A high-speed low-latency portable visual sensing system

Anne Wright

1 Introduction

My work in the field of computer vision has focused on building high speed visual tracking systems to provide position feedback for the control of unstable tasks. In order to make a juggling robot or a robotic helicopter stable, three qualities are essential – high update rate, low latency, and decent accuracy and resolution. We want to put multiple vision systems on small robotic devices with limited payloads. Therefore the system must be very small, light, and cheap. However, we allow marking of the environment so the sophistocation of the vision task can be extremely low. These constraints guided the development of AVS (which, due to lack of a better name, stands for Anne's Vision System).

AVS is a binary vision system. However, many interesting vision tasks can be expressed in terms of binary images by using different video preprocessing hardware. Figure 1 shows the overall organization of AVS. Video input is reduced to three digital signal lines by a hardware preprocessor, and those three signals go directly to timing pins of the vision processor which handles all of the acquisition, vision processing, and control. In addition to black and white intensity thresholding, analog and digital color preprocessing systems have been built. We also have plans to build motion detection and feature extraction preprocessors. The AVS vision processor requires only that vertical and horizontal timing signals which are synchronous with the binary preprocessed video signal be given to the vision processor. The distinction between the many different tasks can be limited to the preprocessor and the acquisition software can operate identically on different versions of the hardware.

The vision processor is a standard Motorola 68332 microcontroller with 64Kbytes of memory running at 16MHz. The microcontroller with memory can be purchased for about 220 dollars. The simplest version of the preprocessing hardware with video output that I have built requires only 4 chips at a total value of around 30 dollars. More complex video hardware increases the cost of the system. Video input is taken from a standard camera or other video source, and video output can be displayed on any standard TV. The system can perform simple vision processing, like blob tracking, at 60 frames per second, treating each interlaced field as a separate frame. An AVS processor running at that update rate can have enough time left over to run other important functions, like data links, motors, servos, IR beacon tracking, etc. at the same time.

The reason AVS can perform other tasks at the same time that it is aquiring video data is that the bandwidth of the incoming video is greatly reduced. First the preprocessor reduces the information from 8 (or 24 for color) bits per pixel to 1 bit. Then the timer processor on the 68332 responds to and records only to the transitions. As a result, the raw input available to vision algorithms is not a raster binary image, but a list of run-length encoded segments corresponding to the row, starting column, and ending column of each pulse which was thresholded high (which on the monitor appears white). For the types of tasks AVS is ideally suited for, this list representation is quite small. All of the algorithms which I have run on the system use this list of segments directly to perform blob growing, which individually identifies all contiguous "white" areas in the image by size and center position. It is therefore possible to accurately track the center of a blob between frames, even in the presence of noise or other blobs in the picture.

Many commercially available hardware systems are capable of running the same algorithms, and in a much more general manner. However, the size, power requirements, and cost of such hardware has so far prevented us from using any of it on board small mobile robots. So while AVS is far more specialized and therefore less flexible than those we

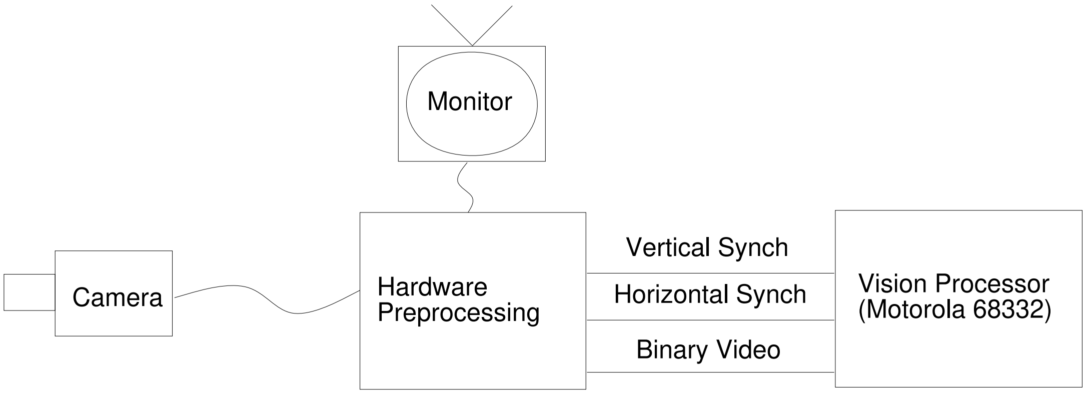

Figure 1: Block Diagram of AVS organization

could buy, it respects our constraints and allows us to apply vision where previously only coarser measurement devices were used.

As a contribution to the field of image processing or machine vision, AVS is not particularly impressive. However, when considered as a contribution to sensing devices available for mobile robots, it is quite impressive. For approximately the same cost as a good gyro, compass, or sonar system, AVS provides high speed high resolution visual sensing capability appropriate for many interesting tasks.

2 Applications

So far AVS has been applied to providing position feedback for a robotic helicopter, controlling a six legged walking robot over a course to a marked goal, and controlling a small wheeled robot to collect colored blocks. Intended future applications include tracking pinball balls to allow a computer to learn to play pinball and implementing obstacle avoidance for the robotic helicopter project. Given a camera capable of sensing heat, it would be easy to use AVS to track the positions of warm blooded creatures. AVS would also work well in applications using structured light, such as laser range finders.

3 Preprocessing Hardware

The preprocessing hardware provides vertical and horizontal (or composite) synchronization signals and a binary video signal to the timing inputs of the 68332. If the signal source is a standard composite or NTSC (the standard format for video signals in North America) source – a camera, TV tuner, VCR, etc. – appropriate synchronization signals may be obtained by using an LM1881 synch stripper chip (available from National Semiconductor and many other producers). However, because the system really has no hard limits on the timing of the video or the number of scan lines it would be easy to use custom CCDs, line scan cameras, tactile sensors with spatial resolution, or any other sensor which could generate appropriate synchronization signals as input.

A major advantage, however, of using NTSC sources is that it is easy to display the thresholded image and confirm that the preprocessing hardware is doing the right thing. In all of my designs I have used a video DAC (digital to analog converter) to provide debugging output. By supplying a composite synchronization signal (which the LM1881 provides), a clock signal, and the binary video signal to a video DAC you can get a video signal suitable for displaying or recording.

To cut corners on the cost and parts count of my preprocessor designs, I use the 68332 system clock to clock the components in the preprocessor hardware instead of using a separate crystal. Also, on the more elaborate designs like the digital color system, the 68332 is used perform all the control functions required for setting up the hardware. Because these are non video rate tasks, this use does not hamper the performance of the vision algorithms. The ease with which it can interface to and control such peripheral hardware is another reason that the 68332 was a good choice for this project.

Figure 2: Block Diagram of the Analog Black and White Preprocessor

3.1 Analog Black and White System

The simplest version of the preprocessing hardware, pictured in Figure 2, is the analog black and white version which thresholds video intensity. It is appropriate for use indoors for tracking lights or objects marked with reflectors. In addition to the synch stripper and the video DAC already mentioned, all that is really required is an analog comparator and a way of setting the threshold. The output of the comparator can be run directly into the binary video input lines on the 68332. The simplest way to set the threshold is with a potentiometer, but it is also possible to use a standard DAC to set the threshold under software control. This version is the most limited, but the cheapest and easiest to build.

The protoype I have built used the following parts: 1 LM1881 Synch Stripper; 1 Brooktree BT106 Video DAC; 1 Motorola MC3430 Quad Comparator (only one-quarter used); 1 LM358 Op-Amp to stiffen the reference voltages; 1 potentiometer to set the threshold; and some resistors and capacitors for support. At 4 simple chips and a small processor board, it's the smallest system I have seen capable of high resolution visual tracking while running a mobile robot.

In the most amusing use this version has been put to, people could use a flashlight to draw pictures in the air while a small 6-legged robot, Genghis (designed by Colin Angle), with a marker taped to its leg copied the picture onto a small easel. The stability and resolution of AVS was sufficient to produce some fairly decent robot art.

3.2 Analog Color System

The next version of the preprocessing hardware was designed to track objects of a particular color in an image. The analog color system uses 6 potentiometers to set upper and lower bounds on the relationship between R and G, and R and B inputs in an RGB (red, green, blue) image. Figure 3 depicts the shape of the surface in RGB space which separates accepted color inputs from rejected ones. The values a,b,c and x,y,z are ratios from 0 to 1 which correspond to the settings on the potentiometers. The set of RGB inputs which respect the inequalities $aG<bR<cG$ and $xB<yR<zB$, which are the points within the thresholding cone, are thresholded to high, and all inputs which do not respect those inequalities are thresholded to low. Comparators are used to test the input according to these relationships. In practice, the output of this circuit is not quite clean enough to track well, but adding a simple RC delay filter on the output and buffering the output of the filter with two inverters made the signal easily clean enough to track well.

It takes quite a bit of practice to set the potentiometers to a given color, but it is impressive that color tracking can be done at all with this little hardware. The parts count on the prototype analog color system is 6 chips and 6 potentiometers outside of the microprocessor board. The chips used were: 1 LM1881 Synch Stripper; 1 Brooktree BT106 Video Dac for display; 1 op-amp follower for voltage reference to the BT106; 1 Motorola MC3430 quad comparator for the four volatage comparisons; 1 74HC08 quand and gate for anding the results of the comparators; and 1 74HC14 hex Schmidt trigger inverter for buffering the output of the RC filter.

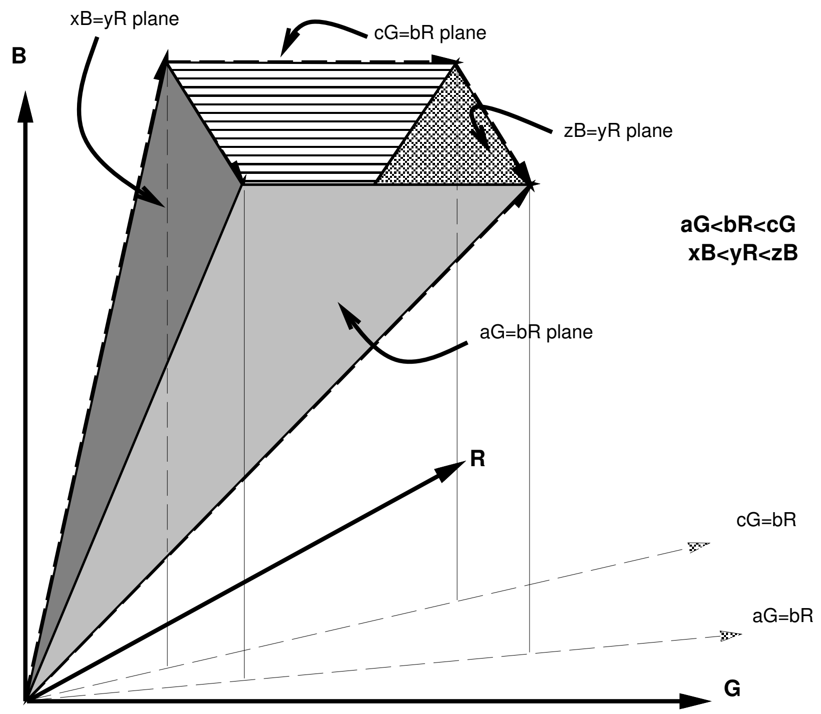

Figure 3: Diagram in RGB space of the surface of the thresholding function for the Analog Color System

The big advantage of this system is that it can be used in arbitrary lighting conditions, and works even better outdoors than indoors. Tasks like picking up blocks or going to a marker of a given color are also considerably more interesting than tasks which must be done in black and white.

3.3 Digital Color System

The digital RGB preprocessing hardware uses a more traditional approach to color thresholding. Three analog to digital converters generate a 24 bit digital value for each video pixel. Sixteen bits of this(6R, 5G, 5B) are used as an address into a look up table (implemented in RAM) and the latched output of the look up table corresponds to the threholded video signal. The advantages this system has over the analog RGB system are that you can sample the color present at a given point in the image and use that data to fill the look up table so it is easier to tune, and you can threshold for multiple colors simultaneously by treating the outputs of the RAM as independent color outputs rather than a single byte of data. However, it requires considerably more hardware to implement and has not yet demonstrated that it can threshold better than the analog system can.

3.4 Motion Detection

Figure 4 shows an idea for the data flow for a hardware preprocessor to implement real time motion tracking input for AVS. This project is still in the planning stages, but it is an exciting prospect to be able to track fast moving objects against a relatively arbitrary background. The idea of using frame subtraction to detect motion is certainly not a new one, but I believe it to be well suited to the capabilities of AVS.

Such a preprocessor could compare an image stored in a frame buffer against the incoming video stream. By using two such frame buffers, one of which is being written to with the new frame while the other is being read, the difference could be continuously computed and thresholded for real time tracking. This difference could just be luminance (black and white) difference or a color difference (R1-R2, G1-G2,B1-B2) which would allow isoluminant color edges to be tracked. The 68332 vision processor could easily handle the control signals for switching which frame buffer is being read from during the vertical retrace period, and the switching rate could be a software selectable parameter.

Figure 4: Data flow diagram for a hypothetical motion detection preprocessor

The main impetus for building such motion preprocessor for AVS is our desire to build a non-intrusive automatic learning pinball player. The idea is to build something which can start the game, release the ball, and flip the flippers by giving it the ablility to mechanically manipulate the controls of an unmodified pinball machine. It seems that in keeping with that goal, the best way to track the ball would be through a frame subtraction tracking scheme.

Andrew Moore of the MIT Artificial Intelligence Laboratory has already made considerable progress in his work on machine learning for automated billiards [Moore, 1992]. He tracks the motion of the balls visually by storing several seconds worth of the output of a Datacube FeatureMax black and white thresholding system and calculating the tracking information after the shot has been made. This approach works well for billiards because the effects of the shot last only for a short period of time before the balls come to rest and the system has as long as it wants to process information before making the next shot.

Pinball is a similar problem, but the vision processing requirements are more demanding. The ball can remain in play for an arbitrarily long period of time, and during that time you must be able to quickly react to its movements. This makes storing the data and processing it off line impossible and it is necessary to process the visual data as it comes in. It would further be desireable to leave such a system running for a long period in order to ascertain how well it learns over time. This is a reason we would like to use something cheap and easily replicated like AVS for this task rather than something like the Datacube which is expensive and has many other demands on its time.

Other appropriate uses for a motion preprocessor would be juggling, catching a ball, playing ping-pong, or building robots which react to movement. The frame buffers used for motion processing could also double as general purpose image buffers accessible by the CPU so that standard greyscale image processing could coexist happily with AVS techniques on the same hardware.

3.5 Feature Detection

Figure 5 shows a possible data flow model for a feature extraction preprocessor for AVS. Many special purpose chips are available to implement filtering and convolutions on video streams. Video shift registers can be used to hold a number of scan lines of video data, so that even spatial filtering is possible without storing entire video frames or missing frames in order to preform the filtering in software. Examples of chips capable of performing these functions are the L64210/L64211 Variable-Length Video Shift Register and the L64230 Binary Filter and Template Matcher available from LSI Logic, as described in the LSI Logic Shortform Catalog, 1991. This capability brings generalized feature extraction within the realm of tasks that AVS could be applied to.

Figure 5: Data flow diagram for a hypothetical feature detection preprocessor

The most obvious uses for spatial filtering include noise reduction or edge detection. Other possible uses include recognizing a small pattern in an image, like the letter A, or any other image fragment. These techniques are described in chapter 7 of "Computer and Robot Vision, Volume I" by Robert Haralick and Linda Shapiro [Haralick, 1992]. It is easy to imagine thresholding the output of such a filter to create a meaningful binary image appropriate to the abilities of AVS.

Edge detection in particular has been successfully applied to problems of navigation like following hallways or avoiding walls by detecting the edge where wall meets floor. G Hartmann used an edge detection and linking scheme for guiding robotic cars to follow lines on highways [Hartmann, 1992]. Ian Horswill on his robot Polly used calculation of the vanishing point of corridors to follow them [Horswill, 1993]. I believe that this sort of task which could be accomplished with an edge detecting version of AVS. We are interested in trying to put this capability on board our autonomous robotic helicopters in order to help avoid collisions. Many obstacles you would want to avoid, like trees, buildings, and people have well-defined edges and we hope that something like an edge detecting AVS could detect these obstacles fast enough to react to them appropriately.

4 Acquisition

A major advantage of using the Motorola 68332 microcontroller is that it has a sophistocated timer processor unit (TPU) which operates independently of the CPU and is capable of 4MHz timing resolution – a perfect speed for dealing with video. The TPU has 16 identical timing channels which can be configured for input capture, pulse accumulation, and many other useful functions. The timer channels can also be configured to interrupt the CPU when timing events occur. These capabilities allow the TPU to capture the row and column position of transitions of the binary video signal with minimal CPU interaction.

At the fastest TPU timing speed, the resolution of the video image distinguishes 265 pixels per line. Ignoring interlace, there are about 250 horizontal lines in each image. If the image is simple enough, acquisition of the input transitions and calculation of the run length encoding for the image can be done entirely on interrupts. The code which takes care of the tracking can process the segments of the image as soon as they appear in memory and act accordingly.

I have found many advantages with this approach to image aquisition.

- Very little hardware outside of the 68332 processor is required, which keeps cost, size, and power consumption low.

- The amount of data the processor has to deal with is extremely low because the preprocessing hardware filters the image and only the interesting transitions are captured.

- Synchronization of the tracking computation with control actions is easy because all the processing is done within a single computational element.

- In small, single processor applications, time for the processor to take care of other concerns, such as motor control or user interaction, can be maximized because time is spent on data acquisition only when there is something of interest in the image.

- Because the acquisition and tracking are really being performed in parallel it is possible to track without missing any frames, which is of great use when measuring the position of very fast moving objects in real time.

There is a major limitation to this approach. If the filtered image is too complex or transitions occur too close together on a horizontal scan line data can be lost. Performance on complex input can be greatly increased by acquiring data in a tight acquisition loop instead of on interrupts, but in resorting to this the particular elegance of AVS is largely lost. However, because the acquisition is done entirely within the 68332 it is easy to switch between these modes if different parts of a task are better matched to one or the other.

5 Vision Processing

The raw form of the vision data that AVS provides is a list of run length encoded segments corresponding to the row, starting column, and ending column of the “white” (thresholded to high) sections in a scan line. For some tasks, such as edge detection, it would be appropriate to detect only the positions of the rising edges, and AVS could even more easily provide that information.

In all of the applications to which it has applied, AVS has executed a simple blob growing algorithm similar to that described in chapter 10 of the book “Introduction to Robotics” by Philip John McKerrow [McKerrow, 1991], except that I perform blob growing directly on the list of segments instead of on a binary matrix. I define a blob to be the set of all segments which are vertically contiguous to another segment in the same set. I convert the list of segments into a list of blobs. Because the segments are stored in the order they come in, the blobs can be accumulated on the fly with the blob list updated as each segment is acquired. During the processing, only of the sum of the row and column of the pixels in each blob (so that centroids can be later determined), the row and left and right extrema of the segments on the last two horizontal lines (so that future connectivity can be determined), and area of each blob is stored. The processing time per segment is related only to the number of blobs with segments on the previous horizontal, any or all of which could be attached to the candidate segment. This is an efficient algorithm, and it has the advantage that it can accurately distinguish the position of each blob in the picture without much regard for the total size or number of blobs. However, in order to use it individual blobs in the image contain a segment in each horizontal scan line they cover. In practice, the preprocessors have been able to perform well enough that this was not a major problem.

Knowing the positions of the blobs in the current image and the position of a blob of interest in the previous image, it is easy to correlate a best fit for that blob in the current frame. This is particularly easy at high update rates because physical things can move only a small amount in a 1/60 of a second. The control tasks that have so far been executed have been based on such tracking of single blob and reacting to the position of that blob in the image.

I am certain that AVS is capable of performing more interesting tasks, like determining the direction a corridor is going in by acquiring edge data and correlating it into lines, but thus far the simple techniques have been sufficient and the hardware development has occupied most of the time.

6 Conclusion

The initial motivation in building AVS was to increase update rate and decrease data flow bottlenecks for simple vision tasks. It succeeded in this goal, and in the process it suggested new approaches to “cheap” vision. As much functionality as possible was pushed within the 68332 processor chip and the external preprocessing hardware was reduced to a bare minimum. The initially unexpected side effect that AVS is very small and cheap made it possible for me to extend some of the advantages of vision processing to small robots.

In my opinion, the cutoff point at which robots are considered too small to be fitted with an autonomous vision system is currently too large. Cameras are now being made small enough that this no longer needs to be the case. Most video acquisition hardware, however, seems to be designed to require installation in a PC, a workstation, or a VME cage and depends on on that environment to process the information it collects. This design philosophy acts largely to prevent robots incapable of carrying a PC, workstation, or VME card cage from being fitted with vision systems. As a result, small robots are usually restricted to using discrete cadmium sulfide cells or phototransistors to detect light, which are very limited in the data they can provide. AVS can help fill the gap between the two extremes of photocells and generalized video processing systems, providing some of the advantages of both.

7 References

[Haralick, 1992] “Computer and Robot Vision, Volume I”, Robert M. Haralick and Linda G. Shapiro, Chapter 7, Addison-Wesley Publishing, 1992

[Hartmann, 1992] “A Hierarchical Vision System”, G. Hartmann, Proceedings of the Intelligent Vehicles Symposium, 1992

[Horswill, 1993] Ian Horswill, “Specialization of Perceptual Processes,” doctoral thesis, MIT, 1993.

[McKerrow, 1991] “Introduction To Robotics”, Phillip John McKerrow, Chapter 10, Section 8, Addison-Wesley Publishing, 1991

[Moore, 1992] A. W. Moore and D. J. Hill and M. P. Johnson, ”An Empirical Investigation of Brute Force to choose Features, Smoothers and Function Approximators”, in ”Computational Learning Theory and Natural Learning Systems, Volume 3”, MIT Press 1992