

## עיבוד וניתוח וידאו – פרויקט מסכם

### חלק א' – פירוט על הבלוקים המרכזיים בפרויקט

#### runme

מודל runme מכיל את הפונקציה הראשית של התוכנית. המודל מייבא את כל הבלוקים של הפרויקט.

מדידת זמנים: זמני ריצה נמדדים בתוכנית באמצעות ספריית timeit, ונמדד עבור פרקי הזמן הבאים: זמן אתחול, זמן ייצוב וידאו, זמן backsubstation, זמן Matting, זמן Tracking, זמן ריצה כולל.

קובץ לוג: כל הזמנים הנמדדים נכתבים לקובץ RunTimeLog.txt שנמצא בתיקיית Outputs ונכתב מחדש בכל הרצה. קובץ הלוג ממומש באמצעות ספריית logging. הלוגר נוצר בקובץ config.py על מנת שיהיה משותף לכל קבצי הפרויקט. בקובץ ה-config יש אפשרות לאפשר הודעות נוספות מסוג DEBUG שאנחנו כותבים לפרויקט, יש הערה מתאימה בקובץ תחת 'logger' שמסבירה היכן ומה בדיוק לשנות.

הרצת בלוקים בנפרד: כל בלוק בתוכנית ממומש על ידי פונקציה בודדת שאינה מקבלת ארגומנטים, זאת מכיוון שכל הארגומנטים ידועים מכיוון כל שמות הקבצים שיש לקרוא/לכתוב ומיקומיהם ידועים. ניתן להריץ כל בלוק בנפרד ע"י סימון הערה על השורה שמבצעת את הרצת הבלוק שרוצים להתעלם ממנו.

קונפיגורציה לפונקציה הראשית:

קובץ הקונפיגורציה config.py מחולק לקטגוריות בהתאם לכל בלוק, המופרדים על ידי הערה בולטת בקוד. בנוסף קיימות קטגוריות כלליות להרצה, פירוט:

# ~~~ DEMO ~~~ #

ניתן לשלוט על הקלט של התוכנית באמצעות config.py:

כאשר DEMO=True, יילקח הסרט המיוצב שניתן לנו מראש במקום הסרט שאנחנו מייצבים Stabilized\_Example\_INPUT.avi וכן הפרמטרים האופטימליים (שמצאנו) לווידאו זה. כאשר DEMO=False יילקח הסרט שאנחנו מייצבים stabilize.avi וכן יילקחו הפרמטרים האופטימליים לווידאו זה.

# ~~~ file paths ~~~ #

בקובץ הקונפיגורציה מופיעים כל ה-paths היחסיים של כל הקבצים שנכתבים או נקלטים בהתאם לדרישות הפרויקט.

# ~~~ logger ~~~ #

כאמור כולל את אתחול הלוגר.

מיקומים יחסיים: דאגנו לכך שמיקומי התיקיות יחסיים. יש לפתוח פרויקט PyCharm בתיקייה הראשית Code.

הצגת bar התקדמות: עם התחלת כל אחד מהבלוקים בתוכנית, מודפסת שורה ל-console עם שם הבלוק. במהלך ריצת הבלוק מוצג progress bar ב-console. ה-progress bar ממומש באמצעות פונקציית tqdm של הספרייה tqdm. ה-progress bar מוצג עבור תהליך מהותי בכל אחד מהבלוקים, ומציג פס מתקדם, התקדמות באחוזים והתקדמות יחסית (שבר).

בדיקת שפיות לגדלי התמונות: על מנת לוודא עמידה בדרישות פרמטרי הפלטים (כגון מימדי הפריים, כמות הפריימים) התוכנית מוודאת שפרמטרי כל הפלטים זהים לוידאו הקלט. במידה והבדיקה עוברת מודפסת ל-console הודעה All output tests passed, אם אחד מהקבצי הפלט לא עובר בבדיקה זו ההודעה לא מודפסת והקובץ הבעייתי נכתב לקובץ הלוג.

## ייצוב וידיאו Video Stabilization:

בחלק זה של הפרוייקט התבססנו בעיקר על ספריית OpenCV. השיטה בה בחרנו להשתמש הינה:

### Point Feature Matching

בשיטה זו בוחרים נקודות עניין ומבצעים עקיבה בין המיקום שלהם בפריים הנוכחי לפריים שמגיע לאחר מכן. בחירת נקודות העניין נעשית באמצעות שימוש בפונקציה `cv2.goodFeatureToTrack()`. ניתן לשנות את הפרמטרים של הפונקציה הזו בקובץ ה-`config.py`.

היתרון של שימוש בפונקציה זו הוא שמחפשים נקודות אשר מהוות "פינות" (שפיצים) לפי פרמטרי איכות הניתנים לשליטה, ובכך למעשה אלגוריתם העקיבה שלנו חסין יותר לבעיית ה-`aperture` (בעיה זו נובעת מכך שבמצלמה המפתח ראייה שלנו הוא סופי ומוכתב מקוטר העדשה ומהחיישן האופטי במצלמה, כתוצאה מכך אם היינו עוקבים למשל אחרי קווים ישרים, יתכן ולא היינו מזהים את כיוון תנועתם האמיתי, אלא רק את גרדיאנט כיוון התנועה).

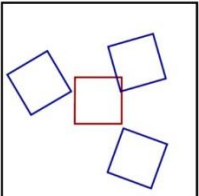
על מנת לשערך את תזוזת נקודות העניין שנבחרו הפעלנו אלגוריתם `Optical Flow` עם `Lukas Kanade` מבוסס שימוש בפיירמידות. הפעלת האלגוריתם הנ"ל נעשתה באמצעות שימוש בפונקציה: `cv2.calcOpticalFlowPyrLK()`. ניתן לשנות את הפרמטרים של הפונקציה הזו בקובץ ה-`config.py`.

היתרון בכך שבחרנו לעבוד עם `feature points` ולא עם התמונה כולה הוא בכך שהצלחנו להשיג סיבוכיות חישובית נמוכה, מכיוון שחישוב `Optical Flow` הינו צרכן משאבים גדול, הסתפקות במספר מועט יותר של פיקסלים מאפשר להאיץ משמעותית את זמן הריצה. בקוד שלנו איפשרנו לעקוב אחרי מקסימום 1000 נקודות עניין, מספר נמוך יחסית למספר הפיקסלים בתמונה (1920X1080).

כעת לאחר שחישבנו `Optical Flow` אנחנו יודעים את מיקום ה-`Feature Points` בפריים הנוכחי ובפריים שאחריו.

בהתחשב בערכים אלה אנחנו מחשבים **טרנספורמציה אוקלידית** באמצעות הפונקציה `cv2.estimateRigidTransform()`. באמצעות טרנספורמציה זו אנו יכולים לדעת את הטרנסלציה ב-`x,y` ואת זווית הסיבוב של התמונה.

Euclidean transformation



Homogeneous:

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \omega_{11} & \omega_{12} & \tau_x \\ \omega_{21} & \omega_{22} & \tau_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Cartesian:

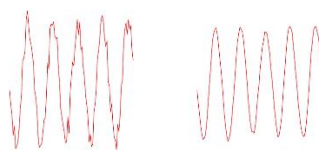
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \omega_{11} & \omega_{12} \\ \omega_{21} & \omega_{22} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} \tau_x \\ \tau_y \end{bmatrix}$$

For short:

$$\mathbf{x}' = \text{euc}[\mathbf{w}, \Omega, \tau]$$

Computer vision: models, learning and inference. ©2011 Simon D. Prince

על מנת לדעת את מסלול התנועה הכולל של הפריים הנוכחי, אנו למעשה סוכמים את כל תוצאות הטרנספורמציות שחושבו בין פריימים שכנים. ככה למעשה אנחנו יכולים לחזור מהמיקום והזווית של הפריים ה-`i` למיקום ולזווית של הפריים הראשון, בדומה למה שביצענו בתרגיל בית מספר 2. על מנת לקבל יצוב חלק ואיכותי, ביצענו החלקה על ידי ממוצע `moving average` על המסלולים שחושבו.



ניתן לשנות את פרמטר רדיוס ההחלקה בקובץ ה-`config.py`.

כעת אנו יכולים לבנות את מטריצת הטרנספורמציה:

$$T = \begin{bmatrix} \cos \theta & -\sin \theta & x \\ \sin \theta & \cos \theta & y \end{bmatrix}$$

בשלב הבא אנו משתמשים בפונקציה `cv2.warpAffine()` על מנת להחזיר את הפריים למיקום ולזווית הרצויים שלפני ההרעדה, אשר יבטלו את רעידות הסרטון.

לאחר מכן אנו מבצעים פעולה של **תיקון הגבולות החיצוניים** כתוצאה מהיווצרות ארטיפקטים כתוצאה מביצוע הטרנספורמציה.

החסרון בשימוש בשיטה זו הוא שהמודל של הטרנספורמציה האפינית לא מכיל מספיק דרגות חופש במודל התנועה. כמו כן האלגוריתם המתואר שמימשנו פחות חסין לתנועות בתדירות גבוהה אשר אופייניות בסרטון שעליו עבדנו בפריימים מסוימים, כמו תזוזת הפנים של האובייקט לכיוון שמאל במהלך הסרטון, או כמו תזוזה חדה של המצלמה ימינה במהלך הסרטון. עם זאת אנו מרוויחים זמן ריצה קצר ותוצאה מספקת בקובץ `stabilized.avi`.

שמירת מטריצות המעבר/טרנספורמציה: על מנת לבצע את "אי-ייצוב הוידאו", כלומר את ההתמרה ההפוכה של ההתמרה בה השתמשנו לייצוב הוידאו ולהפעילה על מפת האלפא שניצור בשלב ה-Matting, אנו שומרים את מטריצת הטרנספורמציה עבור כל הפריימים (ליתר דיוק עבור מספר הפריימים פחות 1, כי מדובר על מעבר בין פריימים). עקב הדרישה להיות בעלי יכולת הרצה של כל בלוק בנפרד, נמנענו בכוונה מלהחזיר מטריצה זו בשלב הוידאו ולהכניסה כארגומנט לפונקציית ה-Matting בפונקציה הראשית `runme`. לצורך שמירת הטרנספורמציות תוך הימנעות מפעולה שכזו, השתמשנו בספריית `pickle` של פייתון לשמירת הקובץ בסוף שלב ייצוב הוידאו, וטעינתו בתחילת שלב ה-Matting. באופן כזה ניתן יהיה להפעיל רק את שלב ה-Matting בנפרד (בהנחה שמישהו בעבר הפעיל את ייצוב הוידאו), לקוד ה-Matting יוזן הקובץ ששמו ומיקומו הם: `Temp\video_stabilization_transforms.p`.

## Background Subtraction

בחלק זה של הפרוייקט עשינו שימוש בעיקר בכלים שנמצאים בספריית `OpenCV`. הכלי המרכזי שהשתמשנו בו הוא פונקציות לביצוע `Background Subtraction (BS)` (המשתמשות באלגוריתם `KNN (K-nearest neighbors)`). יצרנו אובייקט באמצעות הפונקציה `cv2.createBackgroundSubtractor()`, בשלב הבא אנו מפעילים את האלגוריתם `backSub.apply` המשתמש בכלים סטטיסטיים בהתבסס על היסטוריית הפריימים הקודמים ובונה מודל של הרקע. לאחר מכן האלגוריתם מחסיר מהפריים את המודל שנבנה, ומחלץ מהתוצאה תמונה בינארית שבה כל מקום הצבוע בלבן שייך לאובייקט, וכל מקום שצבוע בשחור שייך לרקע. התמונה הבינארית בגודל הפריים הנוכחי שעובדים עליו.

התוצאה של ה-KNN אינה מושלמת, מכיוון שהסרטון בכניסה אינו מיוצב באופן אידיאלי, קיימים פריימים מסוימים בהם מתקבלים רעשים. כמו כן האלגוריתם לומד, במובן שבפריימים הראשונים (כ-40 פריימים ראשונים) המודל עדיין לא מזהה את האדם באופן נפרד מהרקע. בקובץ `config.py` ניתן לשנות את הפרמטרים של אלגוריתם ה-KNN: הפונקציה `createBackground_Substraction` והפונקציה `backSub_apply`.

אנו מבצעים סינונים נוספים בשימוש במספר פילטרים (לפי הסדר הבא):

**פילטר חציון/median filter:**

כפי שתואר בטיפים לפרויקט, מבצעים סינון מידיאני עבור כל אחד מהשכבות B,G,R, בשני אופנים:

- (1) יצירת המסנן מתוך כל הפריימים של הוידאו. יצרנו תמונה חדשה שנקראת background, אותה אנחנו שומרים בשביל התיעוד/אפשרות לקיצור זמן ריצה של חישוב המסנן בכל פעם מחדש על אותו הוידאו בעתיד (אם נרצה בכך). התמונה נשמרת בכל הרצה במיקום ובשם Temp\background.jpg.
- (2) עבור מספר נבחר של פריימים אחרונים לחצי הימני של התמונה, ועבור מספר נבחר של פריימים ראשונים לחצי השמאלי של התמונה. אנחנו מנצלים את העובדה שהאובייקט מתחיל את התנועה בצד שמאל ביחס לרקע, נע ימינה לאורך כל הוידאו ומסיים את תנועתו בצד ימין ביחס לרקע. תמונת הרקע הנוצרת בשיטה זו חדה יותר (פחות מרוחה), ועוזרת לסינון עצמים בתמונה הדורשים חדות רבה, כמו המילים הכתובות על השלטים ברקע. התמונה נשמרת בכל הרצה בשם ובמיקום: Temp\background50\_50.jpg. ניתן לשנות את מספר הפריימים שנלקחים מכל חצי בקובץ config.py.

הערות:

- אנחנו מפעילים כל אחד מהפילטרים על מסכת fgMask הבינארית שקיבלנו מה-KNN, באופן כזה שרק מקומות אותם המדיאני סינן מסוננים גם ב-fgMask.
- אנחנו משתמשים בשני הפילטרים ולא רק באחד מהם מכיוון שלכל אחד החוזקות והחולשות שלו, ולאחר ניסיונות רבים הגענו לתוצאה הטובה ביותר בשימוש בשניהם אחד אחרי השני.

### פעולות מורפולוגיות:

בשלב הראשון אנו מבצעים פעולת erode ובשלב הבא אנו מבצעים פעולת dilate. מספר האיטרציות וגודל החלונות ניתנים לשינוי בקובץ config.py. לאחר ניסיונות רבים הגענו למסקנה שמתקבל סינון טוב יותר בשימוש ב-kernel אליפטי עבור שתי הפעולות.

### Blob Detector:

פילטר של OpenCV. בפרוייקט זה אנו יודעים כי קיים רק אובייקט אחד שנמצא בתנועה, לכן אנו יכולים להשתמש בידע הזה, ולחפש אוספים של נקודות שלא סוננו בגדלים קטנים מגודל האובייקט, ולסנן אותם. מכיוון שה-Blob Detector של OpenCV יודע לזהות Blobs שחורים, ביצענו פעולת Bitwise not על התמונה, הפעלנו עליה את ה-detector, מחקנו את הרעשים ששטחם היה קטן מערך שקבענו, ולאחר מכן בצענו פעם נוספת Bitwise not. ניתן לשנות את הפרמטרים של פילטר blob בקובץ config.py.

### פילטר מסרק:

"פילטר" זה משמש בעצם כמגבר, הוא מוסיף נתונים בעלי גוונים מסוימים בתמונה למסכה שלנו. אנחנו משתמשים ב-5 מסנני "BPF" מסוימים שהותאמו במיוחד לאובייקט הנע:

- (1) shirt\_mask\_morph - פילטר המסנן את החולצה של האובייקט.
- (2) shorts\_mask\_morph - פילטר המסנן את המכנס של האובייקט.
- (3) shoes\_mask\_morph - פילטר המסנן את נעלי האובייקט (בעיקר את החלקים ששונים מהרצפה). הפילטר משתמש בניצול העובדה שנעלי האובייקט תמיד נמצאים ברבע התחתון של המסך, לכן אנחנו מבטלים השפעה של  $\frac{3}{4}$  העליון של כל פריים.
- (4) legs\_mask\_morph - פילטר המסנן את רגלי האובייקט. כאן אנחנו מבצעים איפוס של הפילטר ב-2/3 העליונים של המסך.
- (5) skin\_mask\_morph - פילטר המסנן את צבע העור של האובייקט (פנים וידיים). מכיוון שצבע העור של האובייקט דומה מאוד לידיים שבאחד הפוסטרים בחצי הימני של הרקע, השתמשנו בפילטר זה רק עבור 3/8 הפריימים הראשונים, תוך איפוס השפעת החצי הימני של הפריים.

לאחר יצירת כל הפילטרים איחדנו את השפעתם בשימוש בפונקציה cv2.bitwise\_or, והשתמשנו עבור כל פריים בתוצאה על מנת להוסיף גוונים אלו לתמונה.

יש לציין שפתרון זה הוא רק תוסף לפילטר ה-KNN, ואינו תחליף. ה-KNN מספק אלמנטים שקשה מאוד לסנן בצורה מלאכותית שכזו כגון העיניים או השעון של האובייקט.

## Matting

בחלק הראשון של בלוק זה החלטנו כי אנו מעדיפים לעבוד בפורמט HSV, כאשר החישובים הסטטיסטיים יתבצעו על ערוץ ה-V.

לאחר בדיקת מספר עוקבים שונים (ראה פונקציית `choose_tracker` בקובץ `video_handling.py`) החלטנו להשתמש ב-Tracker מסוג CSRT (השייך לספריית OpenCV). העוקב הנ"ל נבחר מכיוון שהיחס זמן ריצה-ביצועים טובים יותר לעומת העוקבים האחרים. ההסבר לכך נובע מהעובדה שחישוב המרחק הגאודזי וה-KDE נעשים מהר יותר מכיוון שכוללים פחות פיקסלים. אנו מבצעים עקיבה אחר האובייקט, פותחים סביבו מלבן, ולאחר מכן מגדירים שכל מה שמחוץ למלבן זה שייך בוודאות לרקע, וכל שאר הפיקסלים שבתוך המלבן עוברים את העיבוד הבא:

1. **סקריבלים** – במקום לצייר סקריבלים ידנית עשינו שימוש במפה הבינארית שהתקבלה בשלב ה-BS. כל מה שלבן שייך לאובייקט, כל מה ששחור שייך לרקע. כמובן שבהנחה זאת יש שגיאות מסויימות שנובעות מכך שבשלב ה-BS התמונה הבינארית שקיבלנו אינה מושלמת כלל ומכילה שגיאות מקומיות.
2. **חישוב likelihood** – בשלב זה אנו מחלצים PDF באמצעות אלגוריתם KDE בהתבסס על מפות ה-foreground וה-background שקיבלנו בשלב 1. את החישוב הנ"ל ביצענו באמצעות פונקציית `gaussian_kde()` של ספריית Scipy. כפי שלמדנו בכיתה, הפונקציה מייצרת גאוסין מכל נקודה ומאחדת את כל הגאוסיינים לקבלת פונקציית פילוג ההסתברות. למעשה בשלב הזה קיבלנו קירוב להיסטוגרמה של התמונה.
3. בהתבסס על ה-KDE אנו מחשבים עבור כל פיקסל בתמונה מה ההסתברות שהוא שייך ל- $P_B$  Background ומהי ההסתברות הוא שייך ל- $P_F$  Foreground.
4. **חישוב המרחק הגאודזי** בין מפות ההסתברות של ה-Background וה-Foreground לבין המפות ("הסקריבלים") שקיבלנו בשלב 1. החישוב מבוצע בפונקציה `geo_distance` שבקובץ `Matting_functions.py`.  
למימוש של החישוב של המרחק הגאודזי השתמשנו בחבילה בשם GeodisTK שנלקחה מ:  
<https://github.com/taigw/GeodisTK>
- חבילה זו ממשת את החישוב הגאודזי בגישת `fast marching`, שיטה זו מבוססת על פרופוגציה איטרטיבית כמו "שריפה בשדה קוצים". היתרון של השימוש בחבילה זו לעומת חבילות אחרות שבחנו ושקלנו להשתמש בהן הוא הזמן ריצה המהיר. (ראה פירוט על חישוב מרחק גאודזי נוסף שלא בחרנו להשתמש בו בחלק ב' של המסמך).
- מימוש החישוב הגאודזי הוא בעיה יחסית מסובכת למימוש על מערכות מחשוב מודרניות, מכיוון שיש דרישה לבצע גישות רבות לא סדורות לזיכרון של המחשב, דבר שגורם לזמני ריצה גבוהים מאוד (`cache misses`), למרות שהסיבוכיות החישובית לא סופר גבוהה. ניתן להתמודד עם בעיות אלה על ידי שימוש באלגוריתמים שונים המתגברים על בעיה זו, כמו שלמדנו בהרצאה.
5. כל מקום שבו ההפרש בין המרחק הגאודזי של ה-Foreground לבין המרחק הגאודזי של ה-Background הוא שלילי יוגדר כ-Foreground.
6. כעת עשינו שימוש בפונקציית פייתון שמומשה בהתאם למקבילה שלה במטלאב, פונקציית `BWperim`, למעשה הפונקציה הזאת נותנת לנו '1' בכל המקומות המהווים את השפה החיצונים של ה-Foreground. הערה: הפונקציה ממומשת בתוך הקוד של בלוק ה-Matting.
7. בשלב הבא אנו מבצעים פעולה מרפולוגית `dilate`, באמצעות כדור בגודל 2x2, פעולה זו מבוצעת פעמיים על מנת לעבות את הקונטור שקיבלנו בשלב 6.
8. כעת אנו בונים את ה-`Trimap` לתמונה. כל מה שמחוץ לקונטור מוגדר כרקע ('0'), כל מה שבתוך הקונטור מוגדר כאובייקט ('1') וכל מה שבין לבין מוגדר כ-undecided ('0.5').
9. חישוב **מפת אלפא** – כדי לחשב את מפת האלפא אנחנו מגדירים פונקציית משקל  $w(x)$ , פונקציה זו היא למעשה שיקלול בין ה-likelihood לבין המרחק הגאודזי:

$$\omega_l(x) = D_l(x)^{-r} \cdot P_l(x), \quad l \in \{\mathcal{F}, \mathcal{B}\}, \quad 0 \leq r \leq 2$$

$$\alpha(x) = \frac{\omega_{\mathcal{F}}(x)}{\omega_{\mathcal{F}}(x) + \omega_{\mathcal{B}}(x)},$$

הפרמטר  $r$  מאפשר לקבוע את כמות ההשפעה של המרחק הגאודזי וניתן לשינוי בקובץ `config.py`. כמו כן אנו דואגים כי ערכה של אפלא ב- Foreground יהיה 1, וב- Background יהיה 0. כאשר באזור ה-undecided מקבלים ערכים בין 0 ל-1 המאפשרים לבצע בלנדינג כפי שמתואר בשלב הבא.

10. **שלב שקלול:** הערך עבור כל פיקסל בפריים הוא צירוף לינארי בין **הרקע החדש**, לבין ה-Foreground, משוקללים על ידי אלפא כפי שלמדנו בהרצאה:

$$\text{image with new background} = \alpha(x, y) \cdot F(x, y) + (1 - \alpha(x, y)) \cdot B(x, y)$$

על מנת לקבל את וידיאו הפלט אנו חוזרים על כל אחד מהשלבים עבור כל אחד מהפריימים של הסרטון המיוצב. לאחר ריצה על כל הפריימים אנו מקבלים את הסרטון הרצוי עם הרקע החדש.

לצורך כתיבת **מפת האלפא** לקובץ וידאו `alpha.avi`, אנו כופלים את כל ערכי המפה בערך 255 ומבצעים casting לטיפוס `np.uint8`. כל הפריימים של מפת האלפא מתווספים לרשימה `alpha_list` שמשתמשת אותנו לאי ייצוב מפת האלפא עבור כל פריים.

בתום ביצוע תהליך ה-Matting וחילוץ מפת אלפא על כל הפריימים, אנחנו משתמשים ברשימת הטרנספורמציות מבלוק ייצוב הוידאו ומבצעים את ההתמרה ההפוכה עבור כל פריים לקבלת `unstabilized_alpha.avi`.

## Tracking

בחלק זה חיפשנו דרך לבצע עקיבה כמה שיותר טובה, אך כזו שלא תגזול לנו משאבים רבים ותגדיל את זמן הריצה באופן משמעותי, לאחר בחינת מספר אופציות החלטנו להשתמש ב-Tracker מסוג CSRT או בשמו המלא Channel and Spatial Reliability Tracker. טראקר זה מתבסס על פילטר קורולציה (DCF) עם feature sets. משתמשים בפילטר סביב האזור שבפעם האחרונה ידוע שהאובייקט היה בו על מנת למצוא את מיקומו הנוכחי. טראקר זה איטי במעט מטראקר KCF, אך בעל יכולות טובות להתאושש מ-Tracking Failures.

הטראקר עצמו מומש בספריית OpenCV.

## חלק ב' – פירוט על קבצי קוד נוספים בפרויקט – שונות או miscellaneous

### video\_handling.py

כולל פונקציות כלליות בהן אנו משתמשים ביותר מבלוק בודד. פונקציית `extract_video_params` המבצעת חילוץ פרמטרים מקובץ וידאו ובדיקת תקינותו. פונקציית `choose_tracker` שבוחרת את מודל העקיבה בהתאם למה שהגדרנו ב-`config.py`. פונקציית `test_all_outputs` שבודקת בדיקת שפיות לפלטים שנוצרו. באופן כזה אנו נמנעים משכפול קוד ובאגים מיותרים, מכיוון שחילוץ פרמטרים נעשה בכל אחד מהבלוקים, מודל עקיבה ממומש הן בשלב ה-Matting והן בשלב ה-Tracking.

### BS\_filters.py

ריכזנו את הפילטרים הנוספים בהם אנו משתמשים לתיקון פילטר KNN בשלב ה-BS. זאת על מנת להקל על קריאות הקוד, ועל מנת להפריד בין הפילטר הראשי והתוספות שלו.

## Matting functions.py

כולל ריכוז של פונקציות ופילטרים בהם אנו משתמשים בשלב ה-Matting: **פונקציה לחישוב KDE**, פונקציה לחישוב שפות **bwperim**, חישוב **מרחק גאודזי** ופונקציה **לתיקון קצוות** ביצירת מפת **unstabilized\_alpha.avi**.

## תיקיית unused

תיקייה המכילה קבצים שאינם בשימוש בתוכנית הסופית אותה הגשנו, אך היא מכילה ניסיונות נוספים שביצענו. פירוט:

תיקיית geodesic\_distance\_transform-master: מימוש של פונקציה לחישוב מרחק גאודזי, התוצאות שהתקבלו טובות אך זמן החישוב היה איטי מאוד (כ-50-60 שניות לפריים), החלפנו את הפונקציה בשיטה המתבססת על fast marching אשר נתנה תוצאות דומות בזמן ריצה קצר בהרבה (מספר שניות לפריים).

קובץ Matting\_binary.py, ביצוע Matting ללא העקיבה בתוכו. נפסל עקב שיפור מזערי בלבד של התוצאה לעומת ה-Matting עם העקיבה, ובזמן ריצה גדול בהרבה (יותר מ-30 שניות תוספת זמן ריצה לכל פריים בווידאו).

קובץ Matting\_functions\_unused.py שמכיל פונקציות חישוב מרחק גאודזי בה לא השתמשנו כאמור.

קובץ BS\_filters\_unused.py מרכז ניסיונות נוספים לשימוש בפילטר מדיאני שעבור כל פיקסל בתמונת המטרה מחפש האם יש פיקסלים בסביבה של אותן קורדינטות בתמונת הרקע המדיאנית בעלי אותו ערך, הפונקציה יעילה יותר אך לא נבחרה בשל תוספת זמן הריצה (כ-40 שניות נוספות לפריים בשלב ה-BE).

## ניסיונות נוספים לשיפור התוצאה

במהלך העבודה על הפרוייקט ניסינו להשתמש במגוון טכניקות שאספנו, ממקורות שונים כגון האינטרנט, מקורסים קודמים כמו עיבוד תמונה, ורובם מקורס עיבוד הוידאו, במטרה להגיע לתוצאה הטובה ביותר.

בכל אחד משלבי הפרוייקט נתקלנו בדילמות הנוגעות לאיכות התוצאה, אל מול זמן הריצה וסיבוכיות הקוד.

דילמות הנוגעות להרצת קוד מסובך על תמונה ברזולוציה מלאה, אל מול הוספת אלמנט עקיבה שיאפשר לבצע את הקוד על חלק קטן מהתמונה, אך יגזול בעצמו משאבים לא קטנים.

טכניקות נוספות שניסינו אך לא שולבו בגרסה הסופית עקב יחס שיפור/זמן ריצה נמוך:

1. עבודה במרחבי צבע שונים כגון: Grayscale, RGB, HSV ועוד.
2. שימוש בפילטרים מרחביים כגון: Sobel, Bilateral, Gaussian.
3. שימוש ב-Unsharp Mask.
4. ביצוע פעולה של השוואת היסטוגרמה.
5. שימוש בפעולות מורפולוגיות מסוג erode ו-dilate בעלות קרנלים עגולים, אליפטיים, בצורת צלב, מרובעים.
6. שימוש בפילטר מדיאני המבוסס על סביבה של פיקסלים – פירוט על כך למעלה בפרק **תיקיית Unused**.
7. שימוש בפונקציית מרחק גאודזי שונה, פירוט למעלה בפרק **תיקיית Unused**.

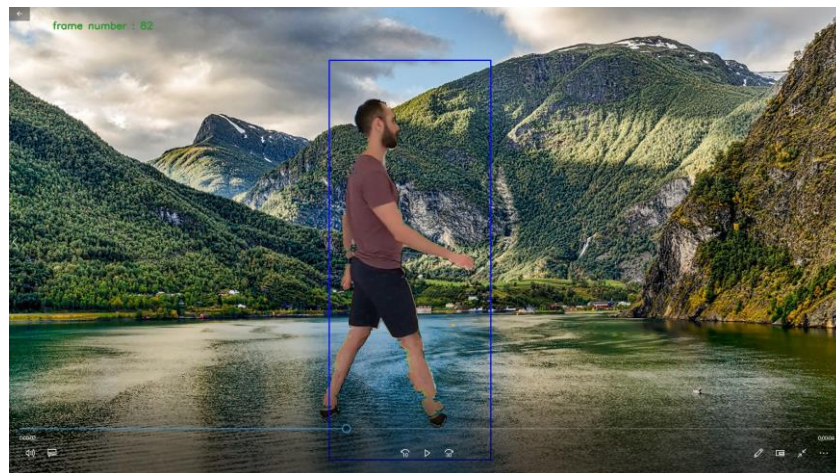


באופן כללי למדנו כי בתחום הוידאו והעיבוד תמונה קיים ארגז כלים נרחב בעל כלים המאפשרים להגיע למגוון פתרונות שונים. לכל אחד מהפתרונות יכולים להיות היתרונות והחסרונות שלו, לדוגמה התאמה לבעיה ספציפית ולא כללית, זמן ריצה, סיבוכיות, איכות תוצאה. באופן כללי כמהנדסים הפתרון בו נבחר יהיה תמיד תלוי אפליקציה.

## חלק ג' – תוצאות הרצה

### תוצאות הרצת DEMO:

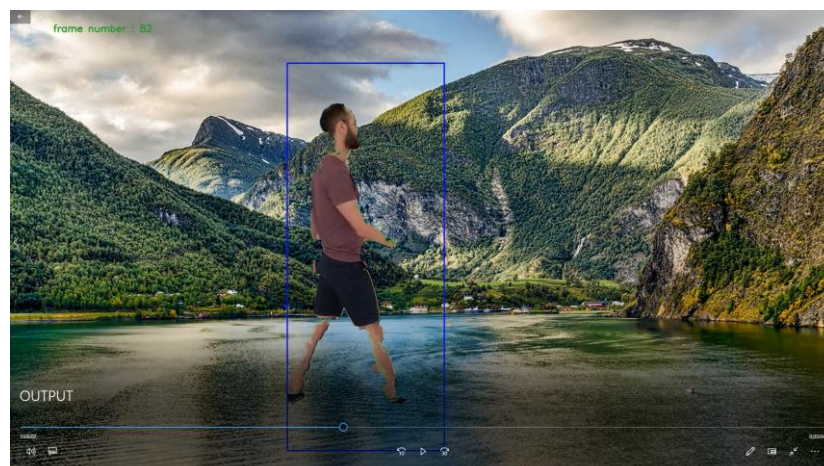
תיקיית ההרצה של הוידאו המיוצב שניתן לנו נמצאת במיקום Temp\Outputs\_DEMO.  
תמונות להמחשה:



זמן ריצה כולל: 48 דקות ו-45 שניות (ראה הקלטה מצורפת).

### תוצאות הרצת הוידאו המיוצב שלנו:

תוצאות ההרצה ה"טובה ביותר" (מבחינת זמן ריצה, תוצאות ההרצה) נמצאות בתיקייה Outputs.  
המחשה תמונות:



זמן ריצה כולל: 41:34 דקות



אור בהרי 204356315  
יונתן רודין 200846038

מהתוצאות ניתן לראות כי לאיכות היצוב מהווה חסם על איכות התוצאות שניתן להשיג, ככל שהייצוב יותר איכותי כך ניתן להגיע לתוצאות טובות יותר.

### הקלטת הרצה:

את הקלטות שתי ההרצות ניתן למצוא בתיקייה Temp\Records, הקלטת הדמו מכילה את המילה DEMO בשם שלה.