Department of Electronic and Telecommunication Engineering

University of Moratuwa, Sri Lanka

MA2033 - Linear Algebra

Applications of Linear Algebra

Project Report

Submitted by

Thalagala B.P. 180631J

Submitted on

March 7, 2021

Contents

**** PDF is clickable***

1 Computer Vision: Detecting Corners/Features of an Image

Related Concepts in Linear Algebra: *Eigenvalues, Eigenvectors, Characteristic Polynomial, Quadratic Forms, Principal Axis Theorem, Orthogonal matrix, Diagonalization*

1.1 Problem Identification

In computer vision field finding corners/features in an image($I(x, y)$) is a key operation which is done prior to many advanced applications such as feature mapping, panorama stitching and 3D reconstruction. A corner can be identified as a point in an image whose neighborhood is locally unique to the given image. When comparing this neighborhood with any of the surrounding neighborhoods, significant difference can be observed between them.

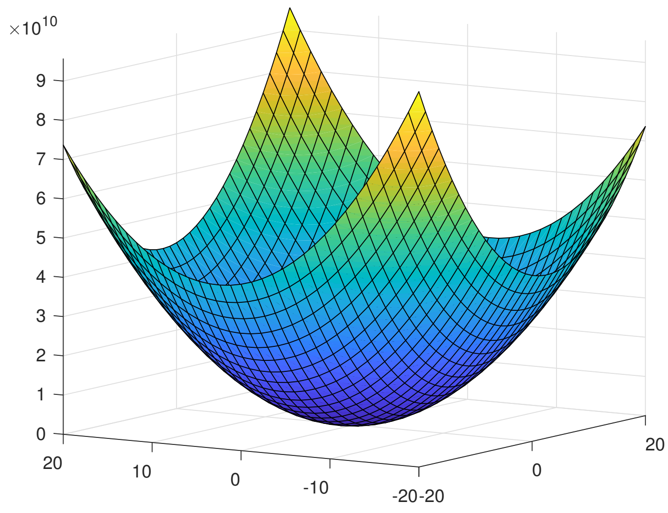
This difference can be quantified mathematically using the following function evaluated over a region called a window(W). Here the most basic Error function is used to make the explanation easy, and this may vary in different implementation of corner detection algorithms.

$$E(u, v) = \sum_{(x,y)\in W} [I(x + u, y + v) - I(x, y)]^2 \tag{1}$$

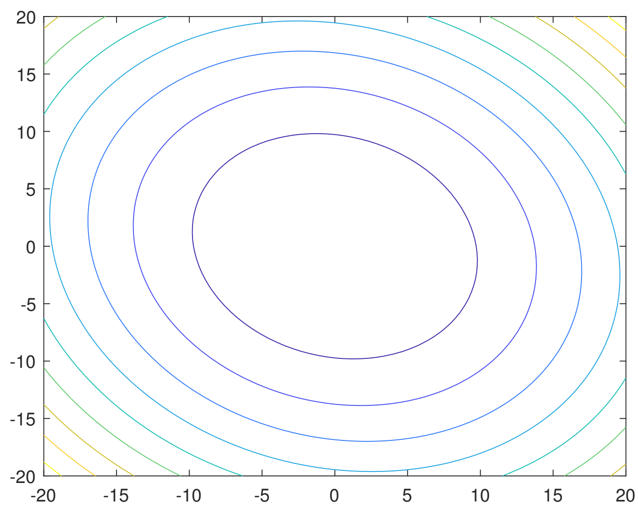
This function can be further simplified using Taylor series and can be rearranged to get an expression in the ***Quadratic form associated with*** M where the M is called the ***Second Moment Matrix*** which consists of summation of quadratic forms of gradients of the image over a window centered at the point of interest. Here gradient of the image along x axis is I_x while I_y is the gradient along y axis.

$$\begin{aligned} E(u, v) &= \sum_{(x,y)\in W} [I_x^2.u^2 + 2.I_x.I_y.u.v + I_y^2.v^2] \\ &= \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} \sum_{(x,y)\in W}[I_x^2] & \sum_{(x,y)\in W}[I_xI_y] \\ \sum_{(x,y)\in W}[I_xI_y] & \sum_{(x,y)\in W}[I_y^2] \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \\ &= \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} \\ E(\underline{x}) &= \underline{x}^\top M \underline{x} \end{aligned} \tag{2}$$

The plot of the above function looks like follows for a given point where the $\sum_{(x,y)\in W}[I_x^2] = 105983042.0$, $\sum_{(x,y)\in W}[I_xI_y] = 13718014.0$ and $\sum_{(x,y)\in W}[I_y^2] = 105671174.0$[7]. (*These values correspond to an actual corner in the Figure 2*). If the point of interest is a corner it will give a very large Error ($E(u, v)$) value even for a very low (u, v) shift in any direction. This can be intuitively recognized as a steep surface(more narrower bowl shape) as depicted in Figure 1(a).



(a) Surface Plot of $E(u, v)$



(b) Contour Plot(Horizontal Slices) of $E(u, v)$

Figure 1: Error Function E(u,v)

1.2 Solution through concepts of Linear Algebra

This narrowness can be mathematically quantified by examining a horizontal slice of the surface plot in figure 1(a). Consider one particular contour in Figure 1(b), then along that contour, function value is a constant i.e $E(\underline{x}) = constant = \underline{x}^\top M\underline{x}$ and then the above Expression 2 represents equation of an ellipse. *If the surface is narrow, the radii of the ellipse must be small for a given $E(u, v)$ value.* That is function value must increase rapidly as we moves outward from the point of interest. Lengths of the major and minor axes(radii) of the ellipse are given by $\lambda_{min}^{-\frac{1}{2}}$ and $\lambda_{max}^{-\frac{1}{2}}$ respectively, where the λ_{min} and λ_{max} are the ***eigenvalues associated with the second moment matrix*** M. We can prove this as follows[5].

Since M is a symmetric matrix, an ***Orthogonal($P^\top = P^{-1}$) matrix*** P can be found such that $\underline{x} = P\underline{y}$ which transforms $E(\underline{x}) = \underline{x}^\top M\underline{x}$ into the form $E(\underline{y}) = \underline{y}^\top D\underline{y}$ where D is a diagonal matrix consists of the eigenvalues of the M. The columns of matrix P consists of the ***eigenvectors*** associated with the eigenvalues of the second moment matrix M.

$$\begin{aligned}
E(\underline{x}) &= \underline{x}^\top M\underline{x} \\
&= (P\underline{y})^\top M(P\underline{y}) \\
&= (\underline{y}^\top P^\top)M(P\underline{y}) \\
&= \underline{y}^\top (P^\top MP)\underline{y} \\
&= \underline{y}^\top D\underline{y} \\
E(y_1, y_2) &= \begin{bmatrix} y_1 & y_2 \end{bmatrix} \begin{bmatrix} \lambda_{min} & 0 \\ 0 & \lambda_{max} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}
\end{aligned} \tag{3}$$

Comparing result of Expression 3 with the standard form of an ellipse,

$$\begin{aligned}
\frac{x^2}{a^2} + \frac{y^2}{b^2} &= \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1/a^2 & 0 \\ 0 & 1/b^2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \\
&= \begin{bmatrix} y_1 & y_2 \end{bmatrix} \begin{bmatrix} \lambda_{min} & 0 \\ 0 & \lambda_{max} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}
\end{aligned} \tag{4}$$

Therefore, $1/a^2 = \lambda_{min}$ and $1/b^2 = \lambda_{max}$. Which yields that $a = \lambda_{min}^{-\frac{1}{2}}$ and $b = \lambda_{max}^{-\frac{1}{2}}$.

Therefore to find corners in an image all we have to do is find the ***eigenvalues*** associated with each second moment matrix for each pixel(x, y) in an image. According to the magnitudes of eigenvalues following classification can be done.

a.) **At a Corner:** $E(u, v)$ increases rapidly in any direction and therefore *both eigenvalues are large.*

b.) **At an Edge:** $E(u, v)$ increases rapidly only in one direction and therefore *one eigenvalue is relatively larger than the other.*

c.) **At a flat area:** $E(u, v)$ does not increase rapidly and therefore *both eigenvalues are small.*

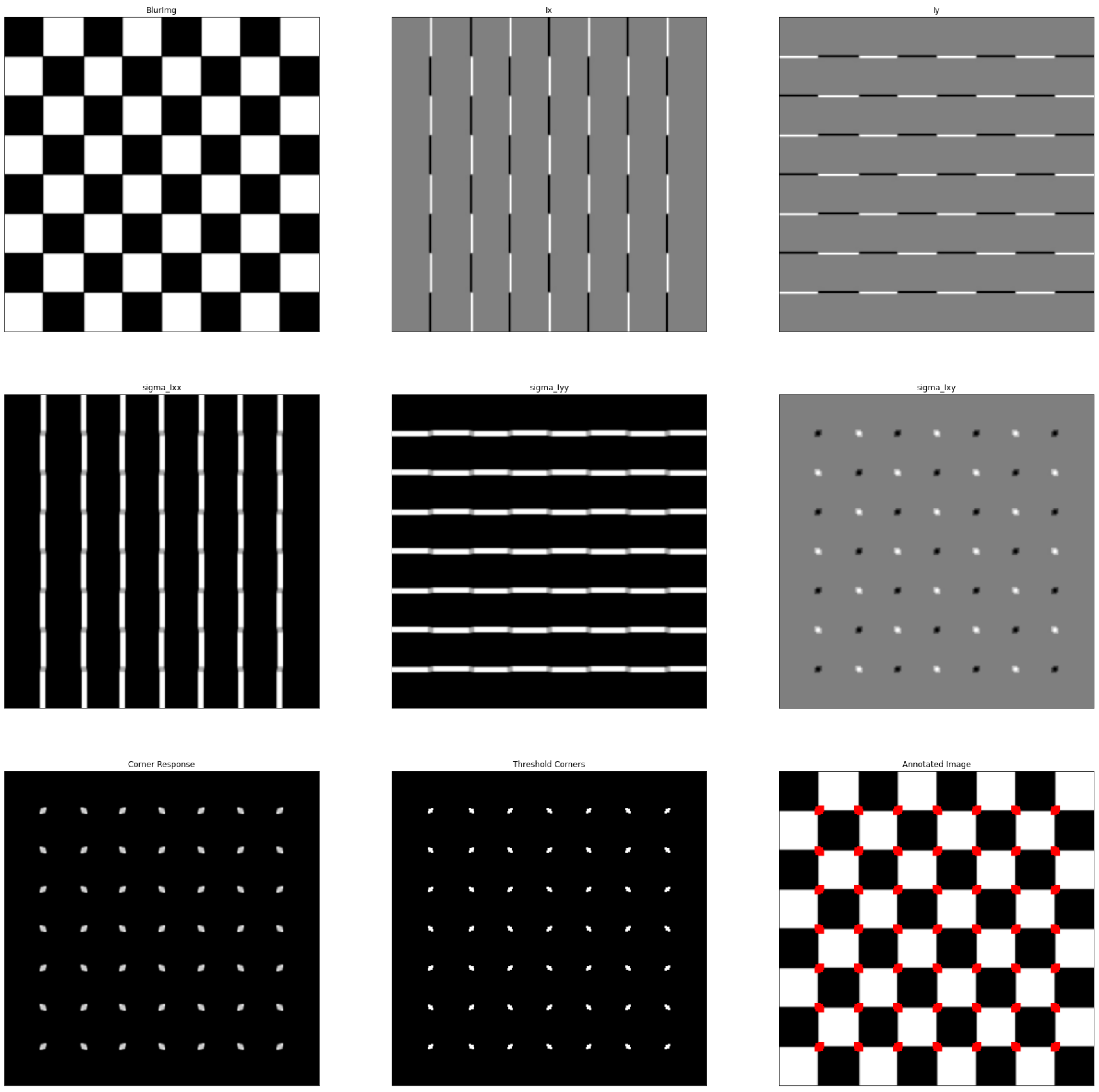


Figure 2: Detecting corners in an image, Step by Step approach[7]

Calculating eigenvalues is computationally expensive therefore what is known as Corner Response Function(R) is used when implementing algorithms. Here $\alpha \in [0.04, 0.06]$ and **At a Corner:** R $\gg 0$, **At an Edge:** R < 0, **At a flat area:** $|R|$ is small. Therefore corners can be extracted by applying an appropriate threshold value for R.

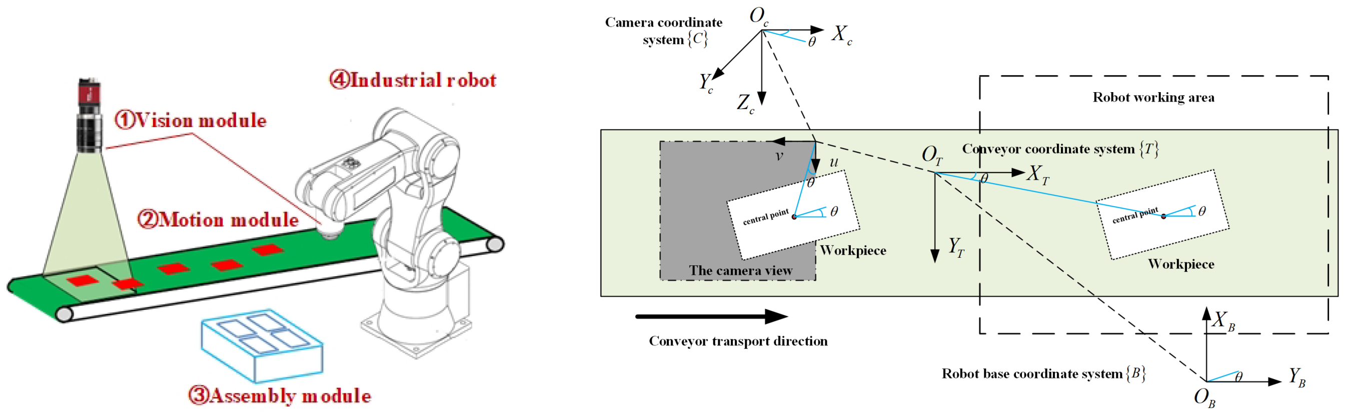
$$\begin{aligned} R &= det(M) - \alpha * trace(M)^2 \\ &= \lambda_{min} * \lambda_{max} - \alpha * (\lambda_{max} + \lambda_{min})^2 \end{aligned} \tag{5}$$

2 Robotic Systems: Coordinate frame Transformation

Related Concepts in Linear Algebra: *Linear Transformations, Matrix Transformations, Change of basis, Matrix Multiplication*

2.1 Problem Identification

When robot arms are used in industrial activities, robot engineer needs to make sure the ***end-effector***(*gripper or any other tool attached at the end of the arm*) of the robot arm is at the exact location in the exact orientation at the operation. Otherwise the required behavior of the arm can not be obtained. Problem is usually robot takes the measurements using sensors or cameras placed somewhere else in the system and therefore these measurements(***position vectors***) are in with respect to those sensor's or camera's coordinate frame while the robot arm operates with respect to some other(Robot base) coordinate frame. Flowing figure illustrates this situation.



(a) Robotic System (b) Coordinate Systems

Figure 3: Use of Different Coordinate systems in an Industrial Robotic System[6]

Therefore to make use of the information gathered by the sensors/cameras first we need to ***transform*** position vectors gathered by camera with respect to its coordinate frame, in to robot arm's base coordinate frame. Then we can use inverse kinematics to calculate the required rotation angles and translations that we should provide to the arm to get the required behavior.

2.2 Solution through concepts of Linear Algebra

In order to fully describe an object in the 3D world we need its ***Rotation and Translation*** with respect to some reference(*in this case the robot arm*) coordinate frame(say α). For that the object of interest(*in this case the sensor/camera*) must be given its own coordinate frame(say β) too. The mathematical way to represent differences between the reference frame and the object's frame is called the ***Homogeneous Transformation Matrix***[1] in Robotics and it is denoted as,

$$H_\beta^\alpha \text{ : read as transformation of frame } \beta \text{ with respect to the frame } \alpha$$

This H_β^α can be mathematically represented as a 4×4 matrix which consists of 3×3 Rotation matrix and a 3×1 Translation matrix with last row as [0 0 0 1] to make it 4×4 square matrix to make the computation easy.

$$H_\beta^\alpha = \begin{bmatrix} r_{11} & r_{12} & r_{13} & d_1 \\ r_{21} & r_{22} & r_{23} & d_2 \\ r_{31} & r_{32} & r_{33} & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

There are ***6 Principal Movements*** as, rotations about 3 axes(X,Y,Z) and translations along 3 axes(X,Y,Z). Correct combination(*obtain by multiplying required principal matrices like* $Rot(e_i, \theta_m) * Trans(e_j, d_k)$) of these 6 principal movements can be used to get any of the advanced transformations, from the reference frame to object's frame. Those basic matrices are as follows where e_i refers to the elements in standard ***basis***($e_1 = [1\ 0\ 0]^\top$, $e_2 = [0\ 1\ 0]^\top$ and $e_3 = [0\ 0\ 1]^\top$) while θ_i indicates how many degrees the object has rotated about the e_i basis vector. Additionally d_i indicates the displacement along the direction of e_i basis vector(axis)[2].

Principal Rotations about axes(X,Y,Z)[3]	**Principal Translations along axes(X,Y,Z)**[3]
$Rot(e_1,\theta_1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_1) & -\sin(\theta_1) & 0 \\ 0 & \sin(\theta_1) & \cos(\theta_1) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$Trans(e_1,d_1) = \begin{bmatrix} 1 & 0 & 0 & d_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
$Rot(e_2,\theta_1) = \begin{bmatrix} \cos(\theta_1) & 0 & \sin(\theta_1) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta_1) & 0 & \cos(\theta_1) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$Trans(e_2,d_2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & d_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
$Rot(e_3,\theta_1) = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$Trans(e_3,d_3) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Once we construct the ***Homogeneous Transformation Matrix which transforms robot base frame into camera frame***, we just have to multiply the position vectors obtained w.r.t camera frame by this transformation matrix to get the corresponding position vectors in the robot base frame. Let ${}^1u = [u_1\ u_2\ u_3\ 1]^\top$ be a position vector(homogeneous) of an object in the camera frame and H_1^0 be the homogeneous transformation matrix, then the position vector of the object in the robot base frame is[4], say ${}^0u$

$$\begin{aligned} {}^0u &= H_1^0 *^1 u \\ {}^0u &= \begin{bmatrix} r_{11} & r_{12} & r_{13} & d_1 \\ r_{21} & r_{22} & r_{23} & d_2 \\ r_{31} & r_{32} & r_{33} & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ 1 \end{bmatrix} \end{aligned} \tag{7}$$

Moreover, since the Homogeneous Transformation Matrix is invertible, following expression is also valid which makes it possible to move between coordinate systems easily.

$$H_1^{0^{-1}} = H_0^1 \tag{8}$$

Bibliography

[1] Robotics 1 kinematics (rotation matrices) - YouTube. https://www.youtube.com/playlist?list=PLT_0lwItn0sDBE98BsbaZezflB96ws12b.

[2] Robotics: Transformation Matrices - Part 1 - YouTube. https://www.youtube.com/watch?v=xYQpeKYCfGs&feature=youtu.be.

[3] Robotics: Transformation Matrices - Part 2 - YouTube. https://www.youtube.com/watch?v=HOv2DTaw38U.

[4] Robotics: Transformation Matrices - Part 3 - YouTube. https://www.youtube.com/watch?v=_B3KsLkX_3A&feature=youtu.be.

[5] Harris Corner Detection (Full Explanation). https://summervisionproject.com/harris-corner-detection/, August 2020.

[6] Rui Song, Fengming Li, Tianyu Fu, and Jie Zhao. A Robotic Automatic Assembly System Based on Vision. *Applied Sciences*, 10(3):1157, January 2020.

[7] Bimalka Thalagala. bimalka98/Computer-Vision-and-Image-Processing. https://github.com/bimalka98/Computer-Vision-and-Image-Processing.