reese received a B.A. in computer science from Rice University and an M.S. and Ph.D. in information and computer science from the University of California, Irvine. His research interests over the years have included artificial intelligence, programming languages, software engineering, and software safety. Jon serves as a software and applications specialist at Newton Research Labs.

Randy Sargent is the president of Newton Research Labs. He received a B.S. in computer science at MIT, and an M.S. in media arts and sciences from the MIT Media Laboratory. Formerly holding titles of Lecturer and Research Specialist at MIT, he is one of the founders of the MIT LEGO Robot Contest (a.k.a. 6.270), now in its ninth year.

Carl Witty is a research scientist at Newton Research Labs. He received his B.S. and M.S. in computer science from Stanford University and MIT, respectively. A member of the winning team in the 1991 international ACM Programming Contest, his interests include robots, science fiction and fantasy, mathematics, and formal methods for software engineering.

Anne Wright is the senior design engineer at Newton Research Labs. The primary architect of the Cognachrome vision system, Anne received her B.S. and M.Eng. in computer science from MIT. She also helped lead and develop technology for the MIT LEGO Robot Contest from 1992 to 1994.

SOFTWARE

Design documentation for M1, including the full source code, is available at www.newtonlabs.com/cc.html#ml.

A video tape highlighting applications discussed in the paper (e.g., M1 picking up tennis balls, the soccer robots performing, etc.) can be ordered from www.newtonlabs.com/cc.html#video.

REFERENCES

www.newtonlabs.com/cc.html
www.mirosot.org
www.ai.mit.edu/projects/wam/index.html#S2.2
www.pbs.org/saf/8_resources/83_transcript_705.html

SOURCES

Cognachrome vision system
Newton Research Labs
Robotics Systems and Software
14813 NE 13th St.
Bellevue, WA 98007
(425) 643-6218
Fax: (425) 643-6447
www.newtonlabs.com

GP1U52X
Sharp Electronics Corp.
Microelectronics Group
5700 NW Pacific Rim Blvd., Ste. 20
Camas, WA 98607
(360) 834-2500
Fax: (360) 834-8903

L297/L298
SGS-Thomson
55 Old Bedford Rd.
Lincoln, MA 01773
(617) 259-0300
Fax: (617) 259-4421

I R S

401 Very Useful
402 Moderately Useful
403 Not Useful