Applications of Linear Algebra in Image Processing

K. V. Huynh, S. Yang

December 19, 2021

1 Introduction

Due to the increase in cameras today, image processing is evermore crucial to everyday usage. Image processing is apparent almost everywhere, if there is a camera. Smart phones have the capability to filter images, causing images to glow more bright or dim the saturation of the image. Along with smart phones, self-driving cars also take advantage of image processing to better its performance.

Linear Algebra plays a major role in computer vision and image processing. In fact the representation of an image is a matrix. As humans, we only see a combination of colors in a specific order. However, to computers, an image is just a matrix, with entries that will delegate which colors we see. And so, the applications of linear algebra ranges from, but doesn't just consist of the manipulation and interpretation of images. This paper will show two examples of linear algebra applied to static images, and to dynamic images(sequence of images). We will use image compression based on Haar Wavelet Transformation as an example to show how linear algebra is applied for static images. Then, we'll explain a technique for motion estimation to showcase some linear algebra for dynamic images.

2 Haar Wavelet Transform

2.1 Haar Wavelet on single-row matrix

Suppose we have a 8×8 matrix as follow:

$$[1, 2, 3, 4, 20, 22, 24, 28]$$

The Haar Wavelet transformation is calculated in three steps. First, we will pair up the pixel to have a total of 4 pairs

$$[[1, 2], [3, 4], [20, 22], [24, 28]]$$

The next step is to calculate the average of each pairs as well as the difference between the first entry of each pairs with the average. The calculation is illustrated as follow:

$$[[\frac{1+2}{2}, \frac{3+4}{2}, \frac{20+22}{2}, \frac{24+28}{2}], [1-\frac{1+2}{2}, \frac{3+4}{2}, 3-\frac{3+4}{2}, 20-\frac{20+22}{2}, 24-\frac{24+28}{2}]]$$

$$[[1.5, 3.5, 21, 26], [-0.5, -0.5, -1, -2]]$$

The first four entries of the matrix is called the approximation coefficients and the last 4 are called detail coefficients. Next step, we will repeat the process but with only the first four entries.

$$[[2.5, 23.5], [-1, -2.5], [-0.5, -0.5, -1, -2]]$$

We then finish the three steps by repeating the process again for the first two entries

$$[[13, -10.5], [-1, -2.5], [-0.5, -0.5, -1, -2]]$$

Finally, we obtained the transform matrix

$$[[13, -10.5], [-1, -2.5], [-0.5, -0.5, -1, -2]]$$

2.2 8×8 matrix case analysis

The question now is how this method of compressing images apply to real-world images. Suppose we have an 8×8 grayscale images with the matrix representations as follow:

$$\begin{bmatrix} 90 & 68 & 72 & 92 & 35 & 40 & 42 & 49 \\ 92 & 87 & 72 & 63 & 21 & 15 & 29 & 45 \\ 29 & 83 & 88 & 32 & 65 & 43 & 22 & 44 \\ 12 & 28 & 39 & 55 & 92 & 78 & 54 & 85 \\ 26 & 63 & 45 & 58 & 98 & 65 & 56 & 65 \\ 76 & 65 & 89 & 45 & 38 & 54 & 86 & 92 \\ 92 & 65 & 66 & 34 & 54 & 77 & 88 & 33 \\ 32 & 92 & 83 & 88 & 45 & 37 & 32 & 65 \end{bmatrix}$$

We will apply the same process for each row of the matrix. After finishing the task, we then do the same process for the columns of the matrix. Finishing up, our final compressed images will be

$$\begin{bmatrix} \frac{3765}{64} & \frac{277}{64} & -\frac{21}{32} & -\frac{15}{16} & -\frac{51}{8} & \frac{87}{16} & \frac{39}{16} & -\frac{69}{16} \\ -\frac{243}{64} & \frac{209}{64} & -\frac{27}{32} & \frac{17}{8} & 1 & -\frac{29}{16} & \frac{35}{16} & -\frac{83}{16} \\ \frac{63}{32} & \frac{477}{32} & \frac{25}{4} & -\frac{127}{16} & \frac{97}{8} & -\frac{51}{8} & -\frac{35}{8} & \frac{15}{4} \\ \frac{19}{16} & -\frac{13}{2} & -\frac{17}{16} & -\frac{39}{16} & \frac{7}{8} & \frac{1}{2} & 4 & -\frac{37}{8} \\ 4 & -3 & -\frac{25}{4} & \frac{11}{4} & \frac{17}{4} & -\frac{29}{4} & -\frac{11}{4} & \frac{9}{4} \\ -\frac{37}{16} & \frac{233}{16} & \frac{23}{4} & \frac{11}{8} & -\frac{19}{2} & 18 & 2 & \frac{9}{4} \\ -\frac{69}{16} & -\frac{97}{16} & -\frac{21}{8} & 16 & -12 & -\frac{57}{4} & \frac{49}{4} & -\frac{3}{4} \\ \frac{35}{16} & -\frac{111}{16} & 13 & \frac{25}{8} & \frac{87}{4} & \frac{37}{4} & -\frac{31}{4} & 22 \end{bmatrix}$$

2.3 Linear Algebra Approach

2.3.1 8×8 matrix example

The linear algebra, especially matrix multiplication, plays an important role in how the Haar wavelet transform work. In this section, we will demonstrate how matrix multiplication could be use to actually find out all the approximation coefficients and detail coefficients. Let us define W_1, W_2, W_3 to be the following matrices

$$W_1 = \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} \end{bmatrix}$$

$$W_2 = \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$W_3 = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Hence computing $W = W_1W_2W_3$, we then have

$$W = W_1W_2W_3 = \begin{bmatrix} \frac{1}{8} & \frac{1}{8} & \frac{1}{4} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{8} & \frac{1}{8} & \frac{1}{4} & 0 & -\frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{8} & \frac{1}{8} & -\frac{1}{4} & 0 & 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{8} & \frac{1}{8} & -\frac{1}{4} & 0 & 0 & -\frac{1}{2} & 0 & 0 \\ \frac{1}{8} & -\frac{1}{8} & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{2} & 0 \\ \frac{1}{8} & -\frac{1}{8} & 0 & \frac{1}{4} & 0 & 0 & -\frac{1}{2} & 0 \\ \frac{1}{8} & -\frac{1}{8} & 0 & -\frac{1}{4} & 0 & 0 & 0 & \frac{1}{2} \\ \frac{1}{8} & -\frac{1}{8} & 0 & -\frac{1}{4} & 0 & 0 & 0 & -\frac{1}{2} \end{bmatrix}$$

Hence we can obtain the transform matrix, denoted B, $B = W^TAW$ and it will yield the same results as mentions in the above sections

$$B = W^TAW = \begin{bmatrix} \frac{3765}{64} & \frac{277}{64} & -\frac{21}{32} & -\frac{15}{16} & -\frac{51}{8} & \frac{87}{16} & \frac{39}{16} & -\frac{69}{16} \\ -\frac{243}{64} & \frac{209}{64} & -\frac{27}{32} & \frac{17}{8} & 1 & -\frac{29}{16} & \frac{35}{16} & -\frac{83}{16} \\ \frac{63}{32} & \frac{477}{32} & \frac{25}{4} & -\frac{127}{16} & \frac{97}{8} & -\frac{51}{8} & -\frac{35}{8} & \frac{15}{4} \\ \frac{19}{16} & -\frac{13}{2} & -\frac{17}{16} & -\frac{39}{16} & \frac{7}{8} & \frac{1}{2} & 4 & -\frac{37}{8} \\ 4 & -3 & -\frac{25}{4} & \frac{11}{4} & \frac{17}{4} & -\frac{29}{4} & -\frac{11}{4} & \frac{9}{4} \\ -\frac{37}{16} & \frac{233}{16} & \frac{23}{4} & \frac{11}{8} & -\frac{19}{2} & 18 & 2 & \frac{9}{4} \\ -\frac{69}{16} & -\frac{97}{16} & -\frac{21}{8} & 16 & -12 & -\frac{57}{4} & \frac{49}{4} & -\frac{3}{4} \\ \frac{35}{16} & -\frac{111}{16} & 13 & \frac{25}{8} & \frac{87}{4} & \frac{37}{4} & -\frac{31}{4} & 22 \end{bmatrix}$$

2.3.2 Larger Square Matrix

How does Haar Wavelet works for $n \times n$ matrices in general. Assuming that there exists a $k \in \mathbb{N}$ such that $n = 2^k$. Now the matrices can be represented as $2^k \times 2^k$. It is quite obviously that $W = W_1W_2 \dots W_k$. However, the main questions is how can we construct matrix W_i for $i \in [1, k]$. We have come up with a general algorithms to generate these matrix. Suppose we want to generate matrix M_t for $t \in [1, k]$, then the following algorithms would be applied

Algorithm 1: Construction of Matrix M_t

Let $M_t = [a_{ij}]$ for $i, j \in [1, k]$
if *i AND $j > 2^t$* **then**
  if $i = j$ **then**
    $a_{ij} = 1$
  end
  if $i \neq j$ **then**
    $a_{ij} = 0$
  end
end
if *$(i < 2^t$ AND $j > 2^t)$ OR $(i > 2^t$ AND $j < 2^t)$* **then**
  $a_{ij} = 0$
end
if *$(\alpha(a_{ij}) \leq 2^t)$* **then**
  Let $temp = 0$
  for $\beta \in [1, 2^t]$ **do**
    if *β mod* $2 = 1$ **then**
      $a_{ij} = \frac{1}{2}$
      $a_{(i+2^t)(j+2^t)} = \frac{1}{2}$
      for $i = \beta$ and $j = i + temp$
    end
    $temp = temp + 1$
    if *β mod* $2 = 0$ **then**
      Copy all the entries from previous row except the second $\frac{1}{2}$. Change it to $-\frac{1}{2}$
    end
  end
end
The rest of the entries which is not in the cases above will be 0

As we can see, the matrix W is orthogonal as well as invertible. It is routinely to find out each of W_i to construct W using Algorithm 1. Suppose we have a $2^r \times 2^r$ matrix A. Let B to be the *after transformed* matrix. We then have

$$B = W^T AW = ((AW)^T W)^T \text{ and } A = (W^{-1})^T BW^{-1} = ((B^T)W^{-1})^T W^{-1}$$

2.4 Applications - Lossless and Lossy Image Compression

For lossy image compression, we will fix a threshold, denoted ϵ. Whenever we find a detail coefficient that has the value less than ϵ, we then change that coefficient to be 0. For instance, set a threshold $\epsilon = 1$. take our matrix B from section 3.3.1

$$B = \begin{bmatrix} \frac{3765}{64} & \frac{277}{64} & -\frac{21}{32} & -\frac{15}{16} & -\frac{51}{8} & \frac{87}{16} & \frac{39}{16} & -\frac{69}{16} \\ -\frac{243}{64} & \frac{209}{64} & -\frac{27}{32} & \frac{17}{8} & 1 & -\frac{29}{16} & \frac{35}{16} & -\frac{83}{16} \\ \frac{63}{32} & \frac{477}{32} & \frac{25}{4} & -\frac{127}{16} & \frac{97}{8} & -\frac{51}{8} & -\frac{35}{8} & \frac{15}{4} \\ \frac{19}{16} & -\frac{13}{2} & -\frac{17}{16} & -\frac{39}{16} & \frac{7}{8} & \frac{1}{2} & 4 & -\frac{37}{8} \\ 4 & -3 & -\frac{25}{4} & \frac{11}{4} & \frac{17}{4} & -\frac{29}{4} & -\frac{11}{4} & \frac{9}{4} \\ -\frac{37}{16} & \frac{233}{16} & \frac{23}{4} & \frac{11}{8} & -\frac{19}{2} & 18 & 2 & \frac{9}{4} \\ -\frac{69}{16} & -\frac{97}{16} & -\frac{21}{8} & 16 & -12 & -\frac{57}{4} & \frac{49}{4} & -\frac{3}{4} \\ \frac{35}{16} & -\frac{111}{16} & 13 & \frac{25}{8} & \frac{87}{4} & \frac{37}{4} & -\frac{31}{4} & 22 \end{bmatrix}$$

We then end up with the matrix B_l

$$B_l = \begin{bmatrix} \frac{3765}{64} & \frac{277}{64} & -\frac{21}{32} & 0 & -\frac{51}{8} & \frac{87}{16} & \frac{39}{16} & -\frac{69}{16} \\ -\frac{243}{64} & \frac{209}{64} & 0 & \frac{17}{8} & 0 & -\frac{29}{16} & \frac{35}{16} & -\frac{83}{16} \\ \frac{63}{32} & \frac{477}{32} & \frac{25}{4} & -\frac{127}{16} & \frac{97}{8} & -\frac{51}{8} & -\frac{35}{8} & \frac{15}{4} \\ \frac{19}{16} & -\frac{13}{2} & -\frac{17}{16} & -\frac{39}{16} & 0 & 0 & 4 & -\frac{37}{8} \\ 4 & -3 & -\frac{25}{4} & \frac{11}{4} & \frac{17}{4} & -\frac{29}{4} & -\frac{11}{4} & \frac{9}{4} \\ -\frac{37}{16} & \frac{233}{16} & \frac{23}{4} & \frac{11}{8} & -\frac{19}{2} & 18 & 2 & \frac{9}{4} \\ -\frac{69}{16} & -\frac{97}{16} & -\frac{21}{8} & 16 & -12 & -\frac{57}{4} & \frac{49}{4} & 0 \\ \frac{35}{16} & -\frac{111}{16} & 13 & \frac{25}{8} & \frac{87}{4} & \frac{37}{4} & -\frac{31}{4} & 22 \end{bmatrix}$$

Apply the reconstruction formula, we then obtain

$$A_a = (W^T)^{-1} B_l W^{-1} = \begin{bmatrix} 89 & 69 & 72 & 92 & \frac{575}{16} & \frac{655}{16} & \frac{657}{16} & \frac{769}{16} \\ 91 & 88 & 72 & 63 & \frac{351}{16} & \frac{255}{16} & \frac{449}{16} & \frac{705}{16} \\ 28 & 84 & 88 & 32 & \frac{1055}{16} & \frac{703}{16} & \frac{337}{16} & \frac{689}{16} \\ 11 & 29 & 39 & 55 & \frac{1487}{16} & \frac{1263}{16} & \frac{849}{16} & \frac{1345}{16} \\ \frac{209}{8} & \frac{503}{8} & \frac{89}{2} & \frac{117}{2} & \frac{1583}{16} & \frac{1055}{16} & \frac{893}{16} & \frac{1013}{16} \\ \frac{609}{8} & \frac{519}{8} & \frac{177}{2} & \frac{91}{2} & \frac{623}{16} & \frac{879}{16} & \frac{1349}{16} & \frac{1469}{16} \\ \frac{751}{8} & \frac{505}{8} & \frac{133}{2} & \frac{67}{2} & \frac{879}{16} & \frac{1247}{16} & \frac{1393}{16} & \frac{513}{16} \\ \frac{271}{8} & \frac{721}{8} & \frac{167}{2} & \frac{175}{2} & \frac{735}{16} & \frac{607}{16} & \frac{497}{16} & \frac{1025}{16} \end{bmatrix}$$

For lossless image compression, $B_t = B$ or the threshold $\epsilon = 0$

3 Motion Estimation

When playing baseball, we can estimate the motion of the ball with relative ease. This estimation of the ball's motion gives us enough information on how we should be positioning our hands to catch the ball. Unfortunately, this is not the case for computers since they see video footage as a sequence of matrices with numeric values, representing colors, as entries to the matrices. When an object moves, a computer only knows that the numeric values have changed, and values don't just change due to a change in motion. The values of an image can change because of a change in lighting. Even for us, if we only saw a matrix of numeric values change, we also wouldn't understand what has changed in the footage. And so, computer scientists and mathematicians have been trying to estimate motion in a sequence of images.

So, how is motion estimation implemented? There two ways to estimate motion. One method is feature tracking. Feature tracking is essentially taking a patch of pixels on an image and then searching the next image, near the same area, for the same patch of pixels. Block matching is an algorithm used for searching[1]. Once the computer finds the same patch, you can determine the change in motion by taking the difference between the patch of pixels in the new image and in the previous image. This paper will be discussing another method call differential motion. Differential motion detects motion in a sequence of images by utilizing the gradients of images in four dimensions.

3.1 Methodology on differential motion

We denote a pixel as $f(x, y)$ where x and y are the position of the pixel on a single image. You can envision this function as a surface in 3-dimension where $f(x, y)$ are the values representing colors. To denote a sequence of pixels, we just add the variable time which gives us $f(x(t), y(t), t)$. The

parameters are $x(t), y(t)$ and t where t is the time that the image occurs. The parameters $x(t)$ and $y(t)$ is the position of an individual pixel of an image at time t. We want to figure out the change of pixels overtime. To make everything work, the brightness constancy assumption must be made. this assumption is best explain with an example. We have a footage of a ball rolling from one side of a room to the other side. We will assume that the lighting of the ball will not change. The assumption is that all pixel intensity will stay the same over time. Essentially,

$$\frac{\partial f(x(t), y(t), t)}{\partial t} = 0$$

At first glance, this assumption may seem to stray away from reality. Going back to our example of the ball, it's unreasonable to assume that the lighting of the ball will stay the same. However, we must take into consideration how an image sequence operates. A sequence of images are taken at a very fast pace with only a small time difference between the images. And so it's not preposterous to believe that there's little or no difference in lighting between a image and the image taken afterwards. Using chain rule, we get,

$$\frac{\partial f}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t} + \frac{\partial f}{\partial t} = 0$$

Since $\frac{\partial x}{\partial t}$ is the change in the position of the pixel over time, it is the velocity of the pixel in the x direction over time. Similarly, $\frac{\partial y}{\partial t}$ is the velocity of the pixel in the y direction over time. And so we derive

$$\frac{\partial f}{\partial x}v_x + \frac{\partial f}{\partial y}v_y + \frac{\partial f}{\partial t} = 0$$

$$f_x v_x + f_y v_y + f_t = 0$$

$$\begin{bmatrix} f_x & f_y \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} + f_t = 0$$

$$\begin{bmatrix} f_x & f_y \end{bmatrix} v + f_t = 0$$

The pixel's motion is defined by v, which we must find. Fortunately, we have enough information to determine f_x and f_y. To find f_x, we set y and t constant to determine f_x. And similarly, we find f_y by setting x and t constant. When we set t to be constant, we're basically just looking at the change in x or y of one image, making it possible to calculate f_x and f_y. We also know f_t since that's just the change in value of f at position x and y over time. However, we do not have information on v_x or v_y since it's difficult to determine the change in x over time for a single pixel. If the computer knew the x parameter of the pixel in the next frame, we could easily find v_x, but it's hard for computers. Identifying a pixel on a image, then identifying the same pixel on the next image is easy for us humans, but hard for computers since computers read images as a matrix of numeric values. Finding v_y is also difficult for the same reasoning. So how do we get v?

We can get v by increasing our constraints. Currently, we only have constraints f_x and f_y. We can make the assumption that a pixel and its neighboring pixels also have the same v to get more constraints. Instead of using one pixel, we use a patch of pixels. This assumption isn't far-fetch if we look back at our ball example. If we were to look at a pixel in the middle of the ball, then its 8 surrounding pixel should also be apart of the ball. Hence, have the same v. With this assumption in place, we can derive the following equation:

$$\begin{bmatrix} f_x(x_1,y_1) & f_y(x_1,y_1) \\ f_x(x_2,y_2) & f_y(x_2,y_2) \\ \vdots & \vdots \\ f_x(x_9,y_9) & f_y(x_9,y_9) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} + \begin{bmatrix} f_t(x_1,y_1) \\ f_t(x_2,y_2) \\ \vdots \\ f_t(x_9,y_9) \end{bmatrix} = 0$$

$$Av + b = 0$$

In a perfect world, we can easily solve for $Av = -b$, but there might not be a solution for v. Instead, we have to find v such that we minimize $E(v) = ||Av + b||^2$ so that $Av + b$ will be as close to zero as possible. Since $||Av+b||^2$ is a quadratic function, the smallest value to $E(v)$ will be when $\frac{dE(v)}{dv} = 0$, so

$$\frac{dE(v)}{dv} = 2A^T(Av + b) = 0$$
$$2A^TAv + 2A^Tb = 0$$
$$A^TAv + A^Tb = 0$$
$$A^TAv = -A^Tb$$
$$v = -(A^TA)^{-1}A^Tb$$

We can now get v by calculating this equation. The only issue we will have is if A^TA isn't invertible.

3.2 Linear Algebra

In order to get v, we must confirm that the 2 by 2 matrix A^TA is invertible. We know A^TA is invertible if the matrix is injective and surjective. For the matrix to be injective, the columns of the matrix must be linear independent. Since we have square matrix, if the columns of the matrix are linear independent, then the columns of the matrix must span $\mathbb{R}^2$, confirming that our matrix is surjective. If the matrix A^TA is surjective and injective, then we can calculate v. We can uncover more of what A^TA looks like to better understand when A^TA isn't invertible.

$$A^TA = \begin{bmatrix} f_x(x_1,y_1) & f_x(x_2,y_2) & \dots & f_x(x_9,y_9) \\ f_y(x_1,y_1) & f_y(x_2,y_2) & \dots & f_x(x_9,y_9) \end{bmatrix} \begin{bmatrix} f_x(x_1,y_1) & f_y(x_1,y_1) \\ f_x(x_2,y_2) & f_y(x_2,y_2) \\ \vdots & \vdots \\ f_x(x_9,y_9) & f_y(x_9,y_9) \end{bmatrix} = \begin{bmatrix} \sum f_x^2 & \sum f_yf_x \\ \sum f_xf_y & \sum f_y^2 \end{bmatrix}$$

As we stated, in order for A^TA to be invertible, the columns must be linearly independent. Figure 1 shows four points exhibiting three examples(the stars) of patches of pixels that aren't invertible and an example of when a patch of pixel(the circle) is invertible. Star 1 is invertible because there is not change in intensity of color on the x axis and on the y axis. Hence, $f_y = f_x = 0$, and so $A^TA = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$. Since A^TA has entries that are zeros at Star 1, there does not exist inverse when in a situation like Star 1. This is similar with Star 2 as well. The change in intensity of color on the x axis is zero, and so we get that $A^TA = \begin{bmatrix} 0 & 0 \\ 0 & \sum f_y^2 \end{bmatrix}$. Since we have a column of zeros, the columns of this matrix does not span $\mathbb{R}^2$. Thus, Star 2 isn't invertible. Star 3 also isn't invertible because

there is only a change in one direction which will get us another rank one matrix. We want patches of pixels at corners such as the circle in Figure 1. At corners, we have distinct derivatives for the x and y. Once we've figured out if $A^T A$ is invertible, we can then calculate the change in motion of the patch of pixel.

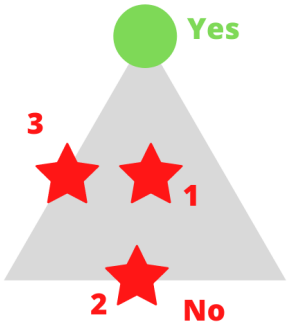

Figure 1: Examples of pixel locations where motion can and can't be computed

3.3 Applications of motion estimation

3.3.1 Optical flow in robotics

Unmanned robotic vehicles has had an explosive increase in the recent years. They can perform tasks such as delivery, surveillance, and assist in search and rescue missions. With the increase in unmanned robotic vehicles, more complex tasks being given to these autonomous robots. One of the difficult problems that robots face is navigating in problematic environments deprive of GPS. One solution to such a problem is the usage of optical flow techniques, inspired from how birds and insect perceive to avoid obstacles and navigate.

Optical flow is the apparent change in motion of objects when there is a change in motion of the observer or the scene. For instance, when in a moving car, a person inside may perceive the outside world as moving. The motion of the outside world is the optical flow. Various algorithms have been developed for computing optical flow. Two such algorithms are differential motion and feature tracking, as mentioned in this paper.Optical flow techniques have been able to greatly improve robotic navigation in troublesome environments. This includes helping with distance estimation, obstacle avoidance, altitude estimation, velocity estimation, landing, and altitude holds [5].

3.3.2 Cell tracking with motion estimation

Motion estimation techniques have also found their way into biology as well. At Kyushu University in Japan, a study used motion estimation techniques to for analyzing cell behavior. Manual tracking of cells can take long hours, and so having a program do such a task is easier. The problem that the researchers were attempting to tackle was the issue of using motion estimation techniques with low frame rates. With a low frame rate, the time difference between when an image is taken and when the next image will be taken is high. This can make it difficult to determine where a single cell went, because there's so much time that passes without knowledge of what the current situation of the cells is. Instead of using algorithms that this paper has mentioned above, the researchers developed

their own method for motion estimation that takes advantage of neural networks. This is one of many examples of how motion estimation is being applied.[6]

3.3.3 Motion estimation in reading lips

Another application of motion estimation is in reading lips. Currently, computers are able to interpret speech and convert speech to text. However, this becomes difficult when there are noise apart from the speech. Thus, a solution was created involving motion estimation to read lips.

A study in King Saud University used a block matching approach to create a lip reading user authentication system. Their system is separated into two phases. The first phase is the training phase where they extract features from images. An image is separated into sections or blocks. The motion of each blocks from one image to another in a sequence will be stored and used for the second phase. The second phase is using the stored motion of the training phase to recognize lip reading. [7]

4 Conclusion

Linear algebra is an important piece that allows these two applications to be what they are. In both methods, matrices are used as a representation for images. In Haar Wavelet image compression, the Haar Wavelet Transform Matrix plays a significant role in how the grayscale image compression and decompression. On a larger scale, there have been more works in the field to apply the method to RGB images. At the same time, the Haar Wavelet Matrix also provide us a way to do progressive image transmission. In motion estimation, linear algebra is used to determine whether or not the velocity of patches of pixels can be calculated. The calculation of the velocity of the patches of pixels is also calculated using linear algebra. These two examples aren't the only examples where linear algebra is applied in image process as there are many more out there.

References

[1] Dazhi, Zhang, et al (2013). Block Matching Algorithms: Their Applications and Limitations in Solar Irradiance Forecasting. *Energy Procedia*, Elsevier https://www.sciencedirect.com/science/article/pii/S1876610213000842.

[2] Mulcahy, Cohm (1997). Image Compression Using the Haar Wavelet Transform.

[3] Haar Wavelet Image Compression *Ohio State University* https://people.math.osu.edu/husen.1/teaching/572/image_comp.pdf.

[4] Fundamentals of Image Processing. *UC Berkeley* https://farid.berkeley.edu/downloads/tutorials/fip.pdf.

[5] Chao, Haiyang, et al (2013). A Survey of Optical Flow Techniques for Robotics Navigation Applications.. *Journal of Intelligent and Robotic Systems*, vol. 73, no. 1-4, pp. 361–372 https://doi.org/10.1007/s10846-013-9923-6

[6] Hayashida, Junya, and Ryoma Bise (2019). Cell Tracking with Deep Learning for Cell Detection and Motion Estimation in Low-Frame-Rate. *Lecture Notes in Computer Science*, pp. 397–405. https://doi.org/10.1007/978-3-030-32239-7_44.

[7] Alghathbar, Khalex and Mahmoud, Hanan A. (2009) *WSEAS Transaction on Information Science and Applications*, Issue 5, Volume 6. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.595.5297&rep=rep1&type=pdf