AN INTELLIGENT WEBSITE TO PROVIDE EASY ACCESS TO COLORING PAGES USING EDGE DETECTION

Grace Zan[1], Samin Salman[2]

[1]University High School, 4771 Campus Drive, Irvine, CA 92612
[2]Computer Science Department, California State Polytechnic University, Pomona, CA 91768

ABSTRACT

Coloring books are used by children and adults to relax and relieve stress. However, there are limited subject matters, and it may be hard to find exactly what you want to color. By creating a program that converts any image into a custom coloring page, you no longer have to spend as much time searching for a suitable coloring book and have more variety in what you want to color. Our program allows the user to import an image and convert it into a coloring page using edge detection. It provides three different types of edge detection that the user can pick from and also allows the user to pick two to blend together to create a more detailed, fuller image. There is also a webcam feature that displays an edge-detected live feed to provide an augmented reality of the user's environment in an outlined format. By using this program, you can quickly and easily create a coloring page that will suit your tastes.

KEYWORDS

Edge Detection, Coloring Pages, Stress Relief, AI

1. INTRODUCTION

Coloring books are a popular hobby, especially among children. While there are a variety of coloring books available, more niche subject matter may be difficult to find, if at all possible. By having the option to create custom coloring pages using edge detection, teachers and parents would be able to have coloring pages of any theme they want without having to spend time searching for specific coloring books. For adults, coloring books have been shown to reduce anxiety and promote concentration and perseverance, more so when given the choice to choose the colors they use [5]. Coloring is able to provide short-term stress relief and is especially helpful when used as a distraction from negative emotions [15]. Furthermore, it is able to reduce depressive symptoms and, when used regularly, can be an effective tool for self-help [6]. Mandala coloring has been shown to have therapeutic effects, and there exist games that utilize the relaxing properties that coloring has [12][1]. In children, coloring has a similar anxiety-reducing ability affecting both boys and girls [3]. Coloring is also better at improving mood than drawing freely to express emotions as well and also does not require any specific skills [14]. In addition to emotional benefits, coloring books have been used to help accurately measure lexicon levels in young children [13].

There are a few programs that convert photos into coloring pages for kids and adults to enjoy. Apps exist that can take photos and transform them into pages to be colored. Some programs

David C. Wyld et al. (Eds): AdNLP, CDKP, NCO, SAI, MLT, Signal, SOFT, ICAITA, CMC - 2024
pp. 245-254, 2024. CS & IT - CSCP 2024 DOI: 10.5121/csit.2024.141022

allow the user to color photos within the app but do not provide an option to download or print the image. Our website does not have this issue and allows users to easily download the finished edge-detected images. For most programs, the images that come out do not have a clear outline that would be desirable for a coloring page, often having blurry or transparent edges. Frequently, the photos are shaded in shadows, ignoring the outlines of objects. Without clear edges, coloring the pages made through these programs could prove to be difficult, especially for children. Having blurry and translucent gray edges also leads to a muddier result [7]. To combat this, we tried to combine different edge detection methods to capture as many outlines as possible while having a clean result and providing the user with different options to find the edge detection method that best suits their tastes.

To solve this problem, it would be ideal to create a website that allows people to use edge detection to convert any image into linework that can be colored in. Edge detection compares pixels of an image with one another to find the edges of objects. The website created is easily accessible on all devices; therefore, it lessens the burden of having to download an application. Compared to other similar applications, ours will allow the user to easily save the completed photos onto their devices for printing so that they may be colored by hand, which would leave more of an impact than simply drawing on a screen. The feel of paper allows the person coloring to have a better grasp on the space they are coloring [8]. Coloring in a physical piece also allows the person coloring to look over their work more broadly than if it were limited to a screen. Therefore, providing the option to print out the drawings to color them physically would allow the user to decompress and unwind more in the process. In addition, excessive screen use can hinder learning ability and lead to a lack of focus, which can make students frustrated and struggle more in the long term [11]. Screen usage also leads to a decrease in blinking, which in turn causes dry eyes in many users [10]. Dry eyes can cause headaches and eye pain, which, instead of allowing people to relax, would cause further distress.

The first experiment we set up was to counter a blind spot in the edge detection program, which caused photos not to come out as expected. To counter this, we try to combine different detection methods in order to get the optimal result. To run this experiment, we conducted an experiment with several different detection methods to see how other people felt about the different methods for a coloring book. We found that depending on the photo, the preferred edge detection differed. Because of this, we decided to put different types of edge detection on the result page of our website. Our second experiment covered the quality of the live feed. The main issue was how, depending on the lighting captured, the number of edges shown and objects captured varied. In the experiment, we compared the number of edges in two images of the same room in different lighting at different thresholds. Each image had the most edges at different thresholds. To counter this, we implemented buttons that allow the user to adjust the threshold of edge detection.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. The edge detection

The edge detection methods we tested might not always capture every edge in an image, leaving blank areas where an edge should be. Each edge detection method by itself may not be suitable for every type of picture. To counter this, one could overlay two images, each going through a different edge detection method so that the edges that were not found using one method would still appear because they were found with the other. For example, parts of images with less contrast that are not caught using the contour method will be caught using the canny method. The

same happens vice versa, with areas not captured with canny being caught with contour. When the two images are overlaid, the different parts overlap, resulting in a more complete picture.

2.2. Low-resolution image

Edge detection works by scanning and comparing pixel intensities to one another in order to determine where the edges are located. An issue that arises is when a user submits a low-resolution image into the program. Because lower-resolution images are composed of fewer pixels, the edges found would appear more pixelated compared to a smoother line that would result from a higher-resolution image. To resolve this, one could enlarge and blur the image to make the pixels less visible, which would allow the website to detect cleaner edges, which are a result of the contrast between the different color blobs in the image.

2.3. Image size

For the website to be easily accessible, it should be user-friendly on all screen sizes. An image sized to fit a computer screen may be too large for a mobile device. Similarly, an image made to fit the size of a mobile phone may appear too small on a computer. The ideal would be to have image size proportionate to the screen of the device being used. To achieve this, we would need to make the components of the website responsive to different screen sizes. This is possible by using CSS to style the necessary portions of the page to respond to different screen sizes. This way, the website would be usable on a variety of different devices.

3. SOLUTION

The website is written using the Python programming language and a flask framework. The first page the user is met with is the home page, with a panel at the top that allows the user to navigate between the home page, about page, and convert page. On the home page, the user is able to upload an image of their choosing to be edge detected. The user is also able to choose which method of edge detection they want to use, as well as a second one in case they want to use more than one. This will lead you to a page of results, where the image using the chosen edge detection method is shown, as well as a version with the second method and an overlaid version using both, which are accessed and downloaded onto your device. The About page explains the individual methods of edge detection and briefly explains how they work. There are links provided for more information. The Convert page has a webcam that captures the live feed from the camera the user has connected and detects the edges of each frame. Buttons are provided to allow the user to adjust the threshold for contour detection to match the lighting of the environment the user is in. Additionally, there is a fun feature to turn on rainbow mode, which changes the color of the detected outlines in each frame to the next color on the spectrum.

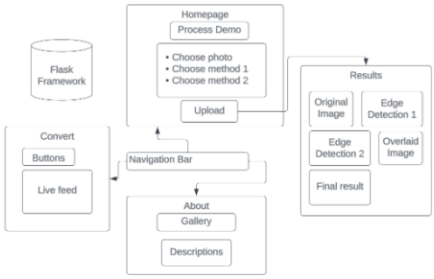


Figure 1. Overview of the solution

The purpose of the homepage is to allow the user to upload an image of their choosing and download an edge-detected version of it. The uploaded image is taken and run through an edge detection program that corresponds to the user's choices. The images are then overlaid, inverted, and given back to the user.

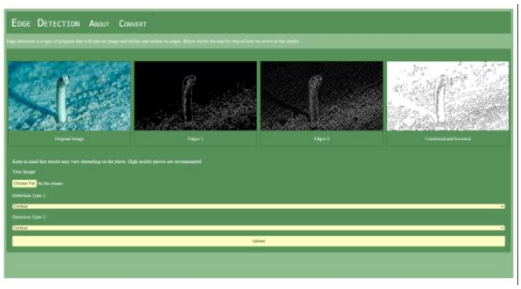
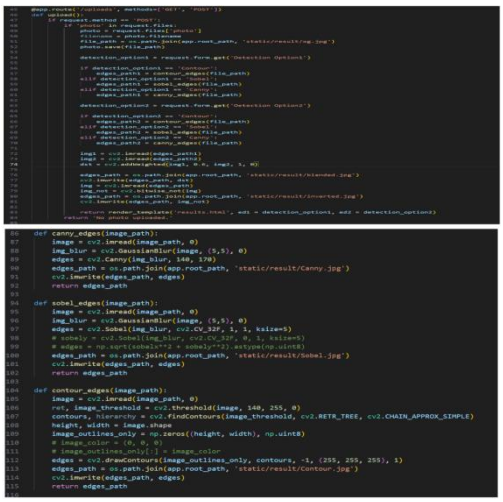


Figure 2. Screenshot of the home page



```python
@app.route('/upload', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        if 'photo' in request.files:
            photo = request.files['photo']
            file_path = os.path.join(app.root_path, 'static/result/og.jpg')
            photo.save(file_path)

            detection_option1 = request.form.get('Detection Option1')

            if detection_option1 == 'Contour':
                edges_path1 = contour_edges(file_path)
            elif detection_option1 == 'Sobel':
                edges_path1 = sobel_edges(file_path)
            elif detection_option1 == 'Canny':
                edges_path1 = canny_edges(file_path)

            detection_option2 = request.form.get('Detection Option2')

            if detection_option2 == 'Contour':
                edges_path2 = contour_edges(file_path)
            elif detection_option2 == 'Sobel':
                edges_path2 = sobel_edges(file_path)
            elif detection_option2 == 'Canny':
                edges_path2 = canny_edges(file_path)

            img1 = cv2.imread(edges_path1)
            img2 = cv2.imread(edges_path2)
            dst = cv2.addWeighted(img1, 0.5, img2, 1, 0)

            edges_path = os.path.join(app.root_path, 'static/result/blended.jpg')
            cv2.imwrite(edges_path, dst)
            img = cv2.imread(edges_path)
            img_not = cv2.bitwise_not(img)
            edges_path = os.path.join(app.root_path, 'static/result/inverted.jpg')
            cv2.imwrite(edges_path, img_not)

            return render_template('results.html', edt = detection_option1, edt2 = detection_option2)
    return 'No photo uploaded.'
```

```python
def canny_edges(image_path):
    image = cv2.imread(image_path, 0)
    img_blur = cv2.GaussianBlur(image, (5,5), 0)
    edges = cv2.Canny(img_blur, 140, 170)
    edges_path = os.path.join(app.root_path, 'static/result/Canny.jpg')
    cv2.imwrite(edges_path, edges)
    return edges_path

def sobel_edges(image_path):
    image = cv2.imread(image_path, 0)
    img_blur = cv2.GaussianBlur(image, (5,5), 0)
    edges = cv2.Sobel(img_blur, cv2.CV_32F, 1, 1, ksize=5)
    # sobely = cv2.Sobel(img_blur, cv2.CV_32F, 0, 1, ksize=5)
    # edges = np.sqrt(sobelx**2 + sobely**2).astype(np.uint8)
    edges_path = os.path.join(app.root_path, 'static/result/Sobel.jpg')
    cv2.imwrite(edges_path, edges)
    return edges_path

def contour_edges(image_path):
    image = cv2.imread(image_path, 0)
    ret, image_threshold = cv2.threshold(image, 140, 255, 0)
    contours, hierarchy = cv2.findContours(image_threshold, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    height, width = image.shape
    image_outlines_only = np.zeros((height, width), np.uint8)
    # image_color = (0, 0, 0)
    # image_outlines_only[:] = image_color
    edges = cv2.drawContours(image_outlines_only, contours, -1, (255, 255, 255), 1)
    edges_path = os.path.join(app.root_path, 'static/result/Contour.jpg')
    cv2.imwrite(edges_path, edges)
    return edges_path
```

Figure 3. Screenshot of code 1

When an image is uploaded, the file is saved into a folder to be edited later on. The form accesses the options the user can choose from. Based on the chosen option, the corresponding detection option is called, and the option is given the path to the image that was previously saved. After the edge detection is completed, the new image is saved into a new path, which will later be used to reference the images that will be overlaid. This step is repeated for the second type of detection. The edge detection results are then combined into one image using the addWeight() function. The function overlays two images on top of each other. Each edge detected image, including the overlaid version, is then inverted using bitwise_not() so the results have a white background and black outlines instead of the black background and white outlines, which are the default. Finally, after the calculations, the results are rendered on the results page.

The About section of the website gives the user more context on what the different methods of edge detection are and how they work. Beneath each description is a link to an external website, which provides further information on the method of edge detection. For those who are interested, this page gives the user more insight into what the best method for their coloring page would be.

Figure 4. Screenshot of the about section

```
<div class="responsive">
  <br><br>
  <div class="gallery">
    <a href="#Contour">
      <img src="../static/Contournoth.jpg" alt="loading Contour" width="252" height="142">
    </a>
    <div class="desc">Contour</div>
  </div>
</div>

<div class="responsive">
  <br><br>
  <div class="gallery">
    <a href="#Canny">
      <img src="../static/Cannynoth.jpg" alt="loading canny" width="252" height="142">
    </a>
    <div class="desc">Canny</div>
  </div>
</div>

<div class="responsive">
  <br><br>
  <div class="gallery">
    <a href="#Sobel">
      <img src="../static/Sobelnoth.jpg" alt="loading sobel" width="252" height="142">
    </a>
    <div class="desc">Sobel</div>
  </div>
  <br><br>
</div>
```

```
.responsive {
  padding: 0 6px;
  float: left;
  width: 24.99999%;
  margin: 6px 0;
}

@media only screen and (max-width: 700px) {
  .responsive {
    width: 49.99999%;
    margin: 6px 0;
  }
}

@media only screen and (max-width: 400px) {
  .responsive {
    width: 100%;
  }
}
```

Figure 5. Screenshot of code 2

To make this page more compatible with different devices, we made the gallery responsive. This means the size or number of images on each line will change depending on the size of the screen. Based on the more common screen sizes, the width of each image is scaled to control how many images would show up on a screen at a time. This means that on smaller devices, only one image is shown, which is at full size. On bigger screens, this would be balanced out by showing multiple images on each line. Each image displayed is connected to a hyperreference to a portion of the page that correlates to the explanation of the version of edge detection used on the image shown. The images can also be opened in another tab, allowing the user to get a closer look at the details captured by the different methods of edge detection.

The webcam captures live feed, which is separated into frames that each go through the contour edge detection process. These frames are then displayed directly on the website as the process is finished. There are buttons that allow the user to adjust the threshold for contour detection to suit the user's environment.

Figure 6. Screenshot of the frames

```python
@app.route('/convert', methods=['GET', 'POST'])
def convert():
    global threshold_value
    global rainbow_mode
    if request.method =='POST':
        action= request.form["threshhold"]
        if action == 'increase':
            threshold_value = threshold_value + 10
            if threshold_value > 255:
                threshold_value = 255
        elif action == 'decrease':
            threshold_value = threshold_value - 10
            if threshold_value < 0:
                threshold_value = 0
        elif action == 'rb':
            rainbow_mode = not rainbow_mode

    gen_frames(threshold_value, rainbow_mode)
    return render_template('convert.html')

@app.route('/video_feed')
def video_feed():
    return Response(gen_frames(threshold_value, rainbow_mode), mimetype='multipart/x-mixed-replace; boundary=frame')
```

```python
def gen_frames(threshold_value, rainbow_mode):
    while True:
        success, frame = camera.read()  # read the camera frame
        if not success:
            break

        else:
            image_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            ret, image_threshold = cv2.threshold(image_gray, threshold_value, 255, 0)
            contours, hierarchy = cv2.findContours(image_threshold, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
            height, width, channels = frame.shape
            image_outlines = np.zeros((height,width,channels), np.uint8)
            image_color = (255,255,255)
            image_outlines[:] = image_color
            outline_thickness = 2
            outline_color = (0,0,0)
            contour_id = -1

            if rainbow_mode:
                outline_color = get_next_color()
                image_color = (0,0,0)
                image_outlines[:] = image_color
            else:
                outline_color = (0,0,0)

            frame_edges = cv2.drawContours(image_outlines, contours, contour_id, outline_color, outline_thickness)
            ret, buffer = cv2.imencode('.jpg', frame_edges)
            frame = buffer.tobytes()

        yield (b'--frame\r\n'
            b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')  # concat frame one by one and show result
```

Figure 7. Screenshot of code 3

Similar to section 3.2, we use forms to get data from the user. However, instead of the selection box on the homepage, buttons are used. Based on the button pressed, the threshold value increases or decreases the threshold_value variable by a value of 10. If the rainbow mode button is pressed, the rainbow mode will turn on, which changes the background color to black and the outline color to alternate between colors of the rainbow. Inside the gen_frames() function, we read the camera into individual frames. Then, the function will use the information collected from the forms to transform each frame of the live feed into an edge-detected version. The thickness and color of the outline are specified within the function. This process is repeated over and over for each frame that gets captured. After each frame is processed, we concatenate frames one by one and send them to the webpage.

4. EXPERIMENT

4.1. Experiment 1

A blind spot within the edge detection program is that it may not detect as many edges as intended. This is important because if an object is missing edges, it will leave an incomplete picture to color.

To counter this issue, we will try several different methods. We will use Contour, Canny, and Sobel. One method alone does not provide satisfactory results, so by combining them, we will be able to create a more complete picture. We overlay the images on top of each other so that both images' edges show up. We decided to do a survey to find the most preferred method of edge detection. The survey was conducted by asking each survey taker to choose the most suitable edge-detected outcome on different images. A survey was chosen because the suitability of an image for a coloring page is subjective based on whoever is going to be using it. Therefore, the best option would be to record the opinions of potential users.

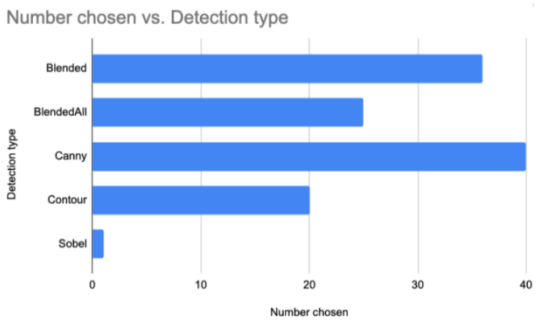


Figure 8. Number chosen vs. Detection type

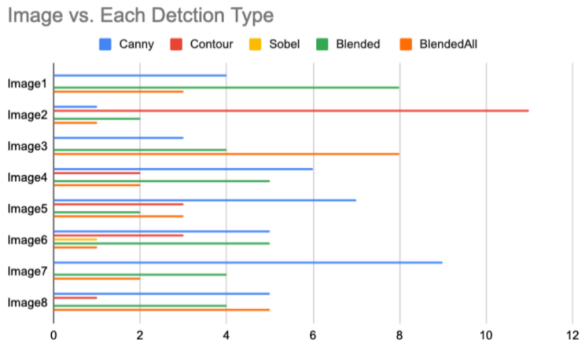


Figure 9. Image vs. Each Detection Type

The first graph shows the total number of votes per type of edge detection added together. Before conducting the survey, we expected the Blended or BlendedAll options to be the most popular; however, Canny was the method that ended up getting the most votes. Although Canny had more total votes, the results for each individual image varied. The second graph shows the votes per type of edge detection for each of the eight images present in the survey. In this graph, we can see that three images out of the eight surveyed had the most votes going to canny, with two where canny tied with other forms of edge detection, Blended, and BlendedAll. Of the remaining images, one had contour voted significantly more than the other options, with another having Blended being voted the most and one having BlendedAll voted the most. The biggest effect on the results is the opinion of the people surveyed. Many people preferred darker, more visible lines rather than the lighter lines of two images overlaid on top of each other.

4.2. Experiment 2

Another fault with the edge detection program is the quality of the live feed. The program may be unable to capture details in faces or other objects in certain lighting. By not showing enough details, objects being recorded may be unrecognizable.
To counter this, we will give the user the option to change the threshold to match the environment being recorded. Different threshold values change the edges being captured. We will implement buttons that can increase or decrease the threshold value in increments of 10. For our experiment, we decided to compare a series of different images. Two lighting scenarios, bright and dim, will be captured using the edge-detection webcam. For both levels of brightness, screenshots will be taken at each of five different threshold values. After getting the pictures, we will import them into a separate website to find the ratio of pixels of the color of the outline to pixels of the color of the background [9]. Doing this will show us which threshold value reveals the most edges for each type of lighting.

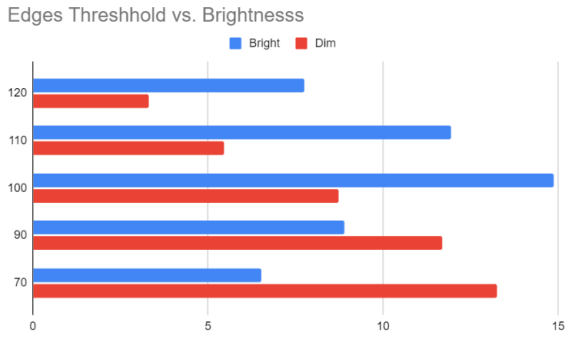


Figure 10. Edges Threshold vs. Brightness

The graph above takes the percentage of pixels that represent edges to pixels that make up the white background of photos taken of the webcam in different lighting and using different thresholds. The blue bars represent the images taken in brighter light, while the red bars represent the images in dimmer light. In both lighting scenarios, photos were taken with a threshold value of 120, 110, 100, 90, and 70. In brighter light, the image that showed the highest percentage of edges was at a threshold of 100, and the second highest percentage was at 110. At a threshold of 90, the percentage of edges was lower than it was at 110; however, it was still higher than the threshold at 120. The ideal threshold for brighter lighting would be somewhere between 100 and 110. Both increasing and decreasing the threshold resulted in a lower percentage of edges. On the other hand, photos taken in dim lighting saw an increased percentage of edges as the threshold was lowered, with the highest percentage of edges in our recorded data being at a threshold value of 70.

5. RELATED WORK

An app made as a graduation project in the Information Technology Department at King Saud University creates coloring pages via Canny edge detection [2]. The app allows the user to take a photo and put it through edge detection to create an outline. The user is able to adjust the difficulty of the coloring page by choosing between easy, medium, and advanced. The app has a feature that allows the user to color in the pages within the app and then save it in a virtual gallery. However, the user is unable to save the image onto their device to print it out physically. In contrast, our program allows the user to enlarge the edge-detected images and download them to print at a later date or import them into a separate coloring app of their choice.

Mimi Panda allows the user to import a photo and turn it into a coloring page [16]. Using the free option, the photo does get converted into edges, however, there are areas that remain gray and blurry, which could muddy an image when colored, especially using lighter or more transparent mediums such as markers or colored pencils. Furthermore, the lines in the converted image are inconsistent. There are certain blurred lines that appear lighter in color than the sharper lines, which may make them harder to identify. It could be confusing to the user whether an edge is actually a line or not if it is lighter in color than the others. Our program, however, allows the user to pick between three different types of edge detection in order to fit the image best. Picking canny or contour gives a result with edges that are solid and smooth.

Crayola is able to create custom coloring pages from imported photos [4]. The site provides a variety of different options the user can choose from in order to account for different types of lighting. On the mobile app, there is a camera feature that can take a photo and turn it into a coloring page immediately. With it, there is adjustable brightness that can slide up or down to change the darkness of the lines. Issues present in the Crayola website include lack of definition on the lines and how dark areas are colored in instead of outlined. In contrast, our program provides results with higher resolution and clear outlines for all objects.

6. CONCLUSIONS

For the live feed feature of our website, the user has to manually adjust the threshold for the image to look right in different kinds of lighting. A way to fix it would be to automate the process by checking the percentage of edges on the screen, finding the optimal threshold that shows the most, and adjusting it. One way to do this automation process would be to write a program to check the number of edges in the current photo and compare that to a set optimal number. Then, depending on if the number of edges is less than the optimal number, it would bring up the threshold and check to see if the edges got closer to the optimal number. If they do get closer, we continue to raise the threshold, and if it gets less, we lower the threshold instead. We repeat this process until the optimal number has been reached.

One limitation of our website is how the user has to manually adjust the threshold in the live feed feature to find the optimal threshold. To improve the live feed feature of our website, we would adjust the threshold to the captured feed automatically.

REFERENCES

[1] Agrawal, Vasundhara, et al. "Color Me: A Game based on art therapy for mental health." Extended Abstracts of the 2020 Annual Symposium on Computer-Human Interaction in Play. 2020.
[2] Alsaaran, Hessah, et al. "Edge Detection for a Children's Coloring Application." 2018 21st Saudi Computer Society National Computer Conference (NCC). IEEE, 2018.
[3] Carsley, Dana, and Nancy L. Heath. "Evaluating the effectiveness of a mindfulness coloring activity for test anxiety in children." The Journal of Educational Research 112.2 (2019): 143-151.
[4] Herrera, Daniel, Juho Kannala, and Janne Heikkilä. "Joint depth and color camera calibration with distortion correction." IEEE Transactions on Pattern Analysis and Machine Intelligence 34.10 (2012): 2058-2064.
[5] Eaton, Judy, and Christine Tieber. "The effects of coloring on anxiety, mood, and perseverance." Art Therapy 34.1 (2017): 42-46.
[6] Flett, Jayde A. M., et al. "Sharpen your pencils: Preliminary evidence that adult coloring reduces depressive symptoms and anxiety." Creativity research journal 29.4 (2017): 409-416.
[7] Forkosh, Jennifer, and Jennifer E. Drake. "Coloring versus drawing: Effects of cognitive demand on mood repair, flow, and enjoyment." Art Therapy 34.2 (2017): 75-82.
[8] Jabr, Ferris. "Why the brain prefers paper." Scientific American 309.5 (2013): 48-53.

[9] Saveanu, Catalina Iulia, et al. "Evaluation of the Efficiency of Hand Hygiene Technique with Hydroalcoholic Solution by Image Color Summarize." Medicina 58.8 (2022): 1108.
[10] Mehra, Divy, and Anat Galor. "Digital screen use and dry eye: a review." Asia-Pacific Journal of Ophthalmology 9.6 (2020): 491-497.
[11] Mineshita, Yui, et al. "Screen time duration and timing: effects on obesity, physical activity, dry eyes, and learning ability in elementary school children." BMC Public Health 21 (2021): 1-11.
[12] Sarah, Noor, et al. "Mandala-coloring as a therapeutic intervention for anxiety reduction in university students." (2017): 904-907.
[13] Pinto, Manuela, and Shalom Zuckerman. "Coloring Book: A new method for testing language comprehension." Behavior research methods 51.6 (2019): 2609-2628.
[14] Samuel, Bosomtwe, et al. "The effects of coloring therapy on patients with generalized anxiety disorder." Animal Models and Experimental Medicine 5.6 (2022): 502-512.
[15] Simmons, Courtney. "Effects of Coloring on Immediate Short-Term Stress Relief." (2016).
[16] Blackburn, Heidi, and Claire E. Chamley. "Color me calm: Adult coloring and the university library." (2016).