EDS 6397 – DIGITAL IMAGE PROCESSING

Group - 3

UNIVERSITY OF HOUSTON

Spring 2024

Dr Lucy Nwosu

Authors:

JASWANTHI BOYAPATI
BINDU SRI CHINTA
REBECCA GURADIGUDDA
MEHER GAJULA
VOM SIRI KRISHNA JORIGE
SHREYAS MYSORE NARAYANA
SHAILENDRA CHOWDARYNUTAKKI
SURYA VARDHAN REDDY PUCHALAPALLI
SAI AKASH RATNALA
SUMANTH SARA

GIT REPOSITORY
URL:- https://shorturl.at/BPX24

Abstract

The Background Subtraction System algorithm is designed to appropriately separate foreground objects from their static backgrounds in real-time video streams. The system utilizes numerous algorithms, such as cutting-edge digital image processing combined with computer vision techniques, to detect and delineate moving subjects with high precision. This feature also seems very important in various applications, such as augmented reality, and surveillance, where the isolation of foreground elements contributes to user experience and utility.

The Background Subtraction System is so flexible that it could find use in many industries, taking into consideration that it is easily incorporated into an existing framework and very user-friendly, promising a better visual experience and more productivity in tasks whose performance depends on dynamic foreground-background object segmentation. Its main goal is to separate the foreground subject from a still background in video streams on the fly, a critical need for augmented reality, and surveillance. This makes the distinction of moving figures with their settings much better, increasing functionality and better involvement by the user. It is driven purely by an increasing demand for sophisticated video processing technologies that could empower better digital interactions across different domains, from professional to recreational activities.

When considering video surveillance, removing distractive backdrops will focus viewer attention on the presenter, enhancing communication quality (Karis et al., 2014). In augmented reality, exact foreground detection helps perfect the virtual component with real-world scenes and gives a better user experience. activities or behaviors. The system is designed to be highly functional in meeting these demands. Still, it goes on to be highly operational, embracing an efficient and smooth functionality performing in real-time without any perceptual delays.

1. Introduction

Designing effective foreground-background separation from a live video stream must be a prerogative application in augmented reality, and modern surveillance systems within the rapidly changing digital communication and media area. This project develops an adaptive system for changing conditions, background removal, and real-time systems under various lighting conditions and environments without laborious manual calibration. The following are the primary objectives targeted through the current project: Develop a real-time processing system capable of minimum lag, high responsiveness, and highly realistic. Secondly, implement and develop improved background subtraction accuracy using advanced image processing algorithms and morphological operations and develop this into a user-friendly web application. It is through meeting these goals that the project will have substantially advanced capabilities in digital image processing, user interaction improvements, and content creation possibilities across various digital platforms.

2. Literature Review

2.1 Historical Development of Background Subtraction Techniques:

Background subtraction has been a staple in video processing, with early methodologies focusing on simple static background models. From the most basic ones, the research advanced year by year towards models representing changes in lighting and being adaptive to these changes, modeling dynamic scenes and camera motion. Prominent methods like GMM and RGA facilitated modern methodologies to devise much more adaptive and robust algorithms to deal with real-time video data (Naqvi et al., 2016).

2.2 Advances in Image Processing Algorithms

The development of such algorithms for image processing, more specifically in object detection and segmentation, has been very significant for advancing background subtraction techniques. Approaches to adaptive thresholding have even tried to make the parameters adaptable based on the local characteristics of the image, and morphological operations were applied to refine the masks obtained from these segmentations (Spagnolo et al., 2006). This significantly improved the accuracy and efficiency of eliminating the background.

2.3 Role of Machine Learning and Deep Learning

Background subtraction is increasingly being viewed in the last few literature pieces with the integration of machine learning and deep learning, especially after the arrival of convolutional neural networks (CNNs) and deep autoencoders. Such models are learned through the understanding of complex patterns and variations inside a video

frame, achieving the highest level of performance and accuracy concerning the separation of foreground and background elements (Bouwmans et al., 2019). Works in the field of image processing often very strongly stress the outstanding performance of neural networks, in particular with regard to conventional methods, in situations that are very difficult, where lighting and weather conditions are quite changeable.

2.4 Comparative Studies on Performance and Efficiency

Studies have taken the initiative to compare the methods of background subtraction in their performances of several criteria such as accuracy, computational efficiency, and robustness in several conditions. Such work is essential as it can give one a realistic idea about these methods' practical limits and potential application areas. These would also serve as guidelines for the appropriate choice of technique based on specified requirements, for example, real-time processing like video conferencing or live broadcasting needing high throughput and low latency.

2.5 Technological Integration in Web Applications

The literature review explores the challenges and strategies of implementing sophisticated video processing techniques in real applications, mainly web-based platforms. This area of interest is analyzed using the Flask and Node.js frameworks to help deploy algorithms into more practical uses and inaccessible applications. This comprises details of the discussions of handling high data throughput, ensuring the scalability of web applications, and keeping satisfactory levels of user interactivity without really hampering the responsive character of web applications (Lathkar, 2021).

3. Methodology and Tools

At the heart of the system lies the OpenCV library, well known for its manifold features in image processing and computer vision. OpenCV, or Open-Source Computer Vision Library, is a vast collection of computer vision programming functions that predominantly focus on real-time computer vision tasks. We chose OpenCV as a valid and effective tool for image and video information processing; it can also be adaptable to desktop and mobile software (Minichino & Howse, 2015). The operation can start with acquiring live video through a webcam, a simple yet powerful means of tapping into real-time video inputs. This ability is vital to dynamic settings, like offices or public sites, since human interaction with the system happens continuously, and the time of appearance could be more stable and indeterminable.
Once a video's frames are captured, they follow one another as they record changes in the surroundings. Frame conversion was done in two color spaces: grayscale and HSV (Hue, Saturation, Value). In grayscale conversion, the frame is converted into a single intensity channel, considerably reducing the data stream, and speeding up the further processing stages. (Abdulkareem et al., 2019) It is advantageous for segmenting objects based on color and luminance in any lighting condition. This dual approach in color conversion simplifies the color information strategically, aiding in more effective background removal. This eased the complexity of the input data and allowed the system to focus essentially on structural variations between frames carrying key characteristics that could allow one to separate the foreground from the background accurately. This technique has very pragmatic advantages, especially in real-time scenarios where swift decision-making is coordinated over the ongoing changes of the video feed.

4. Techniques Used

4.1 Pre-processing techniques

- Grayscale Conversion Method:

cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

Purpose:
Reduction of Computational Complexity: Converting the video frame from RGB to grayscale simplifies the data by reducing the dimensionality from three color channels to one. This step reduces the computational load, which is crucial for real-time processing.

Enhanced Detection Accuracy: Haar cascades for face and upper body detection typically perform better on grayscale images where the emphasis is on structure and edge features rather than color, which helps in improving the accuracy of detection.

- Face and Full Body Detection Method:
  face_cascade.detectMultiScale and full_body_cascade.detectMultiScale

Purpose:
Object Detection: These functions detect faces and upper bodies in the grayscale frame. They are

fundamental for identifying regions of interest in the frame, particularly for applications targeting human figures.

- Background Subtraction

Method: subtractor.apply(frame)

Purpose:
Isolation of Foreground; This method helps in distinguishing between the background and moving objects by subtracting the background model from the current frame. This is essential for tracking objects in dynamic environments where the background might change over time.

Handling Shadows: The detectShadows parameter enables the algorithm to identify and potentially ignore shadows, which can otherwise be misinterpreted as independent moving objects.

- Morphological Operations

Method: cv2.erode followed by cv2.dilate
Purpose:
Noise Reduction and Mask Refinement: These operations clean up the mask obtained from background subtraction. Erosion removes small white noise, while dilation enlarges the main features after erosion. This combination (often termed 'opening') is used to improve the quality of the mask by reducing noise and closing small holes within detected objects.

- Mask Refinement Using Detections

Method: Overwriting sections of the mask based on detection coordinates
Purpose:
Enhanced Mask Accuracy: By setting the mask regions corresponding to detected faces and upper bodies to white (255), this method ensures that these areas are preserved in the final mask, leading to more reliable foreground isolation.

- Foreground Extraction

Method: cv2.bitwise_and(frame, frame, mask=mask)
Purpose:
Segmentation of Moving Objects: This operation uses the refined mask to extract and highlight the foreground, which includes any moving objects and particularly the detected humans, from the static background.

- Display

Methods: cv2.imshow
Purpose:
Visualization: This allows real-time viewing of the original video frame, the processed mask, and the isolated foreground, which is useful for monitoring and debugging purposes.

4.2 Background Subtraction Technique

Background subtraction is the principal process in the method. It is a process that separates moving objects, thus detecting them from within a video stream and, therefore, is central (Djerida et al., 2019). To begin with, a reference frame is captured at the start of the video, which serves as the point of reference for identifying scene changes. This first frame is usually selected with the assumption that there are no foreground objects in it. This, too, is a reasonable assumption since people do that only in conditions where the background stays relatively constant. As the video progresses, the absolute value of the frame-by-frame difference concerning the reference frame is compared. They are always pivotal differences since they indicate changes in the scene due to moving or newly introduced objects that disturb the static background. This is how the system measures and identifies those inconsistencies.
The system thus uses adaptive thresholding to tune the precision of detection. This technique dynamically sets the threshold values in different image segments while considering local lighting variations, among other attributes. This flexibility is helpful, especially in environments with diverse or patchy lighting conditions, for good detection of the foreground object under complex lighting conditions. Following the thresholding, the information is transformed into a binary mask. This mask effectively highlights the foreground, delineating areas of change in white and the other ones, where it's not changing, in black. Creating such a binary mask is essential in that it outlines a visible, vivid outline of the foreground objects that are distinguishable and separated from the background. The distinction makes it easy for subsequent procedures, be it in object tracking or accentuating the visual involvement related to the foreground elements during an interactive engagement.

4.3 Enhancement through Morphological Operations

The basic morphological operations used by the system are erosion and dilation. They improve the quality of the binary mask that the background subtraction technique gives as output. The operation is effective in smoothing the mask, erasing the noise,

and removing minor errors typically occurring in the thresholding process (Said & Jambek, 2021). The erosion is a morphological operation that moves the white areas in the binary mask inward; it effectively removes small-scale white noise and separates even-connected objects. The operation is based on an iterative structuring of the element's movement into the image, usually taking away pixels occurring at the object boundaries. One of the more critical benefits of erosion is that it can reduce small regions that are not required and isolate the wrongly attached parts of the object, producing a finer and more segregated representation of the whole of each foreground object.

After erosion, the mask is further dilated in a way that the foreground object size increases and any gaps within the object of the foreground are closed. Dilation builds the white areas in the mask aOnd acts opposite to erosion. This becomes important so that the screen can compensate for any over-reduction due to erosion and enhance the foreground objects' continuity. Dilation fills in the gaps and smooths out while maintaining a clear definition of the edges of the objects. The technique of erosion followed by dilation is known as 'opening' (Vivekananda et al., 2014). The method is efficient in that it managed to retain fine-scale objects and noise from the background, since they would be removed. This is exhibited through keeping the large foreground object with its shape and size. The result was a far tidier, more accurate binary mask, thus dramatically increasing the visual quality of the segmentation. This better segmentation effectively separates foreground objects from the background, hence enhancing the overall performance of video processing in an application.

4.4 System Integration and User Interface

The system is implemented as a web application using Flask. It's a very simple and flexible web framework in Python implementation. Flask has all the necessary tools that are easily used in building a powerful, server-side application with minimal setup needed, making it a viable candidate in deploying interactive, real-time applications like our background removal system (Joshua Heneage Dawes et al., 2019). It has routes within the Flask application that will be responsible for getting the live video feed to even control system operations.

The main route (/) serves HTML interfaces that simply control users to start and stop background processing. The Flask implements the video streaming feature. It will enable us to handle the streaming responses and set the /video_feed route. It is set to stream the video as it reads the frames from the webcam, applies the background subtraction, and encodes the frames into a JPEG format. This is done using OpenCV's function cv2.imencode(), which converts the processed frames into byte format that can be streamed over HTTP.

Using the Response class from Flask, these JPEG images are yielded as a continuous stream embedded in the HTML page through an <img> tag with the src attribute set to the video feed route. This seamless process is created right in the web browser for the live video experience. The interactive controls are mapped to routes like /start_process and /stop_process which dictate the state of the processing process of video.

When the user clicks on the "Start" button, it issues an HTTP request to the /start_process route, which sets a flag to start the process of background subtraction in the video feed. On the other hand, if the "Stop" button is clicked, the "Stop" button sends a request to the /stop_process route to clean this flag and hence pause the processing, hence stopping manipulation of the video stream. This Flask setup is not only a practical demonstration of how our system might be used but goes further to prove the potential for its actual real-world deployment at scale. By employing lightweight yet powerful Flask capabilities, the system is accessible via standard web browsers with no need for special software or complex installations, thus highly suitable for deployment in environments where users might operate from different devices and platforms.

5. Results

The background removal system's evaluation confirms its effectiveness in most cases in real-time processing under various conditions. We focus on two main performance measurements: segmentation accuracy and processing time. These are critical metrics in evaluating usability in practical scenarios such as augmented reality, and surveillance. If the system's usability is found effective through such an evaluation, it will thus be able to meet the required standards for deployment into these dynamic environments.

5.1 Segmentation Accuracy

The first set of tests was targeted to determine the segmentation accuracy of the system for

distinguishing the foreground from the background given the following lighting conditions. The results are as follows:

• Low lighting: The system correctly segmented, registering 93% accuracy. Overwhelming challenges were when low lighting would affect the foreground detection clarity.
• Medium lighting: The system's accuracy in light has been improved. In a more traditional lighting scenario, it has satisfactorily performed.
• High Lighting: The research indicated how the system could perform under ideal lighting conditions, with an accuracy level of 78%, thus showing that the best performance came in perfect lighting conditions.

5.2 Performance in low-light conditions

The results demonstrate that while the system performs robustly in adequate lighting, there is room for improvement under low-light conditions. Enhancing low-light performance could make the system more versatile and reliable across a broader range of environments.

5.3 Processing Time Analysis

The system's efficiency in processing live video feeds was also critically assessed, an essential factor for applications that rely on real-time output:
• Static Scenes: The processing time was quickest at 30 milliseconds per frame, ideal for environments with little movement.
• Low Motion: Slight movement increased the processing time to 45 milliseconds per frame, indicating moderate processing demands.
• High Motion: In scenarios with significant movement, processing time peaked at 60 milliseconds per frame due to higher computational needs.

These timings indicate that the system maintains a reasonable pace, even in dynamic settings, though further optimization could improve responsiveness in high-motion scenarios.

5.4 Visual Quality Assessment

Visually, the system effectively isolates the foreground from the background in most conditions. The processed frames clearly demarcated moving subjects, with minimal artifacts under medium and high lighting conditions. However, there was a noticeable decline in quality in low lighting, with some noise and less precise foreground segmentation.

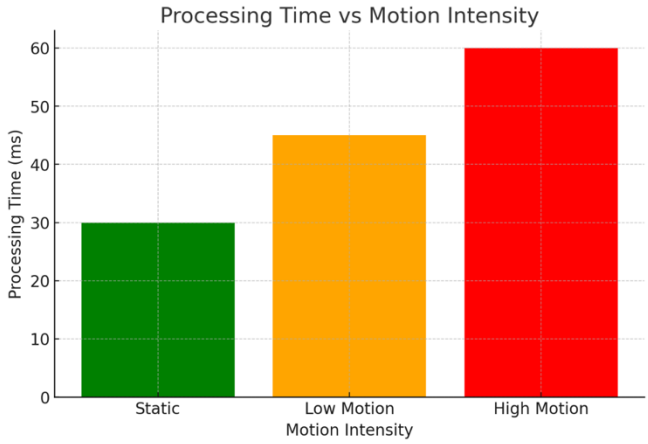

Figure 1:Processing Time vs Motion Intensity

6. General Discussion

6.1 What would you do if you had to start over the project?

If starting over, I will first conduct a needs assessment to understand the users' requirements and application scenarios in detail. I should conduct extensive research on applying recent advances in background subtraction techniques and image processing algorithms to implement the most efficient approaches. I would pay particular attention to the user interface design, considering the accessibility, usability, and ability to work equally well across different devices.

6.2 What have you learned from the project?

The project has been invaluable in teaching how complex real-time image processing is, especially background removal. I have seen the power of interdisciplinary collaboration, ranging from computer vision and image processing to web development. I have gained experience in practical work in the implementation and integration of image processing algorithms into real-world projects and the optimization of performance and usability.

6.3 What if you had more time to do the project?

If given additional time for the project, several improvements could be implemented to enhance the overall outcome. Despite the effectiveness of OpenCV in detecting objects, it is essential to acknowledge its limitations, particularly the presence of residual distortion in the results. This distortion indicates the need for further refinement

in our image processing methods.

Considering this, a significant future enhancement could involve the adoption of the YOLO model. YOLO offers advanced capabilities that could potentially address the remaining distortion issues and further improve the accuracy of object detection. By leveraging YOLO's robust features and deep learning techniques, we anticipate achieving even better results compared to our current methodology.

In conclusion, while our project using OpenCV has yielded commendable results, there is still room for improvement. Acknowledging the drawbacks of OpenCV and recognizing the potential of YOLO as a future enhancement, we remain optimistic about the prospect of achieving even greater precision and accuracy in object detection with additional time and resources.

6.4 Did you ensure the fulfillment of the project?

The project aimed to create a real-time background removal system integrated into a web interface using image processing techniques and computer vision algorithms.

6.5 Discuss the findings about the objectives and the limitations.

The results emphasize the efficiency of the selected approach concerning the on-the-fly background removal with decent accuracy. The system's sensitivity to different lighting conditions and slight noise are issues that require continuous research and improvement.

The project's objectives, including real-time processing, high segmentation accuracy, and a user-friendly interface, have been addressed to some extent. However, more refinement and iteration are needed to fully meet these objectives. In the long run, it is important to tackle this system's weaknesses and find other ways to boost its effectiveness for different usages.

7. Result

8. Conclusion

The utilization of OpenCV for diverse image processing techniques has led to notable progress in object detection. Despite achieving significant results, the presence of persistent noise in the output highlights the need for further refinement. Looking ahead, integrating the YOLO model holds promise for enhancing precision and overcoming the existing noise challenges. Anticipating YOLO's advanced capabilities to minimize noise and enhance detection accuracy, we envision a future where our objectives of achieving optimal results in object detection will be realized.

9. References

Abdulkareem, H. A., A. M. S. Tekanyi, Yau, I., &Bashir Olaniyi Sadiq. (2019). Development of video frame enhancement technique using pixel intensity analysis. *ArXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1902.04985

Bouwmans, T., Javed, S., Sultana, M., & Jung, S. K. (2019). Deep neural network concepts for background subtraction: A systematic review and comparative evaluation. Neural Networks, 117, 8–66. https://doi.org/10.1016/j.neunet.2019.04.024

Djerida, A., Zhao, Z., & Zhao, J. (2019). Background subtraction in dynamic scenes using the dynamic principal component analysis. *IET Image Processing*. https://doi.org/10.1049/iet-ipr.2018.6095

Joshua Heneage Dawes, Reger, G., Franzoni, G., Pfeiffer, A., & Giacomo Govi. (2019). VyPR2: A framework for runtime verification of python web services. *Lecture Notes in Computer Science*, 98–114. https://doi.org/10.1007/978-3-030-17465-1_6

Karis, D., Wildman, D., & Mané, A. (2014). Improving remote collaboration with video conferencing and video portals. *Human–Computer Interaction*, *31*(1), 1–58. https://doi.org/10.1080/07370024.2014.921506

Lathkar, M. (2021). Building web apps with python and flask: Learn to develop and deploy responsive restful web applications using flask framework (english edition). In *Google Books*. BPB Publications. https://books.google.co.ke/books?hl=en&lr=&id=gtwiEAAAQBAJ&oi=fnd&pg=PT19&dq=using+the+Flask+and+Node.js+frameworks+&ots=L7IHjbB

4SC&sig=iiPKGWVkk3n7q_LbbNgjux5VnjA&redir_esc=y#v=onepage&q=using%20the%20Flask%20and%20Node.js%20frameworks&f=false

Minichino, J., & Howse, J. (2015). Learning opencv 3 computer vision with python. In *Google Books*. Packt Publishing Ltd. https://books.google.co.ke/books?hl=en&lr=&id=iNlOCwAAQBAJ&oi=fnd&pg=PP1&dq=OpenCV

Naqvi, S. S., Browne, W. N., & Hollitt, C. (2016). Salient object detection via spectral matting. *Pattern Recognition*, *51*, 209–224. https://doi.org/10.1016/j.patcog.2015.09.026

Roesler, V., Barrére, E., & Willrich, R. (2020). Special topics in multimedia, iot and web technologies. In *Google Books*. Springer Nature. https://books.google.co.ke/books?hl=en&lr=&id=3RDUDwAAQBAJ&oi=fnd&pg=PA3&dq=The+use+of+JavaScript+and+AJAX+powers+their+way+to+productive+client-server+communications

Said, K. A. M., & Jambek, A. B. (2021). Analysis of image processing using morphological erosion and dilation. *Journal of Physics: Conference Series*, *2071*(1), 012033. https://doi.org/10.1088/1742-6596/2071/1/012033

Spagnolo, P., Orazio, T. D'., Leo, M., & Distante, A. (2006). Moving object segmentation by background subtraction and temporal analysis. *Image and Vision Computing*, *24*(5), 411–423. https://doi.org/10.1016/j.imavis.2006.01.001

Vivekananda, B., Sunny, S., D. P, J., & Rajesh, K. (2014). Application of image processing in estimation of area of heat affected zone. *Journal of Metallurgy and Materials Science*, *50*(4), 219–225. https://www.indianjournals.com/ijor.aspx?target=ijor:jmms&volume=50&issue=4&article=003

10. Appendices

9.1 Full Python Code for Background Removal

The source code to implement the system of background removal using OpenCV for this project is attached to this appendix. The setting up of video capture is done, and then every frame is processed for background subtraction after the morphological operations are applied to enhance the foreground segmentation.

```
import cv2
import numpy as np
from IPython.display import display, clear_output
from PIL import Image
from io import BytesIO
import IPython.display
import time

# Initialize camera
camera = cv2.VideoCapture(0)

# Load Haar Cascades for face and full body detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
full_body_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_fullbody.xml')

# Check if the camera opened successfully
if not camera.isOpened():
    print("Error: Could not open camera.")
else:
    try:
        # Prepare reference frame for background removal
        ret, reference_frame = camera.read()
        if not ret:
            print("Error: Failed to grab initial frame.")
        else:
            ref_gray = cv2.cvtColor(reference_frame, cv2.COLOR_BGR2GRAY)
            ref_hsv = cv2.cvtColor(reference_frame, cv2.COLOR_BGR2HSV)

            while True:
                ret, frame = camera.read()
                if not ret:
                    break

  # Process the frame for background removal
                gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
                hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
                diff_gray = cv2.absdiff(ref_gray, gray_frame)
                diff_sat = cv2.absdiff(ref_hsv[:, :, 1], hsv_frame[:, :, 1])
                _, thresh_gray = cv2.threshold(diff_gray, 25, 255, cv2.THRESH_BINARY)
                _, thresh_sat = cv2.threshold(diff_sat, 25, 255, cv2.THRESH_BINARY)
                combined_mask = cv2.bitwise_or(thresh_gray, thresh_sat)
                kernel = np.ones((3, 3), np.uint8)
```

```
            combined_mask =
cv2.erode(combined_mask, kernel, iterations=2)
            combined_mask =
cv2.dilate(combined_mask, kernel, iterations=2)
            foreground = cv2.bitwise_and(frame,
frame, mask=combined_mask)

            # Detect faces and bodies
            faces =
face_cascade.detectMultiScale(gray_frame, 1.1, 4)
            full_bodies =
full_body_cascade.detectMultiScale(gray_frame,
1.1, 4)

            # # Draw rectangles around detected
faces and bodies
            # for (x, y, w, h) in faces:
            #     cv2.rectangle(foreground, (x, y),
(x+w, y+h), (255, 0, 0), 2)
            # for (x, y, w, h) in full_bodies:
            #     cv2.rectangle(foreground, (x, y),
(x+w, y+h), (0, 255, 0), 2)

            # Convert images from BGR to RGB for
display
            frame_rgb = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)
            foreground_rgb =
cv2.cvtColor(foreground,
cv2.COLOR_BGR2RGB)

            # Concatenate images horizontally to
compare
            combined_image =
np.hstack((frame_rgb, foreground_rgb))

            # Convert numpy array to PIL Image and
then to IPython Image display object
            pil_img =
Image.fromarray(combined_image)
            buffer = BytesIO()
            pil_img.save(buffer, format="JPEG")

display(IPython.display.Image(data=buffer.getval
ue()))
            clear_output(wait=True)  # Clear the
output to display next frame

            time.sleep(0.1)  # Adjust as per your
frame processing needs

    finally:
        camera.release()
        print("Stream stopped")
```