Advancing Image Illumination Through Intrinsic Transfer

A
Project Report
Submitted in partial fulfilment of the
Requirements for the award of the Degree of

BACHELOR OF ENGINEERING

IN

INFORMATION TECHNOLOGY

By

Sadu Bhavana 1602-20-737-069

Survi Alekya 1602-20-737-065

Under the guidance of

Dr. S. K. Chaya Devi

Associate Professor

Department of Information Technology

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad-31 2022

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Information Technology

DECLARATION BY THE CANDIDATE

We, **Sadu Bhavana and Survi Alekya,** bearing hall ticket number, **1602-20-737-069 and 1602-20-737-065**, hereby declare that the project report entitled **Advancing Image Illumination Through Intrinsic Transfer** under the guidance of **Dr. S.K. Chaya Devi,** Associate Professor, Department of Information Technology, Vasavi College of Engineering, Hyderabad, is submitted in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering** in **Information Technology**.

This is a record of bonafide work carried out by me and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

1602-20-737-069
Sadu Bhavana

1602-20-737-065

Survi Alekya

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Information Technology

BONAFIDE CERTIFICATE

This is to certify that the project entitled **Advancing Image Illumination Through Intrinsic Transfer** being submitted by **Sadu Bhavana** and **Survi Alekya** bearing **1602-20-737-069** and **1602-20-737-065** in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Information Technology is a record of bonafide work carried out by them under my guidance.

Dr. S. K Chaya Devi
Associate Professor
Internal Guide

Dr. K. Ram Mohan Rao
Professor & HOD, IT

ACKNOWLEDGEMENT

The satisfaction that accompanies that the successful completion of the Main project would not have been possible without the kind support and help of many individuals. We would like to extend our sincere thanks to all of them.

It is with immense pleasure that we would like to take the opportunity to express our humble gratitude to **Dr. S. K Chaya Devi, Associate Professor, Information Technology** under whom we executed this project. We are also grateful to **Dr. S. Sree Lakshmi , Assistant Professor, Information Technology** for his/ her guidance. Their constant guidance and willingness to share their vast knowledge made us understand this project and its manifestations in great depths and helped us to complete the assigned tasks.

We are very much thankful to **Dr. K. Ram Mohan Rao, HOD, Information Technology,** for his kind support and for providing necessary facilities to carry out the work.

We wish to convey our special thanks to S. V. Ramana**, Principal** of **Vasavi College of Engineering** for giving the required information in doing my project work. Not to forget, we thank all other faculty and non-teaching staff, and my friends who had directly or indirectly helped and supported me in completing my project in time.

We also express our sincere thanks to the Management for providing facilities. Finally, we wish to convey our gratitude to our family who fostered all the requirements and facilities that we need.

Abstract

In the ever-evolving landscape of computer vision and image processing, intrinsic image manipulation has emerged as a critical tool for enhancing the interpretability and utility of digital imagery. This study delves into the realm of illumination manipulation, comparing the effectiveness of the Gamma model against the well-established Retinex model in the context of intrinsic image transfer.

Intrinsic image transfer algorithms play a pivotal role in dissecting complex scenes by separating the intrinsic components of an image, namely reflectance and illumination. The Gamma model, inspired by the perceptual characteristics of the human visual system, introduces a novel approach to accurately capture and represent luminance variations. This research employs an intrinsic image transfer algorithm to evaluate the accuracy of both the Gamma and Retinex models in isolating intrinsic image components.

The societal relevance of intrinsic image manipulation cannot be overstated, as it underpins various applications such as computer vision, image editing, and medical imaging. Accurate separation of reflectance and illumination components is crucial for tasks like scene understanding, object recognition, and content-based image retrieval. Furthermore, in the context of illumination manipulation, these techniques find applications in areas such as surveillance, autonomous vehicles, and augmented reality.

Preliminary findings indicate that the Gamma model shows promise in surpassing the Retinex model in certain scenarios, contributing to the ongoing discourse on intrinsic image transfer techniques. This research provides valuable insights for practitioners seeking optimal solutions for illumination manipulation, which, in turn, can impact the development of advanced technologies and applications in our society. Future investigations are warranted to refine these techniques and extend their applicability to diverse real-world scenarios.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

IIT	Intrinsic Image Transfer
CDF	Cumulative Distribution Function
PDF	Probability Density Function
NLM	Non-Local Means
PET	Probabilistic Early Termination
WSNMF	Weighted Sparse Non-Negative Matrix Factorization
SHG	Second Harmonic Generation
MRF	Markov Random Field
CNN	Convolutional Neural Network
DAS	Delay-and-Sum
DCP	Dark Channel Prior
PSNR	Peak Signal-to-Noise Ratio
MSE	The mean squared error
SSMI	Structural Similarity Index Measure
L	Luminance Contrast
C	Comparison of Contrast

1. INTRODUCTION

1.1 OVERVIEW

In the dynamic realm of image processing, the pivotal challenge of effectively managing illumination control has become paramount to minimize photorealistic losses in visual data. The pursuit of robust solutions has led to the emergence of the Intrinsic Image Transfer (IIT) algorithm, a groundbreaking development meticulously crafted to overcome the intricate challenges associated with enhancing image quality. Particularly in scenarios where images suffer from blurriness, the IIT algorithm stands as a beacon of innovation, offering a comprehensive approach to address the complexities of illumination adjustments.

At the heart of the IIT algorithm lies the incorporation of the gamma model, a fundamental component in the understanding and manipulation of luminance levels within an image. The gamma model plays a pivotal role in accurately representing the nonlinear relationship between pixel intensity values and perceived brightness. By leveraging this model, the IIT algorithm achieves a sophisticated level of adaptability, enabling precise adjustments to be made in scenarios where conventional methods may fall short.

This synergy between the Intrinsic Image Transfer algorithm and the gamma model not only showcases a formidable approach to tackling illumination challenges but also highlights the algorithm's capability to navigate through scenarios where meticulous correction is imperative. As the quest for image quality enhancement continues, the IIT algorithm, with its innovative incorporation of the gamma model, stands as a testament to the relentless pursuit of excellence in the field of image processing.

PROBLEM STATEMENT

The primary challenge lies in handling variations in illumination across images, especially when dealing with blurry images. The existing gamma model, commonly used to represent luminance, poses limitations in accurately

capturing the intricate details of illumination. This inadequacy results in suboptimal image enhancement and correction, leading to reduced photorealism.

The Intrinsic Image Transfer (IIT) algorithm is introduced as an innovative solution to overcome the hurdles associated with illumination challenges in image processing. The IIT algorithm aims to transfer intrinsic image components, separating illumination and reflectance information, to facilitate precise correction of blurry images while preserving photorealistic details.

The gamma method is used to remove haze effect(dusty images) in this method first the image is segment the large sky region using the k-means clustering approach and then refined the whole segmented sky region. Guided filter method is used for refinement of the segmented sky region which is used to provide edge related information accurately.

1.2 PROPOSED WORK

The proposed approach combines the Intrinsic Image Transfer (IIT) Algorithm with the Gamma Algorithm. Using the K-Means Clustering Algorithm, the input image is first split into two clusters: the sky region and the non-sky region. The image is then transformed into grayscale using the gamma approach after the guided filter has been applied to the sky region to guarantee that the input's side image is clear. Now, using an exemplar image (basically a pixel image) and the output of the gamma approach, the reflectance, lighting, and content of a picture may be found using the IIT methodology. Every image has a precise point that causes it to appear dull or blurry, this point may be found using the three-step gamma correction procedure.

1.3 SCOPE AND OBJECTIVES:

The primary focus is on developing techniques that reduce the loss of photorealism in image processing, particularly when dealing with illumination challenges. The IIT algorithm, with its innovative approach, aims to preserve the natural appearance of images while making necessary corrections.

The purpose of the project to improve the overall quality of images, emphasizing clarity, sharpness, and realistic rendering. The incorporation of the gamma model in the algorithm is a key aspect, allowing for precise adjustments to pixel values and luminance, leading to enhanced visual appeal.

The inclusion of the gamma model in the algorithm is a strategic choice to address illumination issues. By accurately modeling the relationship between input and output pixel values, the gamma model contributes to effective brightness and contrast adjustments, ensuring optimal illumination control.

****

2. LITERATURE SURVEY

Various methods have been proposed in the literature for the implementation of Intrinsic Image Identification, which include meticulously crafted to overcome the intricate challenges associated with enhancing image quality. Some of them are mentioned below,

"**Ji-Hee Han *etal.*. [8]"** A novel 3-D color histogram equalization technique is presented to overcome constraints in the RGB intensities using cumulative distribution function (CDF). It attempts to produce a uniform probability density function (pdf) in the intensity domain by defining a CDF with an iso-luminance-plane boundary. This technique, which is similar to grayscale histogram equalization, effectively increases brightness contrast. On the other hand, if there are noise, artifacts, or distortions in the image, its performance might be reduced.

"Ramanathan Vignesh *etal.*. [9]" In order to speed up the process, the research provides a Fast Non-Local Means (NLM) Computation approach employing Probabilistic Early Termination (PET). PET removes blocks that are not identical

by considering the likelihood of distortion values. It performs better than benchmark systems in terms of visual perception improvement, PSNR quality improvement, and complexity reduction.

"Tae Ho Kil *etal.*. [10]" The study presents a single picture dehazing technique that incorporates a "reliability map" to reduce depth estimate mistakes, improving upon the dark channel prior. This map evaluates the consistency of obtaining a transmission factor per pixel, which is important in situations where objects and haze have similar colors. Experimental results show less color distortion and better transmission maps.

"Yu-Xiong *etal.*. [11]" Patches on high-confidence structures are given priority for filling order using Weighted Sparse Non-Negative Matrix Factorization (WSNMF), which sequentially propagates patches inward from the border. By looking for comparable patches in the source region, the algorithm creates a data matrix. It then utilizes NMF for matrix completeness, modifying weights in accordance with patch similarity.

"S. Yousefi1 *etal.*. [12" In this research, a statistical modeling strategy for binary second harmonic generation (SHG) images of cervical tissue is presented utilizing Markov Random Field (MRF). MRF uses energy functions and probability conditions to capture pixel interactions; parameters are determined using the least squares technique. Although MRF offers a statistically sound framework for pixel-interaction representation. The discrete label set might not adequately represent the complex and continuous properties that are present in biological tissues in the real world.

Jui-Ying Lu *etal.*. [13]" The technique uses a Convolutional Neural Network (CNN) in place of the conventional Delay-and-Sum (DAS) beamforming and IQ demodulation processes for beamforming in ultrasonic imaging. Training consists

of transfer learning on multiple-angle PW phantom images after pretraining on simulated and single-angle PW phantom images. Improved primary feature capture is seen in the resulting B-mode ultrasound images, which may indicate a breakthrough in PW imaging techniques.

"Li Shen *etal.*. [14]" The approach incorporates both local and global sparsity restrictions to establish a sparse representation of reflectance in natural photos. The global sparsity requirement is enforced when multiresolution analysis is conducted using red-black wavelets. The algorithm offers a practical method for intrinsic picture decomposition by taking into account chromaticity configurations at various scales and reducing the cost function.

"Sumit Kr.Yadav *etal.*. [15]" The suggested technique offers a simple and efficient way to dehaze a single image by using the gamma algorithm. The gamma correction method expands the applicability of the Dark Channel Prior (DCP) method to broader sky regions, yet it still performs exceptionally well in dehazing small sky regions. Gamma correction improves contrast in images, which helps distinguish details in areas of light and dark.

"Junqing Huang *etal.*. [16]" To separate intrinsic image components, Junqing Huang devised the Intrinsic Image Transfer (IIT) technique, which combines filter identification, LLE weight computation, and reconstruction phases. It provides a thorough method for modifying illumination in computer graphics, resulting in renderings that are more lifelike and give you more control over lighting effects and materials.

Prior research that sought to determine illumination, reflectance, and content used the IIT method, which applies Retinex theory to identify the factors influencing picture illumination, reflectance, and content. Many algorithms can be used in place of the Retinex Model. For example, the Histogram Equalization algorithm [8]

converts images to grayscale; the Non-Local Means (NLM) algorithm [9] is based on Probabilistic Early Termination (PET) which focuses on pixel neighborhood distortion calculation used to achieve early termination based on distortion value; the Dark Channel algorithm [10] is used to estimate the correct transmission map regardless of object color using reliable pixels only (linear fitting is used for unreliable pixels) and when compared to other algorithms that employ a way to identify the edge of the images, Gamma Algorithm[15], which primarily focuses on haze impact, this algorithm uses a very simple and effective strategy for haze removal.

****

3. PROPOSED WORK

3.1 BLOCK DIAGRAM

The proposed approach combines the Intrinsic Image Transfer (IIT) Algorithm with the Gamma Algorithm. Using the K-Means Clustering Algorithm, the input image is first split into two clusters: the sky region and the non-sky region. As mentioned in fig4 the image is then transformed into grayscale using the gamma approach after the guided filter has been applied to the sky region to guarantee that the input's side image is clear. Now, using an exemplar image (basically a pixel image) and the output of the gamma approach, the reflectance, lighting, and content of a picture may be found using the IIT methodology. Every image has a precise point that causes it to appear dull or blurry, this point may be found using the three-step gamma correction procedure.

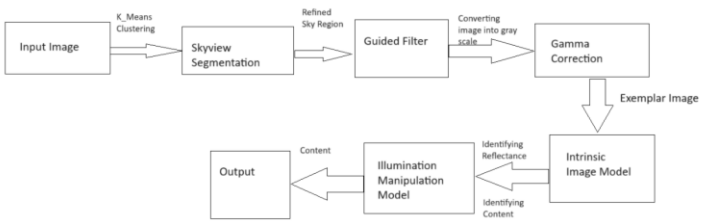


Fig 3 Block of the proposed method.

3.2. ALGORITHM

3.2.1 Sky Region Segmentation

For fig3.1.1 the k-means clustering method is used for segmentation after the computer first selects the vast sky region. K-means clustering divides the dataset into k-subgroups based on similarity and is a simple yet efficient method for segmenting photos. This approach sorts the data points into k different clusters, each represented by a centroid. The procedure is divided into two stages: the first calculates the k centroids, and the second assigns data points to the closest centroid as shown in fig3.1.2. The centroid location of each cluster is used to identify it, making the dataset-splitting process more effective. With an input image of size ***x*×*y*** and k- clusters, in (1) $\boldsymbol{c_k}$ indicates the cluster centroid and ***p(x, y)*** indicates the location of an input pixel within a cluster,

$$d = \| p(x,y) - c_k \| \text{ --------(1)}$$

Fig 3.1.1 Input image

Fig 3.1.2 Image after Applying K-Means Clustering

3.2.2 Guided Filter

The guided filter is used to refine the sky region once it has been segmented. The relationship between the guiding image and the filtered output is defined as follows after the sky region has been segmented and refined using the guided filter:

$$q_i = a_k I_i + b_k, \forall \in \omega_k \text{ ----------(2)}$$

In (2) linear coefficient(a_k, b_k) are constants, I is the input guidance image, q is the linear transform of I in window ω_k.

The filtered output (q) is obtained as follows when noise (n) from input (p) is removed:

$$qi = pi - ni \text{----------(3)}$$

Utilizing the cost function, reduce the difference between q and p with a linear connection in the window ω_k

$$E(a_k, b_k) = \sum_{i \in \omega_k}((a_k I_i + b_k - p_i)^2 + \varepsilon {a_k}^2) \text{----------(4)}$$

In (4), ε is a regularization factor that is used to punish large values of a_k. A linear ridge regression model is the name given to the equation above.

3.2.3 Gamma Correction

A crucial non-linear method for modifying the brightness level of recovered images is gamma correction. Images frequently need brightness modifications after dehazing, therefore choosing an appropriate gamma value is essential for color restoration. Three main gamma coefficients affect this process: the ratio of the average gradient, the ratio of the improved visible edges, and the percentage of saturated pixels. The gamma value increases in tandem with these coefficients. Better outcomes are usually obtained with lower gamma values.

Fig 3.2 Output image

The Fig 3.2 specifies the output image after Gamma Algorithm is applied.

3.2.4 Illumination Manipulation Model:

Intrinsic image transfer(IIT)[16] is a succinct method for manipulating image lighting. Three photorealistic losses content, illumination, and reflectance are specified to describe the underlying "intrinsic" layers. To replicate the spatial smoothing characteristic of picture illumination, a straightforward filtering operator is presented. Additionally, a locally linear embedding (LLE)[17]technique is developed to determine the illumination-invariant reflectance. Without requiring an explicit intrinsic picture decomposition, the enhancements offer an alternate method of optimizing image-illumination. Using a "exemplar image," a closed-form method for managing image illumination is put into practice. On a number of illumination-related tasks, the performance is confirmed both intuitively and numerically. Our IIT approach outperforms state-of-the-art methods in natural photos.

Fig 3.3 Final Output of the Algorithm

Fig 3.3 specifies the output of Gamma+IIT algorithm which identifies the illumination, reflectance and content of the image.

****

4. EXPERIMENTAL STUDY

4.1 DATASETS

Approximately there are twenty images in the dataset and the photorealistic losses has been identified.

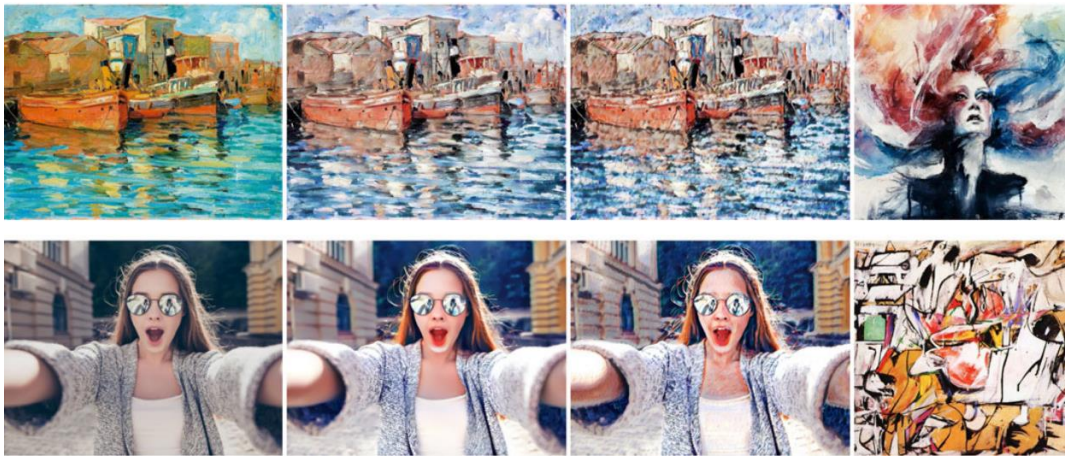
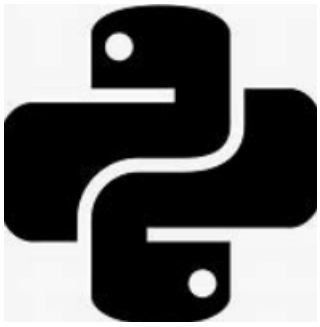


Fig 4.1 Dataset images

4.2 SOFTWARE REQUIREMENTS

1. Python: The software implementation of the intrinsic image transfer algorithm is developed using the Python programming language.



2. Desktop PC: A desktop PC with the following specifications is required to run the Python implementation:

- Operating System: Windows 64-bit

- Processor: Intel Core i7 3.40 GHz

- RAM: 32 GB

3. Python Libraries:

- NumPy: Required for numerical computing tasks.

- OpenCV: Necessary for image processing tasks such as reading and manipulating images.

- SciPy: Useful for scientific computing tasks.

4. Internet Connection: An internet connection is required to access the GitHub repository containing the code for the intrinsic image transfer algorithm.

5. Web Browser: A web browser is needed to access the GitHub repository link provided for downloading the code.

4.3 Hardware Requirements:

1. Desktop PC: The Python implementation of the intrinsic image transfer algorithm runs on a desktop PC with the following specifications:

- Processor: Intel Core i7 3.40 GHz

- RAM: 32 GB

- Operating System: Windows 64-bit

2. Monitor: A monitor is required to display the Python environment and the results of the intrinsic image transfer algorithm.

3. Input Devices: Input devices such as a keyboard and mouse are needed to

interact with the Python environment and run the algorithm.

4. Internet Connection: An internet connection is necessary to access the GitHub repository and download the code for the intrinsic image transfer algorithm.

4.4 RESULTS:

Approximately there are twenty images in the dataset and the photorealistic losses has been identified. Subsequent to the use of intrinsic image transfer, the dehazed image proceeds through further processing stages to ensure completion. This involves multiplying the dehazed image by the corrected sky area mask in order to guarantee that only the dehazed sky region is kept while the rest of the image is kept intact. After processing, the image is saved as the output result, which shows less haze in the sky area and increased clarity. This finalization stage highlights clarity and detail in important regions, improving the image's overall visual quality. With better sky vision and less atmospheric interference, the output result is a better portrayal of the original image. With this step, the image processing pipeline is completed and a polished, aesthetically pleasing output is produced, ready for display or additional analysis.

Fig 4.4 (a) Input Image (b) Exemplar Image (c) Output Image

Figure 4.3(a) shows the image input. The image is then split into two clusters using the k-means clustering algorithm: sky-region segmentation and non-sky-region segmentation. This process is repeated for the sky-region segmentation. The image's sides are identified using a guided filter, and the Gamma method is then used to convert the image to grayscale. The Gamma method's output is combined with the exemplar image (b) to identify realistic losses in the photo, which are then corrected using the LLE algorithm to create the final image (c).

4.5 ANALYSIS

	Gamma Algorithm	Gamma+ IIT Algorithm
PSNR	27.835654293518953	28.157662773625585
SSMI	0.3152540838298061	0.8380320865438617

Table 1: Comparison of PSNR and SSMI values of Gamma algorithm and Gamma+IIT algorithm

PSNR(Peek Signal-To-Noise Ratio):

Peak Signal-to-Noise Ratio (PSNR) is used statistic to assess how well processed or reconstructed images compare to their original quality. It calculates the relationship between a signal's maximal potential power and its noise power.

The mean squared error (MSE) is the easiest way to define PSNR. MSE is defined as follows:

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2 \text{--------(5)}$$

given a noisy approximation K and a noise-free m×n single-tone picture I

PSNR = 20*log (max (max (f)))/ ((MSE)^0.5) ----------(6)

In (6) max(f) represent the maximum pixel value of the image

SSMI(Structural Similarity Index Measure):

One popular technique for determining how similar two photos are to one another is the Structural Similarity Index Measure (SSIM). Its ability to consider structure information and perceived changes in images, rather than just comparing individual pixels, makes it especially popular in the fields of image processing and computer vision. SSIM takes into account elements like brightness, contrast, and structure in order to replicate how people perceive the quality of an image.

Below is an explanation of the elements that go into determining the SSIM index:
Luminance Contrast (L): This factor gauges how close the luminance (brightness) of the two pictures is to one another. It is computed as the average of the two images values.
Comparison of Contrast (C): In an image, contrast is the brightness variation between different objects or areas. The contrast comparison calculates how similar the two images' contrasts are to one another.

The covariance of the two images' pixel values is used to calculate it.

$$SSMI(x,y) = \frac{(2\mu_x\mu_y+c_1)(2\sigma_{xy}+c_2)}{(\mu_x 2+\mu_y 2+c_1)(\sigma_x 2+\sigma_y 2+c_2)} \quad \text{----------(7)}$$

In (7), μ_x and μ_y are the means of the images x and y respectively. $\sigma_x 2$ and $\sigma_y 2$ are the variances of the images x and y respectively. σ_{xy} is the covariance of the images x and y. c_1 and c_2 are constants.

In the following table (Table 1), we have compared PSNR and SSMI to the algorithms Gamma Algorithm and Gamma+IIT algorithm.

******

5. SUMMARY AND FUTURE SCOPE

The combination of the IIT method and the gamma model not only provides a general way to deal with differences in illumination between images, but also highlights how the algorithm can work its way through situations where careful adjustment is necessary. The technique allows for the accurate restoration of fuzzy photos while maintaining lifelike features by isolating intrinsic image components. Looking at cutting-edge methods for adding the gamma model to the IIT algorithm in order to increase illumination adjustments' accuracy and flexibility. Real-time processing optimization of the method makes it appropriate for applications where speed is critical, such live streaming and video processing.

In conclusion, the integration of the gamma model into the Intrinsic Image Transfer (IIT) algorithm represents a significant advancement in the realm of image processing, particularly in addressing the challenges associated with illumination control. Through this innovative approach, the project has demonstrated a profound understanding of the intricate relationship between pixel intensity values and perceived brightness, enabling precise adjustments to be made even in scenarios where conventional methods fall short.

The synergy between the IIT algorithm and the gamma model not only offers a comprehensive solution to handling variations in illumination across images but also underscores the algorithm's capability to navigate through scenarios where meticulous correction is imperative. By separating intrinsic image components and facilitating precise correction of blurry images while preserving photorealistic details, the IIT algorithm has showcased its potential to significantly enhance image quality.

****

REFERENCES

[1] E. H. Land and J. J. McCann, "*Lightness and retinex theory*," Josa, vol. 61, no. 1, pp. 1–11, 1971.

[2] A. Gilchrist, *Seeing Black and White*. USA: Oxford Univ. Press, 2006.

[3] H. Barrow and J. Tenenbaum, "*Recovering intrinsic scene characteristics*," Comput. Vis. Syst., vol. 2, pp. 3–26, 1978.

[4] R. K. Mantiuk, K. Myszkowski, and H.-P. Seidel, *High Dynamic Range Imaging*. New York, NY, USA: Wiley Online Library, 2015.

[5] J. Morovic and M. R. Luo, "*The fundamentals of gamut mapping: A survey,"* J. Imag. Sci. Technol., vol. 45, no. 3, pp. 283–290, 2001.

[6] R. Fattal, D. Lischinski, and M. Werman, "*Gradient domain high dynamic range compression*," ACM Trans. Graph., vol. 21, pp. 249–256, 2002.

[7] M. K. Ng and W. Wang, "*A total variation model for retinex,"* SIAM J. Imag. Sci., vol. 4, no. 1, pp. 345–365, 2011.

[8] *A Novel 3-D Color Histogram Equalization Method With Uniform 1-D Gray Scale Histogram* by Ji-Hee Han in the year 2011

[9] *Fast Non-Local Means (NLM) Computation With Probabilistic Early Termination* by Ramanathan Vignesh in the year 2010.

[10] *SINGLE IMAGE DEHAZING BASED ON RELIABILITY MAP OF DARK CHANNEL PRIOR* by Tae Ho Kil in the year 2013.

[11] *IMAGE INPAINTING VIA WEIGHTED SPARSE NON-NEGATIVE MATRIX FACTORIZATION* by Yu-Xiong Wang in the year 2011

[12] *Synthesis of Cervical Tissue Second Harmonic Generation Images Using Markov Random Field Modeling* by S. Yousefi1 in the year 2011.

[13] *Improving Image Quality for Single-Angle Plane Wave Ultrasound Imaging With Convolutional Neural Network Beamformer* by Jui-Ying Lu in the year 2022

[14] *Intrinsic Image Decomposition Using a Sparse Representation of Reflectance* by Li Shen in the year 2013

[15] *Single Image Dehazing using Adaptive Gamma Correction Method* by Sumit Kr.Yadav in the year 2019

[16] *Intrinsic Image Transfer for Illumination Manipulation* by Junqing Huang in the year 2023

[17] SIAM J. Imag. Sci., vol. 4, no. 1, pp. 345–365, 2011. [8] S. T. Roweis and L. K. Saul, "*Nonlinear dimensionality reduction by locally linear embedding*," Science, vol. 290, no. 5500, pp. 2323– 2326, 2000

APPENDIX

Github link: https://github.com/SaduBhavana/Major-project.git

Code for the proposed work:

```
import numpy as np
import cv2
from sklearn.neighbors import NearestNeighbors
from scipy import sparse
import time
import os


import numpy as np #for computing numerical operations
import cv2 #openCV ,it is a computer vision library
from sklearn.cluster import KMeans #for computing k means algorithm


def sky_region_segmentation(image, k=2):
```

```
    image_float32 = image.astype(np.float32) / 255.0 #converting the image into
float32 for compatability and normalize the pixel value between [0,1]

    # Flatten the image to perform KMeans clustering 2D
    pixels = image_float32.reshape((-1, 3))

    # Use KMeans to segment the image
    kmeans = KMeans(n_clusters=k, random_state=0)
    labels = kmeans.fit_predict(pixels)
    #reshape into original image
    segmented_result = labels.reshape(image.shape[:2])

    # Assume the largest cluster corresponds to the sky region
    sky_label = np.argmax(np.bincount(segmented_result.flatten()))
    sky_region_mask = (segmented_result == sky_label)

    return sky_region_mask #return binary mask

def refine_sky_region(image, sky_region_mask):
    #This function refines the sky region mask using OpenCV's guided filter

    # Convert image to float32
    image_float32 = image.astype(np.float32) / 255.0

    # Convert mask to float32
    mask_float32 = sky_region_mask.astype(np.float32)
```

```
  #If the image or mask has a single channel, it is converted to a three-channel image.


    if image_float32.shape[-1] == 1:

        image_float32 = cv2.cvtColor(image_float32, cv2.COLOR_GRAY2BGR)

    if mask_float32.shape[-1] == 1:

        mask_float32 = cv2.cvtColor(mask_float32, cv2.COLOR_GRAY2BGR)


    # Use OpenCV's guided filter for refinement

    refined_mask = cv2.ximgproc.guidedFilter(image_float32, mask_float32, radius=5, eps=1e-3)


    return refined_mask


def gamma_correction(image, gamma=0.7):

    # Apply gamma correction to the dehazed image


    #Gamma correction is a non-linear adjustment to modify the image intensity.


    corrected_image = np.power(image, gamma)

    return corrected_image


def dehaze_image(image_path):

    # Load of image

    image = cv2.imread(image_path)


    #Loads the input image using OpenCV.
```

#Calls the sky_region_segmentation function to obtain a binary mask for the sky region.

#Calls the refine_sky_region function to refine the sky region using guided filtering.

#Applies gamma correction to the refined mask using the gamma_correction function.

#Displays the original and dehazed images using OpenCV.

```
if image is not None:
    # Step 1: Sky Region Segmentation
    sky_region_mask = sky_region_segmentation(image)

    # Step 2: Guided Filter for the Refinement of Sky Region
    refined_mask = refine_sky_region(image, sky_region_mask)

    # Step 3: Gamma Correction Approach
    corrected_image = gamma_correction(refined_mask, gamma=0.7)  # Adjust gamma value as needed

    # Display or save the results as needed
    cv2.imshow("Original Image", image)
    cv2.imshow("Dehazed Image", corrected_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
else:
    print("Failed to load the image.")
```

```
def image_to_vector(I, param):
    color_transfer = param['color_transfer']
    color_space = param['color_space']
    logarithm = param['logarithm']
    scale = param['scale']
    bias = param['bias']


    rows, cols, layers = I.shape #dimensions of the input
    Px = np.tile(np.arange(1, rows + 1), (1, cols)) #x coordinates
    Py = np.tile(np.arange(1, cols + 1), (rows, 1)) # y coordinates
    P = np.column_stack((Px.flatten(), Py.flatten())) #combine the X and Y coordinates into a single array P.
    # checks if the specified color space is HSV (Hue, Saturation, Value).
    #If so, it converts the input image I from RGB to HSV color space using cv2.cvtColor().
    # Then it extracts the V (value/brightness) channel.
    #If color_transfer is True, it extracts only the H and S channels from the HSV image
    # otherwise, it applies logarithmic transformation  to the V channel and flattens it into a 1D array X.
    if color_space == 'hsv':
        hsv_I = cv2.cvtColor(I, cv2.COLOR_RGB2HSV)
        v = hsv_I[:, :, 2]
        if color_transfer:
            X = hsv_I[:, :, :2].reshape((rows * cols, 2))
        else:
            if logarithm:
                v = -np.log(bias + scale * v)
```

```
        X = v.flatten()
    #It converts the input image I from RGB to Lab color space using cv2.cvtColor().
    #Then it extracts the L (lightness) channel and scales it to the range [0, 1]
    elif color_space == 'lab':
        Lab_I = cv2.cvtColor(I, cv2.COLOR_RGB2LAB)
        L = Lab_I[:, :, 0] / 100.0
        if color_transfer:
            X = Lab_I[:, :, 1:].reshape((rows * cols, 2))
        else:
            if logarithm:
                L = -np.log(bias + scale * L)
            X = L.flatten()
    #Then it extracts the Y (luminance) channel.
    #Depending on the value of color_transfer, it either extracts the Cb and Cr channels or applies a logarithmic transformation.
    elif color_space == 'Ycbcr':
        Ycbcr_I = cv2.cvtColor(I, cv2.COLOR_RGB2YCrCb)
        L = Ycbcr_I[:, :, 0]
        if color_transfer:
            X = Ycbcr_I[:, :, 1:].reshape((rows * cols, 2))
        else:
            if logarithm:
                L = -np.log(bias + scale * L)
            X = L.flatten()
    else:
        if logarithm:
            I = -np.log(bias + scale * I)
```

```
    X = I.reshape((rows * cols, layers))


    return X, P #returns flattened pixel values X and the pixel coordinates P.




def solve_gaussian_weights(X, P, K, delta_s, delta_r, mode, scale, bias):

    N, _ = X.shape # Extracts the number of samples N from the first dimension of the feature vectors X.

    nn = NearestNeighbors(n_neighbors=K)

    nn.fit(P) # Fits the nearest neighbors model with the spatial positions P

    distances, indices = nn.kneighbors(P, return_distance=True)


    X = (np.exp(-X) - bias) / scale

    p = P[indices[:, 1:], :] - P[indices[:, :1], :] #Computes the spatial differences between the neighbors for each sample.

    x = X[indices[:, 1:], :] - X[indices[:, :1], :] #. It subtracts the feature vector of the first neighbor from the feature vectors of the other neighbors.


    if mode == 'gf':

        delta_s2 = 2 * delta_s**2 # Computes the squared spatial bandwidth.

        w = np.exp(-(np.sum(p**2, axis=2) / delta_s2)) #Computes the Gaussian weights based on the spatial differences.

    elif mode == 'bf':

        delta_s2 = 2 * delta_s**2 #computes the squared spatial bandwidth.

        delta_r2 = 2 * delta_r**2

        w = np.exp(-(np.sum(p*2, axis=2) / delta_s2) - (np.sum(x*2, axis=2) / delta_r2)) #Computes the Gaussian weights based on both spatial and feature differences.
```

```
    else:
        delta_s2 = 2 * delta_s**2
        delta_r2 = 2 * delta_r**2
        w = np.exp(-(np.sum(p*2, axis=2) / delta_s2) - (np.sum(x*2, axis=2) / delta_r2))

    w /= np.sum(w, axis=1)[:, np.newaxis] #Normalizes the weights along each row. This ensures that the weights for each sample sum to 1.

    # Create COO sparse matrix
    row_idx = np.repeat(np.arange(N), K-1)
    col_idx = indices[:, 1:].flatten()
    data = w.flatten()

    if len(row_idx) == len(col_idx) == len(data) == N * (K - 1):
        W_coo = sparse.coo_matrix((data, (row_idx, col_idx)), shape=(N, N), dtype=np.float64)

        # Convert to CSR sparse matrix
        W = W_coo.tocsr() #parse matrix representation of the computed weights using COO format. This format stores the (row, column, value)
        M = W.transpose().dot(W)

        return M # affinity matrix M, representing the pairwise similarities between samples based on their feature and spatial similarities.
    else:
        raise ValueError("Length mismatch in COO matrix creation.")
```

```
def solve_lle_embedding(C, S, W, M, alpha, beta, gamma):
    n, _ = M.shape #Extracts the number of rows  from the affinity matrix M.
    m, d = C.shape #Extracts the number of rows and columns from the data matrix C

    A = alpha * W + beta * M + gamma * sparse.eye(m, format='csr') # captures the local structure of the data.
    b = alpha * W.dot(C) + gamma * S #captures the global structure of the data.
    Y = sparse.linalg.spsolve(A, b)

    return Y #the low-dimensional representation of the data

def solve_lle_weights(X, P, K, tol):

    N, D = P.shape

    nn = NearestNeighbors(n_neighbors=K + 1)
    nn.fit(P)
    distances, indices = nn.kneighbors(P, return_distance=True)
    neighborhood = indices[:, 1:]
    currentindex = indices[:, :1]

    w0 = np.zeros((N, K))
    for ii in range(N):
        z = X[currentindex[ii, :], :] - X[neighborhood[ii, :], :]
        c = z.dot(z.T)
        c = c + tol * np.eye(K)
        w = np.linalg.solve(c, np.ones(K))
```

```
        w0[ii, :] = w / np.sum(w)


    # Ensure all arrays have the same length

    common_length = min(len(w0.ravel()), len(currentindex.flatten()),
len(neighborhood.flatten()))


   # W_coo = sparse.coo_matrix((w0.ravel()[:common_length],
(currentindex.flatten()[:common_length],
neighborhood.flatten()[:common_length])), shape=(N, N))

    W_coo = sparse.coo_matrix((w0.ravel()[:common_length],
(currentindex.flatten()[:common_length],
neighborhood.flatten()[:common_length])), shape=(N, N))


    # Only return the necessary value

    return W_coo


def vector_to_image(Y, S, C, param):

    bias = param['bias'] # Bias used during vector-to-image transformation.

    scale = param['scale'] #Scaling factor used during vector-to-image
transformation.

    color_space = param['color_space'] #Specifies the color space of the output
image.

    logarithm = param['logarithm'] #xtracts the boolean flag indicating whether to
apply logarithmic transformation from the parameter dictionary.

    color_transfer = param['color_transfer'] #perform color transfer from the
parameter dictionary.

    color_exemplar = param['color_exemplar'] #Extracts the information about
which image to use as an exemplar for color transfer from the parameter
dictionary.


    rows, cols, layers = S.shape
```

```
if color_space == 'hsv':
    if color_exemplar == 'original':
        hsv_S = cv2.cvtColor(S, cv2.COLOR_RGB2HSV)
    elif color_exemplar == 'exemplar':
        hsv_S = cv2.cvtColor(C, cv2.COLOR_RGB2HSV)

    if color_transfer:
        hsv_S[:, :, :2] = Y.reshape((rows, cols, 2))
    else:
        v = Y
        if logarithm:
            v = np.minimum(1, np.maximum(0, (np.exp(-v) - bias) / scale))
        hsv_S[:, :, 2] = v.reshape((rows, cols, 1))
    I = cv2.cvtColor(hsv_S, cv2.COLOR_HSV2RGB)
elif color_space == 'lab':
    if color_exemplar == 'original':
        lab_S = cv2.cvtColor(S, cv2.COLOR_RGB2LAB)
    elif color_exemplar == 'exemplar':
        lab_S = cv2.cvtColor(C, cv2.COLOR_RGB2LAB)
    if color_transfer:
        lab_S[:, :, 1:] = Y.reshape((rows, cols, 2))
    else:
        L = Y
        if logarithm:
            L = np.minimum(1, np.maximum(0, (np.exp(-L) - bias) / scale))
        lab_S[:, :, 0] = L.reshape((rows, cols, 1))
```

```
        I = cv2.cvtColor(lab_S, cv2.COLOR_LAB2RGB)
    elif color_space == 'Ycbcr':
        if color_exemplar == 'original':
            Ycbcr_S = cv2.cvtColor(S, cv2.COLOR_RGB2YCrCb)
        elif color_exemplar == 'exemplar':
            Ycbcr_S = cv2.cvtColor(C, cv2.COLOR_RGB2YCrCb)
        if color_transfer:
            Ycbcr_S[:, :, 1:] = Y.reshape((rows, cols, 2))
        else:
            L = Y
            if logarithm:
                L = np.minimum(1, np.maximum(0, (np.exp(-L) - bias) / scale))
            Ycbcr_S[:, :, 0] = L.reshape((rows, cols, 1))
        I = cv2.cvtColor(Ycbcr_S, cv2.COLOR_YCrCb2RGB)
    else:
        if logarithm:
            Y = np.minimum(1, np.maximum(0, (np.exp(-Y) - bias) / scale))
        I = Y.reshape((rows, cols, layers))

    return I

def intrinsic_image_transfer(S, C, param):
    X_s, P = image_to_vector(S, param)
    X_c, _ = image_to_vector(C, param)

    bias = param['bias']
    scale = param['scale']
```

```
    k1 = param['filter']['k1']
    delta_s = param['filter']['delta_s']
    delta_r = param['filter']['delta_r']
    filter_mode = param['filter']['mode']
    W_coo = solve_gaussian_weights(X_s, P, k1, delta_s, delta_r, filter_mode,
scale, bias)

    tol = param['LLE']['tol']
    k2 = param['LLE']['k2']

    # Use the coo_matrix directly
    M = W_coo.transpose().dot(W_coo)

    alpha = param['alpha']
    beta = param['beta']
    gamma = param['gamma']
    Y = solve_lle_embedding(X_c, X_s, W_coo, M, alpha, beta, gamma)

    T = vector_to_image(Y, S, S, param)
    T = np.maximum(0, np.minimum(1, T))

    return T, M #ntrinsic image T and the affinity matrix M.


def main():
    start_time = time.time()
```

```
print(start_time)
# Get the absolute path of the script
script_path = os.path.abspath(_file_)

# Assume the 'data' folder is in the same directory as the script
data_folder = os.path.join(os.path.dirname(script_path), '_data')

src_path = os.path.join(data_folder, 'content1.png')


demo_folder=os.path.join(os.path.dirname(script_path),'data')


clahe_path = os.path.join(demo_folder, 'exemplar1.png')

# Check if the files exist
if not os.path.isfile(src_path) or not os.path.isfile(clahe_path):
    print(f"Error: One or both images do not exist.")
    return

# Load images
S = cv2.imread(src_path) / 255.0
C = cv2.imread(clahe_path) / 255.0
S = cv2.resize(S, (new_width, new_height))
C = cv2.resize(C, (new_width, new_height))

# Check if images were loaded successfully
```

```
    if S is None or C is None:
        print("Error: One or both images could not be loaded.")
        return
    # Perform sky region segmentation on the source image
    sky_region_mask = sky_region_segmentation((S * 255).astype(np.uint8))

    # Refine the sky region mask using guided filter
    refined_mask = refine_sky_region(S, sky_region_mask)

    # Apply gamma correction to the refined mask
    corrected_mask = gamma_correction(refined_mask, gamma=0.7)

    param = {
        'logarithm': 1, # 1 suggesting that logarithmic transformation is enabled
        'color_transfer': 0, #meaning color transfer is disabled.
        'bias': 1 / 255,  # for 8-bit image
        'scale': 1.0, #Represents the scaling factor used in image transformation
        'color_space': 'rgb', #Specifies the color space used in image processing
        'color_exemplar': 'original',
        'filter': {
            'k1': 49, #Specifies the number of nearest neighbors used in weight computation.
            'delta_s': 2.0, #Represents the spatial standard deviation in the Gaussian filter
            'delta_r': 0.2, #epresents the range standard deviation in the Gaussian filte
            'mode': 'gf'
        },
        'LLE': {
```

```
        'tol': 1e-5, #Specifies the tolerance value used in LLE computations.
        'k2': 49 #Specifies the number of nearest neighbors used in LLE computations.
    },
    'alpha': 0.9, #Represents the weighting factor for the affinity matrix in LLE computations.
    'beta': 100, #Represents the weighting factor for the similarity matrix in LLE computations.
    'gamma': 0.1 #Represents the regularization parameter in LLE computations.
}

T, M = intrinsic_image_transfer(S, C, param)
# T = np.maximum(0, np.minimum(1, T))
# corrected_mask_resized = cv2.resize(corrected_mask, (T.shape[1], T.shape[0]))

# Replicate the single channel of the mask to match the three channels of the image
#corrected_mask_3ch = np.stack([corrected_mask_resized] * 3, axis=-1)

# Ensure that both arrays have the same shape before multiplication
#T = np.maximum(0, np.minimum(1, T * corrected_mask_3ch))
end_time = time.time()
print(end_time)
# Save the result
result_folder = os.path.join(os.path.dirname(script_path), 'results')
os.makedirs(result_folder, exist_ok=True)
# Before saving the result, add debug statements to check the values of T and corrected_mask
```

```
    print("T min/max:", np.min(T), np.max(T))

    print("corrected_mask min/max:", np.min(corrected_mask),
np.max(corrected_mask))



    cv2.imwrite(os.path.join(result_folder, 'IIT_src.png'), (S *
255).astype(np.uint8))

    cv2.imwrite(os.path.join(result_folder, 'IIT_output.png'), (C *
255).astype(np.uint8))



    print(f"Processing time: {end_time - start_time:.2f} seconds")



if _name_ == "_main_":

    new_width, new_height = 500, 500

    main()
```

******