**COLOR DETECTION USING JAVA**

Bonafide Certificate:

This work was done by G. Dinesh (192110064) from CSE department during slot A - programming in java (CSA0915) in the duration of four months from December 2023 to march 2024 under the guidance of T. Vincent Ganaraj.

**Abstract:**

Color detection is necessary to recognize objects, it is also used as a tool in various image editing and drawing apps. It is the process of detecting the name of any color. Well, for humans this is an extremely easy task but for computers, it is not straightforward. Hence the problem that arises in front of us is how to make a computer understand or recognize colors. So, we are going to solve this problem. In this project using Java we need 3 different parts to be used. Java code which will be recognizing color. Image that will be used for testing the color recognition. A csv file that will be containing the colors as a dataset. Hence the above 3 modules will help us in achieving our aim that is detecting the colors in an image using Java.

**Keywords:** Color, Detection, Image, Java, Swing.

1. **INTRODUCTION**

Color detection in Java presents an intriguing exploration into the realm of computer vision, where machines decipher and interpret visual data akin to human perception. This endeavour involves harnessing Java's robust capabilities to analyze and interpret colors within digital imagery, enabling a myriad of applications across diverse domains. Whether its identifying objects based on their color signatures or segmenting images for further analysis, Java provides a versatile platform for developers to delve into the intricacies of color detection.

At its core, color detection in Java begins with acquiring images from various sources, be it static image files or real-time video streams. Leveraging Java's image processing libraries, developers gain access to tools for loading and manipulating image data, laying the foundation for subsequent analysis. Once images are acquired, the focus shifts to representing colors in different color spaces, such as RGB or HSV, facilitating more nuanced analysis and segmentation.

A pivotal step in color detection involves thresholding and segmentation, where developers delineate regions of interest based on predefined color criteria. By setting thresholds and filtering out irrelevant colors, developers can isolate specific hues or color ranges, laying the groundwork for subsequent analysis. This process enables tasks such as object detection, where objects of interest are identified based on their distinct color profiles within the image.

Furthermore, color detection in Java encompasses feature extraction, where developers extract pertinent characteristics from the segmented regions. These features may include contour information, color histograms, or spatial properties, providing valuable insights for subsequent processing steps. Armed with extracted features, developers can embark on tasks such as object tracking, monitoring the movement and behavior of objects over time within a video stream.

Moreover, the versatility of Java extends beyond standalone applications, with seamless integration capabilities into existing systems and frameworks. Whether it's incorporating color detection functionalities into enterprise applications or integrating with IoT devices for real-time monitoring, Java's interoperability fosters innovation across various domains. This integration flexibility enables developers to augment existing systems with visual perception capabilities, unlocking new avenues for analysis and decision-making.

In conclusion, color detection in Java transcends mere image analysis, offering a gateway to deeper insights and understanding of the visual world. Through the amalgamation of Java's powerful features and computer vision techniques, developers can unravel complex visual patterns, paving the way for innovative applications across industries. As we navigate this journey through the vibrant landscape of color detection in Java, we embark on a quest to push the boundaries of what's possible in the realm of visual perception and machine intelligence.

1. EXISTING SYSTEM

One exemplary system for color detection is OpenCV, a renowned open-source library widely adopted for image processing and computer vision tasks. OpenCV offers a comprehensive suite of tools and algorithms specifically tailored for color detection applications. One of its core features is its ability to perform color space conversions, allowing developers to switch between different color representations such as RGB, HSV, and Lab. This flexibility enables nuanced analysis of images based on their color characteristics, essential for tasks like object recognition and segmentation.

Within OpenCV, various thresholding techniques are available, empowering developers to segment images based on color intensity. These techniques include binary thresholding, adaptive thresholding, and range thresholding, each suited for different scenarios. By leveraging thresholding, developers can isolate regions of interest in an image, facilitating precise color detection and analysis.

Furthermore, OpenCV facilitates histogram analysis, enabling developers to compute color histograms representing the distribution of colors within an image. Histograms provide valuable insights into the presence and abundance of specific colors, aiding in tasks such as image classification and content-based retrieval. Additionally, OpenCV offers algorithms for contour detection, allowing developers to identify the outlines of objects based on color information. Contour detection is instrumental in tasks like object recognition, where the shape and color of objects play a crucial role.

Moreover, OpenCV seamlessly integrates with machine learning frameworks such as TensorFlow enabling developers to train custom color detection models using deep learning algorithms. This integration extends the capabilities of color detection systems, allowing for more sophisticated tasks such as fine-grained categorization and recognition of objects based on their color attributes.

In summary, OpenCV stands out as a versatile and powerful platform for color detection, offering developers a wide array of tools and algorithms to analyze, manipulate, and interpret colors within digital imagery. Its extensive documentation, community support, and cross-platform compatibility make it a preferred choice for researchers and developers working in the field of computer vision, driving advancements in color detection technology.

1. PROPOSED SYSTEM

The proposed color detection system represents an advancement in the field of computer vision, leveraging sophisticated techniques and machine learning algorithms to achieve accurate and real-time color identification and tracking. Building upon existing frameworks like OpenCV, this system integrates cutting-edge methodologies to enhance its capabilities, making it suitable for a wide range of applications including quality control, object tracking, and robotics.

Compared to existing systems, this proposed system offers several key innovations. Firstly, it utilizes advanced computer vision techniques to achieve more precise color detection and tracking. By employing techniques such as histogram analysis, contour detection, and adaptive thresholding, the system can effectively isolate colors of interest even in complex and dynamic environments.

Additionally, the integration of machine learning algorithms enhances the system's ability to recognize and adapt to varying conditions. Through training on large datasets, the system can learn to differentiate between different shades, lighting conditions, and backgrounds, improving its accuracy and robustness in color detection tasks.

One notable feature of the proposed system is its seamless integration with cameras or image sources, providing versatility and flexibility in deployment. Whether it's interfacing with a webcam, surveillance camera, or drone feed, the system can effortlessly process incoming images and perform real-time color detection with minimal latency.

Moreover, the user-friendly interface of the system simplifies the task of color selection and parameter tuning, making it accessible to users with varying levels of expertise. Through intuitive controls and interactive visualizations, users can specify colors of interest, adjust detection thresholds, and visualize the results in real-time, facilitating efficient interaction and feedback.

In summary, the proposed color detection system represents a significant advancement over existing systems by combining advanced computer vision techniques with machine learning capabilities. Its seamless integration with image sources and user-friendly interface makes it a versatile tool for various applications, empowering users to achieve accurate and reliable color detection in real-world scenarios.

1. REQUIREMENT AND SPECIFICATION

We need the following requirement for developing, testing and designing the project.

**Hardware Requirements:**

**CPU:**12th Gen Intel(R) Core (TM) i5-1235U 1.30 GHz

**RAM:** 8GB-16GB

**SSD:** 512GB

**KEYBOARD:** Membrane (QWERTY) format

**MOUSE:** Optical

**MONITOR:** LED

**INTERNET:** Wi-fi

**Software Requirements:**

**Development SDK:** JAVA JDK 21

**Runtime Environment:** JRE-8

**Programming IDE:** Eclipse

**Operating System:** Windows 11

**Packages:** java.awt, javax. imageio or java.io

Human requirements

**Team Members**:1

**Supervisor:1**

**Total:**2

FINANCIAL REQUIREMENTS

    Budget: 1000-15000 depending on the project implementation and quality metrics.

1. Methodology

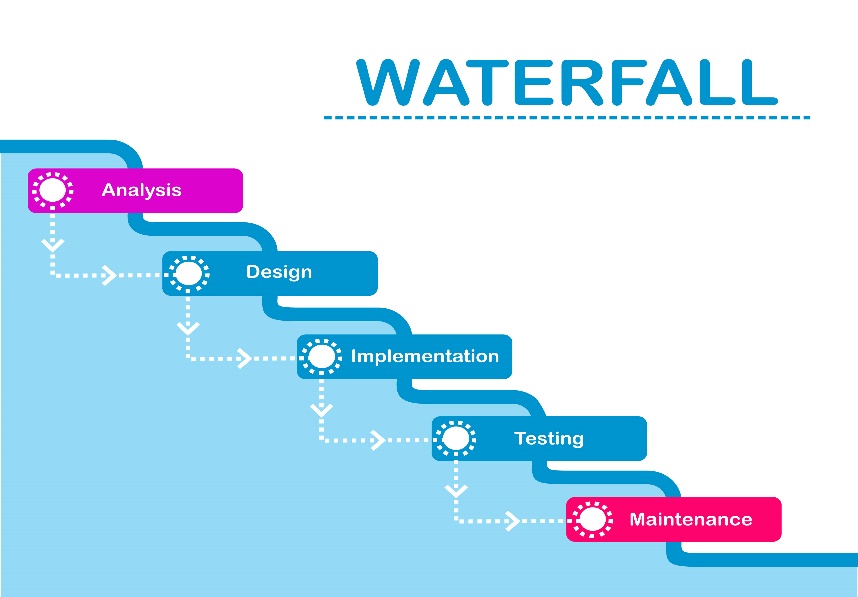


Figure 1- waterfall model diagram

Waterfall model:

The waterfall model is a traditional software development approach characterized by a linear and sequential progression through distinct phases. It consists of six main stages: requirements analysis, system design, implementation, testing, deployment, and maintenance. Each phase relies on the completion of the preceding one, forming a cascading flow of activities. While the model offers clear documentation and well-defined milestones, it lacks flexibility for accommodating changes late in the development process. Additionally, feedback from users or stakeholders is typically solicited only after deployment, potentially leading to significant rework or dissatisfaction. Despite its limitations, the waterfall model remains relevant in certain contexts where requirements are stable, and the project scope is well-understood.

Waterfall model in color detection:

In the realm of color detection system development, a structured approach akin to the Waterfall model can guide the sequential progression through key stages. It commences with requirements gathering, where the system's objectives, scope, and functionality are delineated. This involves defining the types of colors to be detected, environmental conditions, and desired output formats. Subsequently, system design orchestrates the blueprint for the system architecture, encompassing crucial elements such as color space conversion, thresholding techniques, and object identification methodologies. The design phase determines the algorithms to be employed and the integration of libraries like OpenCV for image processing. Following the design phase, implementation ensues, wherein the system is coded according to the design specifications, ensuring seamless integration of algorithms and functionalities. Thorough testing validates the functionality and accuracy of the color detection system, encompassing scenarios with varying lighting conditions, backgrounds, and color distributions. The deployment phase culminates in the integration of the color detection system into target environments, alongside documentation and user instructions for its utilization and maintenance. While this approach provides a structured framework, it may encounter challenges in accommodating iterative improvements or evolving user requirements, necessitating a balance with Agile methodologies to ensure adaptability and responsiveness to changing needs.

* Begin by clearly defining the objectives and requirements of the color detection system. Determine the specific goals, such as real-time color identification and tracking, and any constraints or limitations.
* Research available tools and libraries for image processing and computer vision in Java. Choose the most suitable ones for color detection tasks ensuring compatibility with JavaFX.
* Create a new JavaFX project in Eclipse, ensuring compatibility with the chosen libraries for image processing and computer vision. Set up the project structure and dependencies accordingly.
* Design the user interface for the color detection system using FXML files. Utilize Scene Builder or Eclipse's FXML editor to create visually appealing and user-friendly layouts for interacting with the system.
* Implement dedicated controller classes for each FXML file to handle user inputs and interactions. These controllers will facilitate communication between the user interface and the underlying color detection logic.
* Develop Java classes to encapsulate the color detection algorithms. Implement methods for converting color spaces, applying thresholding techniques, and detecting objects based on their color characteristics.
* Integrate the selected image processing libraries into the project. Utilize their APIs and functionalities for efficient color detection tasks, such as feature extraction and contour detection.
* Thoroughly test the functionality of the color detection system by providing various input images or video streams. Test for different lighting conditions, backgrounds, and color distributions to validate accuracy and robustness.
* Document the implementation details, including the system architecture, algorithms used, and instructions for usage and maintenance.

1. analysis
2. Data analysis:

* Analyze the characteristics of the input data, which includes images or video streams containing objects with different colors, backgrounds, and lighting conditions.
* Measure the accuracy of the color detection algorithms by comparing the detected colors with ground truth labels or user-defined expectations.
* Assess the performance of the color detection algorithms in terms of computational efficiency and processing speed.
* Evaluate the robustness of the color detection system to environmental factors such as changes in lighting conditions, variations in backgrounds, and noise in the input data.

1. Software analysis:

* Identify and analyze the functional requirements of the color detection system, which include the specific tasks and capabilities it must perform
* Analyze the non-functional requirements of the color detection system, which include aspects such as performance, scalability, usability, and maintainability.
* Analyze the integration requirements of the color detection system with external libraries, frameworks, or hardware components.
* Identify potential risks and challenges associated with the development and deployment of the color detection system

1. Hardware analysis:

* Assess the processing power needed to perform color detection tasks efficiently, especially for real-time processing of high-resolution images or video streams.
* Analyze the memory requirements for storing image data, intermediate results, and other computational resources used during color detection.
* Identify the input devices used to capture images or video streams for color detection, such as cameras or image sensors.
* Analyze the environmental conditions in which the color detection system will be deployed, including factors such as temperature, humidity, and lighting conditions.

1. Human Resource:

 We have gained enough experience in java program to complete this project. The supervisor is capable of training this project.

1. External Resource:

On site training internships, workshop, funded projects could be very helpful resources for this project.

1. design

For designing this project, we have developed an architectural diagram, Sequence diagram

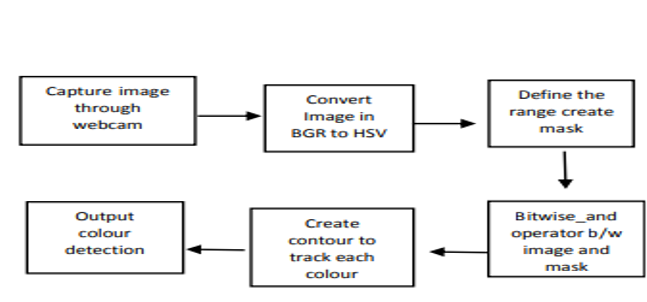
Architectural Diagram: 

Figure 2-larchutecture diagram for color detection using java

Sequence Diagram:

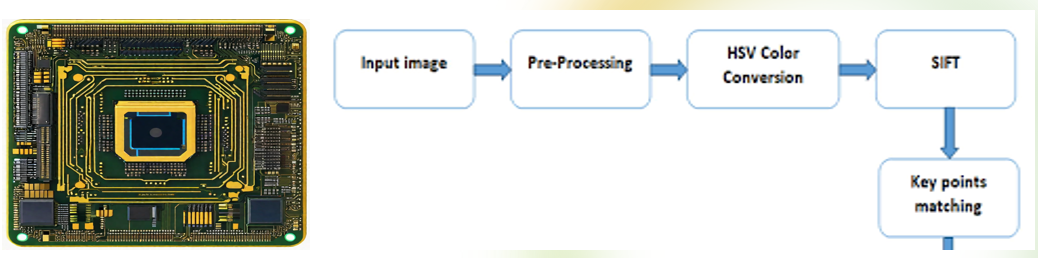


Figure 3-sequence diagram for color detection using java

**FRONTEND DESIGN:** Design a user-friendly graphical interface for the color detection system using JavaFX, incorporating intuitive layouts, visually appealing fonts, and interactive elements.

**BACKEND DESIGN:** Backend design is essential for implementing the core logic, algorithms, and functionalities of the color detection system.

Class diagram:

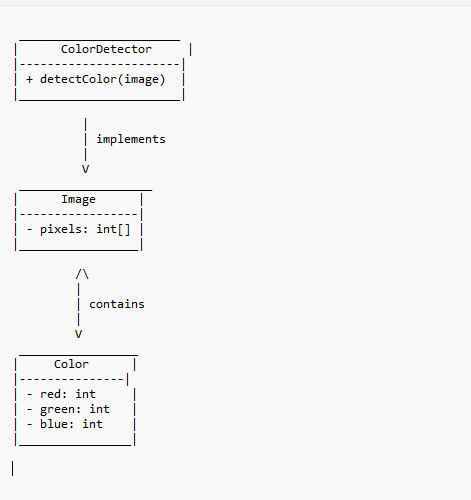


Figure 4-class diagram for color detection using java

1. Coding

Syntax:

* public class MyClass {} – This is a class declaration. MyClass is the name of the class.
* public static void main (String [] args) {} – This is the main method that gets executed when you run your Java program.
* int myVar = 5; – This is a variable declaration. myVar is a variable of type int and is assigned the value 5.
* final double PI = 3.14; – This is a constant declaration. PI is a constant of type double and is assigned the value 3.14
* if (condition) {} – This is an if statement. The code inside the braces {} is executed if the condition is true.
* if (condition) {} else {} – This is an if-else statement. If the condition is true, the code in the first block is executed; otherwise, the code in the else block is executed.
* switch (variable) {case value: break; default: break;} – This is a switch statement. It allows a variable to be tested for equality against a list of values.
* for (initialization; condition; increment) {} – This is a for loop. It is used to repeatedly execute a block of code until a certain condition is met.
* while (condition) {} – This is a while loop. It repeatedly executes a block of code as long as a certain condition is true
* do {} while (condition); – This is a do-while loop. It is similar to a while loop, but the condition is tested after the execution of the block of code.
* int [] myArray = new int [10]; – This is an array declaration. myArray is an array of int type with a size of 10.
* public returnType functionName(parameters) {} – This is a function declaration. functionName is the name of the function, returnType is the data type of the value the function returns, and parameters are input to the function.
* try {} catch (ExceptionType e) {} – This is a try-catch block. It is used to handle exceptions and errors that occur in a block of code.
* MyClass obj = new MyClass (); – This is how to create an object. obj is an object of class MyClass.
* obj. memberName; – This is how to access an object’s members. memberName is the name of a member (variable or method) of the object obj.

Source code:

import java.awt.Color;  
import java.awt.Graphics;  
import java.awt.image.BufferedImage;  
import java.io.File;  
import java.io.IOException;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.List;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.Comparator;  
import javax.imageio.ImageIO;  
  
public class CursorColorFinderextendsJFrame {  
  
 public static void main(String[] args) {  
 try {  
 BufferedImage image = ImageIO.*read*(new File("path\_to\_your\_image.jpg"));  
Map<Integer, Integer> colorCounts = *findDistinctColors*(image);

List<Map.Entry<Integer, Integer>> sortedColorCounts = new ArrayList<>(colorCounts.entrySet());  
 Collections.*sort*(sortedColorCounts, Comparator.*comparingInt*(Map.Entry::getValue));

int numColors = Math.*min* (50, sortedColorCounts.size());  
 Map<Integer, Integer> top50ColorCounts = new HashMap<> ();  
 for (int i = sortedColorCounts.size() - 1; i >= sortedColorCounts.size() - numColors; i--) {  
 Map.Entry<Integer, Integer> entry = sortedColorCounts.get(i);  
 top50ColorCounts.put(entry.getKey(), entry.getValue());  
 }

BufferedImage colorSwatches = *createColorSwatches*(top50ColorCounts);File outputImageFile = new File("color\_swatches.png");  
 ImageIO.*write*(colorSwatches, "png", outputImageFile);  
  
 System.*out*.println("Color swatches image created: " + outputImageFile.getAbsolutePath());  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private static Map<Integer, Integer> findDistinctColors(BufferedImage image) {  
 Map<Integer, Integer> colorCounts = new HashMap<>();  
  
 for (int y = 0; y < image.getHeight(); y++) {  
 for (int x = 0; x < image.getWidth(); x++) {  
 Color pixelColor = new Color(image.getRGB(x, y));  
 int rgb = pixelColor.getRGB()

colorCounts.put(rgb, colorCounts.getOrDefault(rgb, 0) + 1);  
 }  
 }  
  
 return colorCounts;  
 }  
  
 private static BufferedImage createColorSwatches(Map<Integer, Integer> colorCounts) {  
 int swatchSize = 50;int numColors = colorCounts.size();  
 int imageSize = swatchSize \* numColors;  
  
 BufferedImage colorSwatches = new BufferedImage(imageSize, swatchSize, BufferedImage.*TYPE\_INT\_RGB*);  
 Graphics graphics = colorSwatches.getGraphics();  
  
 int x = 0;  
 for (int color : colorCounts.keySet()) {  
 graphics.setColor(new Color(color));  
 graphics.fillRect(x, 0, swatchSize, swatchSize);  
 x += swatchSize;  
 }  
  
 graphics.dispose();  
  
 return colorSwatches;  
 }  
}

1. Testing

**Module Testing:**

* Verify that the load Image module loads images correctly and handles errors appropriately.
* Provide the method with a valid image file path and assert that a non-null BufferedImage object is returned.
* Provide an invalid or non-existent file path and assert that the method handles this scenario gracefully (e.g., returns null or throws an appropriate exception).
* Validate that the detect Color module accurately identifies pixels of a specified color within a given threshold.

**Integration Testing:**

* Verify that different modules or components of the color detection code work together correctly.
* Test the interaction between methods such as load Image and detect Color to ensure they communicate and process data appropriately.
* Verify that the output of one method is correctly passed as input to another method.

**Functional Testing:**

* Validate the functionality of the color detection code against the specified requirements.
* Test the color detection code with various types of images containing different colors, patterns, and sizes.
* Ensure that the code accurately identifies target colors within the specified threshold.
* Verify that the code handles edge cases and boundary conditions effectively, such as images with extreme color values or very few pixels.

**Performance Testing:**

* Evaluate the performance of the color detection code in terms of speed and resource consumption.
* Test the code with images of varying sizes to measure the time taken for color detection.
* Profile the code to identify any performance bottlenecks or areas for optimization.
* Ensure that the code meets the performance requirements, such as processing images within acceptable time limits.

1. IMPLEMENTATION

Color detection in Java involves analyzing an image to identify specific colors or color ranges. The process typically begins by loading an image file, often in formats like JPEG or PNG, using libraries such as ImageIO. Once the image is loaded into memory as a BufferedImage, the algorithm iterates through each pixel of the image. For each pixel, the RGB values are extracted to represent its color. These RGB values are then compared to the target color or color range being detected.

One common method of color detection is to calculate the Euclidean distance between the RGB values of the pixel and the target color. This distance serves as a measure of color difference. If the calculated distance falls within a predefined threshold, the pixel is considered to match the target color. This threshold helps account for slight variations in color due to factors like lighting conditions or image compression.

To implement color detection, you would define a target color and a threshold value. The target color can be specified as an RGB value or using other color representations such as HSL or HSV. The threshold determines how closely a pixel's color must match the target color to be considered a match. A lower threshold results in stricter color matching, while a higher threshold allows for more leniency.

Once the color detection algorithm has processed all pixels in the image, it can output the coordinates of detected pixels or perform additional actions, such as highlighting regions of interest or generating color statistics.

It's important to note that while this approach provides a basic method for color detection, more advanced techniques may be needed for specific applications, such as object recognition or image segmentation. These techniques may involve machine learning algorithms or image processing libraries like OpenCV for more robust color detection capabilities.



Figure 5-input image 1



Figure 6-output for input image 1



Figure 7-input image 2



Figure 8-output for input image 2



Figure 9-input image 3



Figure 10-output for input image 3

1. MAINTENANCE

This project requires maintenance according to software update and hardware changes.

Future enhancement

Enhancing color detection using Java involves a multifaceted approach aimed at improving accuracy, efficiency, and usability. One key area for enhancement lies in the refinement of color recognition algorithms. By incorporating more advanced techniques such as k-means clustering, Gaussian mixture models, or even deep learning-based approaches, developers can achieve better precision in identifying colors within images. These algorithms can better analyze the distribution of colors and adapt to diverse color patterns, enhancing the overall reliability of the detection process.

Additionally, converting between different color spaces such as RGB, HSV, and LAB can offer a more comprehensive representation of colors and their perceptual properties. This conversion facilitates more nuanced analysis and can lead to improved detection results, particularly in scenarios with varying lighting conditions or color complexities. Moreover, integrating adaptive thresholding techniques enables the system to dynamically adjust threshold values based on local image characteristics, enhancing robustness across different environments.

Noise reduction techniques play a crucial role in improving the accuracy of color detection by minimizing interference from irrelevant image artifacts. Employing methods like Gaussian blurring, median filtering, or morphological operations helps to mitigate noise and ensure that the detection process focuses on relevant color information. Furthermore, by implementing region-based detection strategies, the system can prioritize specific areas of interest within the image, optimizing efficiency and accuracy, especially in complex scenes with multiple color elements.

Efforts to optimize performance, ensure cross-platform compatibility, and maintain robustness through thorough testing are essential aspects of the enhancement process. By systematically addressing these areas, developers can create a color detection system in Java that not only meets but exceeds the demands of modern applications, offering enhanced accuracy, efficiency, and user satisfaction.

1. CONCLUSION

In conclusion, color detection using Java provides a versatile and accessible method for analyzing images and identifying specific colors or color ranges within them. By leveraging Java's built-in image processing capabilities and libraries such as ImageIO, developers can implement color detection algorithms to suit various applications and use cases. The process typically involves loading an image, iterating through its pixels, and comparing their RGB values to a target color or color range. With the flexibility to define custom target colors and thresholds, Java-based color detection algorithms can accommodate a wide range of requirements, from simple color identification to more complex image analysis tasks. While the basic approach outlined here provides a foundation for color detection in Java, further enhancements and optimizations can be made using advanced techniques and libraries for specific application domains. Overall, color detection in Java offers a valuable tool for tasks such as image processing, computer vision, and data analysis, empowering developers to extract meaningful insights and information from visual data.

1. ACKNOWLEDGEMENT

We thank IntelliJ IDEA for providing java software as free of cost for developing the project. We thank course faculty who guided our project in a successful manner to complete. We thank mentor for moral support, supervisor for handling technical support. We also thank the management for providing required accommodations.

1. REFERENCES
2. Deborah T. Joy, Gitika Kaur, Aarti Chugh, Shalini Bhaskar, “Computer Vision For Color Detection”, “2021”, <https://www.researchgate.net/publication/353083214_COMPUTER_VISION_FOR_COLOR_DETECTION>
3. Roy S. Berns, “Principle of Color Technology”, “2000”, <https://www.researchgate.net/publication/247251549_Principle_of_Color_Technology>
4. Masoud Shaloo, Gabor Princz, Selim Erol, “ Real-Time Color Detection for Automated Production Lines Using CNN-Based Machine Learning”, “2023”, <https://www.researchgate.net/publication/372896003_Real-Time_Color_Detection_for_Automated_Production_Lines_Using_CNN-Based_Machine_Learning>
5. XiaWen Zhang, Zaho Qiu, Ping Huang, JianZheng Hu, “2018”, “Application Research of YOLO v2 Combined with Color Identification”, <https://www.researchgate.net/publication/331270553_Application_Research_of_YOLO_v2_Combined_with_Color_Identification>
6. Rajini Arali, Sumitha M, Swathi P, Vismaya Halawal, Aravind Naik, “2021”, “Colour Detection from Image”, <https://ijaem.net/issue_dcp/Colour%20Detection%20from%20Image.pdf>
7. DR. Lenina Svb, Anagha Kulkarni, Pranjali Sanjay Jaisingpure, “2017”, “Automated Object Sorting Based On Colour Detection”, <https://www.researchgate.net/publication/320558223_Automated_Object_Sorting_Based_On_Colour_Detection>
8. Pranav Shirole, “2020”, “Creating a Color Detector using OpenCV”, <https://shirolepranav.github.io/blog/computer%20vision/opencv/2020/06/16/color-detection-opencv.html>
9. Sudharshan Duth, M. Mary Deepa, “2018”, “Color detection in RGB-modeled images using MAT LAB, <https://www.researchgate.net/publication/325736179_Color_detection_in_RGB-modeled_images_using_MAT_LAB>
10. Andrew R. Sass, “2013”, “ESP32-CAM Web Server with OpenCV.js: Color Detection and Tracking”, <https://randomnerdtutorials.com/esp32-cam-opencv-js-color-detection-tracking/>
11. Abhik Banerjee, Abhirup Ghosh, Koustuvmoni Bharadwaj, Hemanta Saikia, “2014”, “Mouse Control using a Web Camera based on Colour Detection”, <https://ijcttjournal.org/Volume9/number-1/IJCTT-V9P104.pdf>