Easy Image Processing: Camera Interfacing for Robotics

By: Daniel Herrington

SINCE CHILDHOOD, DANIEL HAS INCORPORATED HIGH-TECH TOYS IN HIS ROBOTICS PROJECTS. THUS, IT'S NO SURPRISE THAT HE USED A GAME BOY CAMERA IN ONE OF HIS RECENT DESIGNS. NOW, HE'LL SHOW YOU HOW TO USE THE CAMERA TO ENHANCE THE NAVIGATION SYSTEM ON YOUR OWN MOBILE ROBOT.

Reprinted from
Circuit Cellar #151

I've been interested in robotics since I was a little boy. Back when I was in junior high school, I built a mobile robot platform out of the drive portion of a child's motorized car and a Commodore VIC-20. Over the years, advances in technology have made experimenting with robotics more enjoyable. The Game Boy Camera is an ingenious addition to the Game Boy Color game unit that came out a couple of years ago. It's a black-and-white digital camera with a resolution of 123 x 128 pixels, operating at a rate of one to 30 frames per second.

The camera's original price was between $40 and $50, making it somewhat cost-prohibitive for hobbyists. However, because the product was recently discontinued, I found some on eBay selling for between $10 and $20. The reduced price makes the camera an attractive solution if you're interested in robot navigation. It's even less costly than a single Polaroid sonar module ($30 to $50) and in the same ballpark as reflective infrared sensors ($5 to $15).

The sensor inside the camera is a highly integrated CMOS array with built-in edge enhancement and extraction. Built-in image processing enables a microcontroller to perform object detection and tracking, provided certain assumptions about the background of the image are valid.

Atmel's AT90S8515 microcontroller has an external memory interface bus that allows you to easily connect an SRAM IC. The on-chip hardware UART makes it possible to output processed data without consuming precious processing resources, and the timers enable it to control hobby servo motors without much work. In addition, the AVR series of microcontrollers has a high-speed RISC architecture (e.g., most instructions take one or two clock cycles) that makes timing calculations simple. In short, the flexibility of the AVR 8-bit microcontrollers makes attaching special-purpose peripherals like the Game Boy Camera a breeze.

Figure 1 is a block diagram of a camera interface and object-tracking system. As you can see, the camera is controlled via some of the microcontroller's general-purpose I/O pins. The analog output of the camera is attached to the external A/D converter. The servos are connected to two more pins of the microcontroller, and the RS-232 converter conditions the UART's signals for connection to the outside world.

Figure 2 details the interface circuit. A few notes might be helpful here. The A/D converter needs to be fast enough to read out a pixel value every 2 µs if the maximum frame rate is desired. This means the sample frequency of the ADC must be at least 500 kHz. A speed requirement like this rules out the use of the built-in ADC on most microcontrollers (e.g., the AT90S8535).

For my circuit, I settled on the Analog Devices AD7822, which doesn't have the added complication of pipelining that many of the newer ADCs seem to have. Also, you don't need the RS-232 converter IC if the circuit will be interfaced directly to another microcontroller's UART. I used a 7.3728 MHz crystal to achieve compatibility with standard data rates. A speed of 115,200 bps is the maximum speed that a standard PC serial port supports, so I programmed the microcontroller to work at that speed.

The completed prototype circuit board is shown in Photo 1. It's a simple wire-wrapped board with all through-hole components except the ADC. The potentiometer is used to adjust the reference voltage for the microcontroller's on-chip analog comparator. The comparator is used in place of the ADC for the hobby robot depicted in Photo 2. I found that the optimum setting for the reference voltage for my test environment was about 4.66 V.

Photography 101

The Game Boy Camera uses Mitsubishi's M64282FP CMOS image sensor. This sensor is special because Mitsubishi designed some image processing capabilities right into the CMOS array itself.

Software isn't required to perform edge detection, enhancement, or extraction from the image. To get an edge-extracted image, simply put the sensor in the appropriate mode and then read out the resulting analog pixel voltages for the entire image. After this image is read out, it can be further manipulated by the microcontroller. Photo 3 shows the M64282FP chip inside the camera.

I didn't use the camera's cartridge base in these experiments. After the cartridge is opened, the ball can be disconnected from the cartridge. I disconnected the purple cable from the cartridge and soldered a standard 0.1" double-wide header on the end. This allowed a 0.050" ribbon cable to be used for a long-distance connection, although I don't recommend exceeding 1 foot or so.

By the way, I cut a hole in the back of the ball so that the cable could exit in a place where it doesn't interfere with mounting the camera on a pan-tilt base. You may download the pinout of the original connector that's coming out of the ball from the Circuit Cellar ftp site. The numbers refer to the corresponding pin numbers on the M64282FP chip.

To simplify the interfacing of the assembly code and speed things up, I turned the camera sensor board in

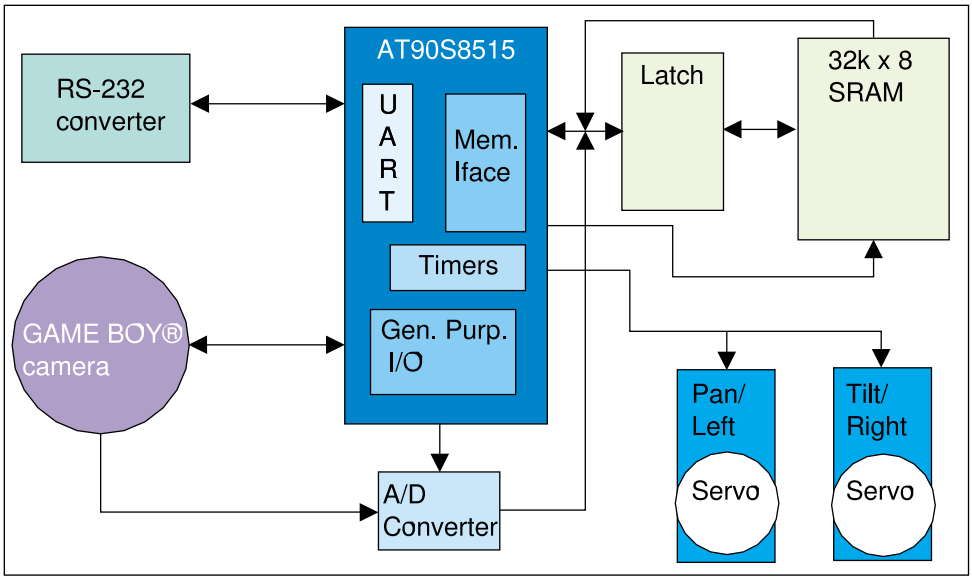

Figure 1: You can use the two servos for either panning/tilting a camera head or driving the left and right wheels of an autonomous robot. For the latter, the servos must be modified to allow for continuous rotation. This servo hack is common for hobby robots.

the camera ball upside-down. This ensures that the first pixels to be read from the camera are those corresponding to the bottom-right corner of the image instead of top-left. Furthermore, this makes the calculation of the nearest object faster, because the image is read out serially from the sensor.

The procedure for programming and using the M64282FP is straightforward. First, load the registers in the M64282FP using a synchronous serial protocol that is similar to other two-wire synchronous serial interfaces. The microcontroller generates the XCK, SIN (data), and LOAD signals for loading all of the registers in the camera IC.

Next, give the Start command. After the exposure is finished, the camera IC will return a READ signal. When the READ signal becomes active, read 15,744 pixels (123 x 128) worth of analog data on the VOUT pin synchronously with the XCK signal that the microcontroller generates. After all of the image data has been output, the READ signal becomes inactive, and the camera automatically starts another exposure.

In the first part of the programming process, you can set the camera's registers for a normal (positive) or inverted (negative) image, an edge-enhanced image, or an edge-extracted image. Register settings also control the camera's exposure time, the output voltage offset level, and the gain of the video signal (i.e., how much it varies from the output voltage offset).

The maximum frequency for the camera's XCK input is 500 kHz (T = 2 µs). With a microcontroller crystal frequency of 7.3728 MHz (T = 135.6336 ns), the time for each half-period of XCK is:

$$\frac{1\mu s}{135.6336ns}$$

or approximately eight microcontroller clock cycles. I tuned the timing of the assembly code by adding NOP instructions where appropriate.

It's interesting to see how different register settings affect the image output from the camera. Table 1 shows two settings: Normal mode and Edge mode. These settings were derived by experimentation and may need to be adjusted for any given environment.

I set up a test area with various medium- and high-contrast colored objects on a light-colored floor (see Photo 4). The top-center image frames within Photo 5a on page 43 show what the Game Boy Camera images look like with specific register settings: Photo 5a is a normal image; Photo 5b is a negative image; and Photo 5c is an edge-extracted image. I used this type of image for object tracking. Note that the light-colored objects (red and orange in this case) don't show up as well in the edge-extracted image. You can increase or decrease the exposure setting to allow these low-contrast objects to be seen in Edge mode.

Obstructed Views

Ian Horswill's 1993 dissertation, "Specialization of Perceptual Processes," details some of his research concerning image processing for robotics. Horswill outlines various assumptions that may be made by a robotic image-processing system for detecting obstacles in a given environment.

After the edges have been extracted from an image, the height of the first edge from the bottom of the image

Register	Address	Normal mode	Edge mode
0	000	0x80	0x3F
1	001	0xD6	0xD6
2	010	0x06	0x18
3	011	0x00	0x00
4	100	0x01	0x01
5	101	0x00	0x00
6	110	0x01	0x01
7	111	0x07	0xF3

Table 1: When you're switching from Normal to Edge mode, it's important to remember the M64282FP registers 0, 2, and 7.

can be determined. Let's assume the camera is mounted somewhere on the front of the robot, several inches or feet above the floor. If the camera is aimed forward and down, and if the floor doesn't have visible edges (i.e., the carpet color is constant, and there are no high-contrast seams or changes of color), then the only edges should be the obstacles on the floor or a floor-to-wall transition in front of the robot.

If the robot moves near a wall, and if there is enough of a contrast between the wall and floor, an edge will be detected at that location in the image. Using this technique, the robot can tell how far away the edge of the obstacle is by its height in the image.

If the image is divided into thirds (i.e., left third, center third, and right third), then the lowest edge in each third of the image gives the robot the distance it can move in that direction. Then, the robot can turn and move toward the farthest direction to avoid the closer obstacles. This "greatest third" approach is well suited for corridor following, because the greatest third of the image is most likely the direction of the corridor.

The camera takes care of extracting the edges from the image, but the microcontroller must perform any additional processing. For instance, if you want to know an object's distance (or depth) from the robot, then you'll need an algorithm to post-process the image and reduce the information down to a depth table. The index to this table could represent a given x location, or column. The entry at each index within the table could be written with the row of the lowest edge pixel in a given column. This algorithm is implemented in the microcontroller as shown in Listing 1.

Now, I'll explain the operation of the depth-finding code. Starting at the bottom-right corner of the image, count the number of pixels vertically until one is reached that surpasses a predefined threshold value. Put the row number (i.e., depth) of that pixel in a table, step to the next column to the left, and repeat the process.

When the depths of all of the columns of the image have been recorded, send that information out of the UART. A graphical representation of the depth map for

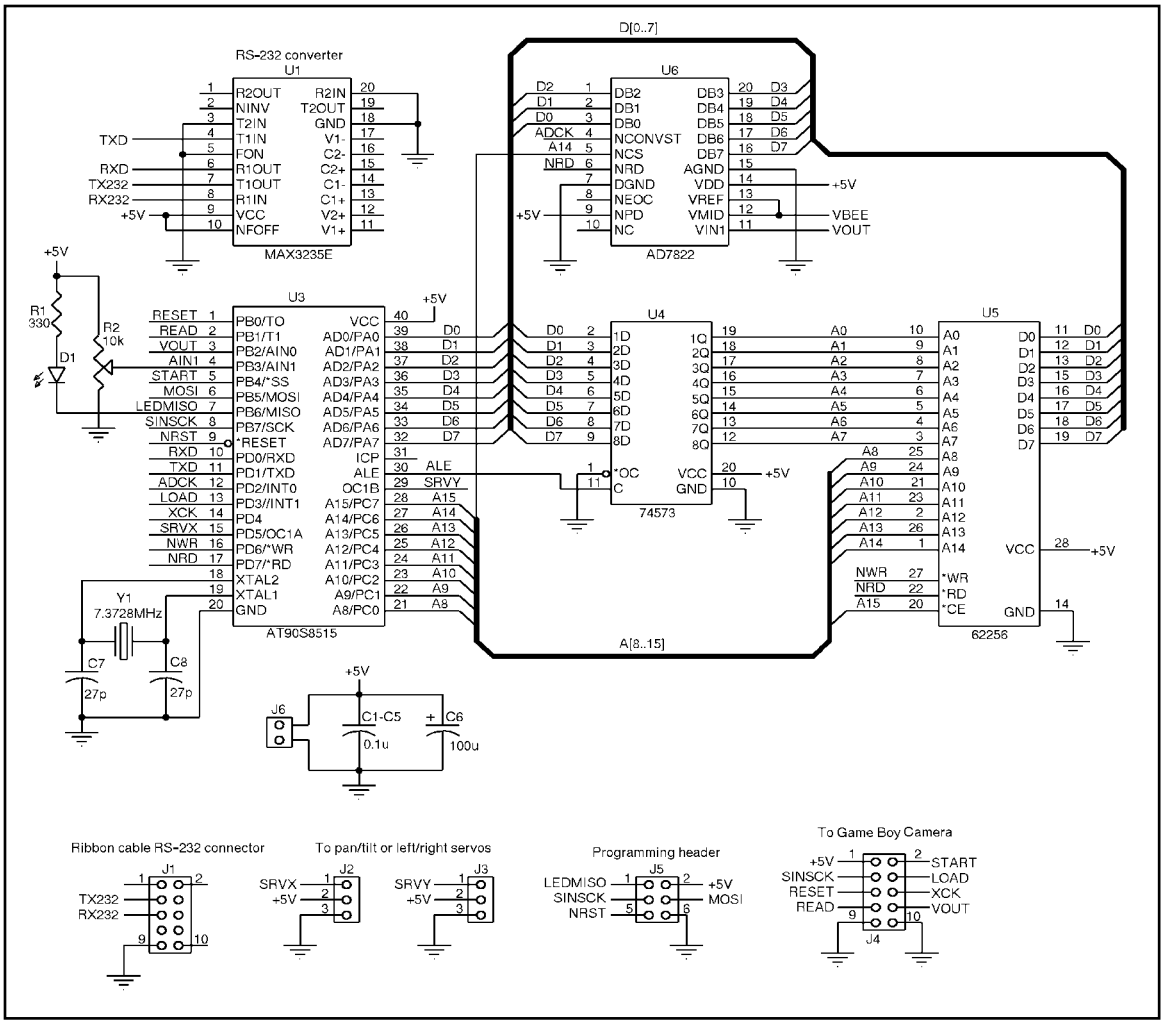

Figure 2: With this processor and interface circuitry and connector design, you can enhance your own robotics applications with the Game Boy Camera.

the test objects in Photo 4 is shown in the Depth/Nearest/Track frame in Photo 5c. The groups of shaded columns are areas that include objects.

Point and Shoot

The opposite of obstacle avoidance is object tracking. The camera can be panned and tilted in response to an object found in the image. Assuming the lowest edge in the image above some brightness threshold is an object to be tracked, the microcontroller can command servo motors to pan and tilt the camera to move the object to the center of the image. This requires some intelligent control of the motors to prevent a slow response, overshooting, and ringing.

To perform object tracking, the microcontroller searches the image in RAM from the bottom up. When it finds the first edge brighter than a given threshold value, it marks the x and y locations and measures the horizontal and vertical distance of this edge from the center of the image. Then, the microcontroller issues a corrective movement command to the servos, which respond by redirecting the camera until the object is centered in the view. Listing 2 shows how it's done.

Photos 6a and b show the pan-tilt servo mechanism. The pan servo is directly mounted to the tilt servo's control surface. Note that the sub-micro-sized servos in my photos allow for a compact installation.

The performance of the pan-tilt camera head is adequate for tracking small objects, provided that the object isn't moving faster than about 1 foot per second at a distance of 4 feet from the camera. This means you can roll a ping-pong ball at a moderate speed across the floor roughly 4 feet in front of the camera, and the camera will lock on and track the ball until the servos reach their limit.

The system won't notice a bouncing ball. Using a large ball (e.g., a basketball) causes different edges (left and right) of the ball to be detected, and the camera oscillates between the two nearest edges if they alternate in intensity or y position.

One helpful piece of equipment for tuning the system is a laser pointer. With a laser pointer, a bright point can

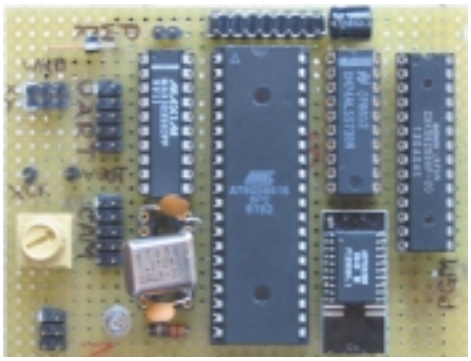

Photo 1: The prototype circuit board is small enough to fit inside a mobile hobby robot.

Photo 2: I've provided you with four views of the mobile robot. The servo that controls the tilt angle of the camera is for future expansion.

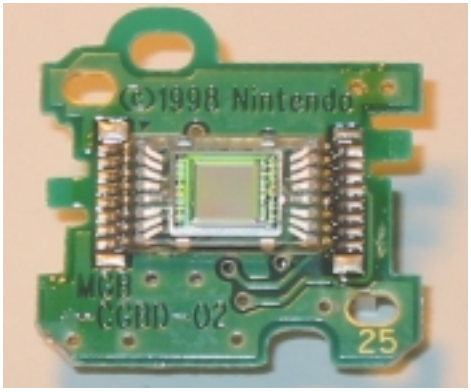

Photo 3: There's only one chip and two capacitors on the circuit board in the camera ball. Take a look at the clear-packaged M64282FP IC.

be moved from one location to another almost instantaneously. Using one, you can observe the reaction of the servos.

The gain of the system is set too high if the servos overshoot and "ring" (i.e., oscillate until they come to a rest) at the new location. The gain should be set by increasing or decreasing the divisor of the error amount so that the correction amount causes the servos to overshoot slightly without ringing. Look for the constant, TRACKDIV, in the assembly code for more information.

Incidentally, the entire image-capture/ process/output sequence takes roughly 11 ms, yielding a frame rate of about nine frames per second. The pan-tilt camera head is only able to track objects while they are within the servo's travel limits. If a subject is lost because it moved too far to the left or right, the camera will wait for up to 30 frames for the object to reappear before it returns to the center position to look for another object. You can overcome this limitation by giving the camera the ability to pan all the way around. To do this, mount it on a mobile robot.

Photo 4: The perfect testing area may be closer than you think. I placed the test objects on my kitchen floor, which has a practically constant texture and surface.

Follow that Subject!

You can apply the theory used for panning and tilting a camera to controlling a mobile robot. The camera itself is stationary with respect to the robot base. Instead of controlling the camera directly, the microcontroller commands the robot base to move in a certain direction at a specified speed. This arrangement allows the mobile robot to find high-contrast objects and approach them.

The robot is able to search for objects by spiraling out in an ever-widening arc until an object is within view. When an object is detected, the robot faces the object and speeds toward it. The robot slows down gradually until it stops with the object located in the center of the camera image. As long as the object doesn't move too fast, the robot will continue to rotate, move forward, or move backward to keep the object in the center of the image. Photo 2 shows the prototype of the mobile robot.

Instead of using the external ADC, the microcontroller uses the on-chip analog comparator to detect bright pixels. In addition, the RAM isn't used, because the only information the robotrequires is the nearest object location. To determine the location of the nearest object, the pixels om the camera are read and processed on the fly.

Because the information isn't sent from the UART, it's pointless to have the RS-232 converter on the board. Therefore, you can construct a reduced circuit for the mobile robot. The only components that you need for the mobile robot's microcontroller board are the microcontroller itself, the crystal, a potentiometer (which is used for adjusting the analog comparator reference voltage), and a few capacitors.

Regarding the performance of the mobile robot, the camera does an excellent job sensing high-contrast objects within its view; however, it is inadequate for detecting the majority of medium- and low-contrast obstacles. In the real world, you should always use multiple layers of sensors. It is a good idea to try supplementing the camera with a bumper panel or whiskers.

Developing

The software for this project consists of two parts. The first section consists of the assembly code in the AVR microcontroller that talks to the camera, RAM, and serial interface. The second part includes the C program for a Linux-based PC that reads and writes camera registers. In addition, this portion captures images and obtains depth information, nearest object information, or object-tracking locations.

I assembled the microcontroller code with the tavrasm AVR assembler and programmed the microcontroller with sp12. I wrote the C program for the host PC using the Simple DirectMedia Layer (SDL) library, which is a public cross-platform graphics and multimedia-programming library.

SDL includes routines for drawing pixels on the X-windows display. A user-contributed extension, Bfont, supplies the routines for writing text to the window. Refer

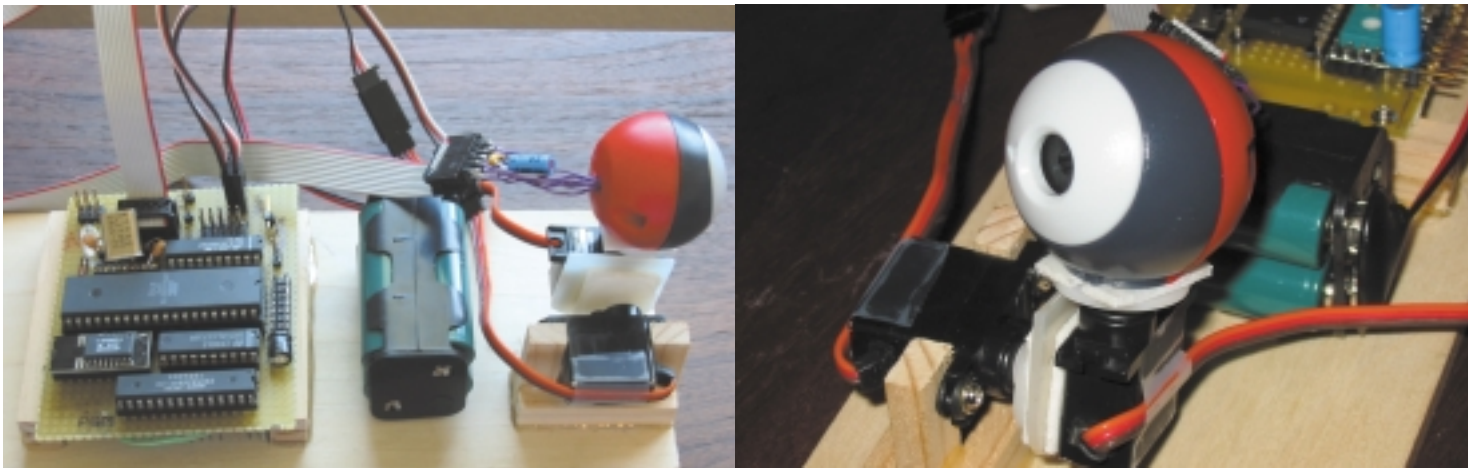

Photo 6a: Four AA NiCd or NiMH batteries power the circuit board used for the pan-tilt head. b—These servos have a quick transit time. It takes only 0.09 s to rotate the control surface through 60°! This fast response time keeps the servos from being the limiting factor in the system's reaction time.

Photo 5a

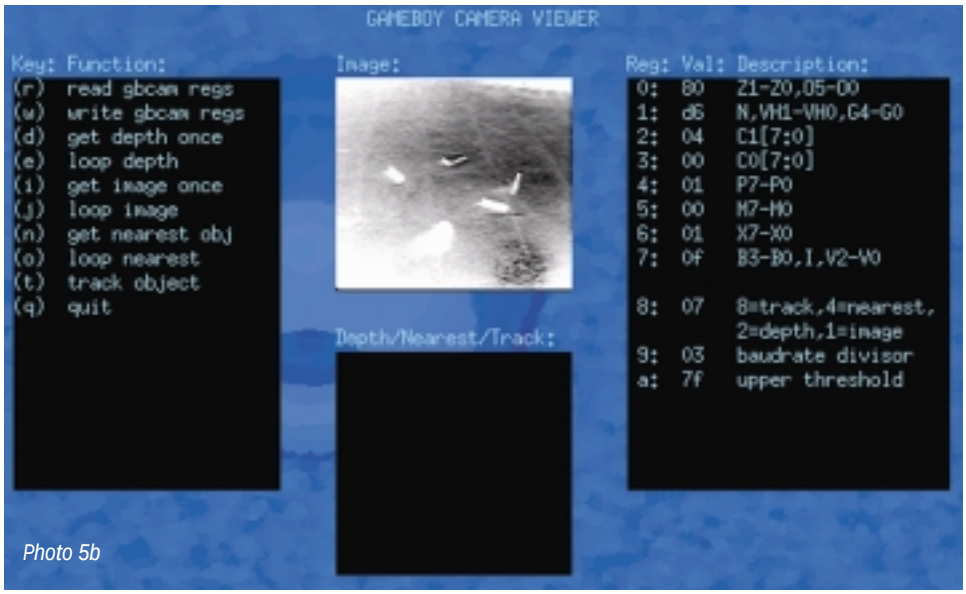

Photo 5b

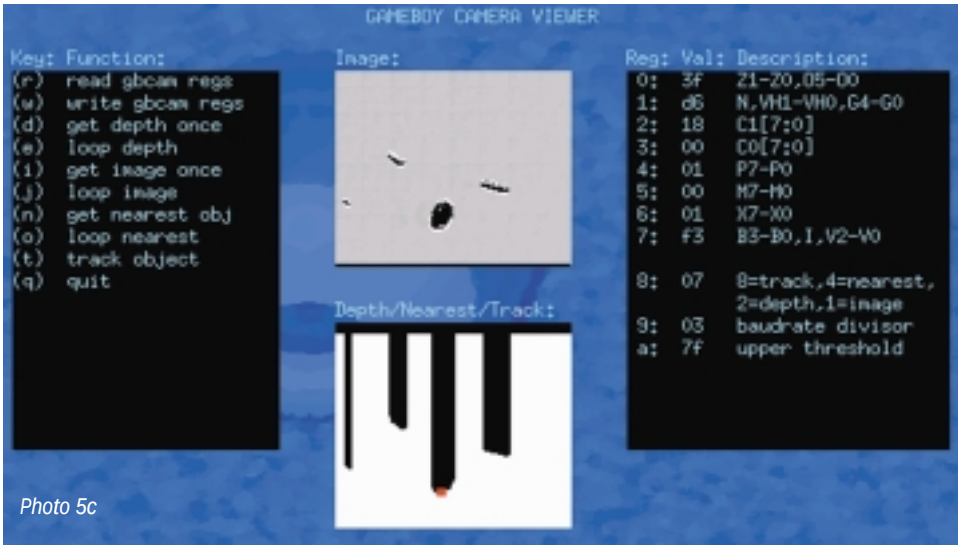

Photo 5c

to Photo 5 for screen shots of the user interface. Although I completed all of the development for this project in Linux, it should be fairly easy to port to a Windows environment.

There are two versions of the AVR assembly code. The pan-tilt program controls the pan-tilt camera head, and the chase program controls the mobile robot. The pan-tilt code has the added overhead of communicating with the PC through the UART. In addition, it requires the use of the pan-tilt C program on the PC. This adds significant delays to the operation of the pan-tilt camera.

If you want to get the maximum performance out of the pan-tilt camera, you can hard-code the register settings and remove the calls to the UART communication routines. The chase code is already optimized for the fastest possible frame rate. Either of the AVR programs can be modified to allow for higher-level behavior by adding calls to new routines in the main loops.

Enlargements

One way to enhance the output of the tracking system would be to mount a low-power laser pointer on the pan-tilt head. Then, as long as the field of view is kept away from humans and highly reflective surfaces (in addition to other appropriate safety precautions), the robot could alternate between strobing the laser pointer and tracking an object. This would let you see a pulsating red dot, signaling where the robot is actually focusing its attention.

You could also use the laser and a cylindrical lens to generate a flat, fanned-out beam (visible as a horizontal line on any objects in its path). This line would be visible only at points in the field of view where an object is obstructing the beam (T. Ward, "Scope Out the Laser Range Finder," Circuit Cellar 46). Therefore, the camera would have a much greater ability to detect low-contrast objects against the floor background. Additionally, this approach would also help in finding low-contrast walls.

If a high-speed parallel connection to a PC were used instead of the serial one, then the PC could attempt to perform pattern recognition in real time, comparing the edge-extracted image to an edge database of known objects.

A similar improvement would be to use two cameras in tandem without pan servos. A microcontroller could send both edge-extracted images via a high-speed parallel connection to a PC, allowing the PC to compare the two images and attempt to find matching patterns of edges. Assuming that the pan angles of the two cameras were fixed, the matching pattern locations would then allow the PC to determine the distance to certain objects based on the distance between the two cameras and the difference in pan angles.