Efficient Image Processing for Increased Resolution and Color Correctness of CMOS Image Sensors

Jacky Baltes

Centre for Image Technology and Robotics
University of Auckland,Auckland
New Zealand
`j.baltes@auckland.ac.nz`
`http://www.citr.auckland.ac.nz/~jacky`

Abstract. This paper describes fast demosaicing methods to quadruple the resolution of a CMOS camera. The resulting increase in accuracy in camera calibration and object detection is important for local vision robots, especially those that use computer vision as their only source of information about the state of the world. The paper describes two methods for demosaicing: interpolation and variance demosaicing. A comparison of three sample views is shown to demonstrate the increased resolution and the difference between the interpolation and variance demosaicing methods. Both demosaicing methods work well. Variance demosaicing performs better around edges in the image, but is computationally more expensive.

1 Introduction

This paper presents demoasicing, a well known technique in digital cameras and investigates its application to problem of computer vision in real-time, real world robotics. Demoasicing quadruples the resolution of CMOS sensors based on the Bayer pattern. This increase in resolution leads to more accuracy in determining the camera calibration and feature locations. For example, the distance to the ball and the angle to the goal can be computed more accurately.

Given the limited amount of processing power available on most robotic platforms and the harsh constraints of the real world, the extra amount of memory required and the additional computational overhead is of interest. Both of these issues are investigated for the two described algorithms.

Section 2 gives a brief introduction into the geometry of a CMOS camera and the Bayer pattern that is most commonly used. Section 3 introduces demosaicing and presents two demosaicing methods: interpolated demosaicing and variance demosaicing. Section 4 describes an additional method for pre-processing of color images taken with a CMOS sensor. The green channel is usually much more sensitive than the other channels. Conclusions are described in section 5.

2 CMOS Camera Hardware

This section describes the actual hardware that is used in most CMOS image sensors. The CMOS sensor is able to measure the itensity of incoming light only. Therefore, a color CMOS sensor uses at least three different image sensors to measure the brightness of the color signals: one red filter, one green filter, and one blue filter.

However, since multiples of three are awkward to manufacture, a common way of manufacturing a CMOS sensor is the so-called "Bayer" pattern. This pattern use four image sensors as shown in Fig. 1.

R	G	R	G	R	G	R	G
G	B	G	B	G	B	G	B
R	G	R	G	R	G	R	G
G	B	G	B	G	B	G	B
R	G	R	G	R	G	R	G
G	B	G	B	G	B	G	B

Fig. 1. Bayer Pattern as used in CMOS Image Sensors. R, G, and B indicate a red, green, and blue filter sensor element.

It can easily be seen that for a single position only, a single color channel is available. That is pixel (0,0) can only measure the red channel of the signal. The sensor data needs to be further processed to create a RGB color image.

A standard method is to create a single color pixel from four adjacent image sensors. One problem is that there are two image sensors with green information. To speed up the processing, most systems simply discard one channel of green information. Another possibility is to use the average of the two green channels as green value for the pixel. Figure 2 is a graphical illustration of the algorithm used. In this example, a 164 by 124 image sensor produces a 82 by 62 image.

This is the standard method used by the CMOS cameras used by the Eyebot controllers, which are used in the 4 Stooges RoboCup team. Some sample images obtained with this method are shown as original image in Fig. 4, Fig. 5, and Fig. 6.

The obvious disadvantage of this method is that information is lost in the conversion process. The position of the pixel segments is not taken into consid-

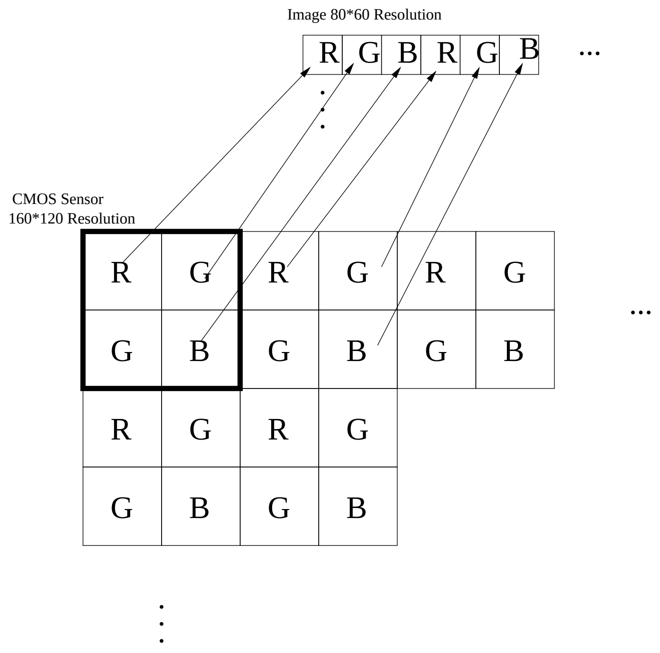


Fig. 2. Standard Processing of CMOS Sensor Data

eration, and the resolution of the resulting image is only a quarter of that of the resolution of the CMOS sensor.

3 Demosaicing

The idea is to infer the missing values for the color information of all pixels by using information about adjacent or near-by pixels. This process is called demoasicing. Figure 3 is a graphical illustration of the main idea.

If we take the blue-filter image sensor at location (1,1) as an example, then we can see that the blue information at this pixel is known, but we do not know the values for the red and green channel. However, we do know the values of the red channel for four adjacent pixels and the values of the green channel for four other adjacent pixels.

Similarly the values for the green and blue channel are missing for red-filter sensor elements, and the green filter elements are missing the red and blue color information.

The process of inferring the missing color information is a topic often studied and well known in digital cameras.

3.1 Averaging

The simplest and fastest algorithm is a bilinear interpolation of the missing pixel values by averaging the values of adjacent pixels. The algorithm shown in Alg. 1

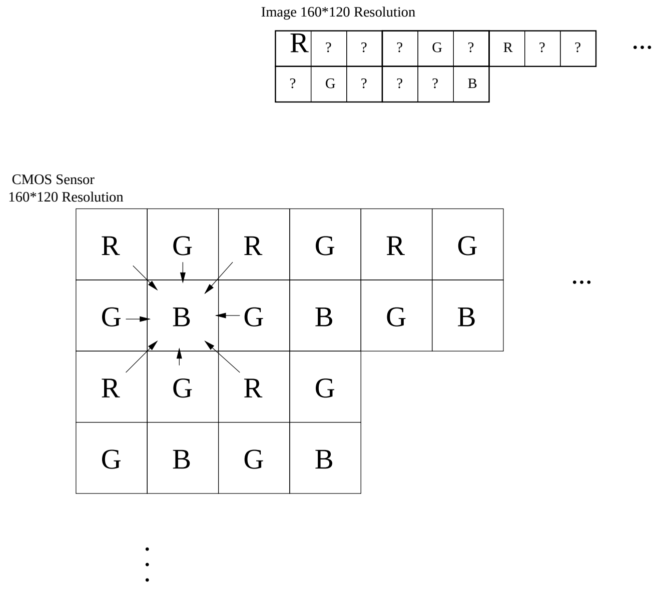


Fig. 3. Demosaicing of CMOS Sensor Data. Green and red information for the pixel (1,1) is inferred from neighboring pixels.

implements this simple interpolation scheme. It can be seen that for red and blue pixels four neighboring pixels are available to infer each of the missing color information. For the green pixels, only two neighboring pixels are available for the red and blue color information.

This algorithm can be implemented by using three rows of pixels as temporary storage.

The computational cost of this algorithm is not much more than the standard algorithm, but results in an image with four times the resolution. The following table summarizes the number of memory reads and writes, the additions and subtractions, and lastly the multiply and divide operations for each pixel:

Pixel	Mem Read/Write	Add/Sub	Mul/Div
Red	9/1	8	2
Green Pixel 1	5/1	4	2
Green Pixel 2	5/1	4	2
Blue	9/1	8	2

The total cost of demosaicing is therefore 8 Mem + 6 Add + 2 Div operations per pixel. This algorithm was implemented an tested on the Eyebot controllers. On a 25 MHz 68332, the demosaicing routine took 30ms. This is about half the time required to read in the image. This means that the maximum frame rate of the video processing dropped from 15 to 10 frames per second.

Algorithm 1 Interpolation Algorithm for Demoasicing

```
for $i$=0 to Height do
  for $j$=0 to Width do
    if ($i$ mod 2 = 0) AND ($j$ mod 2=0) then
      red = pixel[$i$,$j$] {// Red Pixel}
      green = (pixel[$i-1$,$j$] + pixel[$i$,$j-1$] + pixel[$i+1$,$j$]
      + pixel[$i$,$j+1$])/4
      blue = (pixel[$i-1$,$j+1$] + pixel[$i-1$,$j-1$] + pixel[$i+1$,$j-1$]
      + pixel[$i+1$,$j+1$])/4
    else if ($i$ mod 2 = 0) AND ($j$ mod 2=1) then
      green = pixel[$i$,$j$] {// Green Pixel 1}
      red = (pixel[$i$,$j-1$] + pixel[$i$,$j+1$])/2
      blue = (pixel[$i-1$,$j$] + pixel[$i+1$,$j$])/2
    else if ($i$ mod 2 = 1) AND $j$ mod 2=0) then
      green = pixel[$i$,$j$] {// Green Pixel 2};
      red = (pixel[$i-1$,$j$] + pixel[$i+1$,$j$])/2
      blue = (pixel[$i$,$j-1$] + pixel[$i$,$j+1$])/2
    else if ($i$ mod 2 = 1) AND $j$ mod 2=1) then
      blue = pixel[$i$,$j$] {// Blue Pixel };
      red = (pixel[$i-1$,$j+1$] + pixel[$i-1$,$j-1$] + pixel[$i+1$,$j-1$]
      + pixel[$i+1$,$j+1$])/4;
      green = (pixel[$i-1$,$j$] + pixel[$i$,$j-1$] + pixel[$i+1$,$j$]
      + pixel[$i$,$j+1$])/4;
    end if
    image[i,j] = (red, green, blue);
  end for
end for
```

Sample images are shown in Fig. 4, Fig. 5, and Fig. 6. The images after interpolated demosaicing look quite natural. More pixels are available to determine the center of the ball for example.

3.2 Variance Interpolation

A closer look at the output of interpolated demosaicing shows that it leads to blurring around the edges of objects. For example, the edges of the ball, the goal, or the track are blurred by this process.

To overcome this blurring, a more complex demosaicing algorithm, so called variance demoasicing is used to maintain the sharpness of edges. The main idea is to compute the amount of variance (that is the difference between this pixel and its neighboring pixels) and to assume that the variance is similar in all three color bands. So, if the current red pixel is much brighter than its neighboring pixels, then its green and blue channels are also increased. The algorithm is shown in Alg. 2. Algorithm 3 is used to compute the average brightness in the current color channel. In line 2, the difference between the current pixel and this average is computed. The missing color information (green and blue for a red pixel) is computed similarly to the interpolation algorithm shown in Alg. 1.

However, in lines 2 and 2, the computed difference is used to adjust these color values.

Algorithm 2 Variance Algorithm for Demoasicing

```
for $i$=0 to Height do
  for $j$=0 to Width do
    if ($i$ mod 2 = 0) AND ($j$ mod 2=0) then
      red = pixel[$i$,$j$] {// Red Pixel}
      redAvg = calcAverage(i,j,red)
      redDiff = red - redAvg
      green = (pixel[$i-1$,$j$] + pixel[$i$,$j-1$] + pixel[$i+1$,$j$]
      + pixel[$i$,$j+1$])/4
      green = green + redDiff
      blue = (pixel[$i-1$,$j+1$] + pixel[$i-1$,$j-1$] + pixel[$i+1$,$j-1$]
      + pixel[$i+1$,$j+1$])/4
      blue = blue + blueDiff
    else if ($i$ mod 2 = 0) AND ($j$ mod 2=1) then
      green = pixel[$i$,$j$] {// Green Pixel 1}
      greenAvg = calcAverage(i,j,green)
      greenDiff = green - greenAvg
      red = (pixel[$i$,$j-1$] + pixel[$i$,$j+1$])/2
      red = red + greenDiff
      blue = (pixel[$i-1$,$j$] + pixel[$i+1$,$j$])/2
      blue = blue + greenDiff
    else if ($i$ mod 2 = 1) AND $j$ mod 2=0) then
      {Similar to processing of Green Pixel 1}
      {and omitted here}
    else if ($i$ mod 2 = 1) AND $j$ mod 2=1) then
      {Similar to processing of Red Pixel}
      {and omitted here}
    end if
    image[i,j] = (red, green, blue);
  end for
end for
```

Algorithm 3 calcAverage(x,y,c)

```
avg = (pixel[$i-2$,$j$] + pixel[$i+2$,$j$] + pixel[$i$,$j-2$]
+ pixel[$i$,$j+2$])/4;
return avg
```

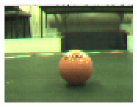
This algorithm requires more storage and more computation that the interpolation algorithm described in section. In fact, it requires temporary storage to hold five rows. Furthermore, the cost of computing the calcAverage is 4/1 Memory + 4 Add + 1 Div. Thus the costs in terms of memory read or writes, additions or subtractions, and multiply and divide for this algorithm can be computed and are shown in the following table.

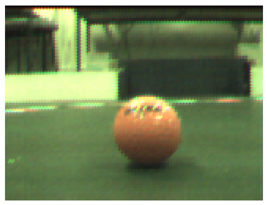
Pixel	Mem Read/Write	Add/Sub	Mul/Div
Red	15/5	18	3
Green Pixel 1	11/3	10	3
Green Pixel 2	11/3	10	3
Blue	15/5	18	3

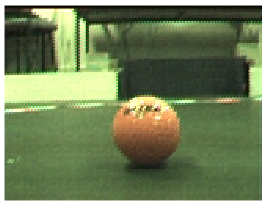
The average cost of computation for a pixel is therefore 17 Mem + 14 Add + 3 Div. In other words, this algorithm uses more than twice the number of memory accesses and add operations than the previously shown one. This result was verified by timing the implementation of both algorithms, where the variance demoasicing algorithm took sufficiently more time than interpolated demosaicing.

Three sample images with the output of the variance demosaicing algorithm are shown in Fig. 4, Fig. 5, and Fig. 6. The images after variance demosaicing show that the edges around the ball, the cup, and the track are much clearer and not blurred.

We also created difference images to highlight the differences between the interpolated and variance algorithm. As can be seen, the result of the algorithms are very similar and only around the edges are the outputs different.


Original Image


Interpolated Demosaicing


Variance Demosaicing

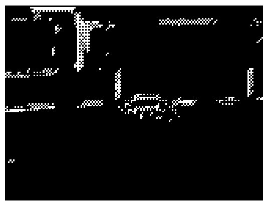

Difference between Interpolate and Variance Demosaicing

Fig. 4. Ball Scene: Comparison of the original image, the interpolate-demoasiced image, and the variance-demosaiced image. The difference between these two methods is also shown.

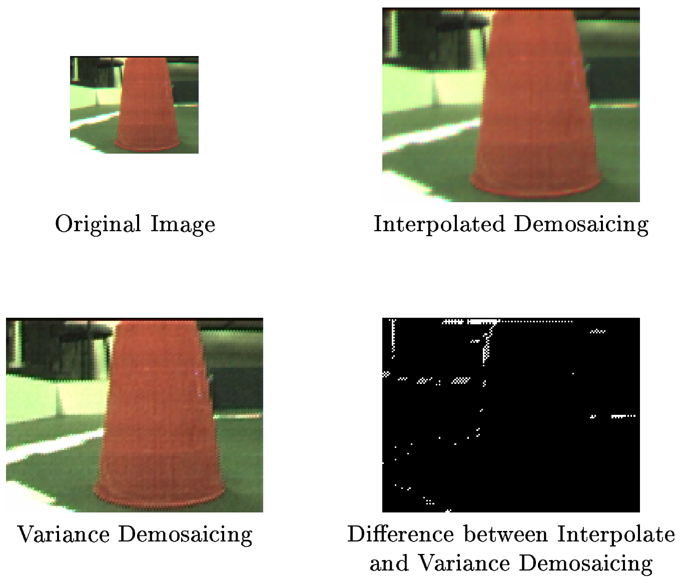


Fig. 5. Scene with Red Cup: Comparison of the original image, the interpolate-demoasiced image, and the variance-demosaiced image. The difference between these two methods is also shown.

4 Color Calibration

Another problem with CMOS sensors is that the sensitivity of the device is not the same as that of the human eye. Therefore, images taken by a CMOS camera have incorrect color information. This can be easily be seen in a color photo which show a very strong green tint.

To correct for this green tint, we took an image of a gray scale gradient. This image showed that the red and blue channels were similar and ranged in value from approximately 20 to 210. However, the green channel ranged from approximately 25 to 250. We therefore reduced the intensity of the green channel by a factor of 0.8. The resulting image was a gray scale image and the green tint has disappeared.

Since, like most other teams we perform blob detection in the image to detect interesting colors in the scene, and since this blob detection only depends on the paramaters used to define the colors, we do not adjust the green channel during standard local vision processing. The main reason is that green channel adjustment resulted in a 10ms increase in our video processing loop.

However, we use the green channel adjustment only when displaying or transferring color images for later viewing. The resulting images are more convincing if the colors are adjusted correctly.

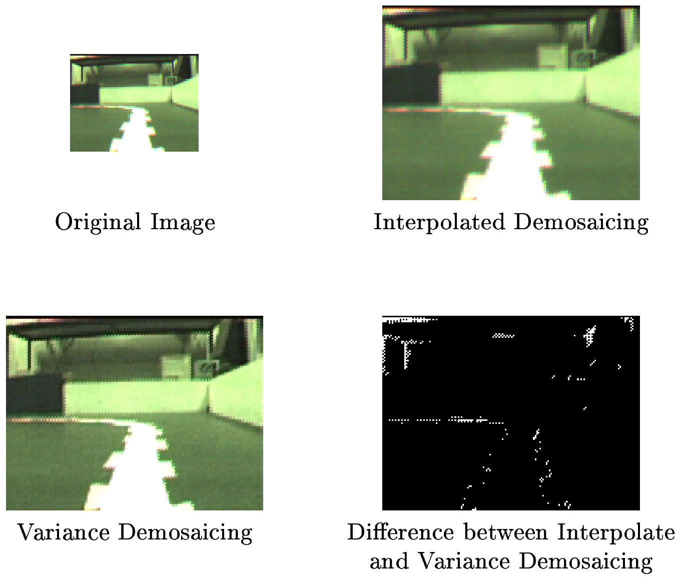


Fig. 6. Track Scene: Comparison of the original image, the interpolate-demoasiced image, and the variance-demosaiced image. The difference between these two methods is also shown.

5 Conclusion

This paper describes two algorithms for the demoasicing of color images: the interpolation and variance demosaicing algorithms. Both algorithms are easy to implement and perform well in practice.

An empirical evaluation showed that the interpolation algorithm results in blurring around the edges and that the variance algorithm can overcome this problem.

However, the paper also showed that the computational cost of the variance algorithm is significantly more than that of interpolated demosaicing.

This trade-off depends then on the required accuracy of image features as well as the available processing power. We currently use only interpolated demosaicing in our Eyebot controllers, which allows us to maintain a framerate of 10 frames per second.

Currently, we are investigating methods for interleaving the image acquisition and demosaicing processes. This would result in a higher frame rates for our vision processing, since some of the cost of demosaicing may be hidden in the image acquisition routine, which currently has to wait for the next byte from the camera.

Of course, more complex and accurate methods of demosaicing can be developed. The gradient information and direction of edges in the image may be used. We do not believe that the improvement would be minimal and does not warrant the additional computational cost.