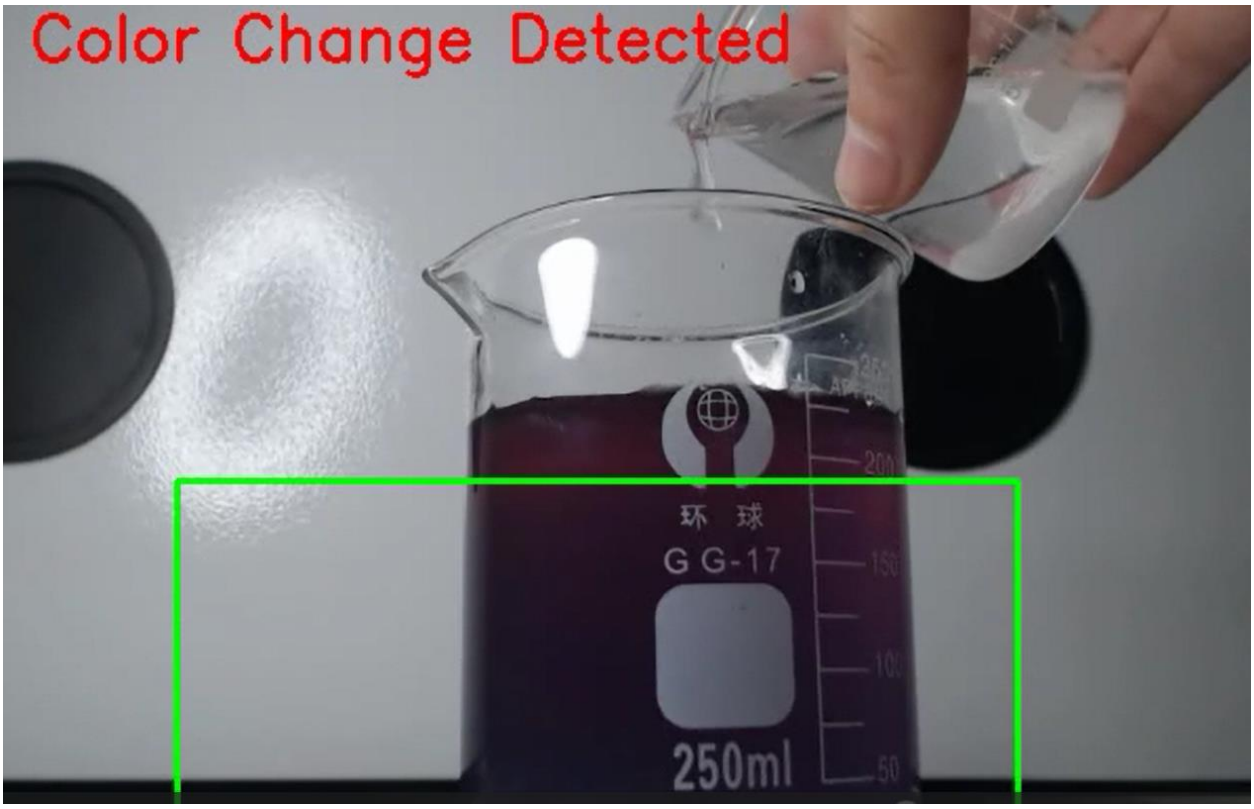
Color Change Detection Program



Author: Paras Khosla

Date: 31-05-2024

Department: Information Communication & Technology

Table of Contents

Table of Figures

Introduction

This report outlines the implementation of a color change detection system using the OpenCV (docs.opencv, n.d.) library in Python. The system is designed to detect color changes in a specific region of interest (ROI) in a video stream, focusing on the color burgundy. When a color change is detected, the system records a video segment and stops after 10 seconds. If no color change is detected for one minute, the system stops recording and saves the video.

Algorithm Overview

The algorithm involves capturing frames from a camera, extracting a specific region of interest (ROI), converting the color space of the ROI to HSV, and determining if there is a significant change in the color within the ROI. The steps are as follows:

1. **Capture frames from the camera**.
2. **Extract the ROI** from each frame.
3. **Convert the ROI to HSV color space**.
4. **Threshold the HSV image** to detect the target color.
5. **Calculate the percentage of the ROI occupied by the target color**.
6. **Determine if a significant color change has occurred**.
7. **Record video** if a color change is detected and stop recording after 10 seconds.
8. **Stop the script** if no color change is detected for one minute and save the recorded video.

Key Functions

HSV Color Space

HSV (Hue, Saturation, Value) color space is used to make color detection more robust:

- **Hue** represents the color type.
- **Saturation** represents the vibrancy of the color.
- **Value** represents the brightness of the color.

The color detection thresholds (`lower_color` and `upper_color`) in the `detect_color_change` function might need to be adjusted based on the specific target color and lighting conditions.

By converting the ROI from BGR to HSV (Answers.opencv, 2018), we can define a color range for more accurate detection:

```
# Convert BGR image to HSV (Hue, Saturation, Value)
hsv = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)

# Define lower and upper bounds for the original color in HSV
lower_color = np.array([160, 100, 50])   # Lower HSV values for the original color
upper_color = np.array([180, 255, 255])  # Upper HSV values for the original color

# Threshold the HSV image to get only the original color
mask = cv2.inRange(hsv, lower_color, upper_color)
```

Figure 1: Hsv values for detecting color change.

This conversion and thresholding process isolates the target color (in this case burgundy) in the HSV color space, allowing for more precise detection. This step creates a binary mask where the target color is white (255) and all other colors are black (0).

Calculate Color Percentage:

This formula calculates the percentage of the ROI occupied by the target color by asumming (Jovian, n.d.) the white pixels in the mask and dividing it by the total number of pixels in the ROI.

```
# Calculate the percentage of the ROI occupied by the original color
color_percentage = np.sum(mask == 255) / (roi.shape[0] * roi.shape[1]) * 100
```

Figure 2: Formula for calculating percentage of defined color.

The mask created from thresholding the HSV image helps in calculating the percentage of the ROI occupied by the target color. This formula divides the number of white pixels in the mask by the total number of pixels in the ROI, converting it to a percentage.

Algorithm for Color Change Detection

The algorithm checks if the calculated color percentage is below a specified threshold to detect a significant color change:

```
# Determine if there's a significant color change
color_change_detected = color_percentage < 2  # Adjust threshold as needed
```

Figure 3: Threshold for color change detection.

If the percentage of the target color is less than 2%, a color change is detected. This threshold can be adjusted based on the specific requirements of the experiment or application.

Region of interest (ROI)

The ROI is a specific area of the frame that is monitored for color changes, and it helps in the increase of responsiveness of color change detection. In this implementation, the ROI is defined as:

```
# Define region of interest (example: focus on center of the frame, wider and lower)
x, y, w, h = 120, 250, 400, 230
# I adjusted the x coordinate to be smaller (120) to move the ROI towards the left of the frame.
# I adjusted the y coordinate to be larger (250) to move the ROI further down in the frame (to simulate an object on the floor).
# I increased the w (width) to 400 to make the ROI wider.
# I decreased the h (height) to 230 to keep the ROI relatively narrow.
```

Figure 4: Region of interest (ROI) values defined.

This means the ROI (answers.opencv, n.d.) is a rectangle starting at (120, 250) with a width of 400 and a height of 230 pixels. This specific ROI configuration focuses on the center of the frame, with a wider and lower position to simulate an object on the floor.

Video Recording

In the setup in the image below:

- fourcc: Defines the codec for video encoding.
- output_path: Specifies the file path for the output video.
- os.makedirs(...): Creates the output directory if it doesn't exist.

```
# Video recording setup
fourcc = cv2.VideoWriter_fourcc(*'XVID')
output_path = 'D:/testing/output.avi'
os.makedirs(os.path.dirname(output_path), exist_ok=True)
out = cv2.VideoWriter(output_path, fourcc, 20.0, (640, 480))
recording = True
```

Figure 5: Video recording setup.

- out: Initializes the “VideoWriter” object with the specified codec, frame rate (20 fps), and frame size (640x480).

Displaying text when color change detection

When a color change is detected:

- **Initialization**: If “color_change_time” is None, it means this is the first detection of a color change. In this case, “color_change_time” is set to the current time.
- **Text Overlay**: The program adds text to the video frame using the cv2.putText function. This text indicates that a color change has been detected. Parameters include the frame (frame), text content ('Color Change Detected'), position (coordinates (50, 50) from the top-

left corner), font type (cv2.FONT_HERSHEY_SIMPLEX), font scale (1), color (red in BGR format: (0, 0, 255)), and thickness (2).

```
# Display text on the top of the frame if color change is detected
if color_change_detected:
    if color_change_time is None:
        color_change_time = time.time()
        change_detected_counter += 1 # color change detection counter
        print(change_detected_counter)  # Print counter when color change is detected

    cv2.putText(frame, 'Color Change Detected', (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
```

Figure 6: Displaying text on video frame after color change detection.

Cutting Video After 10 Seconds

The program is designed to stop recording 10 seconds after a color change is detected. This is achieved through the following mechanism:

1. **Color Change Detection:** When the program detects a color change, it records the time of this event as `color_change_time`.
2. **Conditional Check:** Within the main loop, the program continuously checks if 10 seconds have elapsed since `color_change_time`. If so, it stops the recording by setting the `recording` flag to `False`.

```
# Stop recording 10 seconds after color change detected
if time.time() - color_change_time >= 10:
    recording = False
```

Showing Video Duration

In the context of the provided code, the duration of the recorded video is calculated and printed to the terminal. This process involves several key steps:

1. **Recording Start Time:** When a color change is detected for the first time, the current time is recorded as `recording_start_time`. This marks the beginning of the video recording.
2. **Duration Calculation:** Once the recording is stopped (either due to manual interruption or after 10 seconds following a color change), the duration of the recording is calculated. This is done by subtracting `recording_start_time` from the current time at the moment recording stops.

```python
# Calculate the duration of the recorded video
if recording_start_time is not None:
    duration_time = time.time() - recording_start_time
    minutes = int(duration_time // 60)
    seconds = int(duration_time % 60)
    duration = f"{minutes}:{seconds}"
    print(f"Duration of the recorded video: {duration}")
else:
    duration = None
```

Personal Reflection

Working on this project has been an enlightening experience, giving me valuable insights into the practical applications of computer vision and color space transformations. I learned the importance of different color spaces, especially the HSV color space, which is more intuitive for color detection tasks than the RGB color space. Working with Regions of Interest (ROIs) helped me focus on specific areas of an image, making the detection process more efficient. Implementing the algorithm to detect color changes enhanced my understanding of image processing techniques like color thresholding and mask creation. I also gained hands-on experience with real-time systems by incorporating time-based logic to control video recording and script termination based on color detection events. Handling various edge cases, such as ensuring the script stops correctly and the video is saved even if no color change is detected for a minute, improved my problem-solving skills. This project has solidified my knowledge in computer vision and provided a foundation for future work in more advanced image processing and real-time video analysis applications. It has also taught me the importance of precise parameter tuning and robust error handling in creating reliable software systems.

Conclusion

This project successfully implements a color change detection system using OpenCV. By defining specific HSV ranges and processing a region of interest, the system can robustly detect changes in color and handle video recording based on these detections. The script ensures that video recording stops after 10 seconds of detection and halts if no changes are detected for one minute, saving the recorded video for further analysis.

The system is highly adaptable and can be fine-tuned for various applications by adjusting the HSV thresholds and ROI parameters. This makes it suitable for a wide range of laboratory experiments

and other scenarios where color change detection is critical.

The testing videos “Color_change_with_Liquid_testing” and “Color_change_detection_video_cutting” can be found on this link.

References

Answers.opencv. (2018, Jan 05). *Opencv.org*. Retrieved from Opencv.org: https://answers.opencv.org/question/181705/how-to-define-lower-and-upper-range-of-two-or-more-different-color/

answers.opencv. (n.d., n.d. n.d.). *answers.opencv*. Retrieved from opencv.org: https://answers.opencv.org/question/18619/region-of-interest-in-video-file/

docs.opencv. (n.d., n.d. n.d.). *docs.opencv*. Retrieved from docs.opencv.org: https://docs.opencv.org/4.x/

Jovian. (n.d., n.d. n.d.). *jovian.com*. Retrieved from jovian.com: https://jovian.com/raphaelhaddadny/image-analysis-opencv-color-percentage