A Second Generation Low Cost Embedded Color Vision System

Anthony Rowe
Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
agr@ece.cmu.edu

Charles Rosenberg
Google, Inc.
Mountain View, CA 94043
chuck@google.com

Illah Nourbakhsh
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
illah@ri.cmu.edu

Abstract

In this paper we describe a low cost embedded vision system, the CMUcam2. The CMUcam2 is the second generation of the CMUcam system and attempts to overcome the shortcomings of the original system as well as improve upon and add to its functionality. The goal of the system is to provide simple vision capabilities to small embedded systems in the form of an intelligent sensor. The system utilizes a low cost CMOS color camera module, a frame buffer chip and all image data is processed by a low cost microcontroller. The system includes the original functionality of color blob tracking, but improves upon it with tracking speeds of up to 50 frames per second. New functionality includes frame differencing, edge detection, and color histogramming. Other improvements were also made to facilitate communication with slower speed processors, as is often the case in a variety of robotics applications, including miniature robotics, hobby robotics and aerial robots.

1 Introduction

In many applications relatively simple computer vision algorithms have proved themselves to be extremely useful, [3], [5], [6], [8], [13], [15]. It has been challenging to implement even these simple computer vision algorithms in embedded systems which utilize small microcontrollers because traditional vision system implementations require a camera, a frame grabber, and a high speed processor. The goal of the system developed here is to provide this functionality in a small low power package and provide a low bandwidth data stream to a host processor. This has become possible because of the availability of low cost CMOS color camera modules and high speed, low cost microcontrollers.

In our previous system, the CMUcam, we attempted to use the minimal amount of hardware which could implement basic functionality. This system processed pixels "on the fly" using very little memory which placed certain limitations on the image processing algorithms. The CMUcam2 extends the CMUcam's functionality by including a frame buffer chip which decouples the pixel capture and processing operations. This in combination with a more capable microcontroller allowes for more complex and flexible processing.

The system described in this paper has been implemented and is fully functional. A fully assembled version of the system is available from multiple commercial vendors for a cost of $199. [10]

2 System Architecture

Our vision system is designed to provide high-level information that is extracted from a camera image and communicated to an external processor. For example the external processor in a mobile robot system could configure the vision system's color tracking mode to stream the centroid location of a particular bounded set of RGB values. The vision system would process the data in real time and output high-level information. In the following sections we describe the details of our hardware and software system and compare our new system to the previous version.

2.1 Hardware

The hardware for our original system consisted of a three chip design: a CMOS camera chip, a microcontroller, and a simple RS232 level shifter. Our new design adds a fourth chip, a frame buffer. In this design the camera is connected directly to the frame buffer chip. The processor triggers a frame grab and the frame buffer chip stores the data streamed from the camera without further intervention by the processor. Once the data is in the frame buffer, the processor can synchronously clock the data out of the frame buffer as needed. The microcontroller processes the data stream and extracts user defined information that is sent to the outside world via an asynchronous serial interface implemented in software. The complete vision system is 2.20" x 2.20" and less than 2" deep with the camera module and

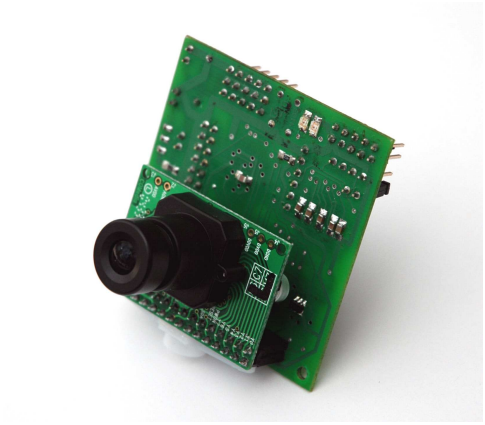

Figure 1: The controller board mated with the CMOS sensor.

lens attached, see Figures 1 and 2. The system operates at 5 volts and draws 170 milliamperes of current while fully active.

The image input to the system is provided by an Omnivision OV6620 or OV7620 CMOS camera on a chip.[9] The CMOS camera is mounted on a carrier board which includes a lens and supporting passive components. By itself, the board is free running and will output a stream of 8 bit RGB or YCrCb color pixels. Synchronization signals, including a pixel clock, are then used to read out data and indicate new frames and horizontal lines. When used in concjuntion with our system, the OV6620 supports resolutions of up to 352 x 288 with a maximum refresh rate of 50 frames per second (fps) and the OV7620 supports resolutions of up to 240 x 160 with a maximum refresh rate of 60 fps. Camera parameters such as color saturation, brightness, contrast, white balance, exposure time, gain and output modes are programmable using a standard serial I2C interface. An analog monochrome output exists that can be used for external monitoring of the image. Unlike the original CMUcam which used a nonstandard frame rate, making this output difficult to use, the current system generates a standard video signal.

The main microcontroller used to process the video data is a Ubicom SX52 operating at 75 MHz.[14] The SX52 is a RISC processor and operates at 75 MIPS including single cycle I/O operations. It has a 4096 word flash programmable EEPROM and 262 bytes of SRAM. Although the computational capabilities of this processor are the same as the SX28 used in the original CMUcam, the SX52 has twice the ROM and RAM, as well as additional I/O ports which permitted us to include extra functionality. The processor has very few hardware peripherals, but has fast and deterministic interrupts and flexible I/O ports that allow software to emulate standard hardware peripherals in a virtual manner. Using these "virtual peripherals", we implemented a serial UART port, standard hobby servo PWM output ports, LED status functions and a push button input. It is also possible, using a pass-through PC104 style connector, to join multiple vision boards on a single camera bus. This allows for parallel processing of the image data in what we call slave mode. Using this "slave mode" two microprocessors can be attached to the output of a single CMOS camera, allowing two different image operations to be performed in a fully synchronized fashion. The frame buffer is the AL422B manufactured by Averlogic. It contains 384K bytes of storage in a FIFO configuration. [1] The FIFO nature of the memory means that the processor is limited to sequential accesses to the image buffer. An internal address counter keeps track of the current location in the image. The image buffer memory is dual ported allowing the camera to simultaneously write to the memory while the processor is reading from it.

We chose the Averlogic FIFO for three main reasons. First, the interface is very simple. The FIFO has very few control pins, which is critical when dealing with a microcontroller with limited I/O capabilities. Secondly, the FIFO can now manage the data transfer from the camera to the frame buffer at the level of a single frame instead of a single pixel. This allows the transfer to take place much faster than the processor would be able to handle on its own. This decoupling also allows us to do more complicated processing on each pixel. A third benefit of the image buffering is that the camera can operate at full frame rate. Running the camera at full frame rate yields better automatic gain and exposure performance due to the factory default tuning of the system. An unexpected benefit from the frame buffer was an increase in the processing speed per frame. This happened because in the original CMUcam the pixel timing had to be such that the timing conditions were met for the worst case path through the code. In the new design the processor accesses pixels as it needs them and therefore timing is no longer constrained to the worst case. The disadvantage is the lack of random access to the data, the additional cost of the component as well as the extra power consumption. Even though we are limited to sequential data access much like in the original CMUcam, we now also have the ability to reset the read pointer of the FIFO without overwriting data. This allows multiple processing functions to be called on the same buffered image.

In many embedded applications power consumption is an important factor and it is often advantageous to be able to shutdown systems when they are not needed. To facilitate power savings we provide support to sleep the processor, the oscillator, and the camera module. This is advantageous in security or battery powered sensing environments when the camera is used at very low duty cycles. During normal operation, the camera consumes 850mW of power. When in

sleep mode, this is reduced to 505mW, and in a deep sleep mode where the oscillator is disabled, this can be further reduced to 420mW.

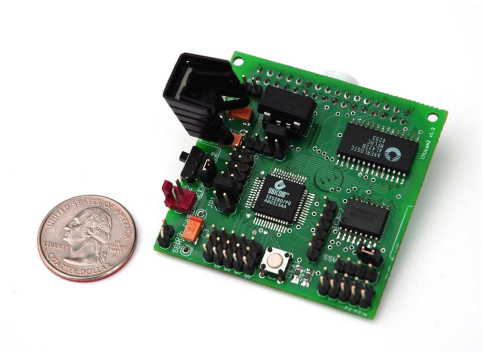

Figure 2: Detail of the assembled microcontroller board, 2.20” × 2.20”. Visible are the microcontroller in the middle, the frame buffer in the upper right, the clock oscillator in the upper left and the RS232 level shifter in the lower right.

2.2 Firmware

The main challenges of the original CMUcam design were limited RAM and ROM in the processor as well as the strict code timing requirements necessitated by processing the data "on the fly." The stringent timing requirements were greatly ameliorated by the addition of the frame buffer, because the processor no longer has a strict deadline required for each pixel capture. However, we still paid close attention to efficiency in order to maximize image processing speed. Even with the increased RAM and ROM available in our system, space was still extremely limited. In our never ending desire to maximize the functionality of the system, we utilized 99.87% of the ROM. And because the frame buffer did not support random access, state information needed to be stored in the processor’s RAM which is much too small (262 bytes) for even a single row of pixel data.

All firmware for the vision board was written in C and compiled using the ByteCraft SXC compiler. When compiled the current firmware requires 4087 words of ROM and at some points utilizes all but 2 bytes of the SX52’s RAM.

2.2.1 Color Blob Tracking

The color blob tracking algorithm allows the user to enter a minimum and maximum bound for each of either the three RGB or YCrCb channel values, depending on how the camera is configured. Each pixel in the buffer is compared against the user specified bounds. The coordinates of the pixels that fall within the color bounds are compared against previously stored coordinates to generate a bounding box. This simple method requires that the CMUcam2 store little global information about the image. The stored data includes the upper left x1, y1 coordinate and the lower right x2, y2 coordinate that enclose pixels which satisfy the color bounds. We also count how many pixels actually fall within the color boundaries. Once the entire frame has been processed, some additional post processing operations are performed. In particular, a scaled ratio between the total sum of pixels within the color boundaries and the actual area calculated by the bounding box is computed. This value can then be used as a confidence measure indicating whether there is only one compact object being tracked which fills the bounding box or multiple small detections. The system also accumulates the x and y positions of each detected pixel. These accumulated sums are then divided by the total number of detected pixels to calculate the centroid of the tracked object. Once it has received an entire frame of data, the system can return the x,y coordinates of the centroid, the four coordinates of color bounding box, the number of detected pixels as well as a confidence value for the object tracked. The camera can be put into "line mode" which in turn returns a binary image of the pixels being tracked. As shown in figure 3, another variation on line mode returns the minimum and maximum tracked pixel as well as a centroid for each row. This mode of operation is particularly useful for applications such as line following.

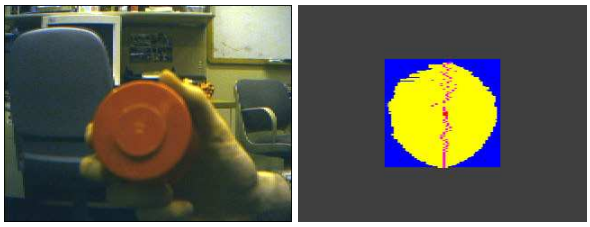

Figure 3: The left image shows a frame dump of a red object. The right image shows the same image being processed with line mode enabled. The yellow lines show the horizontal starting and ending positions of the object, while the magenta dots show the centroid of each line.

2.2.2 Color Statistics

The vision system also includes a color statistic acquisition function. This function keeps a running sum of the individual color channel components. Upon completion of the frame, it divides these accumulated values by the total number of pixels returning the mean color. It also returns an approximation of the absolute deviation from the mean of each color. This can be used to quantify the spread of the

colors about the mean. When used in conjunction with other features such as windowing, described following, the color statistics can be used for determining the color of an object at a specific location in the field of view.

In order to provide a richer set of color related data, there is also a histogram function. The histogram operation allows the user to determine the distribution of a color across the image. The histogram contains 28 bins each holding the number of pixels that occurred within that bin's range of color values. So bin 0 on channel 0 would contain the number of red pixels that were between 16 and 23 in value.

2.2.3 Frame Differencing

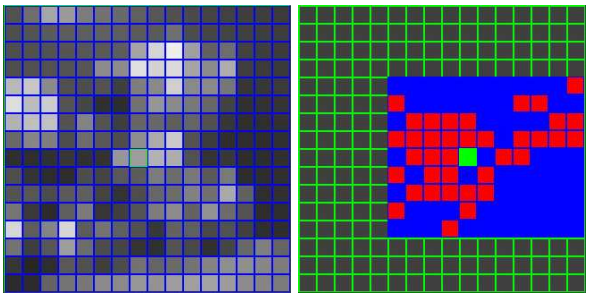

Figure 4: The left image shows an example of a reference frame. The right image shows a bitmap of a hand being moved in front of the scene.

The CMUcam2 incorporates the ability to identify differences between the current image and a reference frame. This is useful when attempting to locate motion given a fixed camera position. When in frame differencing mode, the CMUcam2 tessellates the image into an 8 by 8 grid. All of the pixels for a user defined channel in each square are averaged creating a reference low resolution image. Figure 4 shows an example of such an image. When new frames are captured, this same calculation is performed except now the values are compared with the stored values. The user can define a threshold delta value that can be used to increase or decrease sensitivity. The outputs of the frame differencing commands are nearly identical to that of color tracking. Additionally, a motion bounding box, centroid and "line mode" binary bitmap are available. It is also possible to export the detected delta values, the original averaged values or the current averaged values. Figure 4 shows the bitmap version of a hand moving in front of the camera. In this image, virtual high resolution frame differencing is enabled. In high resolution mode, the camera will operate at 16x16 instead of 8x8. The captured image is still stored internally at 8x8. The extra resolution is achieved by doing 4 smaller comparisons against each internally stored pixel. This generally yields good results when the background image is relatively smooth, or has a uniform color.

2.2.4 Image Processing and Camera Settings

Another group of functions define how the data is formatted and performs minor adjustments on the overall performance of the system. These functions include a noise filter, an interface transfer flow control setting and a command to modify the CMOS camera's internal image settings. The noise filter mode makes any color tracking algorithm more robust by requiring a detection to include a user defined number of multiple horizontally adjacent pixels in the specified color range. This added robustness however can cause small objects not to be detected. The interface flow control settings allow configuration of the serial data entering and leaving the system. The default mode uses visible ASCII characters and continuously streams data as each frame is processed. Selecting "poll mode" instead causes each function to only return one packet of data and then return to its idle state. Another setting allows for raw binary bytes to be transferred instead of visible ASCII text and suppresses or enables different synchronization bytes reducing overhead. It is also possible to set an output data mask so that only user defined values in a packet are returned. This type of flexibility was lacking in the original CMUcam and has proven to be very helpful when dealing with less powerful microcontrollers. The camera settings control command allows the user to change the frame rate, toggle white balance, toggle gain, switch between RGB and YUV modes or set any of the camera's internal register values. [9]

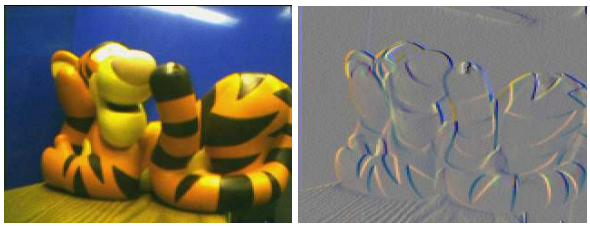

Figure 5: Sample frame grab from the OV7620 camera module with the corresponding frame while pixel differencing is enabled.

The ability to enter what is known as pixel differencing mode allows the entire standard set of image processing routines to operate on what is similar to a horizontal edge detected image. As seen in figure 5, by operating on the difference between the current and previous horizontal pixel, high frequency components in each color channel are emphasized. In combination with the histogram mode, or even the mean statistics data, this can indicate the level of texture on an objects surface.

Along with the basic algorithmic functions, there is a set of modifying parameters that allow for more advanced image processing. The first of these parameters is the ability to arbitrarily set the window size and location that the user wishes to process. This allows data to be captured in an isolated region of the camera's view. The window bounding box can be easily changed between frames allowing for more localized analysis of the environment. New to the CMUcam2, the image can also be down sampled allowing for more rapid processing. These features can be used in conjunction with the frame buffer making it possible to operate on the same image in the FIFO multiple times before reloading a new image.

2.2.5 Demo Mode and Additional Features

To accommodate systems where extra actuators may be necessary, the camera board has the internal ability to control up to five standard hobby servos. Using this ability, the vision system can operate in a stand-alone "demo mode". Upon startup, if the board detects that the button is depressed, it will automatically enter demo mode. While the camera adjusts to the current light level, a status LED blinks. When the button is pressed again, the camera acquires the color of the first object it sees upon power up and tracks it using a simple feedback loop driving two servos on the horizontal and vertical axis. The positions of the servos can be set or read manually, even while demo mode is active. The servo output ports can also be used as TTL digital outputs instead of generating a servo PWM signal. For debugging purposes there are two firmware controlled LEDs, one of which can be set to illuminate when the sensor detects an object. These LEDs and the button can be manually accessed via user commands.

2.3 Interface

The vision system by default uses a human readable ASCII communication protocol that allows the user to communicate with it interactively from a serial terminal program. As described previously, a less verbose mode can be enabled to reduce serial port traffic when communicating directly with a computer or another microcontroller. When communicating with a computer, the system can also dump an entire raw image via the serial port. This can be used for diagnostic purposes or higher resolution processing. At the current default frame rate and at the maximum window size of 255 x 176 a full frame dump takes about 10 seconds. With down sampling and selecting individual channels, it is possible to send multiple frames per second.

By default, all communication with the board takes place at 115.2 kilobaud, but jumpers can be used to select values ranging from 1.2 to 115.2 kilobaud speeds. The CMUcam2 contains 47 commands each of which are followed by their associated parameters. Below is a small example of a typical set of command transactions used to track the mean RGB color located in the middle of the image. The vision system output is shown in italics:

CMUcam2 v1.01c6
:cr 18 44 17 2 19 32
ACK
:vw 30 60 50 80
ACK
:pm 1
ACK
:gm
ACK
S 150 20 30 5 2 6
:pm 0
ACK
:sw 0 0 80 143
ACK
:tc 145 18 24 155 22 36
ACK
M 50 80 38 82 53 128 35 98
M 52 81 38 82 53 128 35 98
M 51 80 38 84 53 128 35 98

The first command "CR" sets the CMOS camera registers. The numbers that follow are register addresses and parameters, for example 18 44 tells the camera to set the color mode to RGB and turn on automatic white balance. These values are outlined in the vision system documentation [9] as well as the CMOS camera documentation.[8] The "SW" command sets the coordinates of the virtual window to be processed. In this case x1=30 y1=60 x2=50 y2=80,which selects the center of the image assuming the ov6620 camera operating in low resolution mode. The "PM 1" command turns on the poll mode of the camera so that any additional functions will only return a single line and not stream data. The "GM" command then asks the camera to get the mean value in the current window. The resulting "S" packet shows the R_{mean}, G_{mean}, B_{mean}, followed by the $R_{deviation}$, $G_{deviation}$ and $B_{deviation}$. Next, poll mode is disabled and the window is set back to encompass the entire image. The final "TC" command actually calls the track color function, passing in a minimum RGB value of (145,18,24) and a maximum value of(155,22,36). This value is the mean value previously returned from the camera now padded by its deviation. The returned M packets appear at up to 50 frames per second and show the centroid x, y coordinates the x1, y1, x2, y2 bounding box coordinates, the number of detected pixels and the confidence value of the object being tracked: M x y x1 y1 x2 y2 pixels confidence.

To aid in system integration, we have also developed a new Java based graphical user interface (GUI) that allows the user to interface with the camera from a Unix or Windows based PC. This GUI has been greatly expanded upon

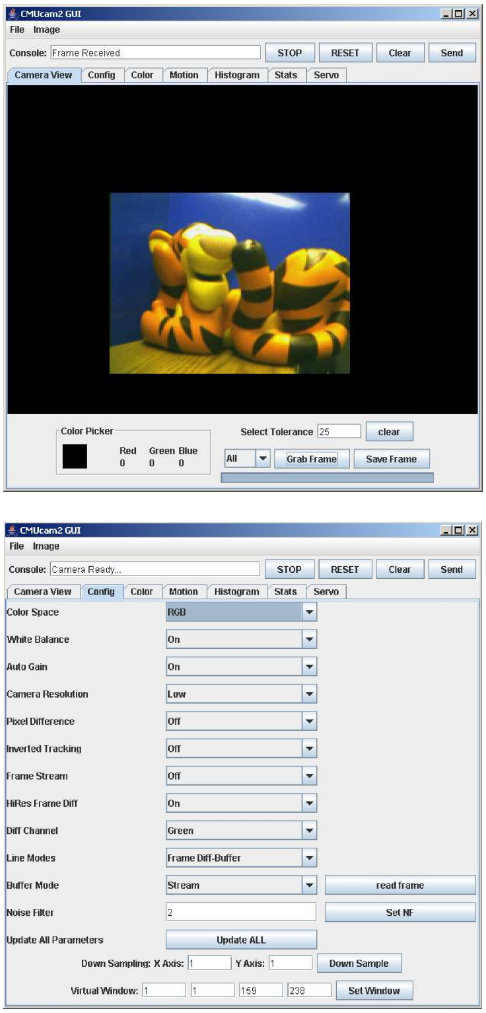

Figure 6: The top screenshot is of the main CMUcam2 GUI window. The lower screenshot shows the camera configuration option panel.

since the original CMUcam and allows for almost all elements of the camera to be explored in a user friendly environment. The GUI graphically displays real-time data from the camera in a more natural manner. For example, the user can enter color bounds into a dialog box and then issue a track color command. The GUI then formats that request and sends it to the camera. The output of the camera data is parsed and displayed in a window that shows the actual bounding box whose colors depend on the confidence value returned. Depending on how it is configured, the "line mode" binary image and the centroid may also be overlaid on the bounding box, as seen in Figure 3. When the user calls the statistics function, the returned color data is mixed and displayed. One of the most important features of the GUI its ability to display a frame dump. The main GUI window and the configuration panel are shown in figure 6.

3 Related Work

The many hardware and software systems that have been constructed by the computer vision community are too numerous to list here. However, some well known systems have had similar goals to the work described here. The Cognachrome vision system [12] which consists of custom frame grabber and processing hardware has functionality most similar to the system we describe here. The Cognachrome system is more capable than the system described here, it can track 25 objects at 60 Hz. However the system described here is significantly less complex and physically smaller making it more attractive for applications like on board vision for small mobile robots. The MIT Cheap Vision Machine [2] has a similar overall architecture to the Cognachrome system and is similarly more capable than the system described here, but is also significantly more complex. A number of systems [3], [4], [7] consist of highly optimized software systems which rely on standard desktop computer systems to process image data. The system here is unique in that it targets applications where including the capabilities of a standard desktop machine would be prohibitive because of size, cost, or power requirements.

4 Conclusions and Future Work

The goal of this work was to overcome the shortcomings of the CMUcam system as well as improve upon and add to its functionality. With the addition of a frame buffer, we were able to increase performance of color tracking from a maximium of 17 frames per second to 50. We were also able to enhance the interface software making integration of the system with low end microcontrollers even simpler. The extra code space on our main processor made frame differencing, and histogram generation possible, while leav-

ing enough space to enhance the configurability of all other legacy functions. The system has already proven its usefulness in a vast array of educational and research projects.

The single most important drawback of this system is the difficulty involved in developing additional on chip algorithms in firmware given the scarce resources and complexities of firmware coding on the SX-52 microprocessor. Ideally, we would like a more flexible open source development environment with more code space and enough onboard memory to hold a single frame. This would then open up an even wider range of applications and allow for community development. To meet these needs, we have already developed a prototype ARM7 based fully reprogrammable system. We also intend to significantly reduce the power consumption of this future system so as to enable long term battery operation.

Acknowledgments

NASA/Ames Intelligent Systems provided ongoing funding for this research.

References

[1] Averlogic, "AL422B Video FIFO Documentation",http://www.averlogic.com/video_FIFO/422b.html

[2] C. Barnhart, The MIT Cheap Vision Machine, http://www.ai.mit.edu/people/ceb/cvm.html

[3] J. Bruce and T. Balch and M. Veloso, "Fast and Inexpensive Color Image Segmentation for Interactive Robots," *The Proceedings of IROS 2000*, 2000.

[4] G. D. Hager and K. Toyama, "The XVision system: A general purpose substrate for real-time vision applications," *Computer Vision and Image Understanding*, vol. 69, no. 1, pp. 23-27, January 1998.

[5] I. Horswill, "Polly: A vision-based artificial agent," *The Proceedings of the Eleventh National Conference on Artificial Intelligence*, 1993.

[6] I. Nourbakhsh, D. Andre, C. Tomasi and M. Genesereth, "Mobile Robot Obstacle Avoidance via Depth from Focus," *Robotics and Autonomous Systems*, vol. 22, pp. 151-158, 1997.

[7] K. Konolige, The SRI Small Vision System, http://www.ai.sri.com/~konolige/svs/

[8] L.M. Lorigo and R.A. Brooks and W.E.L. Grimson, "Visually Guided Obstacle Avoidance in Unstructured Environments," *Proceedings of IROS 97*, pp. 373-379, 1997.

[9] Omnivision Technologies Incorporated, "OV6620 Single-Chip CMOS CIF Color Digital Camera Technical Documentation," http://www.ovt.com/

[10] A. Rowe, C. Rosenberg, I. Nourbakhsh, CMUcam Website, http://www.cs.cmu.edu/~cmucam/

[11] A. Rowe, C. Rosenberg, I. Nourbakhsh, "A Low Cost Embedded Color Vision System," *The Proceedings of IROS*, 2002.

[12] R. Sargent and A. Wright, "The Cognachrome Color Vision System," http://www.newtonlabs.com/cognachrome/

[13] R. Sargent and B. Bailey and C. Witty and A. Wright, "Dynamic Object Capture Using Fast Vision Tracking," *AI Magazine*, vol. 18, no. 1, 1997.

[14] Ubicom Incorporated, "SX28AC Configurable Communication Controllers Technical Documentation", http://www.ubicom.com/sx/

[15] I. Ulrich and I. Nourbakhsh, "Appearance-Based Obstacle Detection with Monocular Color Vision", *Proceedings of AAAI Conference*, pp. 866-871, 2000.