Supporting Information

Magnetic smartphone microflow cytometry enables rapid CD4/CD8 T cell quantification

Hee Sik Shin[a], Sung Joo Lee[a], Jae In Kim[b], Jung Ho Kim[b], Jun Yong Choi[b], Su Jin Jeong[b]*, and Sungyoung Choi[a,c,]*

[a]Department of Electronic Engineering, Hanyang University, Seoul 04763, Republic of Korea

[b]Department of Internal Medicine and AIDS Research Institute, Yonsei University College of Medicine, Seoul 03722, Republic of Korea

[c]Department of Biomedical Engineering, Hanyang University, Seoul 04763, Republic of Korea

Contents:

1. Supplementary Figures

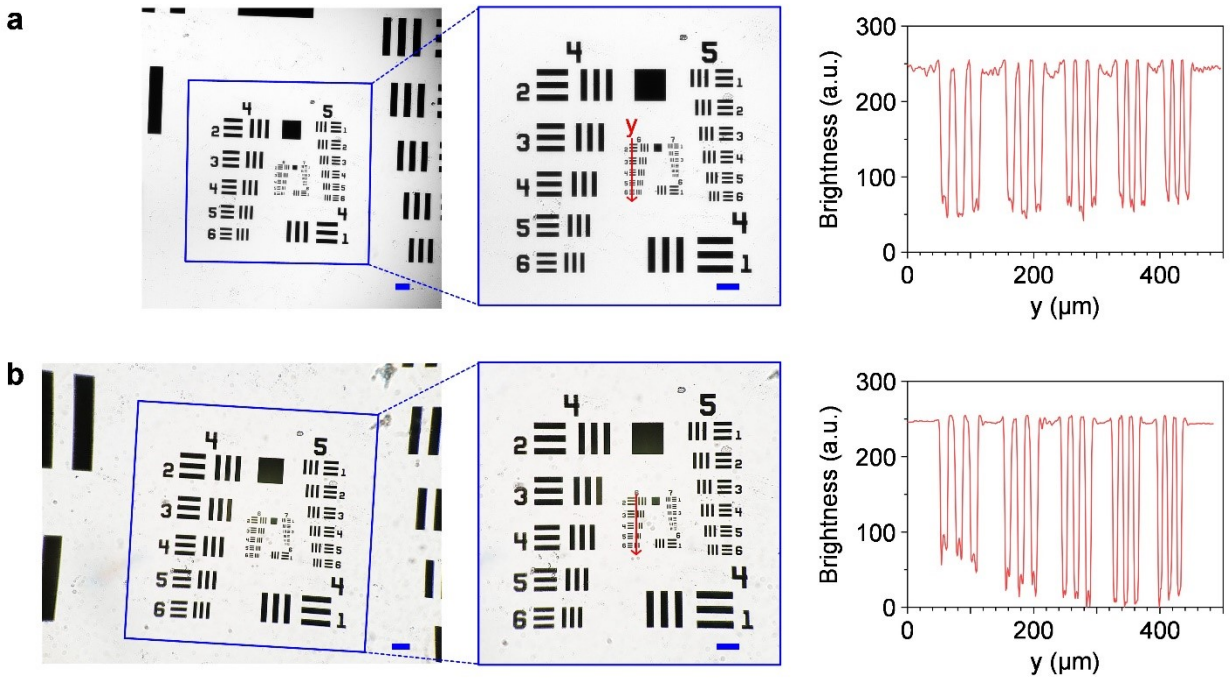

Fig. S1. Resolution performance comparison. (a, b) Images of a standard resolution test pattern captured using (a) the bench-top microscope with 4× objective lens and (b) the smartphone-based microscope. Insets show magnified views of the central regions. Brightness profiles measured along the y-axis are presented to the right, showing comparable resolution performance between the two systems. Scale bars, 100 μm.

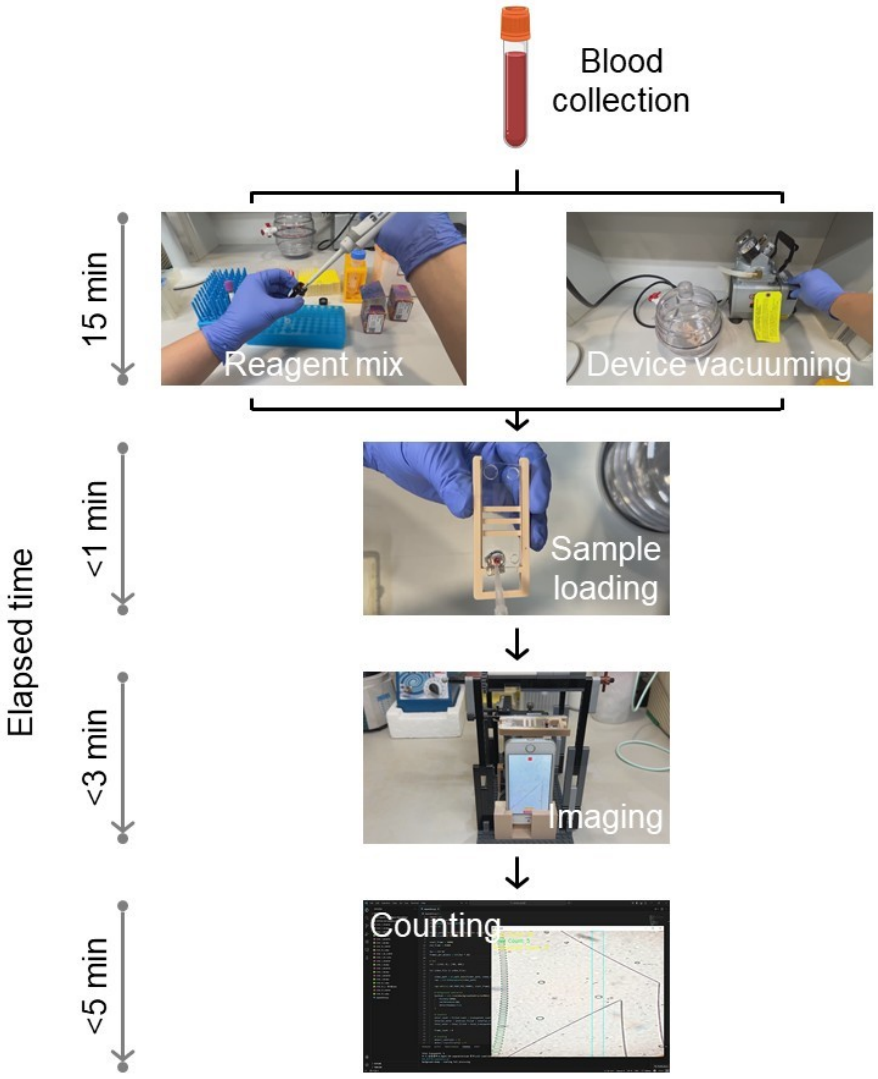

Fig. S2. Workflow and time-course of the cell counting protocol. The protocol includes (1) blood collection, (2) reagent mixing and device vacumming, (3) sample loading, (4) chip imaging, and (5) automated cell counting. The total estimated hands-on and processing time for a single counting is approximately 24 minutes.

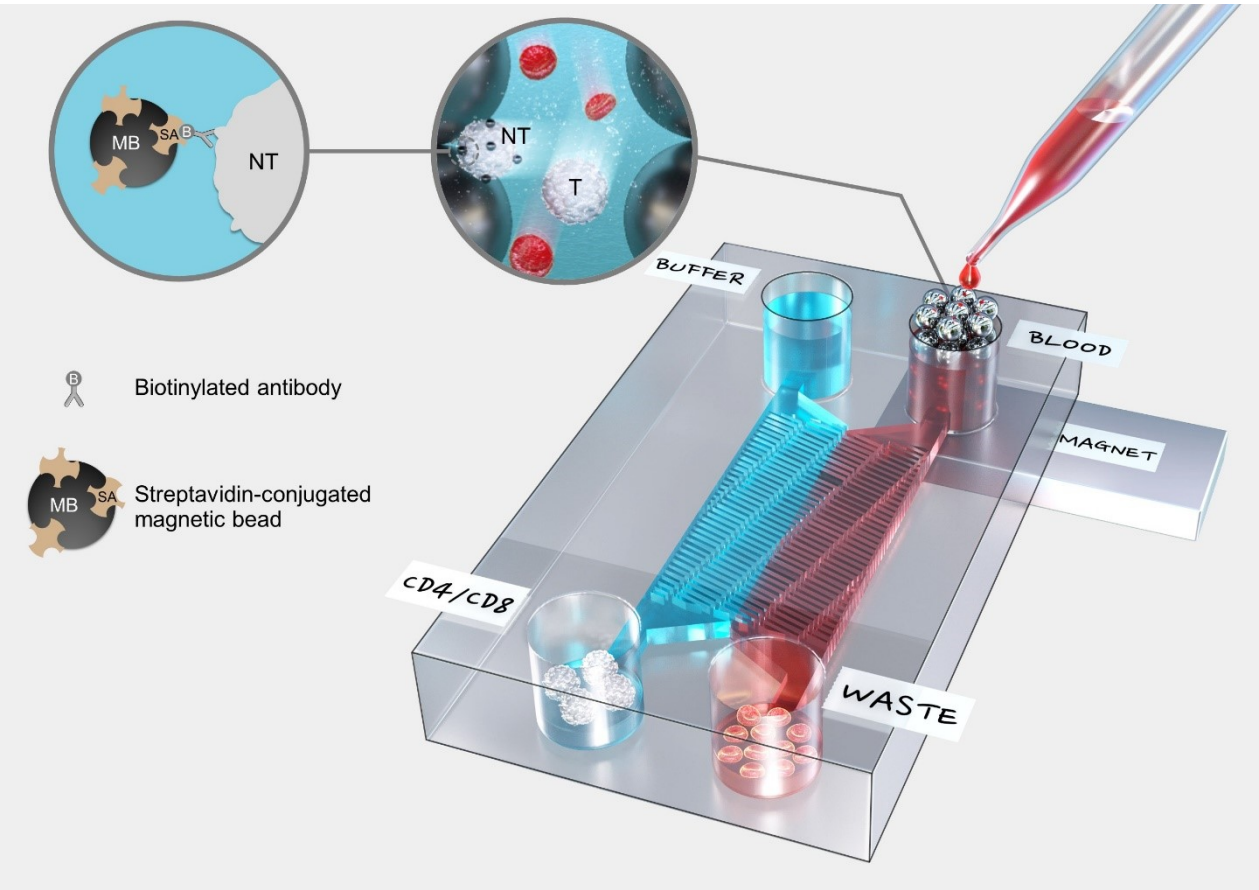

Fig. S3. Schematic of the magnetic labeling strategy. Biotinylated antibodies first bind to non-target (NT) cells, after which streptavidin-conjugated magnetic beads attach to the bound antibodies, enabling selective magnetic labeling of the NT population. T indicates target cells.

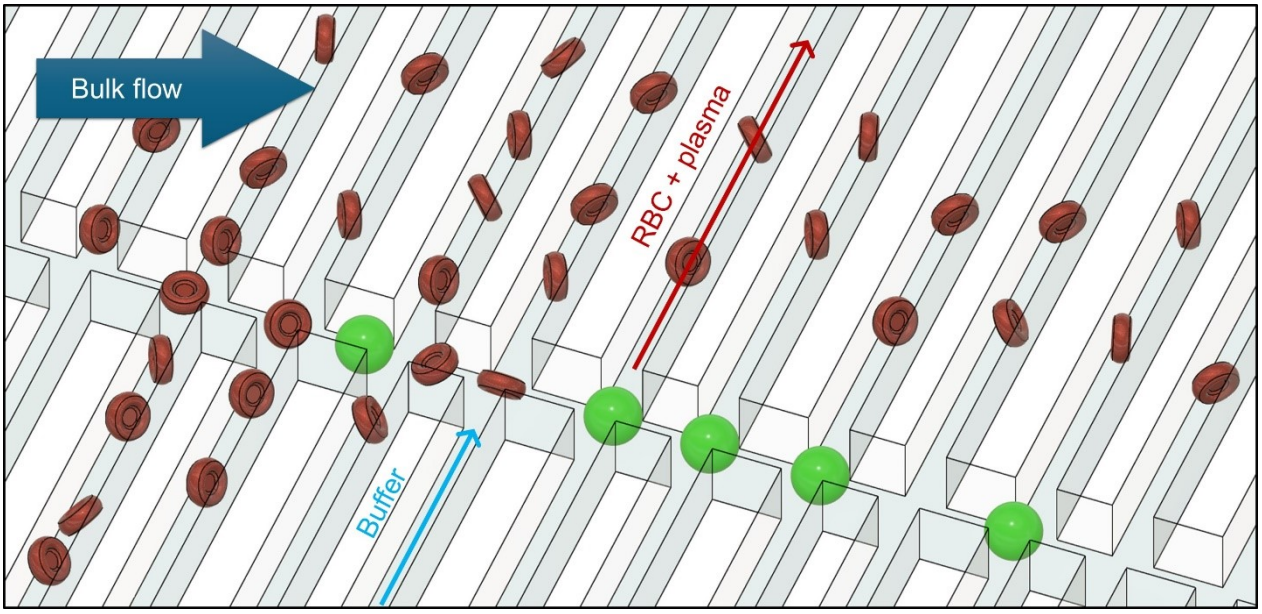

Fig. S4. Size-based cell separation principle. The microfluidic lattice comprises inclined wide main channels interconnected by relatively narrow side channels. Cells flowing at an oblique angle along the main channels align toward channel walls. Cells with equilibrium positions falling outside side-channel streamlines continue along the main channels, rather than entering side channels. This network induces steric hindrance, functioning as a sieve to selectively retain larger lymphocytes while allowing smaller red blood cells and plasma to pass through the side channels.

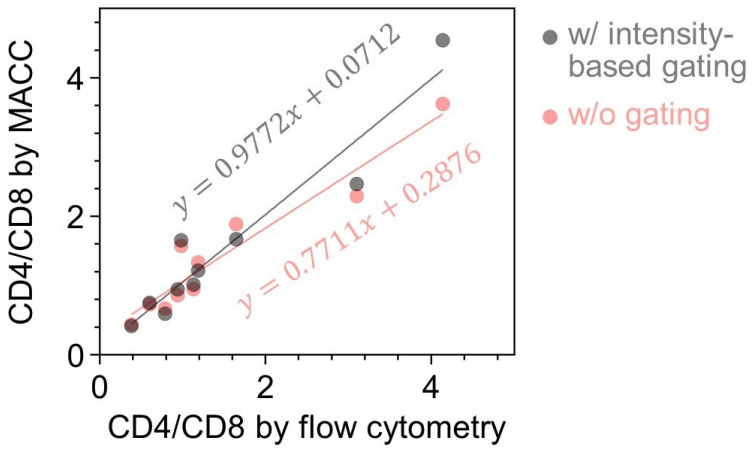

Fig. S5. Effect of intensity-based gating on the CD4/CD8 ratio, yielding values that closely align with those obtained by conventional flow cytometry.

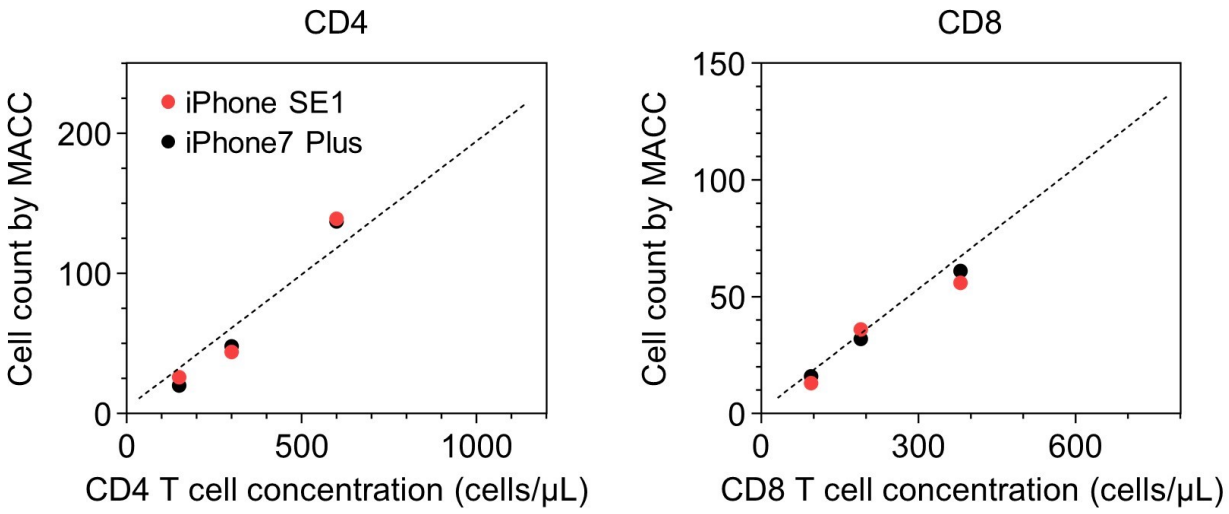

Fig. S6. MACC performance was validated using an additional smartphone model (iPhone 7 Plus), showing consistent CD4+ and CD8+ T-cell enumeration across three blood concentrations. The dotted lines represent the calibration curves obtained from Fig. 4a, and the newly tested data points align well with these reference lines. Usability was also assessed by a first-year graduate student with no prior microfluidic separation experience, who performed CD4 counting using the primary smartphone (iPhone SE1) after a brief 30-min training, while all remaining measurements were conducted by an experienced operator.

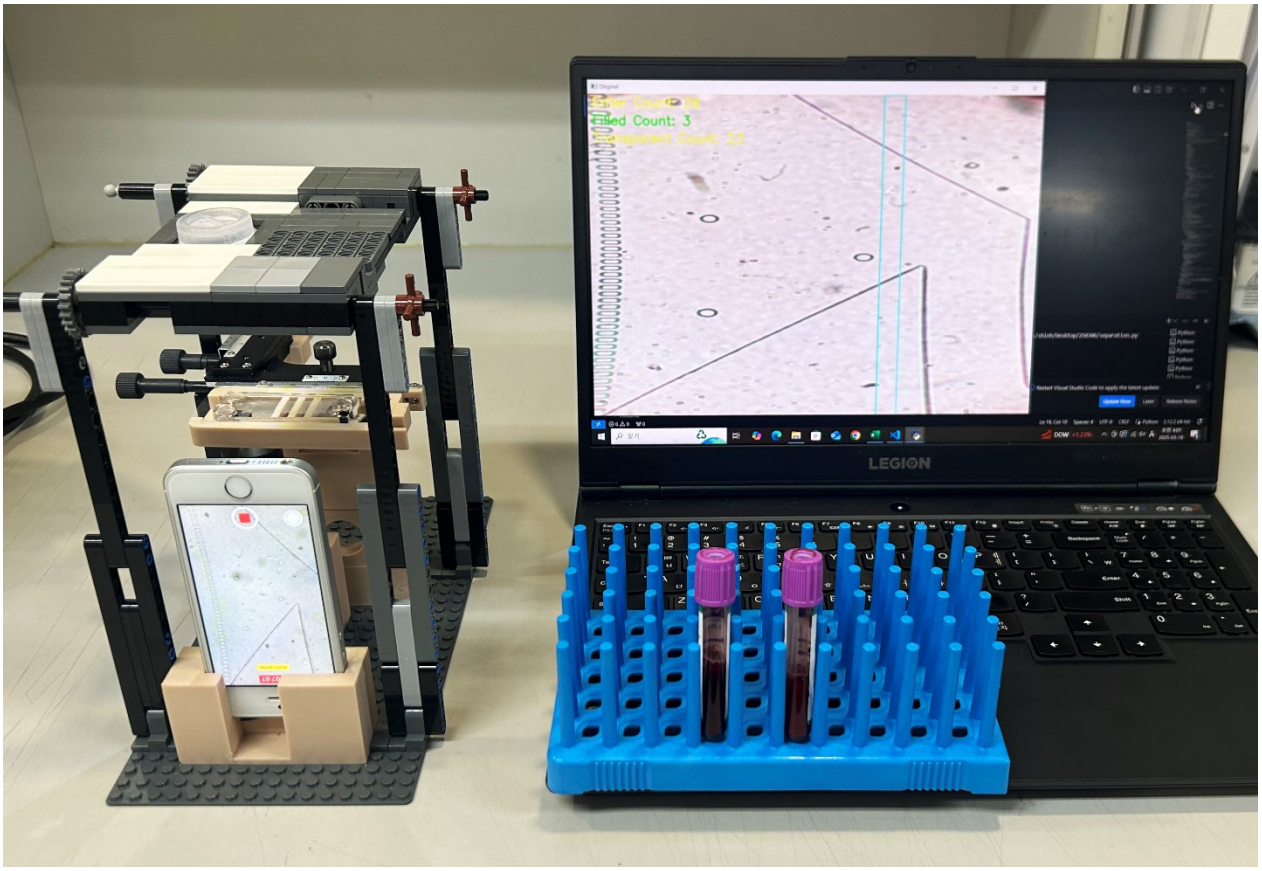

Fig. S7. Photograph of the MACC setup for HIV sample testing in a hospital laboratory.

2. Supplementary Table

Table S1. Performance comparison

	VISITECT	Immuno-chromatographic strip	Microfluidic sorting-based Coulter counter	Flow cytometry	Our work
LOD*	-	34 cells/μL	-	-	-
Lowest measurable concentration	-	-	<40 cells/μL	~2 cells/μL	31 cells/μL
Dynamic range	-	50 to 1000 cells/μL	40 to 1000 cells/μL	2 to 1808 cells/μL	31 to 1100 cells/μL
Total time for CD4 analysis	40 min	30 min	15 min	-	24 min
System cost	$0	$300	$5,000	$100,000	Total $916.2; $348.3 for caged mirror $366 for objective lens $158.5 for eyepiece lens $6.8 for magnet $10.1 for second-hand Lego components $17 for 3D printing $1.4 for LED bulb $8.1 for xy stage
Test cost	$4.6	$0.5	$8	$0.3	Total $8.3; $7.6 for MACS reagent $0.7 for microfluidic chip
Ref.	s1	s2	s3	s4	-

*LOD = Limit of detection

[s1] Lechiile, K., Leeme, T.B., Tenforde, M.W., Bapabi, M., Magwenzi, J., Maithamako, O., Mulenga, F., Mohammed, T., Ngidi, J., Mokomane, M., Lawrence, D.S., Mine, M., Jarvis, J.N., 2022. Laboratory Evaluation of the VISITECT Advanced Disease Semiquantitative Point-of-Care CD4 Test. J Acquir Immune Defic Syndr 91(5), 502-507.

[s2] Xiao, W., Xiao, M., Yao, S., Cheng, H., Shen, H., Fu, Q., Zhao, J., Tang, Y., 2019. A Rapid, Simple, and Low-Cost CD4 Cell Count Sensor Based on Blocking Immunochromatographic Strip System. ACS Sens 4(6), 1508-1514.

[s3] Watkins, N.N., Hassan, U., Damhorst, G., Ni, H., Vaid, A., Rodriguez, W., Bashir, R., 2013. Microfluidic CD4+ and CD8+ T lymphocyte counters for point-of-care HIV diagnostics using whole blood. Sci Transl Med 5(214), 214ra170.
[s4] Dieye TN, Vereecken C, Diallo K, Ondoa AT, Diaw B, Camara AA, Karam K, Kestens L, Mboup S. Absolute CD4 T-cell counting in resource-poor settings. J Acquir Immune Defic Syndr. 2005;39(1):32-37.Dieye TN, Vereecken C, Diallo K, Ondoa AT, Diaw B, Camara AA, Karam K, Kestens L, Mboup S. Absolute CD4 T-cell counting in resource-poor settings. J Acquir Immune Defic Syndr. 2005;39(1):32-37.

3. Supplementary Videos

Movie S1. Cell counting using magnetic-activated smartphone microflow cytometry. The video was recorded at 239.98 fps and is displayed at 30 fps, resulting in slow-motion visualization of cell movement.

Movie S2. Workflow for T cell counting using magnetic-activated smartphone microflow cytometry.

4. Supplementary Codes

The following Python source code is designed to count cells and classify them based on contrast using a predefined threshold.

```
import cv2
import numpy as np
from openpyxl import Workbook
import os

# Folder path containing the video files
folder_path = 'C:/Users/admin/Desktop/HIV_sample'

# List of video file names to process
video_files = ['HIV_sample.MOV']

for video_file in video_files:
    video_path = os.path.join(folder_path, video_file)
    cap = cv2.VideoCapture(video_path)

    # Set the starting frame position
    start_frame = 30000
    end_frame = 75000
    cap.set(cv2.CAP_PROP_POS_FRAMES, start_frame)

    # Define FPS and frames per minute
    fps = 239.98
    frames_per_minute = int(fps * 60)  # 14398.8 frames per minute

    # Create a background subtractor object
```

```
    backSub = cv2.createBackgroundSubtractorMOG2(history=10000,
varThreshold=100, detectShadows=True)

    # Define the ROI coordinates (x1, y1, x2, y2) in terms of the resized frame
    roi = ((530, 0), (600, 800))  # Example values, adjust as needed

    # Initialize variables for object counting
    enter_count = 0
    exit_count = 0
    filled_count = 0
    transparent_count = 0
    frame_count = 0

    # Initialize variables for interval counting
    interval_enter_count = 0
    interval_filled_count = 0
    interval_transparent_count = 0

    # Total counts
    total_enter_count = 0
    total_filled_count = 0
    total_transparent_count = 0

    # Dictionary to keep track of object centroids and their ids
    object_centroids = {}
    object_classification = {}
    next_object_id = 0
    ttl = {}

    # Initialize variables for object counting
    accumulated_frame = None
```

```
# Create an Excel workbook and worksheet
wb = Workbook()
ws = wb.active
ws.title = "Dot Counts"

# Write headers
ws.append(["Minute", "Enter Count", "Filled Count", "Transparent Count"])

def get_centroid(contour):
    M = cv2.moments(contour)
    if M["m00"] != 0:
        cx = int(M["m10"] / M["m00"])
        cy = int(M["m01"] / M["m00"])
        return (cx, cy)
    return None

def cluster_centroids(centroids, distance_threshold=0.1):
    clustered_centroids = []
    for centroid in centroids:
        merged = False
        for i, c in enumerate(clustered_centroids):
            if np.linalg.norm(np.array(centroid) - np.array(c)) < distance_threshold:
                clustered_centroids[i] = ((c[0] + centroid[0]) // 2, (c[1] + centroid[1]) // 2)
                merged = True
                break
        if not merged:
            clustered_centroids.append(centroid)
    return clustered_centroids

# Accumulate frames to compute the average frame
```

```
while cap.isOpened():
    ret, frame = cap.read()
    if not ret or frame_count >= end_frame - start_frame:
        break

    # Resize the frame (only once to avoid multiple resizings)
    frame = cv2.resize(frame, (1080, 760), interpolation=cv2.INTER_CUBIC)

    # Accumulate the frames
    if accumulated_frame is None:
        accumulated_frame = np.zeros_like(frame, dtype=np.float32)

    accumulated_frame += frame.astype(np.float32)
    frame_count += 1

# Compute the average frame
if frame_count > 0:
    average_frame = accumulated_frame / 45000
    average_frame = cv2.convertScaleAbs(average_frame)  # Convert to 8-bit image

# Rewind the video to the starting frame
cap.set(cv2.CAP_PROP_POS_FRAMES, start_frame)

# Reset frame_count for the main processing loop
frame_count = 0
bg = None

while True:
    ret, frame = cap.read()
    if not ret or frame_count >= end_frame - start_frame:
        break
```

```
# Resize the frame
frame = cv2.resize(frame, (1080, 760), interpolation=cv2.INTER_CUBIC)

# Subtract the average frame from the current frame
frame_diff = cv2.absdiff(frame, average_frame)

# Apply Gaussian blur to the frame
g_frame = cv2.GaussianBlur(frame_diff, (7, 7), 0)

# Initialize background model if necessary
if frame_count == 0:
    bg = g_frame.copy().astype("float")

# Update the background model
if frame_count < 10000000:
    cv2.accumulateWeighted(g_frame, bg, 0.1)

# Calculate the difference between the current frame and the background
diff = cv2.absdiff(g_frame, bg.astype("uint8"))
diff = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
_, diff = cv2.threshold(diff, 8, 255, cv2.THRESH_BINARY)

# Apply the background subtractor to get the foreground mask
fg_mask = backSub.apply(diff)

# Remove noise from the foreground mask using morphological operations
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
fg_mask = cv2.morphologyEx(fg_mask, cv2.MORPH_OPEN, kernel)

# Apply edge detection using Canny
```

```
        edge_detect = cv2.Canny(fg_mask, 100, 10)

        # Find hierarchical contours
        contours, hierarchy = cv2.findContours(edge_detect, cv2.RETR_CCOMP,
cv2.CHAIN_APPROX_SIMPLE)

        # Draw the ROI rectangle on the frame
        cv2.rectangle(frame, roi[0], roi[1], (255, 255, 0), 2)

        current_centroids = []

        for i, contour in enumerate(contours):
            if 10 <= cv2.contourArea(contour) <= 150:  # Adjusted area limits
                # Get the centroid of the contour
                centroid = get_centroid(contour)
                if centroid is not None:
                    # Check if the centroid is within the ROI
                    if roi[0][0] <= centroid[0] <= roi[1][0] and roi[0][1] <= centroid[1] <= roi[1][1]:
                        current_centroids.append(centroid)
                        # Draw the bounding box around the object
                        x, y, w, h = cv2.boundingRect(contour)
                        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
                        # Draw the centroid for debugging
                        cv2.circle(frame, centroid, 3, (0, 0, 255), -1)

        # Cluster close centroids
        clustered_centroids = cluster_centroids(current_centroids)

        # Match current centroids with existing tracked objects
        for centroid in current_centroids:
            matched = False
```

```
for obj_id, obj_centroid in object_centroids.items():
    distance = np.linalg.norm(np.array(centroid) - np.array(obj_centroid))
    if distance < 30:  # Adjust distance threshold as needed
        object_centroids[obj_id] = centroid
        ttl[obj_id] = 30  # Reset TTL
        matched = True
        break
if not matched:
    object_centroids[next_object_id] = centroid
    ttl[next_object_id] = 30  # Initialize TTL

    # Classify the dot as filled or transparent based on the average pixel intensity
    x, y = centroid
    radius = 2  # Small radius around the centroid for mean calculation
    roi_gray = cv2.cvtColor(frame_diff, cv2.COLOR_BGR2GRAY)
    mask = np.zeros_like(roi_gray)
    cv2.circle(mask, (x, y), radius, 255, -1)
    mean_intensity = cv2.mean(roi_gray, mask=mask)[0]
    if mean_intensity > 35:  # Adjust the threshold value if needed
        object_classification[next_object_id] = 'filled'
        filled_count += 1
        interval_filled_count += 1
        total_filled_count += 1
    else:
        object_classification[next_object_id] = 'transparent'
        transparent_count += 1
        interval_transparent_count += 1
        total_transparent_count += 1
    next_object_id += 1
    enter_count += 1
    interval_enter_count += 1
```

```
                total_enter_count += 1

        # Decrease TTL and remove objects that have exited the ROI
        for obj_id in list(object_centroids.keys()):
            ttl[obj_id] -= 1
            if ttl[obj_id] <= 0 or not (roi[0][0] <= object_centroids[obj_id][0] <= roi[1][0] and
roi[0][1] <= object_centroids[obj_id][1] <= roi[1][1]):
                exit_count += 1
                del object_centroids[obj_id]
                del ttl[obj_id]
                del object_classification[obj_id]  # Also remove the classification

        # Log counts at the end of each interval
        if frame_count % frames_per_minute == 0:
            minute = frame_count // frames_per_minute
            ws.append([minute, interval_enter_count, interval_filled_count,
interval_transparent_count])
            interval_enter_count = 0
            interval_filled_count = 0
            interval_transparent_count = 0

        # Display the counts on the frame
        cv2.putText(frame, f'Enter Count: {enter_count}', (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2, cv2.LINE_AA)
        cv2.putText(frame, f'Filled Count: {filled_count}', (10, 70),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE_AA)
        cv2.putText(frame, f'Transparent Count: {transparent_count}', (10, 110),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 255), 2, cv2.LINE_AA)
        # Display the results
        cv2.imshow('Original', frame)
        cv2.imshow('dots distinguish', g_frame)
```

```
        cv2.imshow('Edge Detect', edge_detect)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

        frame_count += 1

    cap.release()
    cv2.destroyAllWindows()

    # Save any remaining counts for the last partial minute
    if frame_count % frames_per_minute != 0:
        minute = frame_count // frames_per_minute + 1
        ws.append([minute, interval_enter_count, interval_filled_count,
interval_transparent_count])

    # Save the Excel file with a name corresponding to the video file
    excel_filename = os.path.splitext(video_file)[0] + '.xlsx'
    excel_filepath = os.path.join(folder_path, excel_filename)
    wb.save(excel_filepath)

    print(f"Finished processing {video_file}")
    print(f"Total Enter Count: {total_enter_count}")
    print(f"Total Filled Count: {total_filled_count}")
    print(f"Total Transparent Count: {total_transparent_count}")
```