Independent Multimodal Background Subtraction - Implementation

Necessary Imports and Global Variables

Let's start with importing necessary libraries and defining common global variables

In [1]:
```
import cv2
import numpy as np
import math
import datetime
import matplotlib.pyplot as plt

WIDTH = 240
HEIGHT = 0

RED = 2
GREEN = 1
BLUE = 0

HUE = 0
SATURATION = 1
VALUE = 2
```

Defining Necessary Classes

According to paper, we have background model B, which contains background tuples. And those background tuples consist of r,b,g,d values. Where r,g,b are RGB values of pixel and d is the number of pixels Sn(i, j), associated with those.

In [2]:
```
class RGBDTuple:
    def __init__(self, red, green, blue, d):
        self.red = red
        self.green = green
        self.blue = blue
        self.d = d

    # Lately, it is used for rgb hsv conversion to suppress shadows
    def getHSV(self):
        r, g, b = self.red / 255.0, self.green / 255.0, self.blue /
255.0
        mx = max(r, g, b)
        mn = min(r, g, b)
        df = mx - mn
        if mx == mn:
            h = 0
        elif mx == r:
            h = (60 * ((g - b) / df) + 360) % 360
        elif mx == g:
            h = (60 * ((b - r) / df) + 120) % 360
        elif mx == b:
            h = (60 * ((r - g) / df) + 240) % 360
        if mx == 0:
            s = 0
        else:
            s = df / mx * 100
        v = mx * 100
        return h, s, v

class BackgroundTuple:
    def __init__(self, tuples):
        self.tuples = tuples
```

Defining Some Helper Methods

We define two helper methods. One to find maximum of three values. Other is to create Background Model B array.

In [3]:
```
def maximum(a, b, c):
    if (a >= b) and (a >= b):
        largest = a

    elif (b >= a) and (b >= a):
        largest = b
    else:
        largest = c

    return largest

def initB(frame):
    # B is the array of BackgroundTuple objects. Which have list of
rgbdTuples
    B = []
    for i in range(len(frame)):
        subArr = []
        for j in range(len(frame[i])):
            t = RGBDTuple(frame[i][j][RED], frame[i][j][GREEN], fra
me[i][j][BLUE], 1)
            backTuple = BackgroundTuple([t])
            subArr.append(backTuple)
        B.append(subArr)
    return B
```

Defining RegisterBackground Method

In the paper, proposed IMBS algorithm contains two methods. One of them is the RegisterBackground method which clusters the background tuples in background samples Sn.

In [4]:
```
def registerBackground(n, frame, B, N, D, A):

    # n -> Sampling number
    # frame -> Video frame at time t
    # B -> Background Model
    # N -> Number of background samples to analyze
    # D -> The minimal number D of occurrences to consider a tuple ⟨
r,g,b,d ≥ D⟩ as a significant background value
    # A -> The association threshold A for assigning a pixel to an
existing tuple

    if len(B) < 1:
        #If background model is empty, first create it with the hel
per method
        B = initB(frame)
        return B

    for i in range(len(frame)):
        for j in range(len(frame[i])):
            # Get RGB values of each pixel
            rgbPixel = frame[i][j]
            if n == 0:
                # If it is first sample simply add the rgbd tuple t
```

```
o list.
                t = RGBDTuple(frame[i][j][RED], frame[i][j][GREEN],
frame[i][j][BLUE], 1)
                B[i][j].tuples.append(t)
            elif n == N - 1:
                for k in range(len(B[i][j].tuples)):
                    if B[i][j].tuples[k].d < D:
                        # If number of associated pixels are less t
hen D threshold, remove them from list.
                        B[i][j].tuples.pop(k)
            else:
                for k in range(len(B[i][j].tuples)):
                    # Associated pixel RGB value
                    rgbdTuple = B[i][j].tuples[k]
                    if maximum(abs(rgbPixel[RED] - rgbdTuple.red),
                               abs(rgbPixel[GREEN] - rgbdTuple.gree
n),
                               abs(rgbPixel[BLUE] - rgbdTuple.blue)
) <= A:
                        # If the difference between current sample
pixel and associated pixel is less than threshold
                        # simply update the tuple
                        rgbdTuple.red = ((rgbdTuple.red * rgbdTuple
.d) + rgbPixel[RED]) / (
                                rgbdTuple.d + 1)
                        rgbdTuple.green = ((rgbdTuple.green * rgbdT
uple.d) + rgbPixel[
                            GREEN]) / (rgbdTuple.d + 1)
                        rgbdTuple.blue = ((rgbdTuple.blue * rgbdTup
le.d) + rgbPixel[
                            BLUE]) / (rgbdTuple.d + 1)
                        rgbdTuple.d += 1
                        break
                    else:
                        # If it not add the current pixel as a new
tuple
                        t = RGBDTuple(rgbPixel[RED], rgbPixel[GREEN
], rgbPixel[BLUE], 1)
                        B[i][j].tuples.append(t)
    return B
```

Defining GetForeground Method

Another essential method of IMBS is 'getForeground', which checks the tuples and decides that the pixel is whether foreground or not. Also the compainon method 'getForegroundVisualFrame' is created, to map value 1 to 255 in foreground mask, in order to show as a white.

In [ ]:
```
def getForeground(frame, B, A):

    # frame -> Video frame at time t
    # B -> Background Model
    # A -> The association threshold A for assigning a pixel to an
existing tuple

    # Create foreground mask
    F = np.full((HEIGHT, WIDTH), 1)
    for i in range(len(frame)):
        for j in range(len(frame[i])):
            # Get RGB values of each pixel
            rgbPixel = frame[i][j]
            if len(B) > 0 and len(B[i][j].tuples) > 0:
                for k in range(len(B[i][j].tuples)):
                    # Associated pixel RGB value
                    rgbdTuple = B[i][j].tuples[k]
                    if maximum(abs(rgbPixel[RED] - rgbdTuple.red),
                               abs(rgbPixel[GREEN] - rgbdTuple.gree
n),
                               abs(rgbPixel[BLUE] - rgbdTuple.blue)
) < A:
                        # If the difference between current sample
pixel and associated pixel is less than threshold
                        # simply mark the pixel as not foreground
                        F[i][j] = 0
                        break
    return F

def getForegroundVisualFrame(frame):
    for i in range(len(frame)):
        for j in range(len(frame[i])):
            frame[i][j] *= 255
    return np.array(frame, dtype=np.uint8)
```

IMBS Method

Now, IMBS method can be defined since we have essential methods are defined.

In [ ]:

```
def IMBS(video):
    P = 5  # The sampling period
    N = 15 # The number background samples to analyse
    D = 2  # The minimal number D of occurrences to consider a tupl
e ⟨r,g,b,d ≥ D⟩ as a significant background value
    A = 5  # The association threshold A for assigning a pixel to a
n existing tuple
    t = 0  # Current time
    n = 0  # According to a sampling period P, the current frame I
is added to L, thus becoming a background sample Sn, 1 ≤ n ≤ N
    ts = 0 # The timestamp of the last processed background sample
    B = []

    if video.isOpened():
        ratio = WIDTH / video.get(3)
        global HEIGHT
        HEIGHT = math.floor(video.get(4) * ratio)

    while True:
        check, frame = video.read()

        resizedFrame = cv2.resize(frame, (WIDTH, HEIGHT))

        frame = np.array(resizedFrame, dtype='int64')
        if t - ts > P:
            # During the sampling period, tuples are create or upda
ted
            B = registerBackground(n, frame, B, N, D, A)
            ts = t
            n += 1
            if n == N:
                n = 0

        t += 1
        foreground = getForeground(frame, B, A)
        visualFrame = getForegroundVisualFrame(foreground)
        cv2.imshow("Foreground", visualFrame)

        cv2.imshow("Image", resizedFrame)
        key = cv2.waitKey(100)

        if key == ord('q'):
            break
```

An Improvement: Morphological Operators

In [ ]:

```
def IMBS2(video):
    P = 5  # The sampling period
    N = 15 # The number background samples to analyse
    D = 2  # The minimal number D of occurrences to consider a tupl
e ⟨r,g,b,d ≥ D⟩ as a significant background value
    A = 5  # The association threshold A for assigning a pixel to a
n existing tuple
    t = 0  # Current time
    n = 0  # According to a sampling period P, the current frame I
is added to L, thus becoming a background sample Sn, 1 ≤ n ≤ N
    ts = 0 # The timestamp of the last processed background sample
    B = []

    if video.isOpened():
        ratio = WIDTH / video.get(3)
        global HEIGHT
        HEIGHT = math.floor(video.get(4) * ratio)

    while True:
        check, frame = video.read()

        resizedFrame = cv2.resize(frame, (WIDTH, HEIGHT))

        frame = np.array(resizedFrame, dtype='int64')
        if t - ts > P:
            # During the sampling period, tuples are create or upda
ted
            B = registerBackground(n, frame, B, N, D, A)
            ts = t
            n += 1
            if n == N:
                n = 0

        t += 1
        foreground = getForeground(frame, B, A)
        visualFrame = getForegroundVisualFrame(foreground)

        # MORPHOLOGICAL OPERATORS ------------------------------
------------------------------------------------
        visualFrame = cv2.morphologyEx(visualFrame, cv2.MORPH_CLOSE
, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3)))
        visualFrame = cv2.morphologyEx(visualFrame, cv2.MORPH_CLOSE
, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3)))

        cv2.imshow("Foreground", visualFrame)

        cv2.imshow("Image", resizedFrame)
        key = cv2.waitKey(100)

        if key == ord('q'):
            break
```

Another Improvement: Gaussian Filter

In [ ]:

```
def IMBS3(video):
    P = 5  # The sampling period
    N = 15 # The number background samples to analyse
    D = 2  # The minimal number D of occurrences to consider a tupl
e ⟨r,g,b,d ≥ D⟩ as a significant background value
    A = 5  # The association threshold A for assigning a pixel to a
n existing tuple
    t = 0  # Current time
    n = 0  # According to a sampling period P, the current frame I
is added to L, thus becoming a background sample Sn, 1 ≤ n ≤ N
    ts = 0 # The timestamp of the last processed background sample
    B = []

    if video.isOpened():
        ratio = WIDTH / video.get(3)
        global HEIGHT
        HEIGHT = math.floor(video.get(4) * ratio)

    while True:
        check, frame = video.read()

        # GAUSSIAN FILTER ----------------------------------------
----------------------------------------------
        frame = cv2.GaussianBlur(frame, (13, 13), 0)

        resizedFrame = cv2.resize(frame, (WIDTH, HEIGHT))

        frame = np.array(resizedFrame, dtype='int64')
        if t - ts > P:
            # During the sampling period, tuples are create or upda
ted
            B = registerBackground(n, frame, B, N, D, A)
            ts = t
            n += 1
            if n == N:
                n = 0

        t += 1
        foreground = getForeground(frame, B, A)
        visualFrame = getForegroundVisualFrame(foreground)

        # MORPHOLOGICAL OPERATORS --------------------------------
----------------------------------------------
        visualFrame = cv2.morphologyEx(visualFrame, cv2.MORPH_CLOSE
, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3)))
        visualFrame = cv2.morphologyEx(visualFrame, cv2.MORPH_CLOSE
, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3)))

        cv2.imshow("Foreground", visualFrame)

        cv2.imshow("Image", resizedFrame)
        key = cv2.waitKey(100)

        if key == ord('q'):
            break
```

Shadow Suppression

Now, let's add the shadow suppression method to get rid off shadows.

In [ ]:
```
def suppressShadows(frame, B):
    F = np.full((HEIGHT, WIDTH), 0)
    # User defined thresholds
    alpha = 0.75
    beta = 2.1
    ts = 40
    th = 60

    # Get current hsv frame
    hsvFrame = cv2.cvtColor(np.array(frame, dtype=np.uint8), cv2.CO
LOR_BGR2HSV)

    for i in range(len(frame)):
        for j in range(len(frame[i])):
            count = 0
            if len(B) > 0 and len(B[i][j].tuples) > 0:
                for k in range(len(B[i][j].tuples)):
                    rgbd = B[i][j].tuples[k]
                    hsvPixel = hsvFrame[i][j]
                    h, s, v = rgbd.getHSV()

                    if hsvPixel[HUE] - h <= th and hsvPixel[SATURAT
ION] - s <= ts and hsvPixel[VALUE] / v >= alpha and hsvPixel[VALUE]
/ v <= beta:
                        # Check the defined conditions in paper, if
it is satisfied increase count
                        count += 1

            if len(B) > 0 and count == len(B[i][j].tuples):
                # If all of the tuples satisfies the condition then
mark the pixel as a foreground
                F[i][j] = 1

    return F
```

In [ ]:
```
def IMBS4(video):
    P = 5  # The sampling period
    N = 15 # The number background samples to analyse
    D = 2  # The minimal number D of occurrences to consider a tupl
e 〈r,g,b,d ≥ D〉 as a significant background value
    A = 5  # The association threshold A for assigning a pixel to a
n existing tuple
    t = 0  # Current time
    n = 0  # According to a sampling period P, the current frame I
is added to L, thus becoming a background sample Sn, 1 ≤ n ≤ N
    ts = 0 # The timestamp of the last processed background sample
    B = []

    if video.isOpened():
```

```
        ratio = WIDTH / video.get(3)
        global HEIGHT
        HEIGHT = math.floor(video.get(4) * ratio)

    while True:
        check, frame = video.read()

        # GAUSSIAN FILTER -----------------------------------------
-----------------------------------------------
        frame = cv2.GaussianBlur(frame, (13, 13), 0)

        resizedFrame = cv2.resize(frame, (WIDTH, HEIGHT))

        frame = np.array(resizedFrame, dtype='int64')
        if t - ts > P:
            # During the sampling period, tuples are create or upda
ted
            B = registerBackground(n, frame, B, N, D, A)
            ts = t
            n += 1
            if n == N:
                n = 0

        t += 1
        # SUPPRESS SHADOWS ----------------------------------------
-----------------------------------------------
        foreground = suppressShadows(frame, B)
        visualFrame = getForegroundVisualFrame(foreground)

        # MORPHOLOGICAL OPERATORS ---------------------------------
-----------------------------------------------
        visualFrame = cv2.morphologyEx(visualFrame, cv2.MORPH_CLOSE
, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3)))
        visualFrame = cv2.morphologyEx(visualFrame, cv2.MORPH_CLOSE
, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3)))

        cv2.imshow("Foreground", visualFrame)

        cv2.imshow("Image", resizedFrame)
        key = cv2.waitKey(100)

        if key == ord('q'):
            break
```

Comparision With MOG

In [ ]:

```
def IMBS5(video):
    P = 5  # The sampling period
    N = 15 # The number background samples to analyse
    D = 2  # The minimal number D of occurrences to consider a tupl
e ⟨r,g,b,d ≥ D⟩ as a significant background value
    A = 5  # The association threshold A for assigning a pixel to a
n existing tuple
    t = 0  # Current time
```

```
    n = 0  # According to a sampling period P, the current frame I
is added to L, thus becoming a background sample Sn, 1 ≤ n ≤ N
    ts = 0 # The timestamp of the last processed background sample
    B = []

    if video.isOpened():
        ratio = WIDTH / video.get(3)
        global HEIGHT
        HEIGHT = math.floor(video.get(4) * ratio)

    fgbg = cv2.createBackgroundSubtractorMOG2()

    while True:
        check, frame = video.read()

        # GAUSSIAN FILTER ----------------------------------------
----------------------------------------------
        frame = cv2.GaussianBlur(frame, (13, 13), 0)

        resizedFrame = cv2.resize(frame, (WIDTH, HEIGHT))

        frame = np.array(resizedFrame, dtype='int64')
        if t - ts > P:
            # During the sampling period, tuples are create or upda
ted
            B = registerBackground(n, frame, B, N, D, A)
            ts = t
            n += 1
            if n == N:
                n = 0

        t += 1
        # SUPPRESS SHADOWS ---------------------------------------
----------------------------------------------
        foreground = suppressShadows(frame, B)
        visualFrame = getForegroundVisualFrame(foreground)

        # MORPHOLOGICAL OPERATORS --------------------------------
----------------------------------------------
        visualFrame = cv2.morphologyEx(visualFrame, cv2.MORPH_CLOSE
, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3)))
        visualFrame = cv2.morphologyEx(visualFrame, cv2.MORPH_CLOSE
, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3)))

        cv2.imshow("IMBS", visualFrame)

        fgmask = fgbg.apply(frame)
        cv2.imshow('MOG', fgmask)
        key = cv2.waitKey(100)

        if key == ord('q'):
            break
```

Main Program

In [ ]:
```
video = cv2.VideoCapture('video2.avi')
IMBS5(video)
```

In [ ]:
```

```