Python Drone Tracker

פרויקט גמר לתואר שני

מגיש: רועי אייזנר

מנחה: יוני מנדל

מבוא

קודם לפרויקט זה, בוצע פרויקט לאפיון ומימוש אלגוריתם עקיבה אחרי רחפן בוידאו.

האלגוריתם מומש ב Matlab ורץ בקצבים נמוכים מאוד ורחוקים מ Real-Time.

מטרת פרויקט זה היא לעשות עוד צעד לכיוון ריצה בזמן אמת של האלגוריתם על רחפן, שיאפשר לו (בעזרת אלגוריתם בקרה משלים) לעקוב בצורה אוטונומית אחרי רחפנים אחרים בעזרת מצלמה בלבד.

כדי לאפשר הרצה של אלגוריתם כזה על פלטפורמת Embedded, הפרויקט מכיל שני חלקים:

1. בחינת ביצועי העוקבים הקיימים ב Python בספריית OpenCV, מבחינת ביצועי עקיבה וזמן ריצה.
2. כתיבת האלגוריתם מחדש ב Python, תיקון באגים, ייעול ובחינת ביצועים.

תוכן עניינים

השוואת ביצועים של העוקבים הקיימים ב OpenCV

בטבלה מוצגים הביצועים של העוקבים הקיימים ב OpenCV.

האלגוריתם שקדם לעבודה זו פותח ב Matlab ונבדק בעזרת קבוצת סרטונים שצולמו מרחפן. העוקבים ב OpenCV נבדקו בעזרת אותם סרטונים (ממוספרים כמו ב Matlab) כשהם מתחילים בדיוק מאותו bounding box מסביב לרחפן.

ירוק = עוקב טוב לאורך כל הסרטון

צהוב = עוקב טוב אבל מאבד באמצע

אדום = לא מצליח לעקוב

csrt	kcf	boosting	mil	medianflow	mosse	
	לא הדוק	לא הדוק			לא הדוק	1
						2
						3
						4
						5
						6
						7
						8
						9
						10
						11
						12
						13
						14
						17
						18
						19
						20
						21
						22
						23
						24
						25
						26
						27
8-15	25-35	7-18	7-8	17-19	30-40	FPS

ניתן לראות שהעוקב הטוב ביותר לבחינה והתעמקות הוא **MILTrack**, שמציג ביצועי עקיבה טובים על רוב הסרטונים בזמן ריצה סביר לסרטון.

סקירה – אלגוריתם MILTrack

ממומש על פי המאמר http://faculty.ucmerced.edu/mhyang/papers/cvpr09a.pdf.

זהו אלגוריתם Object Tracking ששייך לקבוצה של שיטות עקיבה שנקראות "tracking by detection" שהן בעלות ביצועי זמן ריצה טובים ומתאימות ל real-time.

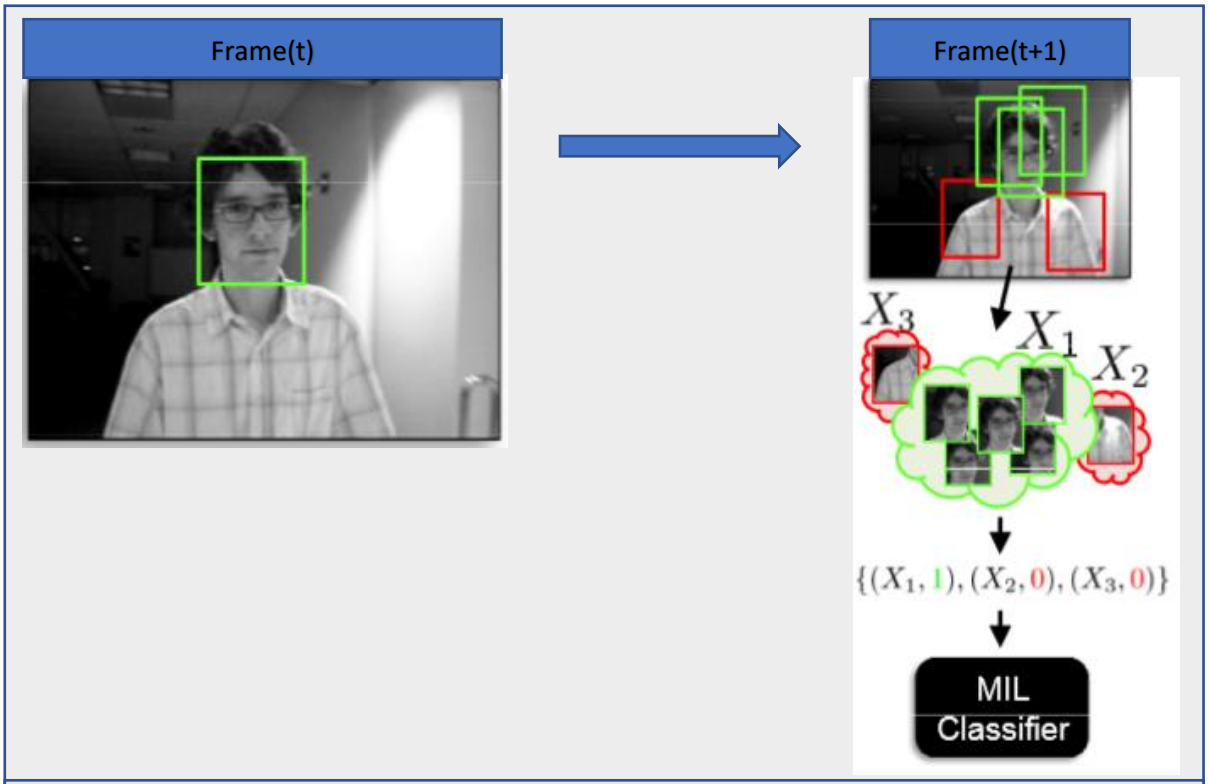
בשיטות אלה מאמנים מסווג אדפטיבי שמטרתו להפריד בין האובייקט לרקע. כדי לאמן את המסווג online משתמשים בכל פעם במצב הנוכחי של העוקב כדי לחלץ דוגמאות חיוביות (אובייקט) ושליליות (רקע).

השיטה הזו רגישה לאי דיוקים קטנים בביצועי העוקב שגורמים לדוגמאות אימון פגומות ולסחיפה מהירה של העוקב. MILTrack מנסה להתמודד עם הבעיה הזו ע"י שימוש ב Multiple Instance Learning (MIL) כך שבמקום לאמן עם דוגמה חיובית בודדת (רגיש מאוד לאי דיוקים) מאמנים על "שק" חיובי, שמכיל תמונות מסביבת העוקב כך שלפחות דוגמה אחת בשק היא חיובית.

האלגוריתם מבוסס על הקונספט של Boosting – שילוב של הרבה מסווגים חלשים (לרוב כלל החלטה על feature בודד) כדי לקבל מסווג אחד חזק, כאשר בכל שלב מעדכנים את המסווגים החלשים אחד אחרי השני לפי ה feature שמפריד בצורה הטובה ביותר בין דוגמאות חיוביות לשליליות.

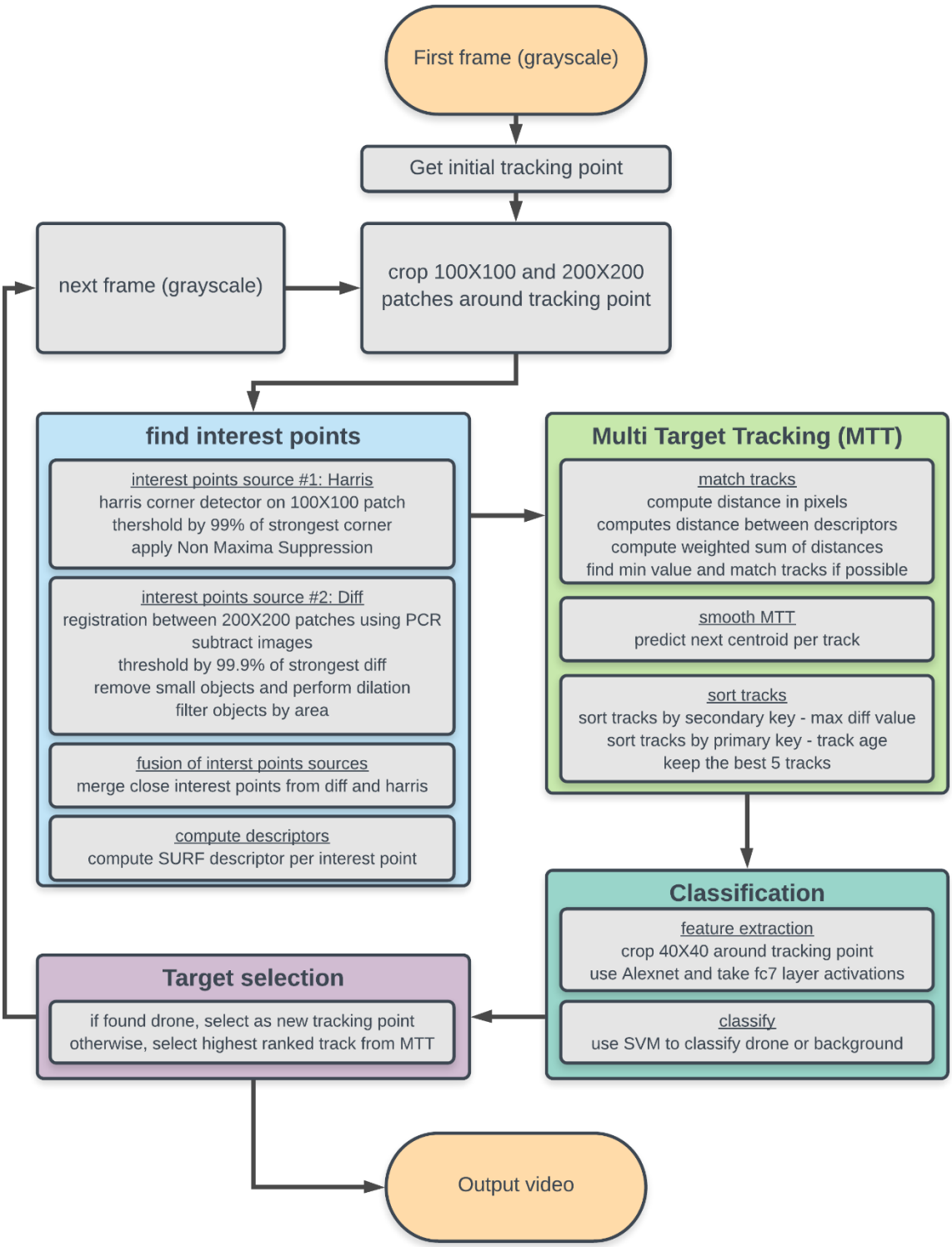
על ידי השימוש ב"שק" חיובי שמכיל את האובייקט, האלגוריתם מצליח להתמודד בצורה טובה עם שינויים והסתרות חלקיות, אך לא יכול להתמודד עם הסתרה מלאה.

בזכות המסווג האדפטיבי, העוקב הזה מראה ביצועים טובים מאוד על הסרטונים של הרחפן, ומצליח להתמודד עם התזוזות המהירות, שינויי הסקאלה והזווית שממנה אנחנו רואים את הרחפן. ניתן לראות שבזכות המסווג העוקב גם מתמודד היטב עם הרקע המשתנה, וכל הסרטונים בהם העוקב נכשל לחלוטין (17,21,23,24) הם כאלה שהרחפן מתחיל קטן ועל רקע בעייתי, כך שהעקיבה נכשלת מיד בפריימים הראשונים לפני שהייתה למידה משמעותית של המסווג להפרדה בין רחפן לרקע.



דוגמה לפעולת MILTrack: העוקב חוזה את מיקום הפנים בפריים הבא, לוקח תמונות בסביבת האובייקט ומתייחס אליהן כ"שק" חיובי X_1 שבו צריך להופיע האובייקט. כל דוגמה שלילית היא ב"שק" משל עצמה.

האלגוריתם הממומש ב Matlab – דיאגרמת בלוקים



שונויות מימוש בין ה Matlab ל Python

תיקון באג במימוש ה Matlab של חיתוך patch

ב vec_for_imcrop הלוגיקה של חיתוך patch מהתמונה מכוונת להגיע לתמונה בגודל זוגי (נניח 100X100) כך שהמרכז הוא הפיקסל הראשון אחרי האמצע (אם ה patch הוא 100X100 אז האמצע הוא ב [51,51]).

התנאי שבודקים הוא:

```
if center_point(1)-ceil(size_to_crop(1)/2) < 0
    size_to_crop(1)=size_to_crop(1)-abs((center_point(1) -
ceil(size_to_crop(1)/2))*2) ;
end
```

והוא לא מתחשב במקרה ש `center_point(1)-ceil(size_to_crop(1)/2) = 0`

למשל, עבור patch בגודל 100X100 ונקודת אמצע של [50,50] בתמונה המקורית, הקוד יחתוך בפועל patch של 99X99 שבו נקודת המרכז נמצאת בפיקסל הראשון לפני האמצע, וזה גורם לסטייה קלה בהמשך האלגוריתם.

רחפן בשולי התמונה

הקוד ב Matlab לא תומך במצבים שבהם הרחפן נמצא בשולי התמונה. הדבר הראשון שקורס הוא הניסיון להכניס תמונה בגודל קטן מהצפוי לתור `ref_img_cropHistory_2`, שגורם לשגיאה.

נקודות עניין לפי Harris Corner Detector

א. קוד ה Matlab מממש בעצמו את אלגוריתם ה Harris ומחשב Noble corner measure:

$$M_c' = \frac{\det(A)}{trace(A)+\epsilon} \quad , \quad A = \sum_{x,y} w(x,y) \cdot \begin{pmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{pmatrix}$$

ובpython השתמשתי בפונקציה מוכנה של opencv שמשתמשת במידה שהוגדרה במאמר המקורי של Harris&Stephens, http://www.bmva.org/bmvc/1988/avc-88-023.pdf:

$$M_c = \det(M) - k\left(trace(M)\right)^2$$

הפרמטרים לפונקציה נבחרו כדי להיות דומים ככל הניתן למימוש ב Matlab:

1. ה Matlab מחשב גרדיאנט ואחר כך מעביר בפילטר גאוסי עם $\sigma = 1$. המימוש של הספרייה ב Python משתמש בפונקציה cv2.Sobel() שמשלבת את הגרדיאנט ואת הפילטור הגאוסי. לכן נבחר הפרמטר HARRIS_SOBEL_K_SIZE = -1, שמשתמש ב kernel בגודל 3X3 שמקביל במובן מסוים ל $\sigma = 1$.
2. הפרמטר החופשי לרגישות k נקבע לערך של 0.04 שהוא ערך מקובל לפרמטר, אבל אין לו מקבילה בקוד ה Matlab.
3. הערך של HARRIS_BLOCK_SIZE = 2 שמתייחס לגודל החלון ב Harris הוא מינימלי, כדי שכמו במימוש ב Matlab לא נפסול פינות שקרובות זו לזו.

ב. קוד הMatlab משתמש בפונקציה nonmaxsuppts שלא קיימת ב python ומימשתי בעצמי, תפקוד שתי הפונקציות זהה עד כדי המימושים של פונקציית dilate ב python לעומת imdilate ב Matlab, שמתנהגות בצורה דומה עבור מספר תמונות שבדקתי.

נקודות עניין לפי תמונת הפרשים

א. רגיסטרציה – במטלב הרגיסטרציה מתבצעת ע"י הפונקציה imregcorr שמבוססת על phase correlation ומחשב טרנספורציה מסוג similarity (4 דרגות חופש. יודע למדל הזזה, סיבוב ומתיחה). קוד ה python משתמש בשיטה אחרת - ECC, המתוארת במאמר:
http://xanthippi.ceid.upatras.gr/people/evangelidis/george_files/PAMI_2008.pdf
אין אפשרות לנסות לחשב similarity, אז ננסה לחשב התמרה אפינית (6 דרגות חופש).
לאלגוריתם יש פרמטר של מספר איטרציות שמשפיע מאוד על הביצועים וזמני הריצה הכוללים. קבעתי אותו על 50 לפי תחושה בתור פשרה בין ביצועים לזמן ריצה.

ב. לצורך הורדת אובייקטים קטנים משתמשים ב Matlab ב()bwareopen ובמימוש ה python השתמשתי בפונקציה morphology.remove_small_objects מהספריה skimage שמבצעת את אותה פעולה מורפולוגית (open) עם אותם פרמטרים:
מורידים כל רכיב עם פחות מ min_size = 3 פיקסלים כאשר גם פיקסלים באלכסון נחשבים מחוברים (connectivity = 8, שזה גם ה default ב Matlab).

חישוב descriptors עבור נקודות עניין

בMatlab משתמשים ב SURF, וב Python אני משתמש ב ORB שהוא ה descriptor היחיד שניתן להשתמש בו ללא רשיון - http://www.willowgarage.com/sites/default/files/orb_final.pdf

התאמת נקודות עניין ל track קיים

עבור כל track, מנסים לראות האם נקודת עניין חדשה מתאימה אליו. ההתאמה מבוססת על שקלול בין שני מדדים, כאשר הנקודה עם המדד הכולל המינימלי מתאימה ל track. שני המדדים הם:

a. המרחק של הנקודה החדשה מהמיקום הצפוי של הרחפן
b. המרחק בין ה descriptors של הנקודה החדשה לעומת הנקודה האחרונה ב track

ב Matlab השקלול הוא:

```
feature_dis = sqrt(sum((features-repmat(Track_features',
size(features,1),1)).^2,2))' + dis/window;
```

כלומר חיבור של המרחק ב SURF עם המרחק בפיקסלים מחולק בגודל GR (אם יש עקיבה רציפה גודל הפרמטר הזה הוא 10). אחרי החלוקה, הגודל dis/window הוא בדרך כלל מספר בין 0 ל10.

המרחק ב SURF הוא מספר קטן מאוד (בדרך כלל קטן מ1), ואותו חשבון עבור המרחק ב ORB נותן מספר גבוה בהרבה (בדרך כלל עשרות עד מאות). כך יוצא שאם נשארים עם אותה נוסחה, הקריטריון של המרחק בין הנקודות לא ישפיע כלל על העקיבה.

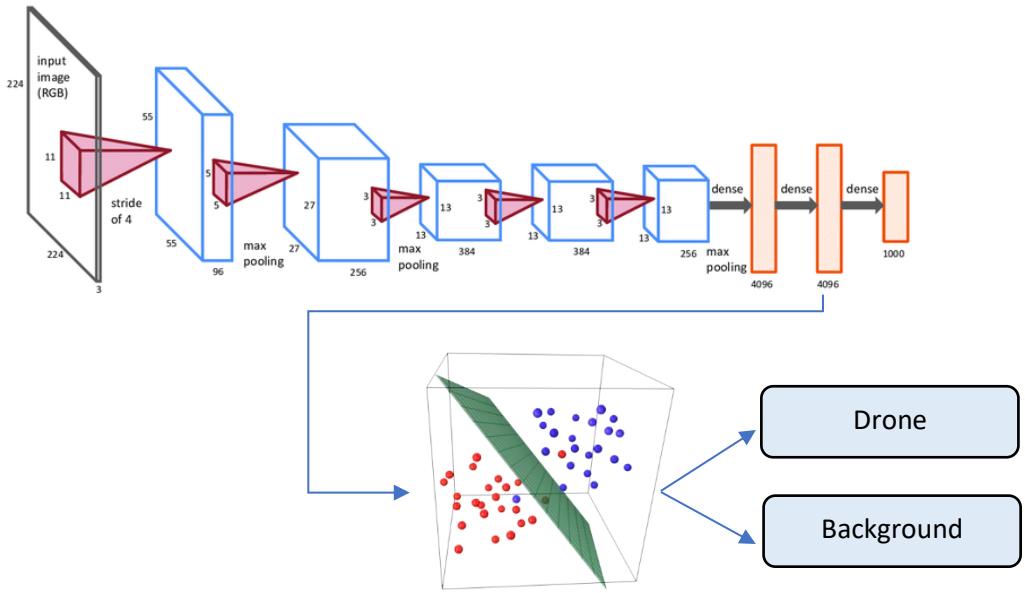
כדי לאזן את ההשפעה של שני המדדים, חילקתי את המרחק ב ORB ב512, וזה הביא את המדד לגודל סביר.

```
dis = np.sqrt((x_cg_list - x_p) ** 2 + (y_cg_list - y_p) ** 2)

dis_normalized = dis / window

orb_dis = np.sqrt(np.sum((features - track_features) ** 2, 1))

orb_dis_normalized = orb_dis / 512

feature_dis = orb_dis_normalized + dis_normalized
```

כעת גם הסף עבור בדיקה למרחק מינימלי כבר לא תקף (בגלל שסוכמים גדלים שונים), אך בהיעדר בחירה חכמה יותר השארתי אותו בערך המקורי min_dist = 2.

אימון מסווג SVM להפרדה בין רחפן לרקע עם שימוש ב Alexnet

הרשת alexnet שהשתמשתי בה ב python היא בעלת אותו מבנה כמו הרשת ב Matlab אך אומנה על דוגמאות שונות, ולכן המשקולות שלה שונים מעט מהרשת ב Matlab.
לא מתועד באילו דוגמאות השתמשו כדי לאמן את המסווג ב Matlab, אז הוצאתי ע"י העוקב בMatlab תמונות של רחפנים (מהפריימים שבהם הוא עוקב בצורה תקינה) ורקע (מהפריימים שבהם הוא לא עוקב בצורה תקינה) ואימנתי עליהן.
גודל ה Database שנוצר הוא 4614 תמונות רחפנים ו 6874 תמונות רקע, והן חולקו ל60% תמונות ל train ו 40% תמונות ל test.
בפועל, למרות שכשבודקים על ה test set מקבלים score של 99.4%, הדיוק של ההחלטה רחפן/רקע בסרטונים עצמם נראה בינוני, וזה נובע לדעתי מ overfitting.
בגלל שלקחנו הרבה תמונות רצופות מהעוקב ב Matlab, סביר שעבור רוב התמונות ב test set יש תמונה שדומה להן מאוד ב train set, כך שלמרות שלמסווג אין ביצועים טובים בהפרדה בין רחפנים לרקע אנחנו מקבלים score גבוה.



השוואת ביצועים

ביצועי עקיבה

השוואה על סרטונים מהקובץ GOPR0014.mp4 שמופיעים לפי אותו מספר בGetVideoInfo.m ב Matlab ובקובץ video_info.py ב Python.

מספר סרטון	האם עוקב ב Matlab?	האם עוקב ב python?	האם MILTrack עוקב?
1	כן	כן	כן
2	כן	כן	כן
3	כן	כן	כן
4	כן	כן	כן
5	כן	כן	כן
6	כן	כן	כן
7	כן	כן	כן
8	כן	כן	כן
9	כן	כן	כן
10	כן	מאבד אחרי חצי סרטון	כן
11	כן	כן	כן
12	כן	לא	כן
13	כן	כן	כן
14	לא רץ	כן	כן
17	לא	לא	לא
18	כן	כן	כן
19	לא	לא	כן
20	כן	לא	כן
21	כן	כן	לא
22	כן	לא	כן
23	לא	לא	לא
24	לא	לא	לא
25	כן	כן	כן
26	לא רץ	מאבד אחרי חצי סרטון	מאבד אחרי חצי סרטון
27	כן	כן	כן

ניתן לראות שביצועי האלגוריתם לא השתנו משמעותית בין Matlab למימוש ב Python. התוכנה מעט יציבה יותר ולא קורסת, ומצד שני בגלל הביצועים הפחות טובים של הרשת מאבדים את הרחפן יותר פעמים במהלך הסרטון. באופן כללי, MILTrack עוקב טוב ויציב יותר משני המימושים למרות שהוא לא הותאם ייעודית למעקב אחרי רחפנים, וישנם מספר סרטונים קשים במיוחד שכל האלגוריתמים נכשלים בהם – 17,23,24,26.

ביצועי זמן ריצה

זמני ריצה ב Matlab

בהרצה של סרטון מספר 1 שאורכו כ-5 שניות, הלולאה המרכזית של הקוד רצה 206 שניות עבור 345 פריימים, כלומר ב 1.68 FPS. תוצאת ה Profiler של Matlab לפונקציות הכי איטיות בקוד:

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
fft2	2422	38.495 s	38.495 s	
ifft2	1038	17.472 s	17.472 s	
insertMarker	1694	15.384 s	11.153 s	
CompactSVMImpl>iDispatchPredict	348	10.867 s	10.867 s	
VideoWriter>VideoWriter.writeVideo	694	13.405 s	7.918 s	
stridedConv	1044	7.615 s	7.615 s	
images\private\phasecorr	346	42.312 s	7.259 s	
...veForTranslationGivenScaleAndRotation	346	29.992 s	6.862 s	
...t;ConvolutionReLUHostStrategy.forward	696	5.484 s	5.484 s	
imregcorr>getFourierMellinSpectra	346	20.388 s	4.433 s	
imregcorr>createBlackmanWindow	1730	4.397 s	4.397 s	
extractFeatures>extractSURFFeatures	346	5.505 s	4.125 s	

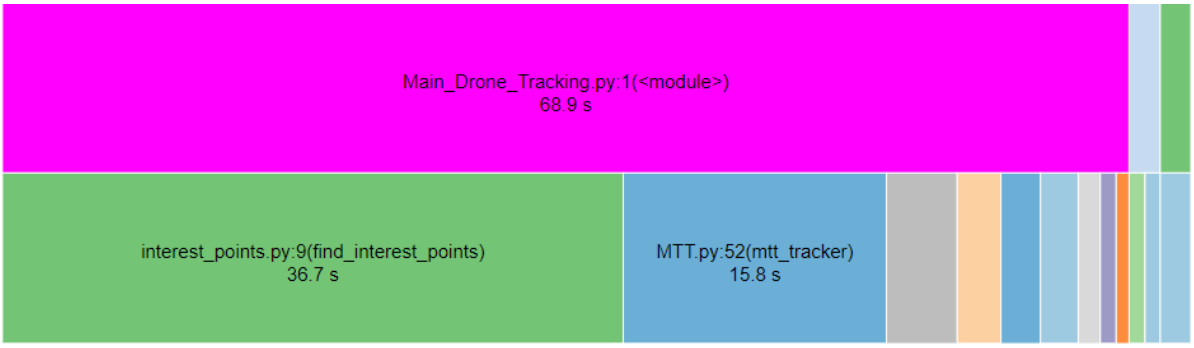
וניתן לראות שיש שני תהליכים מרכזיים שמגבילים את קצב הריצה (מסומנים בצבע).

חישוב ההתמרה לצורך רגיסטרציה לוקח לבדו 102.6s, **חצי מזמן הריצה הכולל.**

תהליך הסיווג רחפן / רקע בעזרת Alexnet ו SVM גם הוא יקר בזמן, ולוקח כ-40s (מתוכם בערך 30% מהזמן על המסווג ו 70% על הרשת).

זמני ריצה ב Python

בהרצה של סרטון מספר 1 שאורכו כ-5 שניות, הלולאה המרכזית של הקוד רצה 64 שניות עבור 345 פריימים, כלומר ב 5.4 FPS. חלוקת הזמנים בקוד מתוך CProfile:



וניתן לראות שגם כאן יש שני תהליכים מרכזיים שמגבילים את קצב הריצה.

תהליך מציאת נקודות עניין הוא היקר ביותר ולוקח לבדו 36.7s, **כ55% מזמן הריצה הכולל.**

תהליך סידור ה tracks שכולל את הסיווג רחפן / רקע בעזרת Alexnet ו SVM גם הוא יקר בזמן ולוקח 15.8s, **כ25% מזמן הריצה הכולל**.

נסתכל על רשימת הפונקציות היקרות ביותר בזמן כדי להבין בדיוק מה הפעולות שמעכבות אותנו:

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
345	35.68	0.1034	35.68	0.1034	~:0(<findTransformECC>)
392	13.81	0.03523	13.81	0.03523	~:0(<method 'forward' of 'cv2.dnn_Net' objects>)
345	4.158	0.01205	4.158	0.01205	~:0(<method 'write' of 'cv2.VideoWriter' objects>)
345	2.588	0.007502	2.588	0.007502	~:0(<method 'compute' of 'cv2.Feature2D' objects>)
346	2.338	0.006758	2.338	0.006758	~:0(<method 'read' of 'cv2.VideoCapture' objects>)
730	1.295	0.001774	1.295	0.001774	~:0(<method 'copy' of 'numpy.ndarray' objects>)
392	1.258	0.003208	1.258	0.003208	~:0(<sklearn.svm.libsvm.predict>)

וניתן לראות שיש שתי פעולות בסיסיות שהן יקרות במיוחד:

1. findTransformECC

כחלק מתהליך מציאת נקודות עניין מתמונת ההפרשים, זו הפונקציה שמוצאת התמרה אפינית לצורך רגיסטרציה בין פריימים עוקבים. הפונקציה הזו נקראת פעם אחת לכל פריים, ולוקחת כ 55% מזמן הריצה של האלגוריתם – 0.1s לקריאה. זה מגביל לבדו את ביצועי זמן הריצה ל 10 FPS.
כחלק מהפעלת האלגוריתם אפשר לבחור פרמטר של מספר איטרציות והוא המשפיע המרכזי על זמן הריצה, ונבחר כרגע לערך של 50 בתור פשרה בין זמן ריצה לביצועים.
לצורך השוואה, אם מורידים את הפרמטר הזה להיות 10, זמן ריצת הלולאה המרכזית הוא 34s, כלומר ביצועים של 10.1 FPS וחיסכון גדול של זמן - 30s. ביצועי העקיבה נפגעים מעט בגלל שינוי הפרמטר ומאבדים את הרחפן בשני סרטונים – 3 ו 11.

2. מעבר ב Alexnet

העברת התמונות דרך alexnet כדי להוציא features למסווג svm היא פעולה יקרה יחסית, שלוקחת (בהרצה על ה cpu שלי) כ 33ms בכל קריאה ויכולה להתבצע מספר פעמים בכל פריים. בהנחה של 4 קריאות בממוצע לפריים (זה בערך היחס על פני כל הסרטונים שנבדקו), המעבר ברשת לבדו מגביל אותנו לאיזור 8 FPS. במידה והמעבר ברשת ירוץ מקבילית על GPU, הביצועים יכולים להשתפר משמעותית.

סיכום והמלצות

הייעול ומימוש האלגוריתם מחדש ב Python מאיץ **בלפחות פי 5** את זמן הריצה ומקרב אותנו לאפשרות לרוץ ב Real-Time. המעבר ל Python מאפשר מעבר פשוט בין פלטפורמות ומערכות הפעלה.

הדבר המרכזי שמעכב כרגע את האלגוריתם (גם במימוש ה Matlab וגם ב Python) הוא מציאת ההתמרה שנדרשת לרגיסטרציה בין תמונות עוקבות, דבר הכרחי לצורך שימוש בתמונת הפרשים. בחירת חלופה מוצלחת יותר למציאת נקודות עניין בתמונה תאפשר האצה משמעותית של האלגוריתם וריצה בזמן אמת.

כחלק מבחינת החלופות ב Python, נמצא כי העוקב MILTrack מתאים לבעיה שלנו ויש לו ביצועים יותר טובים גם מבחינת עקיבה וגם מבחינת זמני ריצה.

כדי לשפר אפילו עוד יותר את הביצועים, אפשר לקחת את הרעיון של MILTrack ולהתאים אותו לעקיבה אחרי רחפנים בתרחיש שלנו:

1. הוספת לוגיקת MTT ושמירת כמה Tracks בכל רגע נתון.
2. ניצול העובדה שאנחנו עוקבים אחרי רחפנים בלבד ושימוש במסווג SVM שאימנו ייעודית לרחפנים.

נספח – הוראות התקנה

התקנת Pycharm

1. ההוראות נבדקו עם גרסה JetBrains PyCharm Community Edition 2019.1.1.
2. קישור להורדה:
https://www.jetbrains.com/pycharm/download/#section=windows

העתקת קבצים מוכנים

יש להעתיק למחשב את התיקיות הבאות:

1. drone_tracker_3_5 – תיקיית הפרויקט מ Pycharm
2. bvlc_alexnet – תיקייה עם קונפיגורציות ומודל caffe של הרשת
3. GOPR0010.MP4, GOPR0014.MP4 – סרטוני רחפנים

התקנת Anaconda

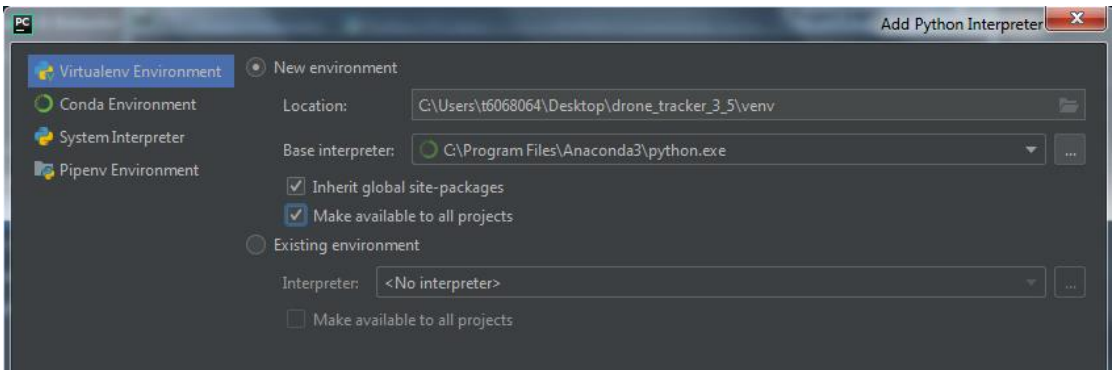
יש להתקין Anaconda עם python 3.5

1. כניסה לארכיון גרסאות Anaconda:
https://repo.continuum.io/archive/
2. הורדת גרסה עם python 3.5 עבור windows 64 bit:
Anaconda3-4.2.0-Windows-x86_64.exe
3. בסיום ההתקנה, יש להוסיף את תיקיית Anaconda3 ל Environment Variables.

הגדרת Python Interpreter לפרויקט

1. פתיחת drone_tracker_3_5 כתיקיית פרויקט ב Pycharm
2. קנפוג python interpreter ב pycharm
   a. File -> Settings -> Project Interpreter -> Show All -> +
   b. צור venv חדש שמשתמש ב Anaconda.

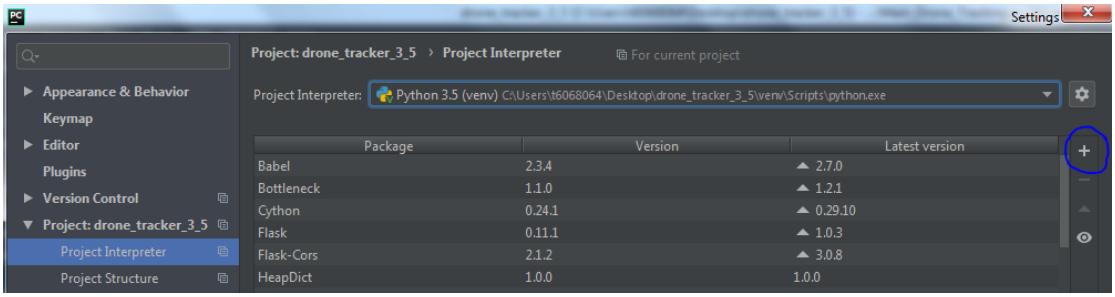
רוב הסיכויים ש Anaconda לא יופיע ברשימה של Base interpreter, במקרה כזה צריך להכניס ידנית את ה Path של python.exe בתיקיית Anaconda.



c. אם אין אפשרות ללחוץ OK, יש למחוק את התיקייה venv ולחזור על שלבים a,b.

התקנת ספריות

1. ב Project Interpreter לחץ על +



2. התקן את החבילות הבאות לפי סדר הופעתן, יש להקפיד על הגרסה המדויקת:

Package	Version
opencv-python	3.4.5.20
joblib	0.13.2
numpy	1.11.3
scipy	1.2.1
scikit-image	0.13.1
sklearn	0.0
scikit-learn	0.20.3
matplotlib	1.5.1
Pillow	2.9.0

התקנת Caffe

יש לעבוד לפי ההוראות המופיעות ב: https://github.com/BVLC/caffe/tree/windows

התקנת Visual Studio 2015

1. לבחור 2015 ב https://visualstudio.microsoft.com/vs/older-downloads/
2. להכניס משתמש Microsoft או ליצור חדש במידת הצורך
3. הורדת Visual Studio Community 2015
4. במהלך ההתקנה – חשוב לסמן C++ בLanguages.

הסיבה שצריך Visual Studio היא הקומפיילר, והוא לא מותקן ב default.

אם לא סימנת C++ בהתקנה המקורית, צריך להפעיל שוב את ההתקנה, לסמן Modify ולהוסיף את C++.

התקנת CMake

1. הקישורים להתקנות נמצאים ב https://cmake.org/download/
2. לבחור ב Windows win64-x64 Installer, למשל cmake-3.14.5-win64-x64.msi

התקנת Git

1. קישור ישיר להורדה https://git-scm.com/download/win

שינויים ב scripts\build_win.cmd

1. WITH_NINJA = 0
2. PYTHON_VERSION = 3
3. CONDA_ROOT = < your conda root, e.g C:\Program Files\Anaconda3>

העתקת קבצי מודל Alexnet

1. יש להחליף את התיקייה caffe/models/bvlc_alexnet בתיקייה המוכנה bvlc_alexnet
2. יש לוודא שהקבצים deploy_fc7.prototxt ו- bvlc_alexnet.caffemodel קיימים

שינויים נדרשים בקוד ה Python

1. Main_Drone_Tracking.py, שורות 21-22, עדכון נתיבים של קבצי המודל של הרשת

```
# load alexnet serialized model from disk
prototxt_alexnet_fc7 = r"C:\Users\t6068064\caffe\models\bvlc_alexnet\deploy_fc7.prototxt"
model_alexnet = r"C:\Users\t6068064\caffe\models\bvlc_alexnet\bvlc_alexnet.caffemodel"
```

2. video_info.py, עדכון נתיבי הסרטונים

```
    switcher = {  # [start_frame, end_frame, tracking_point, video_filename]
        # GOPR0014.MP4
        1: [10258, 10604, [130, 1739], "GOPR0014.MP4"],
        2: [10499, 10799, [419, 1265], "D:/MSc_Project/Drone_Movies_Raw/GOPR0014.MP4"],
        3: [15999, 16349, [617, 844], "D:/MSc_Project/Drone_Movies_Raw/GOPR0014.MP4"],
        4: [17099, 17449, [620, 1036], "D:/MSc_Project/Drone_Movies_Raw/GOPR0014.MP4"],
```

3. אופציונאלי: crop_video.py, שינוי נתיבים לקובץ כניסה ויציאה

```
# Movie to crop and first frame to be displayed
input_video = 'GOPR0014.mp4'
cap = cv2.VideoCapture(input_video)
start_frame = 10259

# output movie setup
output_video = 'cropped_drone_videos/' + os.path.splitext(input_video)[0] + time.strftime("_%d%m%Y_%H%M%S") + '.mp4'
fourcc = cv2.VideoWriter_fourcc(*'MP4V')
out = cv2.VideoWriter(output_video, fourcc, 20.0, (1920, 1080))
```

4. אופציונאלי: train(/test)_svm_drone_tracker.py, שינוי נתיב לתיקיות Database

```
drones_directory = r"C:\Users\roiei\PycharmProjects\opencv_alexnet\drones_db\Raw_Data_Drones"
drone_images_paths = list()
for (dirpath, dirnames, filenames) in os.walk(drones_directory):
    drone_images_paths_to_add = [os.path.join(dirpath, file) for file in filenames]
    drone_images_paths += drone_images_paths_to_add

backgrounds_directory = r"C:\Users\roiei\PycharmProjects\opencv_alexnet\drones_db\Raw_Data_Background"
background_images_paths = list()
for (dirpath, dirnames, filenames) in os.walk(backgrounds_directory):
    background_images_paths_to_add = [os.path.join(dirpath, file) for file in filenames]
    background_images_paths += background_images_paths_to_add
```

נספח – כלי לאימון מסווג SVM להפרדה בין רחפן לרקע משכבה fc7 ב Alexnet

בתיקייה drone_tracker_svm_classifier יש שני סקריפטים שימושיים, אחד ל train ואחד ל test, כדי להקל על מגבלות הזיכרון הנדרשות לטעינת כל התמונות ופעולות עליהן.

השינויים שיש לעשות בקוד לפני שמשתמשים בו לאימון ובדיקה של מסווג חדש:

בקובץ train_svm_drone_tracker.py

1. שינוי הנתיב ל Database שמכיל את תמונות הרחפנים והרקע

```
drones_directory = r"C:\Users\roiei\PycharmProjects\opencv_alexnet\drones_db\Raw_Data_Drones"
drone_images_paths = list()
for (dirpath, dirnames, filenames) in os.walk(drones_directory):
    drone_images_paths_to_add = [os.path.join(dirpath, file) for file in filenames]
    drone_images_paths += drone_images_paths_to_add

backgrounds_directory = r"C:\Users\roiei\PycharmProjects\opencv_alexnet\drones_db\Raw_Data_Background"
background_images_paths = list()
for (dirpath, dirnames, filenames) in os.walk(backgrounds_directory):
    background_images_paths_to_add = [os.path.join(dirpath, file) for file in filenames]
    background_images_paths += background_images_paths_to_add
```

2. שינוי הנתיב לקבצי המודל של Alexnet ב Caffe

```
prototxt_alexnet_fc7 = "C:/Users/roiei/Desktop/caffe/caffe/models/bvlc_alexnet/deploy_fc7.prototxt"
model_alexnet = "C:/Users/roiei/Desktop/caffe/caffe/models/bvlc_alexnet/bvlc_alexnet.caffemodel"
```

3. שינוי הנתיב למודל המאומן (חשוב כדי לא לדרוס בטעות את המודל שבשימוש)

```
dump(clf, 'alexnet_svm.joblib')
```

בקובץ test_svm_drone_tracker.py

1. שינוי הנתיב ל Database שמכיל את תמונות הרחפנים והרקע

```
drones_directory = r"C:\Users\roiei\PycharmProjects\opencv_alexnet\drones_db\Raw_Data_Drones"
drone_images_paths = list()
for (dirpath, dirnames, filenames) in os.walk(drones_directory):
    drone_images_paths_to_add = [os.path.join(dirpath, file) for file in filenames]
    drone_images_paths += drone_images_paths_to_add

backgrounds_directory = r"C:\Users\roiei\PycharmProjects\opencv_alexnet\drones_db\Raw_Data_Background"
background_images_paths = list()
for (dirpath, dirnames, filenames) in os.walk(backgrounds_directory):
    background_images_paths_to_add = [os.path.join(dirpath, file) for file in filenames]
    background_images_paths += background_images_paths_to_add
```

2. שינוי הנתיב לקבצי המודל של Alexnet ב Caffe

```
prototxt_alexnet_fc7 = "C:/Users/roiei/Desktop/caffe/caffe/models/bvlc_alexnet/deploy_fc7.prototxt"
model_alexnet = "C:/Users/roiei/Desktop/caffe/caffe/models/bvlc_alexnet/bvlc_alexnet.caffemodel"
```

3. שינוי הנתיב של המודל לבדיקה

```
print("loading SVM classifier...")
start = time.time()
clf = load('alexnet_svm.joblib')
end = time.time()
print("loading SVM classifier took {:.5} seconds".format(end - start))
```

יש להריץ את האימון ואחריו את הבדיקה.

תוצאות אופייניות על 4614 תמונות רחפנים ו 6874 רקע (מתוכן 60% אימון ו 40% בדיקה):

הרצת train_svm_classifier.py

```
reading train images...
reading train images and converting to blob took 33.606 seconds
loading model...
train data forward propagation...
forward propagation to get train data fc7 activations took 324.33 seconds
training SVM...
training SVM took 129.63 seconds
saving trained SVM...
```

הרצת test_svm_classifier.py

```
reading test images...
reading test images and converting to blob took 16.48 seconds
loading model...
test data forward propagation...
forward propagation to get test data fc7 activations took 170.23 seconds
loading SVM classifier...
loading SVM classifier took 0.56684 seconds
calculating score over test data...
score over test data is:  0.9941253263707572
calculating score over test data took 13.931 seconds
```

מבנה ה Database

במידה ורוצים להשתמש בתמונות אחרות של רחפנים ורקע, יש לשמור על אותו פורמט: