

# **Circuit Cellar / Atmel AVR Contest**

**June 30<sup>th</sup> 2004**

**Entry #A3694: AVRcam**

**Abstract**

**A low-cost image processing engine capable of visually tracking eight objects of eight different user-defined colors, at 27 frames per second...**

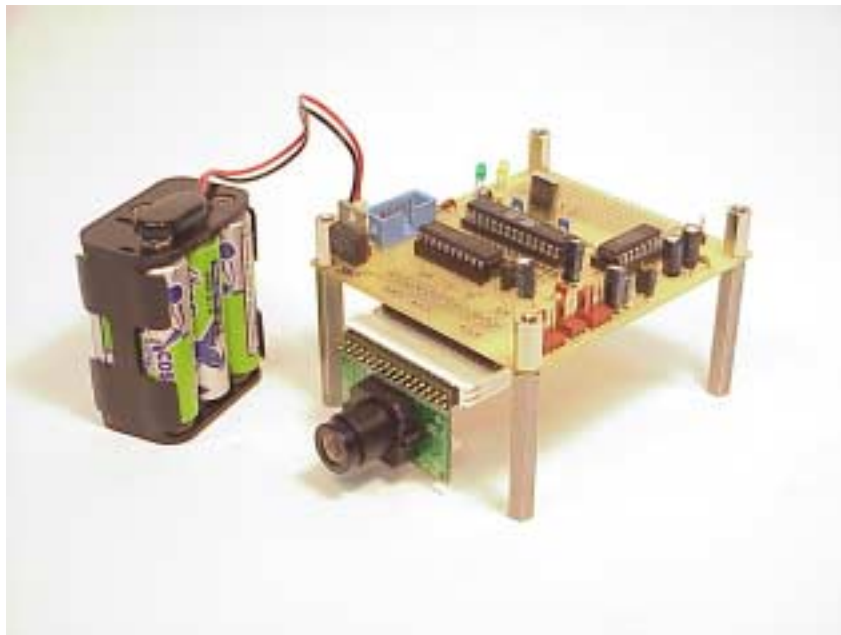
**...Hobbyist robotics just got a lot more interesting**

### Overview

The AVRcam is a stand-alone image-processing engine based on the Atmel ATmega8 microcontroller. This engine is capable of tracking eight objects of eight different user-defined colors, at 27 frames per second. The system provides a user-interface, through a well-defined protocol, over a standard serial port. The user-interface provides the following image information to the user in real-time:

- number of currently tracked objects
- color of each tracked object
- center point of each tracked object
- bounding box of each tracked object

The system also allows the user to take snapshots of its current field of view, allowing the user to better understand how the AVRcam perceives its environment. The AVRcam is shown in Photo 1.



**Photo 1: The AVRcam Version 1.0**

A PC-based application (AVRcamVIEW) was also developed to provide a platform for demonstrating and configuring the AVRcam. This application interfaces to the AVRcam through the standard RS232 port on any PC. This application allows the user to take snapshots with the AVRcam, and view them or save them for later use. Once a snapshot is taken, the user can quickly set up the color map to be used by the AVRcam, by simply clicking on the colors of interest. AVRcamVIEW will automatically build and download a color map to the AVRcam for to be used during object tracking. Finally, AVRcamVIEW can display the real-time tracking results to the user. An example screenshot is shown in Photo 2.

The AVRcam was developed for the purpose of enabling mobile robots to perceive their environment in a small form-factor, low-power, high-capability package. A typical embedded configuration would have the AVRcam connected to another microcontroller, which accesses the AVRcam through its serial user-interface. This type of system is especially useful when the environment in which a robot operates can be defined by the colors around it. An example of this is the RoboCup robot soccer competition held each year. The AVRcam system can be easily interfaced to a robot to provide high-level, real-time information about the robot's environment.

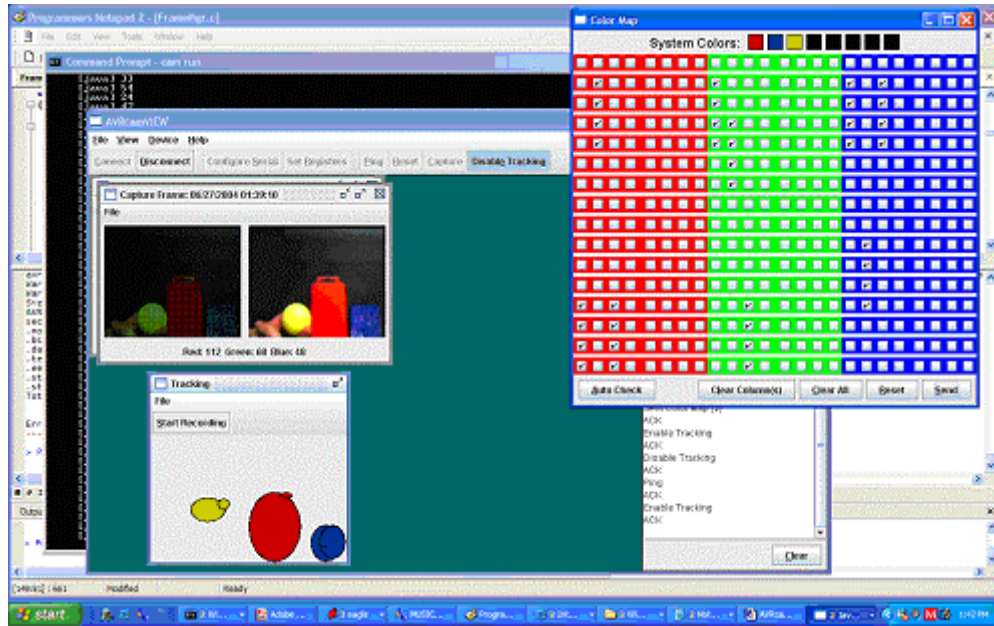


Photo 2: The AVRcamVIEW PC Application Tracking Three Objects Simultaneously

### System Block Diagram

The complete AVRcam system is shown in Figure 1. The system consists of an Omnivision OV6620 CMOS color image sensor, an ATmega8 microcontroller, and a handful of support components. The key to allowing the system to execute at such high frame rates is to provide inherent synchronization between the image sensor and the ATmega8 by clocking them both with the same crystal. Thus, synchronization with the pixel data is only required at the start of each line. This greatly reduces the amount of overhead when acquiring each pixel.

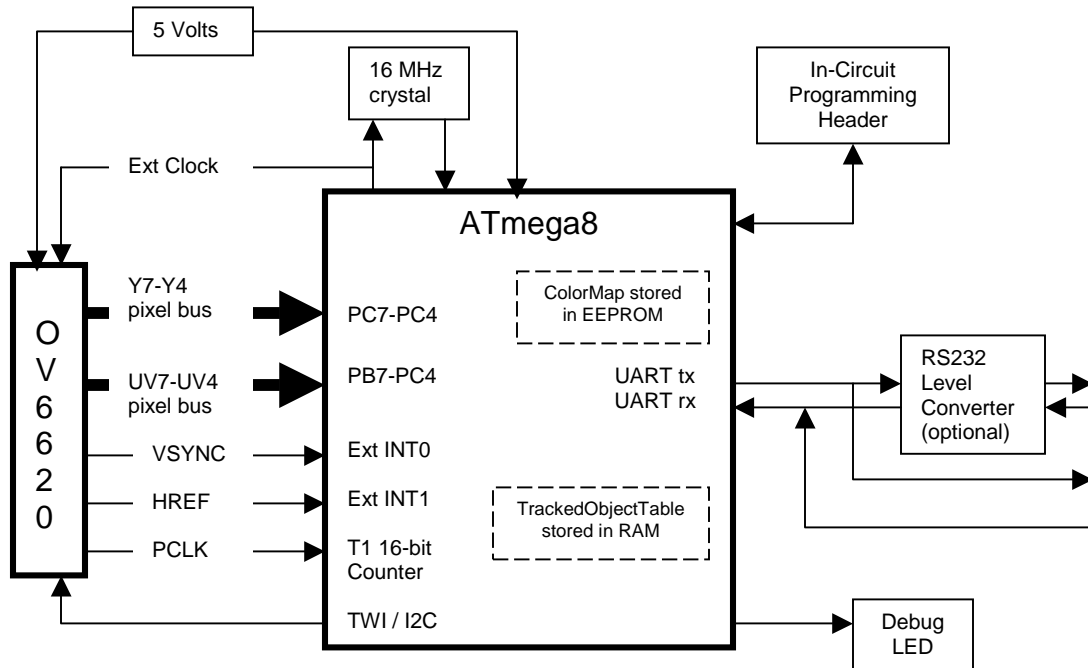


Figure 1: AVRcam System Block Diagram



Entry #A3694

```

-----
|                                     |
-----
OR
-----
|                                     |
-----
|                                     | */
( (currPixelRunStart <= lastLineXStart) &&
  (currPixelRunFinish >= lastLineXFinish) ) )
{
  /* THERE IS CONNECTEDNESS...update the lastLineXStart and lastLineXFinish
  data pointed to by pTrackedObjectData */
  *pTrackedObjectData++ = currPixelRunStart;
  *pTrackedObjectData++ = currPixelRunFinish;

  /* check if the bounding box needs to be updated */
  if (*pTrackedObjectData > currPixelRunStart)
  {
    /* need to update the bounding box for the upper left point to
    enclose this new left-most point...we never have to update the
    upper left Y point, since each scan line we process moves from
    top to bottom */
    *pTrackedObjectData = currPixelRunStart;
  }
  pTrackedObjectData += 2; /* move the pointer up to the lowerRight data */
  if (*pTrackedObjectData < currPixelRunFinish)
  {
    /* need to update the bounding box for the lower right X point to
    enclose this new right-most point */
    *pTrackedObjectData = currPixelRunFinish;
  }

  /* the lower right 'y' point always gets updated when connectedness is found */
  *(pTrackedObjectData+1) = trackedLineCount;

  /* set a flag indicating that that color run is part of another
  object and thus doesn't need to be added as a new entry into the
  tracking table */
  colorConnected = TRUE;
  break;
}
}
}
}

if (colorConnected == FALSE)
{
  /* a new entry needs to be made to the tracking table, since we have
  a run-length with a color, and it isn't connected to anything...but we
  can only do this if there is space left in the trackedObject table */
  if (numCurrTrackedObjects < MAX_TRACKED_OBJECTS)
  {
    /* space is available...add the object */
    pTrackedObjectData = (unsigned char *)&CurrentTrackedObjectTable[numCurrTrackedObjects];

    /* now that we have a pointer to the tracked object to be updated, update all
    the fields */
    *pTrackedObjectData++ = currColor; /* color */
    *pTrackedObjectData++ = currPixelRunStart; /* lastLineXStart */
    *pTrackedObjectData++ = currPixelRunFinish; /* lastLineXFinish */
    *pTrackedObjectData++ = currPixelRunStart; /* x_upperLeft */
    *pTrackedObjectData++ = trackedLineCount; /* y_upperLeft */
    *pTrackedObjectData++ = currPixelRunFinish; /* x_lowerRight */
    *pTrackedObjectData++ = trackedLineCount; /* y_lowerRight */
    /* we'll calculate the center point for each tracked object when
    the frame is done, instead of wasting the precious clock cycles doing
    it now */

    numCurrTrackedObjects++;
  }
}
}
} while(currPixelRunFinish < ACTUAL_NUM_PIXELS_IN_A_LINE);
}
}

```

