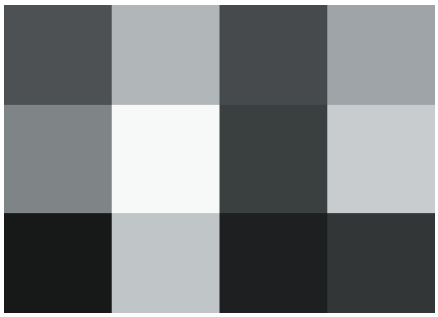
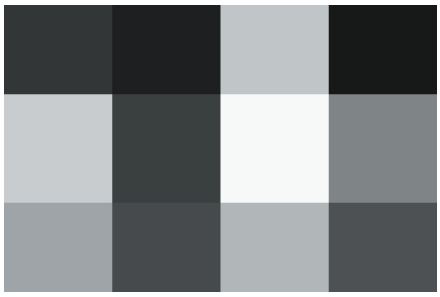
BEHIND THE SCENES OF TODAY'S IMAGE PROCESSING

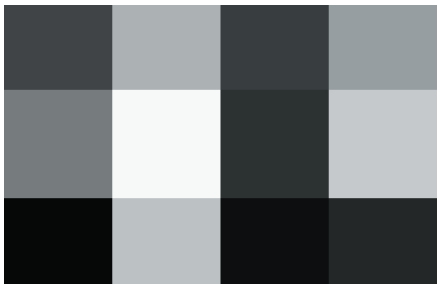
Learn about what goes on behind the user interface of image software packages and how an image is modified to create the final end results.

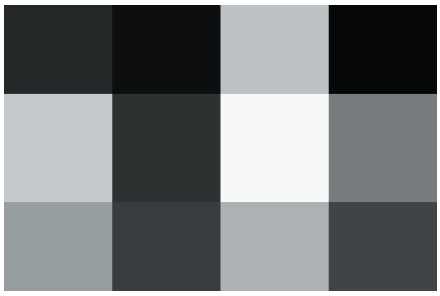
LUMENERA **WHITE PAPER SERIES**

THE MOST IMPORTANT CAMERA PARAMETERS FOR AERIAL IMAGING









WHAT'S INSIDE

INTRODUCTION

Image processing has become so ingrained in everyday life that we often forget how quickly it has advanced over the last few decades. Today, we are able to snap photos and then rotate, crop, and apply various filters to the images directly on the device without any knowledge of photo editing techniques or digital image processing.

Even when it comes to professional-grade image processing, such as with aerial mapping, intelligent security, or medical imaging, both licensed and open source software packages exist to allow users without in-depth knowledge of the processing itself to simply feed in image data and generate orthorectified maps, stitched composite views of a scene, or enhancing x-ray images. This document goes behind the user interface and explores the digital representation of an image and how it is modified to create the end results many have become accustomed to receiving from these software packages.

IMAGE COORDINATE SYSTEM

When referring to a specific pixel in an image, using the Cartesian coordinate system is an obvious choice since pixels are aligned in a grid. Each pixel is given an (x,y) coordinate to illustrate its position. However, in most cases, the origin (0,0) is not at the bottom left of the image. The general convention in image processing is to place the origin at the upper left corner of the image with positive values in x increasing from left to right and positive values in y increasing from top to bottom.

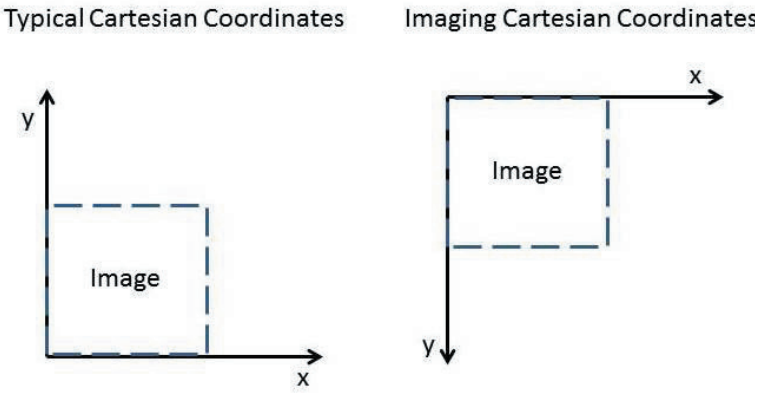


Figure 1. Typical vs Imaging Coordinate System

There are some exceptions to this such as with OpenGL, but the majority of image processing software, including OpenCV, adheres to the convention of using the upper left corner as the origin. This convention dates back to CRT monitors, which swept a beam of electrons across the screen from left to right and top to bottom, mirroring how western languages are written and read. Furthermore, since images are often stored as matrices for image processing (see following sections), this coordinate system is a better match to the indexing convention for this data structure.

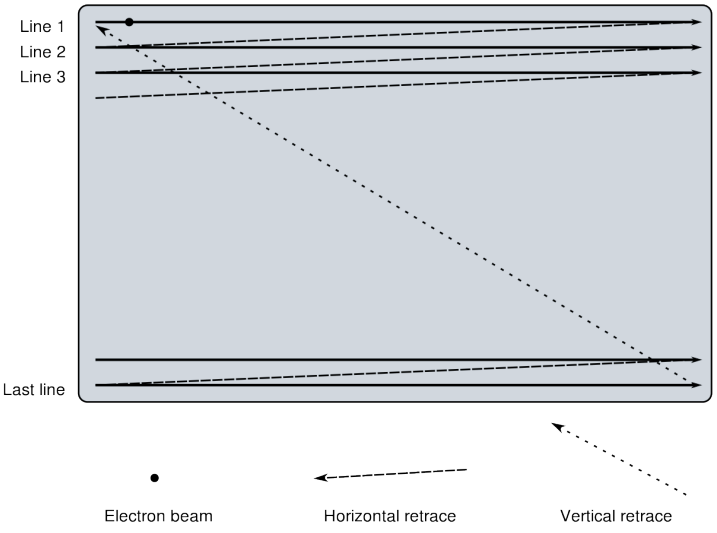


Figure 2. Scanning Pattern Used by CRT Monitors

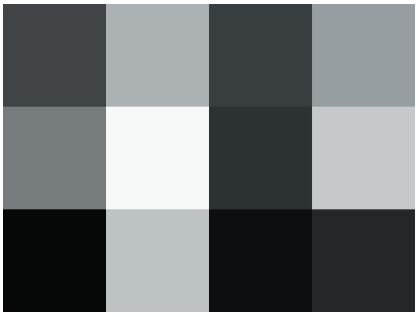
IMAGES AND MATRICES

COORDINATES VS. INDICES

In its rawest form, an image is saved to a matrix (or array) of dimensions equal to that of the sensor's resolution. For example, a 12 megapixel image would be saved to a matrix of 3000 elements tall by 4000 elements wide. When specifying the dimensions of a matrix, the height (y parameter) is given first, followed by the width (x parameter). Each element reports the pixel's value with a certain granularity depending on the bit depth. For illustrative purposes, the 3x4 matrix below (named "A") will be used to represent the 12 megapixel image with a bit depth of 8-bits.

$$A = \begin{bmatrix} 65 & 182 & 57 & 163 \\ 125 & 250 & 43 & 207 \\ 4 & 199 & 11 & 33 \end{bmatrix}$$

Represented as an image, matrix A would look like:



Using the Cartesian coordinate system explained above, pixel (0,0) has the value of 65 and pixel (3,2) has the value of 33. Using matrix indices, the pixel with the value of 65 would be in element $A(_{1,1})$, the pixel with value 33 would be in element $A(_{3,4})$, and the pixel with value 207 would be in element $A(_{2,4})$.

$$A = \begin{bmatrix} 65 & 182 & 57 & 163 \\ 125 & 250 & 43 & 207 \\ 4 & 199 & 11 & 33 \end{bmatrix}$$

Therefore, to transfer from Cartesian coordinates to matrix indices, the Cartesian coordinates need to each be increased by 1 and flipped: $(x,y) = A(_{y+1,x+1})$.

MATRIX MANIPULATIONS

When post processing RAW images, it is important to understand the basics of image manipulation through the use of matrix multiplications. There are a number of ways to multiply matrices such as the dot product, the convolutional product, and the Hadamard product. Image transformations from simple rotations and translations to more complex homography transformations use the dot product. The convolutional product is used when filtering an image to perform operations such as sharpening, blurring, and edge detection. The Hadamard product requires two matrices of the same size to multiply each element by the element of matching index in the second matrix. This elementwise operation concept is used when calculating vegetative indices, such as NDVI, when dividing one image by another.

IMAGE TRANSFORMATIONS

Image transformations move images around without significant changes to the data contained in the image. The manipulations are performed on the pixel's coordinates as opposed to the pixels themselves. A dot product between a transformation matrix and each (x,y) coordinate of the image produces a new set of coordinates to which existing pixels are mapped. In certain transformations, non-integer coordinates are generated, requiring some form of average pixel value from the original image to be mapped to the destination coordinate. Below are a few examples of simple image transformation matrices.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

2x2 Scaling Transformation Matrix:
Scales the image size by factor s

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

2x2 Scaling Transformation Matrix:
Scales the image size by factor s

A similar notation to this is to use a 3x3 matrix as below. The addition of a "1" to the image coordinates does not impact the image itself, but allows for the addition of a translation component to the transformation matrix.

Original Image

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

3x3 Scaling Transformation Matrix:
Scales the image size by factors s

Image Scaled to 50%

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

3x3 Rotation Transformation Matrix:
Rotates the image clockwise by θ degrees

Image Rotated Clockwise 60°

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

3x3 Translation Transformation Matrix:
Moves the image by t pixels

Image Shifted +8 px in y & +7 px in x (Translation Highlighted in Gray)

These individual matrices can be combined to form a single transformation matrix to apply all the image transformations simultaneously. The matrices are simply multiplied with one another using the dot product. Below is an example of a rotation, followed by a scaling, and then a translation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s\cos\theta & -s\sin\theta & t_x s\cos\theta - t_y\, s\sin\theta \\ s\sin\theta & s\cos\theta & t_x\, s\sin\theta + t_y s\cos\theta \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

This format of matrix may be a little daunting at first, but once computed numerically all the terms scale down to numbers. For example, if the image were rotated by 60 degrees, scaled to 50% of its original size, and shifted 8 pixels down and 7 pixels to the right, the transformation matrix would be computed as:

$$\begin{bmatrix} 0.5\cos 60 & -0.5\sin 60 & 8\cdot 0.5\cos 60 - 7\cdot 0.5\sin 60 \\ 0.5\sin 60 & 0.5\cos 60 & 8\cdot 0.5\sin 60 + 7\cdot 0.5\cos 60 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0.25 & -0.43 & -1.03 \\ 0.43 & 0.25 & 5.21 \\ 0 & 0 & 1 \end{bmatrix}$$

Result of Simultaneous Operation Transformation Matrix

Transformation matrices are used quite often behind the scenes in image processing. From common applications like Windows Paint, Adobe Photoshop to specialized photogrammetry software such as OpenCV and Pix4D, transformation matrices are used for simple and complex transformations alike.

Image coregistration is a prime example of a complex transformation matrix. Methods currently exist to take two images captured simultaneously from two different cameras to determine a transformation matrix to align the image data such that pixels with identical coordinates in each image refer to the same point on the ground. Coregistration can be used to align images captured from a mapping system in the air or from an intelligent security system on the ground.

LEARN MORE

Learn more about image coregistration by reading our solution sheet.

IMAGE FILTERING

Filtering image data is done through matrix convolution. A square matrix of odd dimensions (3x3, 5x5, 7x7, etc.) called the kernel is run through the image data matrix. The process involves an elementwise multiplication of the kernel matrix with data from the image matrix. The results from each multiplication are summed and then saved to the coordinates that the center of the kernel matrix occupies over the image data. It is essentially a weighted sum of the nearest neighbors of each pixel. A few examples of kernel matrices are listed below.

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection Matrix

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Sharpen Matrix

$$\begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{bmatrix}$$

Gaussian Blur Matrix

Applying the edge detection matrix (highlighted in green) to element $A(_{2,2})$ from the Coordinates vs Indices section would look like this:

$$\begin{bmatrix} -1 \cdot 65 & -1 \cdot 182 & -1 \cdot 57 & 163 \\ -1 \cdot 125 & 8 \cdot 250 & -1 \cdot 43 & 207 \\ -1 \cdot 4 & -1 \cdot 199 & -1 \cdot 11 & 33 \end{bmatrix}$$

The result for $A'(_{2,2})$ would be 1314. As this is an 8-bit image, the value would be capped at 255, or white, indicating the presence of an edge. Since this is random data, also known as noise, edge detection is not practical in this scenario. However, when applied to a real image, the matrix highlights significant differences in values between neighboring pixels, indicating an edge.

Below is an example of three different kernel matrices used to enhance an x-ray image. The first kernel matrix (a) performs averaging (to help remove noise), the second (b) performs edge enhancement, and the third (c) performs edge detection. For best results, these filters would be used in succession.

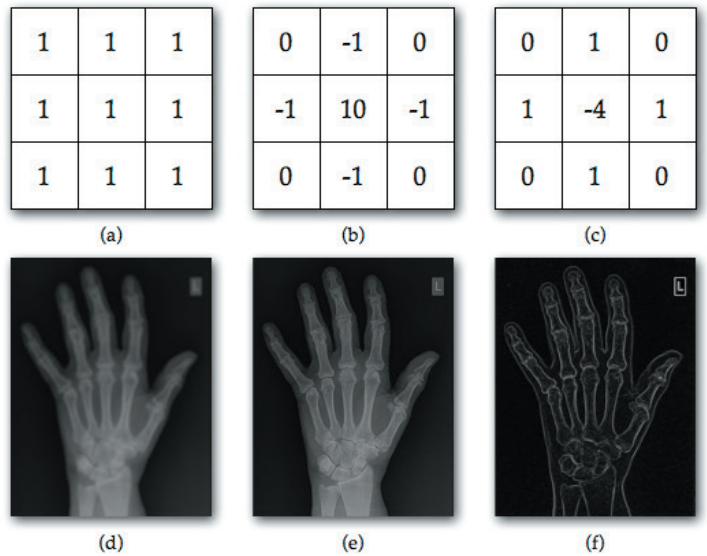


IMAGE CALCULATIONS

Performing calculations on image data can range from multiplying the image by a singular value (known as digital gain) to adding, subtracting, multiplying, or dividing images together. Here, the operation takes place on a pixel by pixel basis, so it is important that the information contained at the same coordinates in each image share a high correlation. For this reason, it is important to perform image coregistration before attempting to perform image calculations with images from two or more cameras.

Multiplying two images together as matrices is known as the Hadamard product and is completed as follows:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \circ \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} a & 2b & 3c \\ 4d & 5e & 6f \\ 7g & 8h & 9i \end{bmatrix}$$

When performing these types of operations in certain image processing software, it is important to specify an image/matrix type that will allow for values higher than the specified bit depth as well as the possibility for negative numbers in the case of subtractions to ensure that no data is clipped.

This is especially important for precision agriculture data processing such as NDVI where one image is subtracted from another to create a vegetation index.

LEARN MORE

To learn about performing NDVI with a single camera, read our solution sheet, or to learn about Broadband Greenness Vegetation Indices, read our blog post.

IMAGE PROCESSING WITH OPENCV

OpenCV is an open source image processing platform with many application-specific solutions either in whole or in part. It allows for easy implementation of image transformations, filtering, and calculations with pre-defined functions while allowing the user to dive deeper and create their own custom image manipulations as described above.

Lumenera cameras interface with OpenCV, among others, using platform-specific SDKs developed for Windows, Linux, and ARM-based embedded systems. Once the images are imported into OpenCV, image processing becomes very straightforward.

There are a number of one-line commands to perform image transformations, such as mirroring the image using flip() or rotating the image using phase(). A more mathematical approach can be taken by using the methods described in the Image Transformations section above, as well as the OpenCV function called transform() where the user provides their own 2x2 or 3x3 transformation matrix. Filtering is just as straightforward in OpenCV with a number of functions that filter the image file, including bilateralFilter() to reduce noise and Laplacian() for edge detection. Image filtering is an incredibly complex topic that requires many more concepts beyond the scope of this document. Nevertheless, this is an excellent starting point to experiment with the usage of image filters.

Finally, performing image calculations in OpenCV is even more straightforward when it comes to addition and subtraction as the standard operators ('+' and '-') are used. For division and multiplication, the functions divide() and multiply() are used. For operations that can be performed in OpenCV, see the OpenCV documentation page: Operations on Arrays.

CONCLUSION

When images are digitally stored in RAW format, they are saved in a matrix with each element representing the pixel's intensity value. These matrices are indexed in the form $A_{(H,V)}$, where H is the pixel's horizontal position and V its vertical position. Contrary to the typical origin location in Cartesian coordinates, the origin of an image file is the upper left hand corner of the image with values increasing horizontally from left to right along the x axis and from top to bottom for the y axis. This helps to ensure that the origin of the image in Cartesian coordinates lines up with the indexing method used for matrices.

On a lower level, modifying an image consists of performing various modifications to the values stored within the matrix or to the structure of the matrix itself. A transformation matrix is used to reassign image coordinates, changing the image's general shape, size, and orientation. Examples of this include a simple clockwise rotation of the image or coregistering two images to one another. A kernel matrix, a square matrix with odd dimensions, is convolved with a matrix to filter an image. A kernel matrix, a square matrix with odd dimensions, is convolved with a matrix to filter an image.

This is how edge detection is performed for computer vision inspection applications. Finally, matrices of identical size can be added, subtracted, multiplied, and divided by each other in an elementwise fashion to analyse the data based on certain mathematical models such as with precision agriculture. Examples of this include the Normalized Difference Vegetation Index (NDVI), Enhanced-NDVI (ENDVI), and Enhanced Vegetation Index (EVI).

If you have additional questions on topics that covered in this document, questions about our cameras, software development kit, or any other imaging concept, reach out to our imaging experts at info@lumenera.com. They will be glad to help.

ABOUT LUMENERA

Lumenera Corporation, a division of Roper Technologies, headquartered in Ottawa, Canada, is a leading developer and manufacturer of high performance digital cameras and custom imaging solutions. Lumenera cameras are used worldwide in a diverse range of industrial, scientific and security applications.

As a global market leader Lumenera provides an extensive range of high quality digital cameras with unique combinations of speed, resolution and sensitivity to satisfy the demands of today's imaging applications.

Lumenera also offers custom design services to OEM partners requiring specialized hardware and software features.

Core competencies include digital bus technologies such as USB 3.1, USB 2.0, Ethernet, HDMI and Gigabit Ethernet (GigE) as well as a complete command of digital imaging hardware and software built around CMOS and CCD based imagers. Our diversity provides our customers with the benefits of superior price-to-performance ratios and faster time-to-market.

Lumenera Corporation
7 Capella Court
Ottawa, ON
Canada, K2E 8A7

613-736-4077
info@lumenera.com
www.lumenera.com