פרויקט גמר במחשבים 5 יח"ל

TelloVision

נושא: טייס אוטומטי לרחפן באמצעות ראייה ממוחשבת ורשתות נוירונים

קישור לקוד מקור של הפרוייקט בגיטהאב: https://github.com/Kafow/TelloVision

שם התלמיד: אופק כפיר

תעודת זהות: 323821991

כיתה: י"ב 3

שם המנחה: יהודה אור

תוכן עניינים:

חלק

תיאורטי

מבוא

נושא העבודה

רחפנים הינם כלי טיס בלתי מאויישים קטנים אשר יוצרים עילוי באמצעות סיבוב להבים בדומה למסוק. רחפנים משמשים כיום בתעשייה למגוון רחב של תחומים כגון: רפואה, אבטחה, צילום, משלוחים, חקלאות וכו'. בשל העלות הנמוכה של רחפנים כיום והמהירות הרבה שבה ניתן לבצע משימות איתם ישנו ביקוש גובר לרחפנים ופתרונות חדשניים באמצעותם אשר יפתרו בעיות רבות שלא היו ניתנות לפתירה עד היום.

בעבודתי פיתחתי אלגוריתם מבוסס Neural networks אשר עוזר לרחפן לזוז בעצמו ללא התערבות אנושית באמצעות המצלמה אשר נמצאת בתוכו, ובאמצעות כך להוות בסיס לפיתוחים עתידיים אשר יעזרו לפתירות בעיות רבות.

לאחר ביצוע הפרויקט אני יכול להגיד שאני גאה בתוצאה ומוכן להציג אותה.

מטרת העבודה

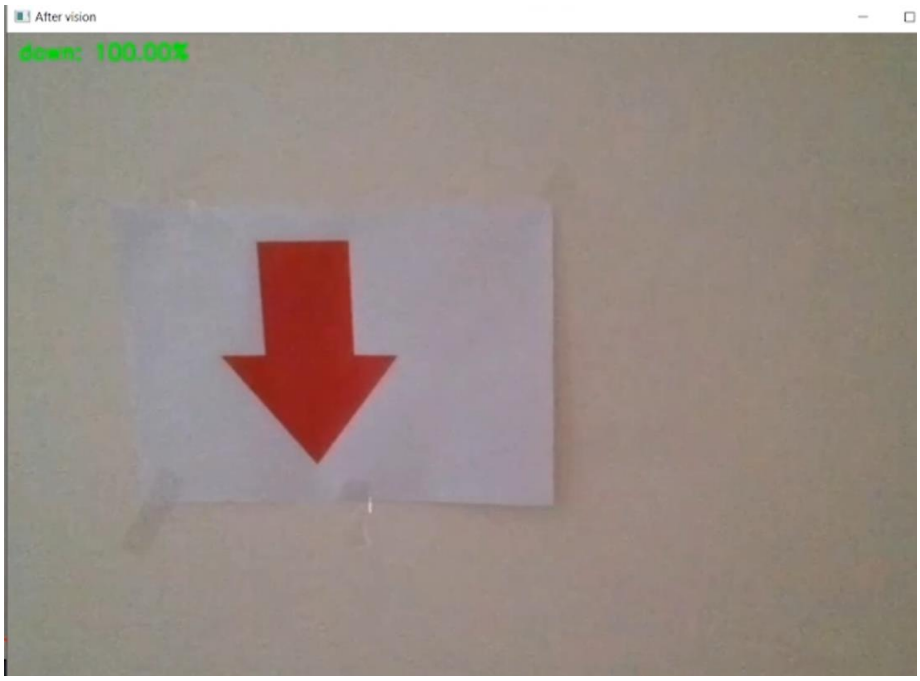
מטרת העבודה הייתה להעשיר את הידע שלי בתחום למידת המכונה, רשתות הנוירונים, תכנות, אלגוריתמיקה ומתמטיקה בנוסף לידע הקיים בתכנות ומתמטיקה על מנת ליצור פרויקט אשר יבצע את משימתו בצורה הטובה ביותר.

במהלך הפרויקט השתמשתי בטכניקת object oriented programming בנוסף ליישום של תכנות פונקציונלי וחלוקה של הקוד למחלקות אשר לכל אחת תפקיד משלה ושמירה של ברירות הקוד באמצעות דוקומנטציה מקיפה בנוסף לעבודה במחשבה שקוד זה צריך להסביר את עצמו לקורא.

למדתי רבות במהלך פרויקט זה ומקווה שאוכל לבטא זאת באמצעות ספר זה.

הצגה ויזאלית של הפרוייקט

<u>זיהוי של החץ למטה</u>



<u>זיהוי של החץ למעלה</u>

זיהוי של החץ ימינה

זיהוי של החץ שמאלה:

קישור לסרטון יוטיוב אשר מראה את הרחפן בזמן פעולה

https://youtu.be/8h6Luors6po

מבוא לתכנות

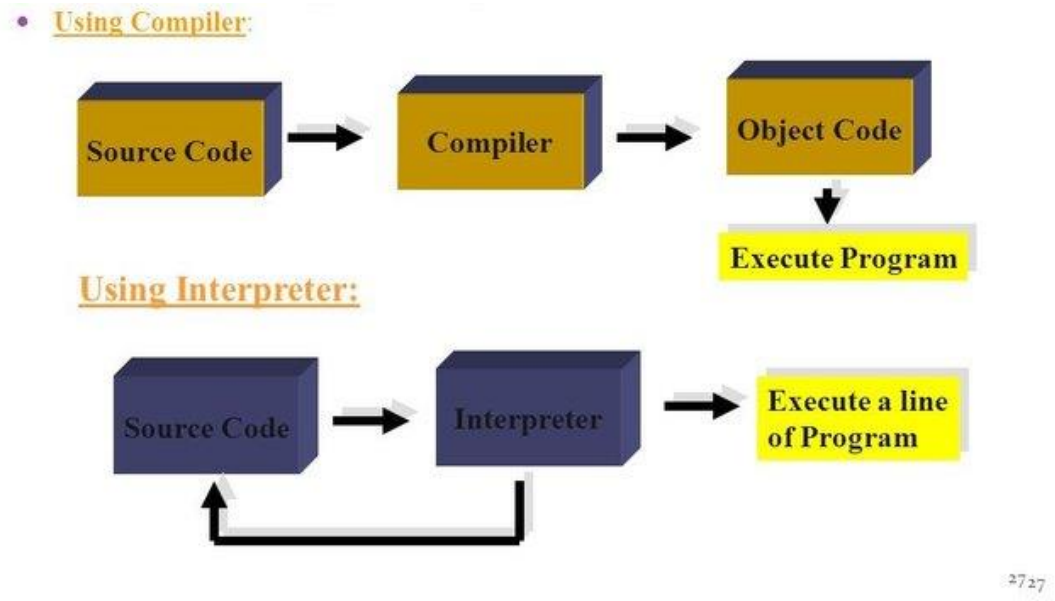
שפת התכנות Python 3

Python הינה שפת תכנות דינמית אשר כיום הנפוצה ביותר בעולם. היא נכתבה בשנת 1991 ושמה לקוח מחבורת הקומדיה "מונטי פייתון".

Python תוכננה ככה שלעומת שפות אחרות היא שמה דגש על עיצוב הקוד וקריאות הקוד (לדוגמה: בלוקים מופרדים באמצעות הזחה ולא באמצעות סוגריים מסולסלים כמו ברוב השפות). השפה תומכת במבני נתונים ובעלת ספרייה סטנדרטית גדולה אשר מאפשרת לתכנת בקלות רבה וביעילות מרבית.

לשפה יש קהילה רחבה של מתכנתים אשר אימצו את השפה ולכן גם ספריות חיצוניות רבות אשר מפשטות את העבודה על המתכנת ודואגות לרוב הדברים הפנימיים. השפה הינה מרובת פרגדימות משמע תומכת גם בתכנות מונחה עצמים בנוסף לתכנות פונקציונלי.

פייתון בשונה משפות אחרות לא מקומפלת ורצה באמצעות Interpreter אשר מפרש את השפה בזמן אמת שורה אחר שורה. בעוד ששפות אחרות ובייחוד C מקמפלות את הקובץ לשפת אסמבלי ואז ממירות את האסמבלי לשפת מכונה אשר אותה המעבד יכול להבין. פייתון ממיר כל שורה בקוד לשפת מכונה בזמן הריצה, דבר זה גורם אמנם להאטה של תוכנית פייתון ביחס לתוכנית C, אך מאפשר שינוי של הקוד בזמן ריצה ולא קימפול מחדש של הקוד כמו בC (דבר שיכול לקחת זמן רב).



בנוסף לכך, פייתון הינה שפה High level משמע שפה אשר לא קרובה לפקודות מכונה. עיקרון זה בא לידי ביטוי בניהול הזכרון, ניהול משתנים, הקצאת זכרון וניקוי אוטומטי של זכרון בזמן הריצה ללא התערבות של המתכנת.

במהלך עבודה זאת תכננתי רק בפייתון וכל התקשורת ולמידת המכונה התבצעו רק בשפה זו.

PEP8

PEP8 – הינו מדריך השפה הרשמי לכתיבת קוד של פייתון. מדריך זה מגדיר כיצד אמור להראות קוד פייתון תקני ומסודר אשר עומד בתקן. יש לציין כי שימוש במדריך זה לא משפיע על ריצת הקוד והוא אינו חובה, אך הוא דואג לסדר בתוך הבלאגן.

לדוגמה: אחד מהחוקים הינו שאחרי Import בתחילת קוד, יש להשאיר 2 שורות ריקות לקוד שיבוא אחרי.

שימוש במדריך זה עוזר להשאיר מבנה קוד אחיד לכל אורך הפרוייקט ולעזור לקורא זר של הקוד להבין אותו בקלות רבה.

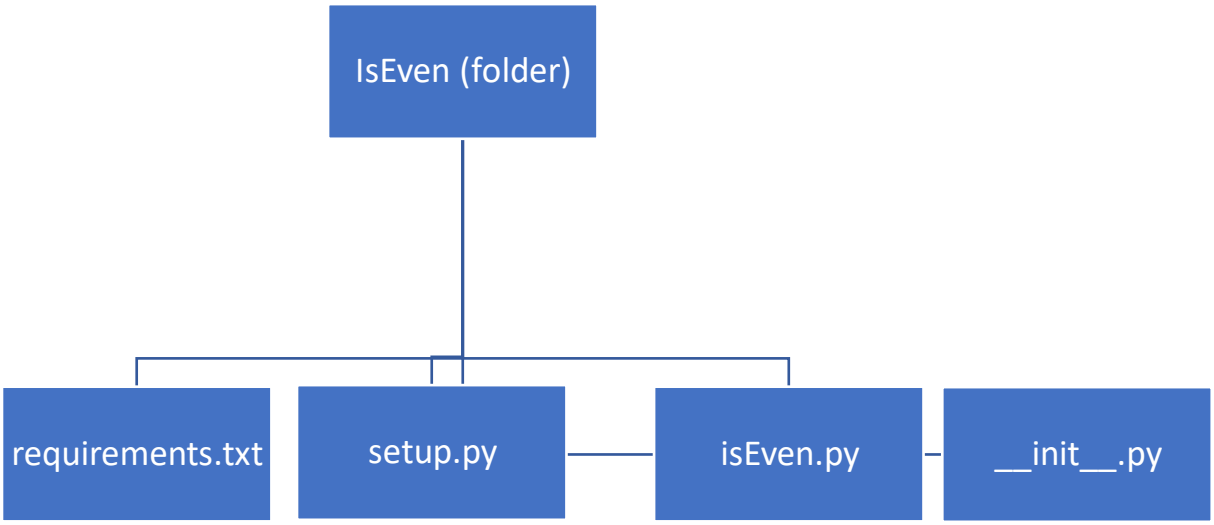
במהלך הפרוייקט השתמשתי רבות במדריך זה ולא חרגתי מחוקיו.

PIP

PIP – הינו מנהל החבילות של פייתון.

בפייתון ניתן ליצור חבילות אשר נקראות Modules ומאפשרות למפתחים לשתף את עבודתם באמצעות ספריה לנוחות מפתחים אחרים.

לדוגמה, אני יכול ליצור ספריה אשר אומרת האם מספר הינו זוגי או אינו זוגי, ואני רוצה לשתף את עבודתי למפתחים אחרים על מנת שיעזרו בספריה שלי לשימושים שלהם. מבנה החבילה יראה ככה:



אסביר על כל קובץ:

- __init__.py - קובץ אשר פייתון מריץ ברגע שמייבאים את הספריה, פה יהיו פונקציות אשר הספריה רוצה שירוצו על מנת לסדר את ה dependencies ועל מנת לייבא את הקוד המרכזי, לדוגמה:

```
from . import isEven
```

- isEven.py – קובץ עם הקוד המרכזי של הספריה, לדוגמה פה זה יהיה:

```
def isEven(number):
    return number % 2 == 0
```

- Setup.py – קובץ setup אשר מציין לpip מה להתקין ברגע שאנחנו מנסים להתקין את החבילה. קובץ זה כולל הוראות התקנה ושם המחבר של החבילה ופרטיו.
לדוגמה:

```
from setuptools import setup

setup(
    name='IsEven',
    version='1.0',
    packages=['math'],
    url='',
    license='MIT',
    author='Ofek Kfir',
    author_email='ofek.kfir@gmail.com',
    description='Package that determines is number is even.'
)
```

- Requirements.txt – קובץ אשר מציין את החבילות אשר עליהן החבילה נתמכת. כל חבילה מופרדת בשורה רווח. לדוגמה אצלנו אנו זקוקים למתמטיקה על מנת לבדוק האם מספר הוא זוגי, ולכן נכלול את math

```
1    math
```

לאחר שביצענו את כל שלבים אלו, ניתן יהיה להעלות את חבילה זאת לאחד משירותי הgit הפופולריים כגון, Github, Gitlab, BitBucket וכד' ומשם להעלות למקור חבילות פייתון הרשמי בשם PyPi אשר מאחסן את כל חבילות הפייתון. לאחר מכן כל מפתח בעולם יוכל להתקין את החבילה שלנו באמצעות הפקודה הפשוטה

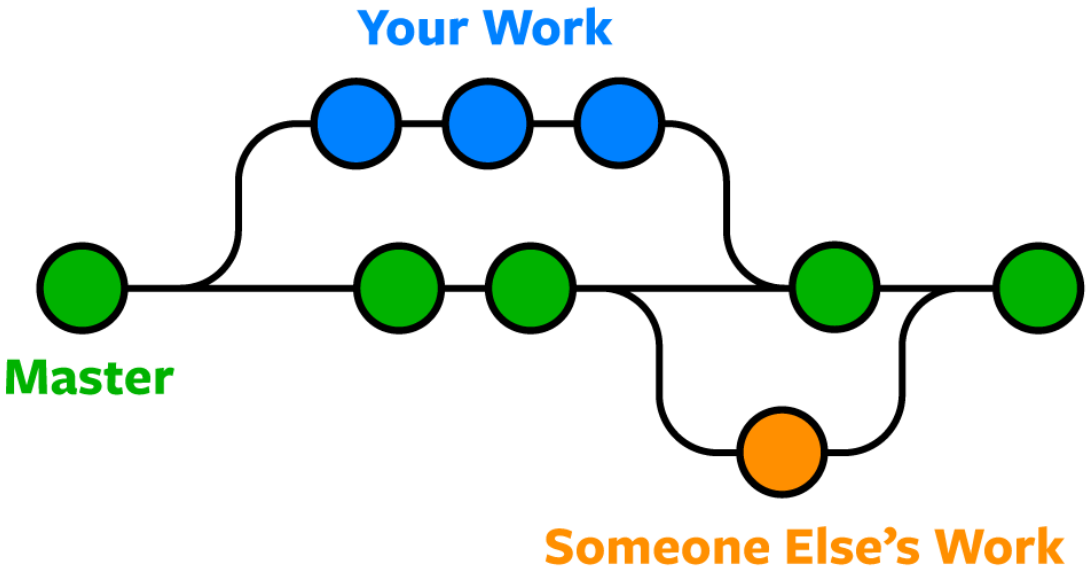
Pip install isEven

Git – Version control system

גיט הינו כלי לניהול גרסאות אשר הומצא על ידי linus Torvalds על מנת לנהל את קוד המקור של קרנל לינוקס. כלי זה משומש כיום בכל התעשייה והינו הכלי העיקרי לניהול קוד מקור של תוכנות.

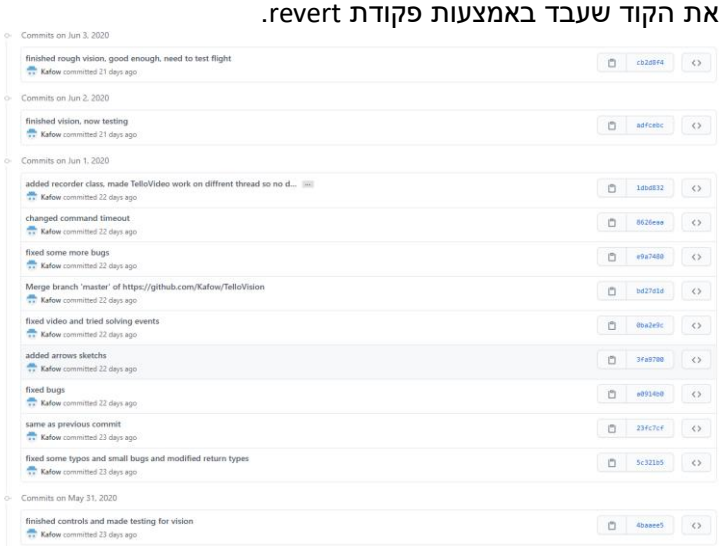
לכלי זה מספר פונקצנליות עיקריות אשר הופכות אותו לכלי העיקרי המשומש בתעשייה:

- יצירת branch – באמצעות גיט ניתן ליצור ענפים שמנהלים פיצרים חדשים ולדחוף אותם לבראנצים אחרים ובכך לנהל גרסאות שונות לאותו מוצר על מנת לא לפגוע במוצר המקורי



דוגמה לניהול ענפים

- Commits – הינן הודעות אשר המפתח רושם בכל שינוי לקובץ והן מסמלות את השינויים שהיו לקובץ. ניתן גם לחזור לקומיטים אחרונים במידה והגרסה החדשה אינה טובה ולשחזר את הקוד שעבד באמצעות פקודת revert.



דוגמה לקומיטים כפי שמוצגים באתר גיטהאב על פרויקט זה.

כיום האתר המוביל לאחסון Repositories הינו github אשר מאחסן מגוון רחב של פרוייקטים של קוד פתוח ומאחסן גם את פרויקט זה בכתובת https://github.com/Kafow/TelloVision.

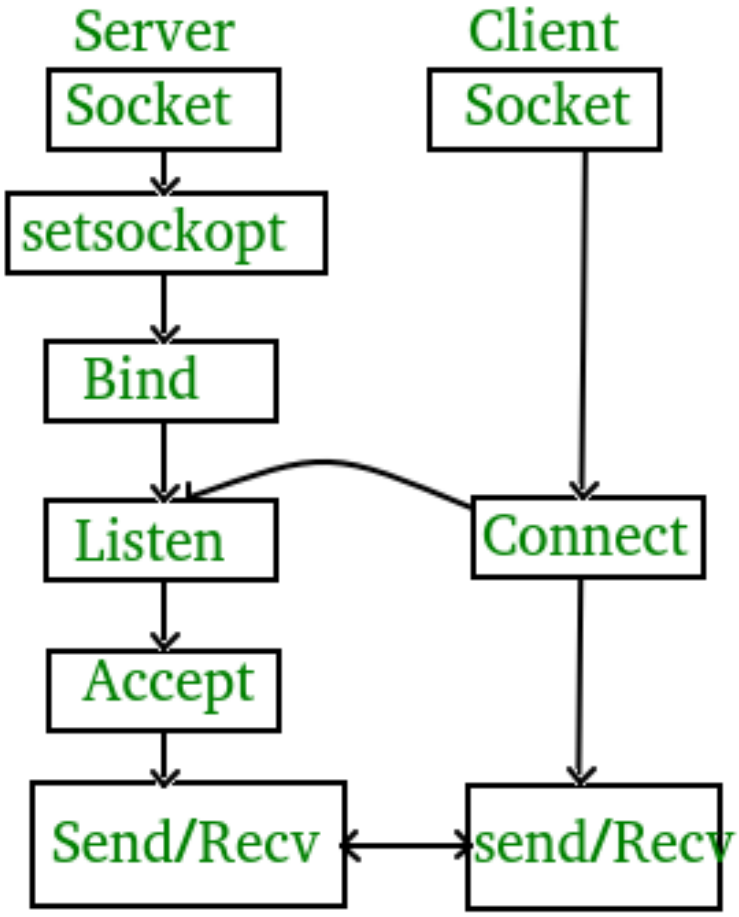
רשתות

Socket

סוקט (שקע) הינו שכבת קבלה והעברה של מידע אשר מחכה במחשב לפקודות על גבי פורט מסוים וגם מעבירה באמצעות הפורט הזה פקודות.

ספריית הסוקט (שקע) בפייתון הינה ספרייה שנועדה על מנת להיות interface בין Socket API של המערכת הפעלה לבין פייתון. באמצעות ספריה זו ניתן לפתוח סוקטים בתוך המערכת ובאמצעותם לקבל ולשלוח מידע על גבי הרשת בין הרחפן למחשב.

בעבודה השתמשתי הרבה בספריה זאת על מנת להעביר פקודות מן הרחפן למחשב אשר מעבד את המידע.



UDP

פרוטוקול תקשורת אשר דוגל בגישת "שלח וסע". גישה זאת אומרת שהמידע שנשלח לא מוודא קבלה של המידע והוא נשלח כפי שהוא.

לגישה זאת מספר יתרונות וחסרונות לעומת הפרוטוקול המקביל TCP:

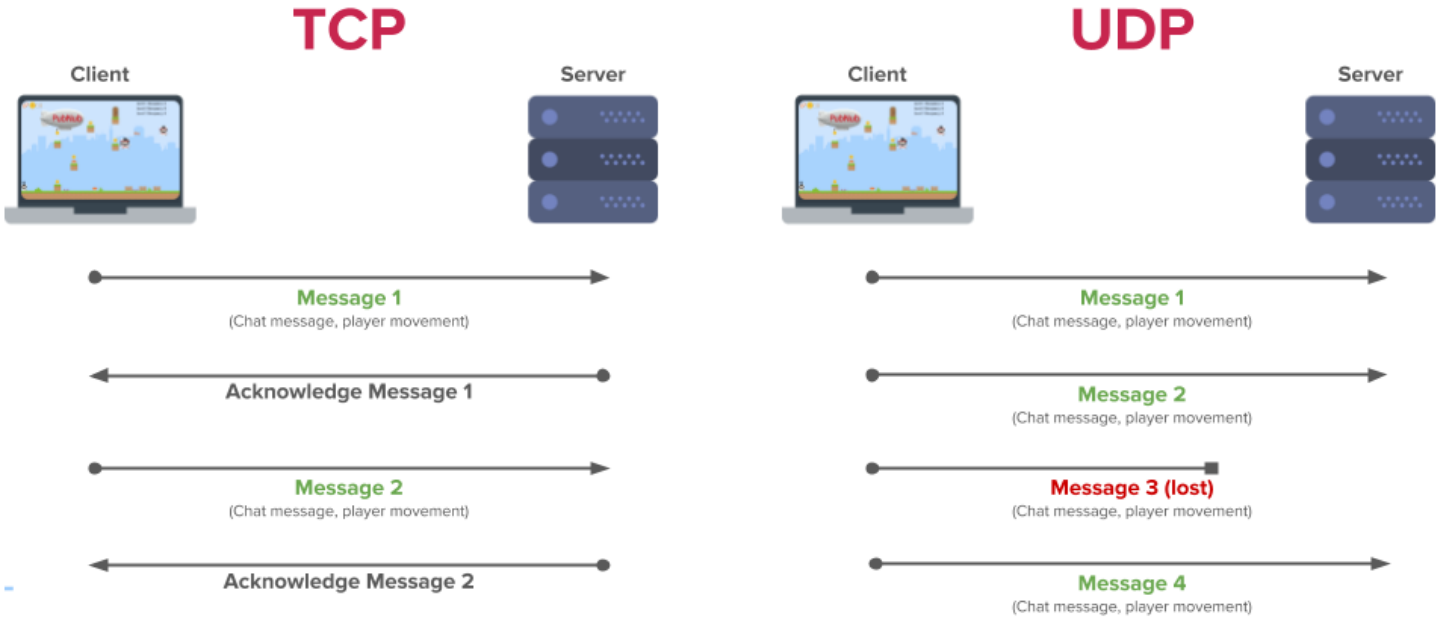
יתרונות:

- המידע נשלח ברצף ולכן מאפשר דיוור מהיר של מידע בלי בדיקה של האם הוא התקבל
- פשוט יותר לשלוח פקטות באמצעות UDP (לא צריך לוודא את קבלת המידע)
- באמצעות UDP אפשר לעשות Broadcast, משמע לשלוח להרבה מחשבים את אותה הודעה.

חסרונות:

- המידע לא מוודא הגעה ולכן יכול להווצר מצב של מידע חסר (Packet loss).
- הסדר של המידע יכול להיות לא מסודר, פקטות אשר נשלחו אחרי יגיעו לפני לדוגמה.
- חבילות של מידע זהה עלולות להגיע מספר פעמים.

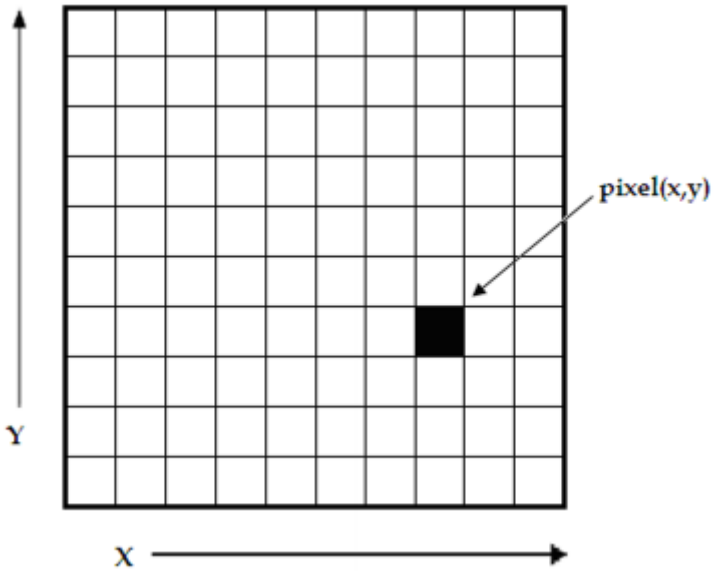
בגלל היתרונות של מידע אשר מועבר באופן מהיר לעומת TCP משתמשים רבות בUDP על מנת להעביר מידע אשר אינו דורש וידוא הגעה כמו שידור וידאו, ולכן השתמשתי בפרוטוקול זה על מנת לשלוח את הפקודות לרחפן ולקבלת השידור וידאו ממנו.



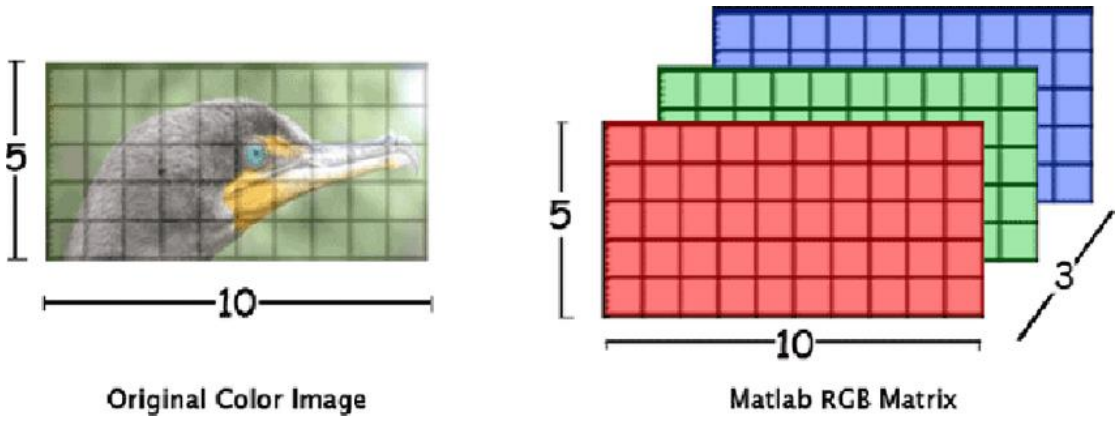
Vision

מהי תמונה?

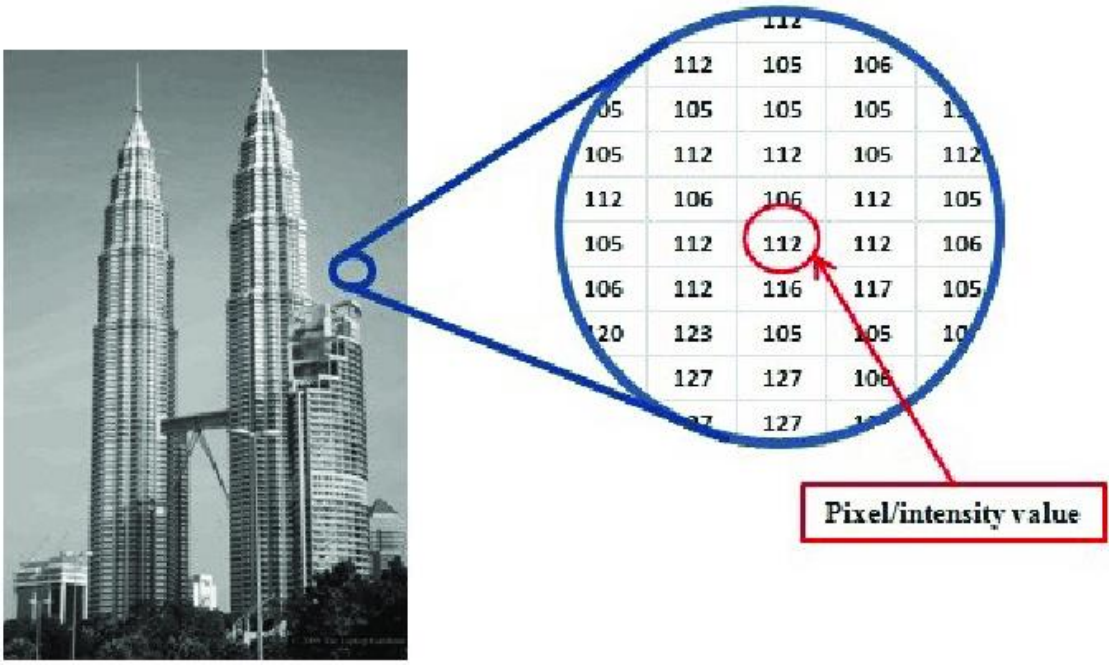
תמונה הינה מטריצה של פיקסלים אשר מסודרת ביחס מסוים.



כל תמונת RGB מחולקת בעצם ל3 תמונות שונות אשר מבטאות צבעים שונים מסדר Red, Green, Blue וכמות הצבע אשר כל פיקסל מקבל מאותו צבע. באמצעות שילוב של שלושת צבעים אלו ניתן לבטא כל צבע אשר נראה לעין ולכן זוהי השיטה הנפוצה ביותר כיום לתמונות. כל פיקסל בתמונת הצבע מקבל מספר שונה הנע בין 0,255 אשר מבטא את חוזק הצבע של אותו פיקסל, ובשילוב של שלושת הפיקסלים משלושת תמונות הצבע, אנו מקבלים פיקסל בתמונה המקורית אשר מבטא את הצבע של התמונה המקורית באותו פיקסל. בעצם, ניתן להתייחס לכל פיקסל בתמונת המקור כוקטור תלת מימדי במרחב הצבעים.



דבר זה אינו נכון לתמונות Grayscale אשר להם יש רק ערוץ אחד שמבטא את העוצמה של השחור באותו פיקסל.



Greyscale image

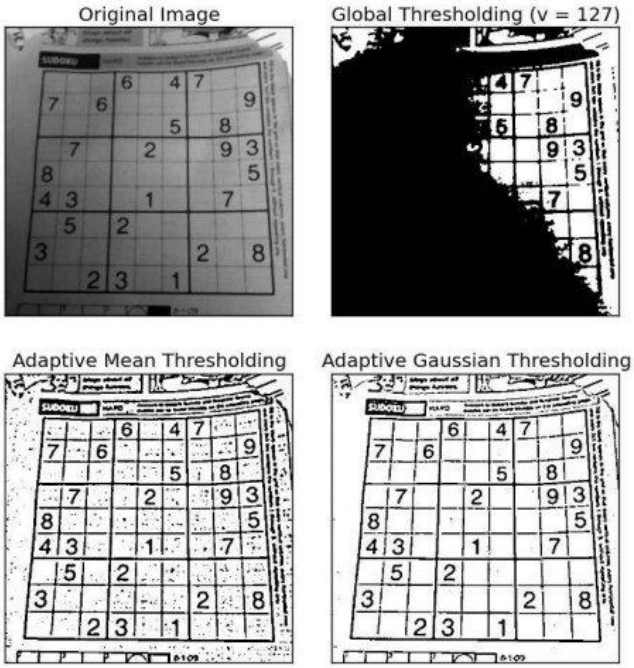
Thresholding

Thresholding היא פעולה שבה אנו מציבים תחום מסוים של צבע ועל פיו מסננים את כל מה שלא נכלל בתוך הצבע.

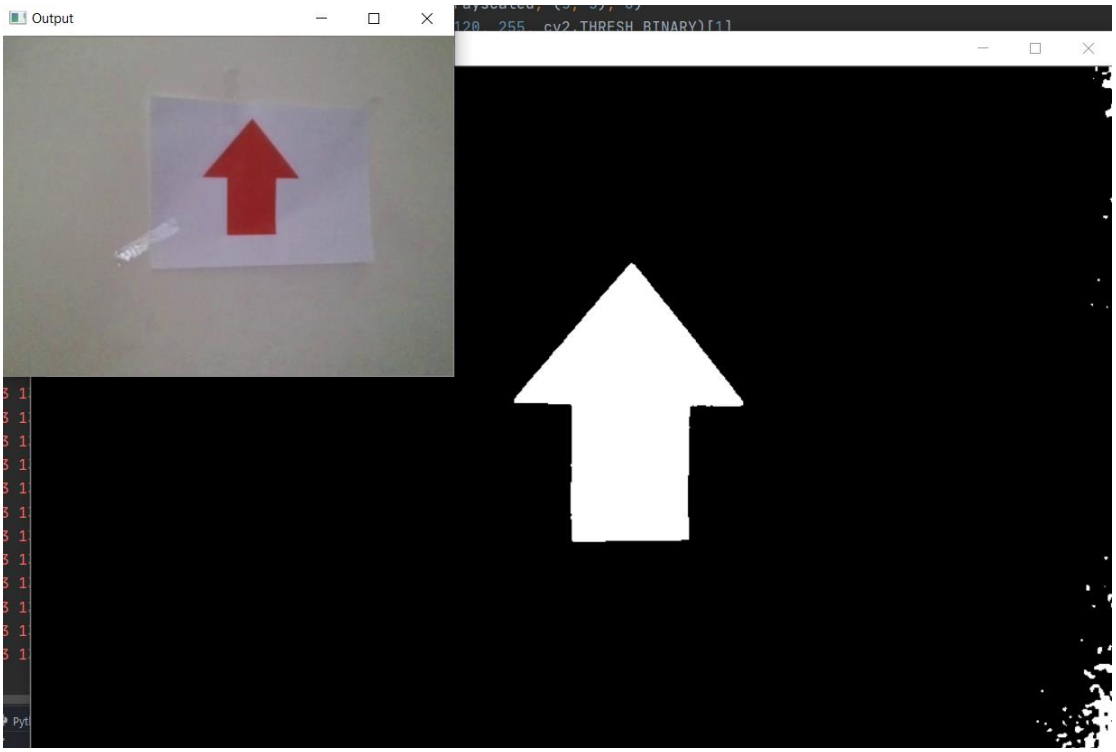
באמצעות כך, ניתן לסנן אובייקטים אשר לא מגיבים לאותו הצבע של שאר התמונה ולזהות אובייקטים בתמונה עצמה.

ישנם כמה סוגי Thresholding אשר בעיקר מגדירים את הדרך שבה נסנן את התחום המסוים של הצבע.

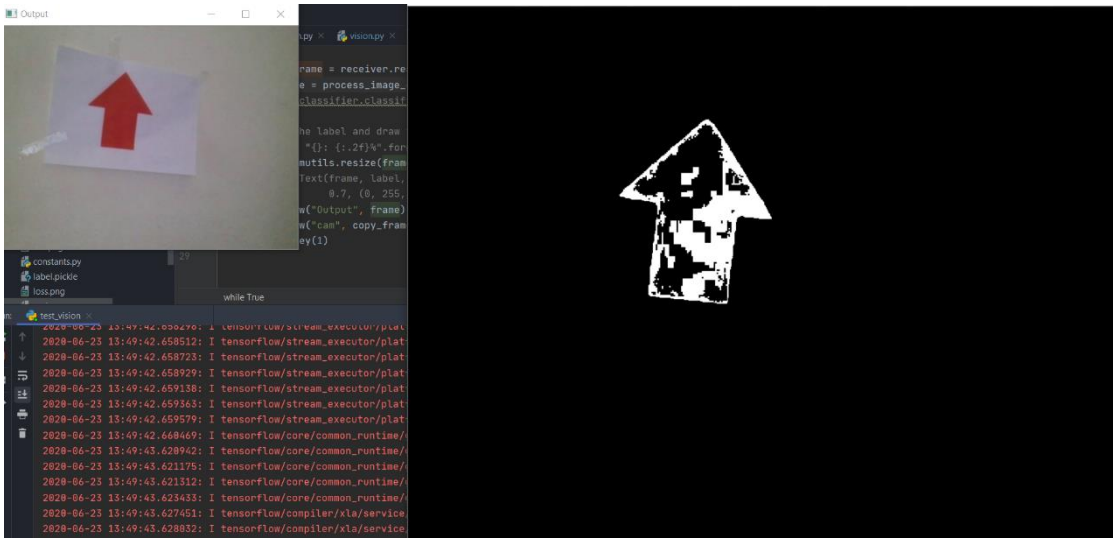
- Adaptive thresholding – הthreshold נקבע על פי ממוצע של סביבת האובייקט
- Gaussian adaptive thresholding – ה threshold נקבע על פי ממוצע של סביבת האובייקט שמושפע על ידי פונקצית גאוס
- In Range – מגדיר תחומי צבע ומסנן מה שלא בתוך תחומי צבע אלו.



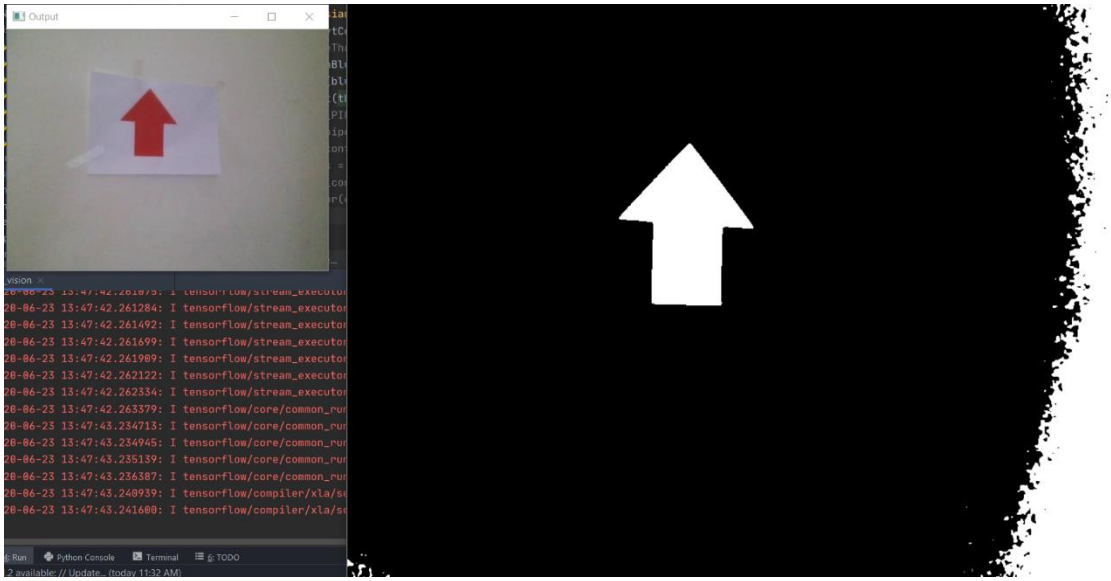
בעבודה שלי השתמשתי בthresholding על מנת לסנן את הרקע של התמונה ולהתמקד בחץ, ניסיתי להשתמש במספר אופציות אשר הניבו לי תוצאות שונות אך הכי אמינה הייתה בסופו של דבר . Gaussian blur + Mean Thresholding



Gaussian blur + Mean Thresholding



In range Thresholding



Gaussian adaptive Thresholding

Denoising

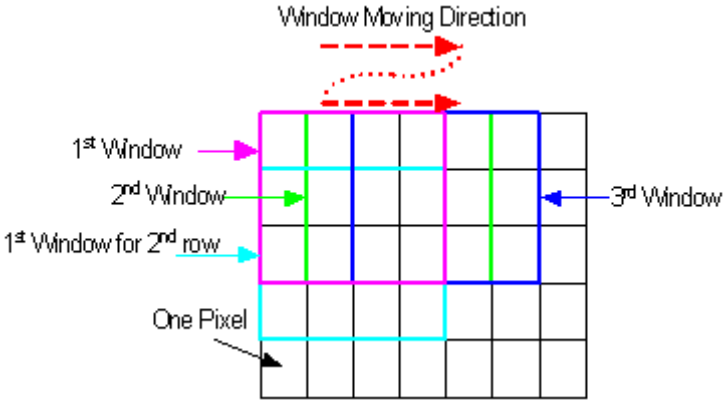
לעיתים רבות תמונה תהיה בעלת רעשים חיצוניים אשר לא מתאימים לאובייקט אשר אנו רוצים להשיג עם הThresholding, לכן עלינו יהיה לעשות denoising על מנת להפטר מרעשים אלו ולהשיג את המטרה הרצויה.

ישנן מספר דרכים לעשות Denoising לתמונה על מנת להפטר מן הרעשים, אך אנו נתמקד באחד העיקרי ששימש בעבודתי, Erode and dilate.

Sliding window

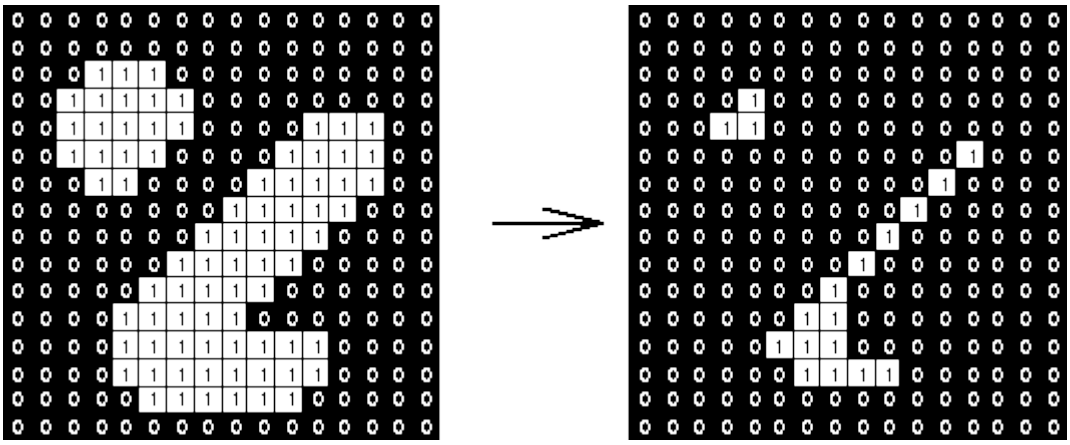
על מנת להבין מה זה Erode and Dilate עלינו להבין קודם כיצד הן עובדות.

Sliding window הינה פעולה שעוברת באמצעות חלון קטן על קלט התמונה ולוקח חלקים ממנה ומעביר עיבוד על כל חלק ככה שהוא יהיה הפיקסל בתמונה של הפלט.



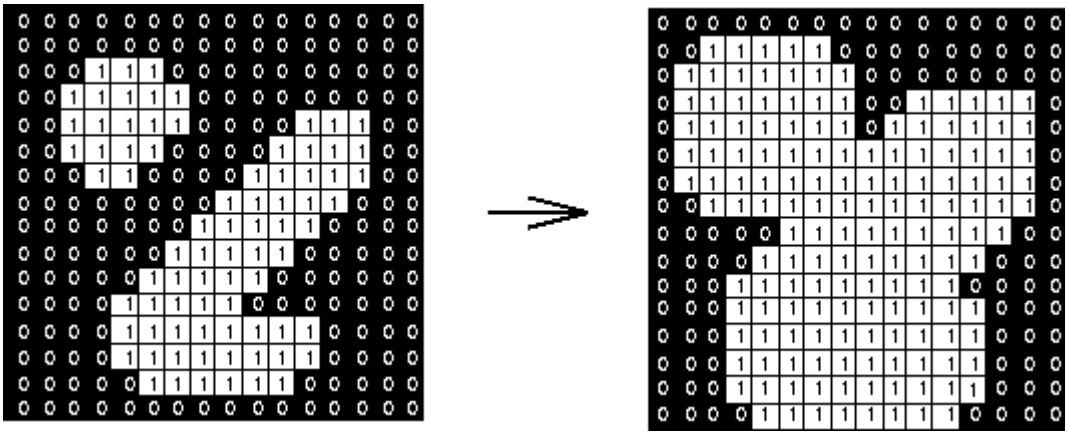
Erode

Erode (צמצום) הינה פעולה אשר מעבירה sliding window בגודל מוגדר על התמונה ולוקחת את המינימום מבין כל הערכים.



Dilate

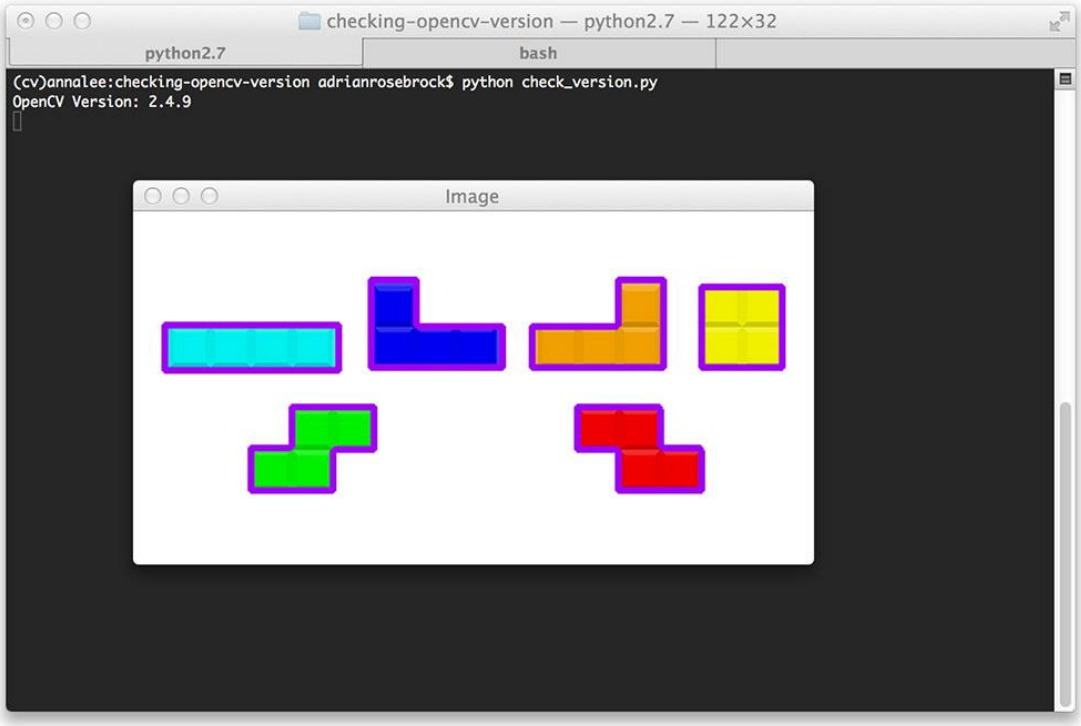
Dilate (הרחבה) היא פעולה שעושה הפוך מErode ומעבירה sliding window בגודל מוגדר על התמונה וכל איבר מוגדר להיות המקסימום של החלון.



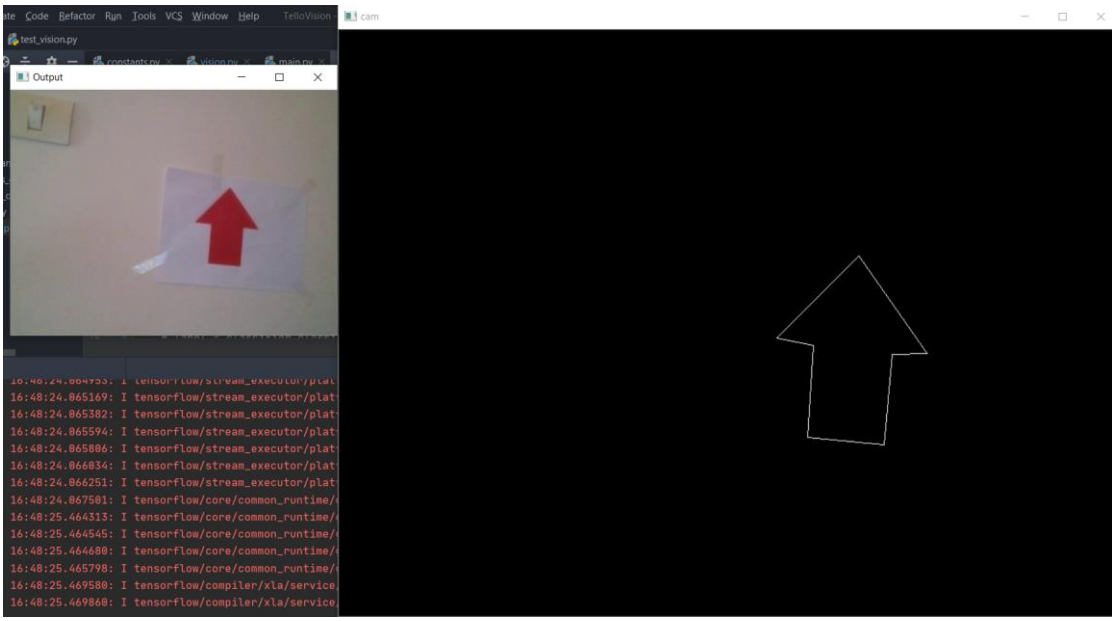
באמצעות שילוב של Erode and Dilate אנו מסוגלים לסנן רעשים קטנים אשר אינם קשורים לאובייקט ולקבל את הדבר אשר רלוונטי לנו.

Contours finding

על מנת למצוא אובייקים בעלי צורה מסויימת אשר הינם קבועים בצורתם, לרוב ננסה למצוא את הקווי מתאר שלהם באמצעות תיאורם כנקודות על המסך ואז נחבר בין הנקודות על מנת למצוא את האובייקט.



באמצעות מציאת האובייקט נוכל למלא אותו בצבע ולהתייחס אליו איך כיצד שאנו רוצים וצריכים.



גבולות החץ באמצעות פונקציית find contours.

Pipeline

Pipeline הינו צינור של מידע אשר כולל פונקציות שכל אחת מעבירה לבאה בתור את הפלט שלה כקלט.

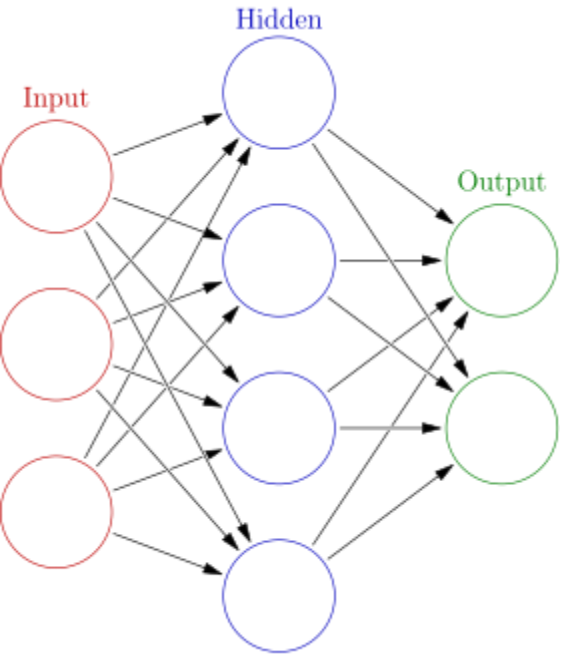
Pipeline משמש אותנו רבות בויזן בשל כך שאנו צריכים להעביר תמונה מפונקציה אחת לשנייה כמו במקרה של denoising שאנו צריכים לרוב אחרי הEroden לבצע dilate.

PipeLine לדוגמה שאני משתמש בעבודה.

Machine learning

Machine learning הינו תחום סטטיסטי חדש יחסית במדעי המחשב המתמקד בפיתוח אלגוריתמים על מנת לגרום למחשב "ללמוד" בעצמו באמצעות אימון על ידי דוגמאות שבני אדם עשו ופותר בעיות אשר פיתוח אלגוריתמים אליהם יהיה בגדר הבלתי אפשרי.

אלגוריתמים של Machine learning באים לפתור בעיות אשר נראות קלות לבני אדם, אך למחשב הינן קשות ביותר ולא ניתן לכתוב אלגוריתם סטנדרטי לזה. לדוגמה, קריאה של טקסט והבנתו הן משימות שלבני אדם נראות טריוואליות, אך מכונה לא מסוגלת להבין לדוגמה מה ההבדל בין צבי החיה, לצבי השם בקונטסט של המשפט. ולכן בעיות אלו פותרים באמצעות אלגוריתמי machine learning.



כמובן שזה רק קמצוץ קטן מהאפשרויות שאפשר לעשות עם machine learning, היום אפשר לראות אלגוריתמים אשר משחזרים בצבע תמונות מ1910 על פי תמונות עדכניות, מזהים קול של בן אדם ומתרגמים את מה שהוא אומר לכתב בדיוק מדהים, מערכת המלצות למוזיקה ופריטים פיזיים על פי קניות קודמות של אותו בן אדם, ועוד הרבה דברים מדהימים שאי אפשר להעלות על הדעת.

רשת נוירונים

רשת נוירונים הינה מודל של machine learning אשר נועד לחקות את המוח האנושי באמצעות נוירונים מלאכותיים אשר מדמים את הנוירונים של המוח. באמצעות מודל זה ניתן לאמן את המחשב לעשות כל בעיה שניתן לפתור באמצעות machine learning ביעילות ובדיוק יותר רב ממודלים אחרים.

רשת נוירונים מורכבת משכבות של נוירונים אשר מחוברות אחת לשנייה באמצעות משקלים אשר מושפעים מפונקציה אשר נקראת Loss function (נגע בכך בהמשך). כל נוירון מחובר לכל נוירונים בשכבה הבאה אחריו וכך ההשפעה עוברת.

רשתות נוירוינים מתאפיינות במבנה קבוע:

- שכבת הקלט
- שכבות נסתרות
- שכבת הפלט

שכבת הקלט מקבלת במקרה שלנו תמונה עם ערוץ אחד (בגלל שהתמונה בשחור לבן).

שכבת הפלט מוציאה 5 פלטים שאומרים מה הסיכוי של כל אחד להיות החץ הנכון או רנדומלי.

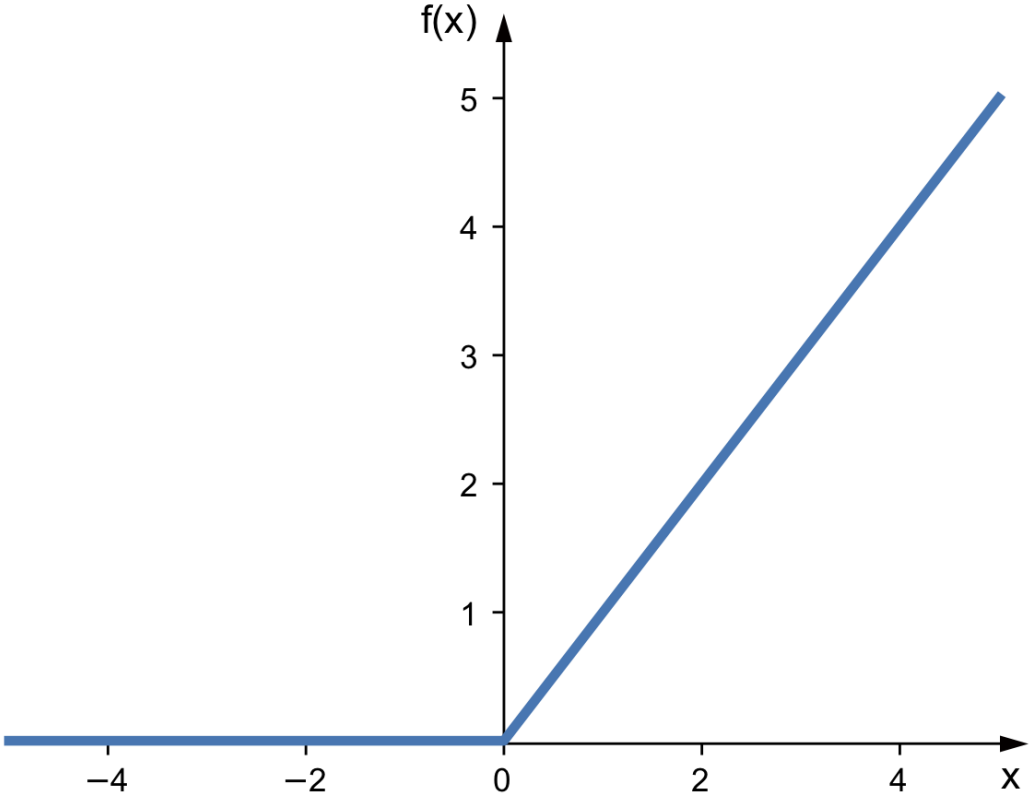
חיזוי פלט

כדי להבין כיצד עובד חיזוי הפלט אנו צריכים להבין כיצד נוירון בסיסי עובד.

כל נוירון עושה את הפעולות הבאות בכל קלט:

1. מקבל קלט כוקטור
2. מכפיל את הוקטור בweight
3. מוסיף bias
4. מפעיל את הactivation function עליו על מנת לנרמל אותו

פונקציית ההפעלה (Activation function) הינה פונקציה אשר מקבלת ערך ולכל ערך היא מסוגלת להחזיר ערך מנורמל אשר יבטא את הערך לעומת ערכים אחרים. בעבודתי אני השתמשתי בReLU עקב המהירות שלה לעומת פונקציות אחרות והאפשרות לגזירה.



פוקציית ReLU

באמצעות תהליך זה על כל הנוירונים ברשת ניתן לחזות את הפלט.

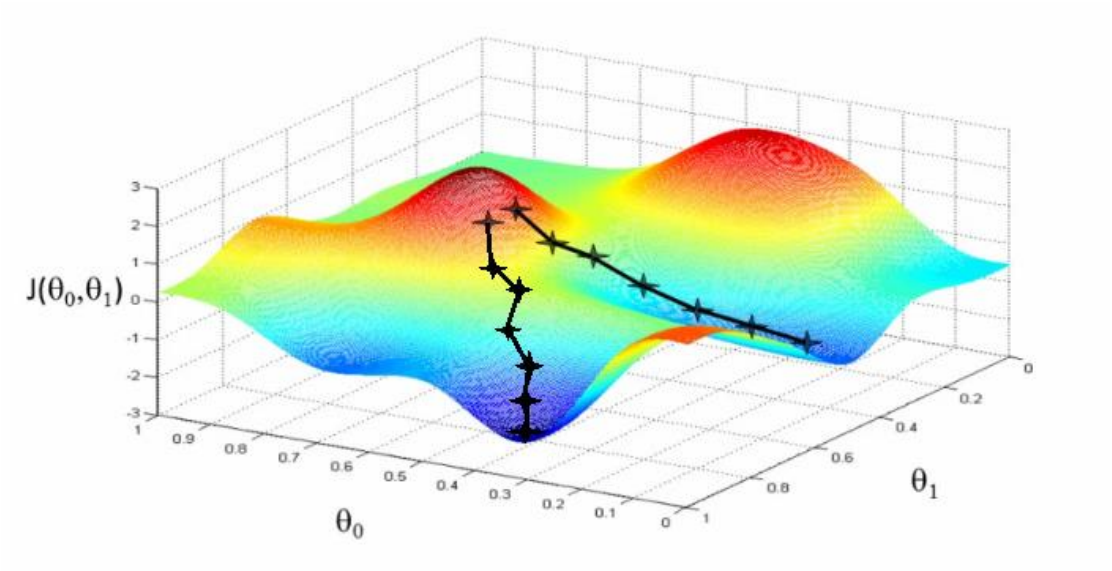
אימון הרשת

באימון הרשת, הרשת מקבלת קלט ופלט רצוי לאימון. הרשת מנסה לחזות מה הפלט ואחרי החיזוי אנו בודקים כמה הרשת נוירונים צדקה בהתאם למידע שיש לנו. את המידע הזה אנחנו מכניסים לפונקציה אשר נקראת פונקציית "הפסד" (loss function) שהיא מחשבת את הכמות השפעה שיש לשים על הנוירון הבא על מנת שהוא יהיה יותר מדויק. בעבודה שלי אני השתמשתי בLoss בשם categorical crossentropy אשר מחשב את הloss באמצעות הנוסחה המתמטית הבאה:

$$L(y,\hat{y}) = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij} * log(\hat{y}_{ij}))$$

לאחר מכן תהיה לנו פונקציה רציפה וגזירה שעליה נרצה למצוא את המינימום הנדרש על מנת להבין מה השגיאה הכי נמוכה שאנו צריכים לשים כדי שהרשת תהיה מדויקת. על מנת למצוא את מינימום זה נעזר באלגוריתם לדוגמה בשם Gradient descent. אלגוריתם זה מנסה למצוא את נקודת המינימום של הפונקציה באמצעות התקדמות לכיוון ההפוך מהגראדינט (מכיוון שהגראדינט מראה את השיפוע כלפי מעלה), כך שבכל שלב שאנחנו מתקדמים אנחנו יורדים עוד ועוד כלפי מטה.

אפשר לדמיין זאת כאדם בערפל בחשכה, על מנת למצוא את מורד העמק על האדם לנסות לגשש את דרכו כלפי תחתית העמק, לכן על מנת למצוא את דרך זו האדם יתקדם כל פעם מרחק קטן כדי לראות כיצד להגיב כלפי האדמה.



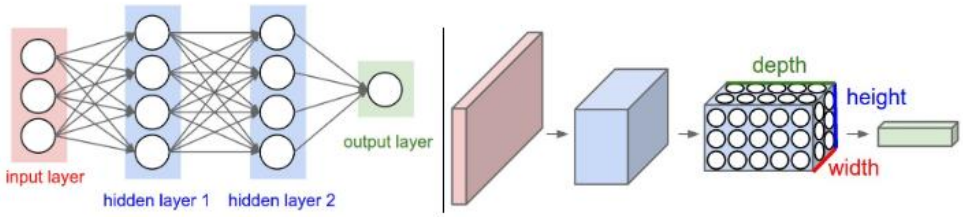
לאחר שאנו מוציאים את השגיאה המינמלית, אנו צריכים לדעת מהי הדרך הטובה ביותר ליישם זאת על כל הנוירונים שלנו? פה נכנס לתמונה אלגוריתם בשם Backpropagation. אלגוריתם זה נקרא כך בשל היותו מהסוף להתחלה, אנו יודעים מה ערכי השגיאה ולכן עלינו כעת להשפיע על הרשת באמצעות משקלים גדולים אם השגיאה הייתה גדולה, או קטנים אם השגיאה הייתה קטנה. תהליך זה נמשך כמה פעמים על מנת לגרום לרשת להתאמן האופן האופטימלי ביותר.

CNN (Conventional Neural Network)

תת סוג של רשת נוירונים אשר מיועד בעיקר לתמונות ווידאו. בניגוד לרשת נוירונים רגילה שבה כל נוירון מחובר לכל הנוירונים בשכבה הבאה, בCNN הגישה היא יותר פסיבית. שם הרשת בא מהפעולה המתמטית הבינארית קונבולוציה אשר מתארת את סך השטח הכלוא מתחת למכפלת שתי פונקציות.

הסיבה שבה משתמשים ברשתות אלו במקום רשתות נוירונים "נורמליות" הינה בגלל מספר המשקלים שנצטרך לשים לתמונה במידה ונרצה לעשות פעולות על תמונות. הרי תמונה ממוצעת הינה ברוחב של 1920 עם גובה של 1080 ו3 ערוצי צבע משמע 6,220,800 משקלים (3*1920*1080)! מספר כזה גבוה של משקלים הינו לא רציונלי ולכן עלינו להשתמש בסוג אחר של רשתות.

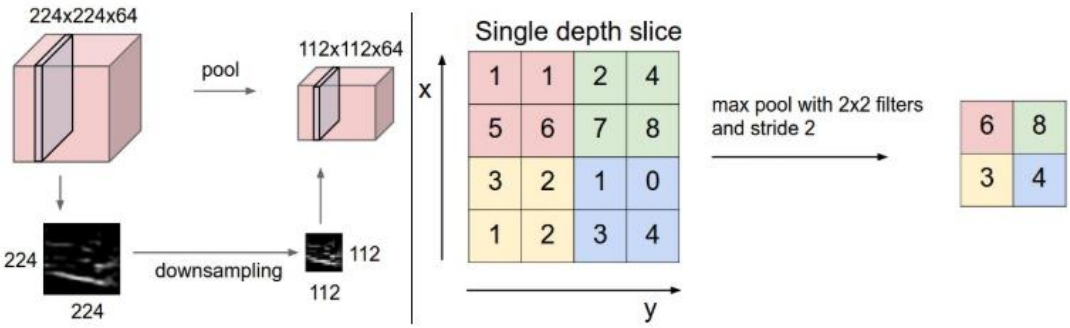
רשתות אלו מתאפיינות באלמנט בשם kernel filter אשר מתאר ריבוע נע על גבי התמונה שמבצע פעולות מתמטיות על ערכי הפיקסלים אשר נכללים באותו המטריצה וממיר אותם למטריצה חדשה אשר כוללת את ערכי הפעולה המתמטית אשר התבצעה על המטריצה (בדומה ל sliding window).



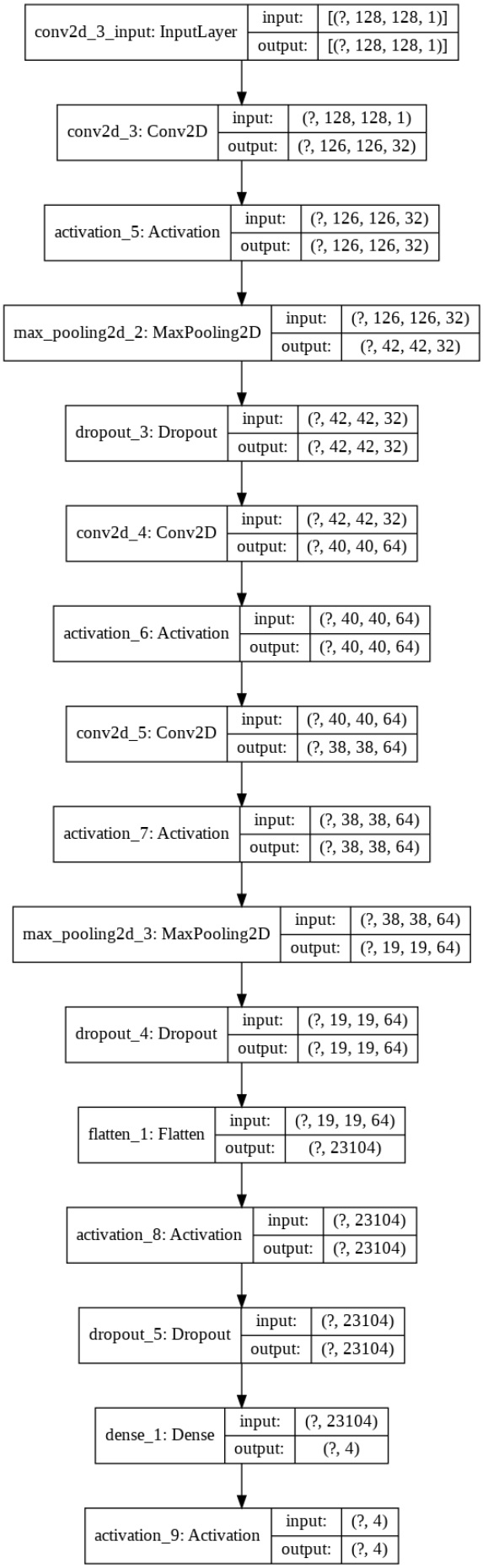
רשת נוירונים רגילה (משמאל) לעומת מבנה דומה של רשת נוירונים קונבציונלית.

רשת נוירונים מסוג זה לרוב בנויה מ3 שכבות עיקריות של פעולה

- Conv2D – שכבת רשת נוירונים עם 2 מימדים
- Activation Function- פונקציית אקטיבציה
- MaxPool2D – מציאת המקסימום של הkernel והצבתו במטריצה חדשה



ארכיטקטורת הרשת אשר בה
השתמשתי עבור הפרויקט שלי
(נבנה באמצעות פונקציית
plot_model של keras)



חלק

מעשי

ספריות הפרויקט

במהלך הפרויקט השתמשתי במגוון ספריות אשר נועדו על מנת לפשט תהליכים ולייעול הקוד

OpenCV

ספרייה אשר נכתבה על ידי מפתחי אינטל בשנת 2000 אשר עוזרת ליישום אלגוריתמי עיבוד תמונה מתקדמים בזמן אמת, באמצעות ממשק פייתוני לספריה המקורית אשר נכתבה בשפת C. כיום הפרויקט הינו open source ונתמך על ידי קהילת מפתחים גדולה אשר מתחזקת את הפרויקט.

GBvision

ספריית ויז'ן מבית קבוצת הרובוטיקה של הכפר הירוק GreenBlitz. ספרייה זו מהווה ממשק לייעול הגישה לספריית OpenCV באמצעות יישום אלגורתימים מראש, גישה ישירה למצלמות רשת, PipeLine קל ויעיל לשימוש, זיהוי אובייקטים במרחב התלת-מימדי ועוד כלים.

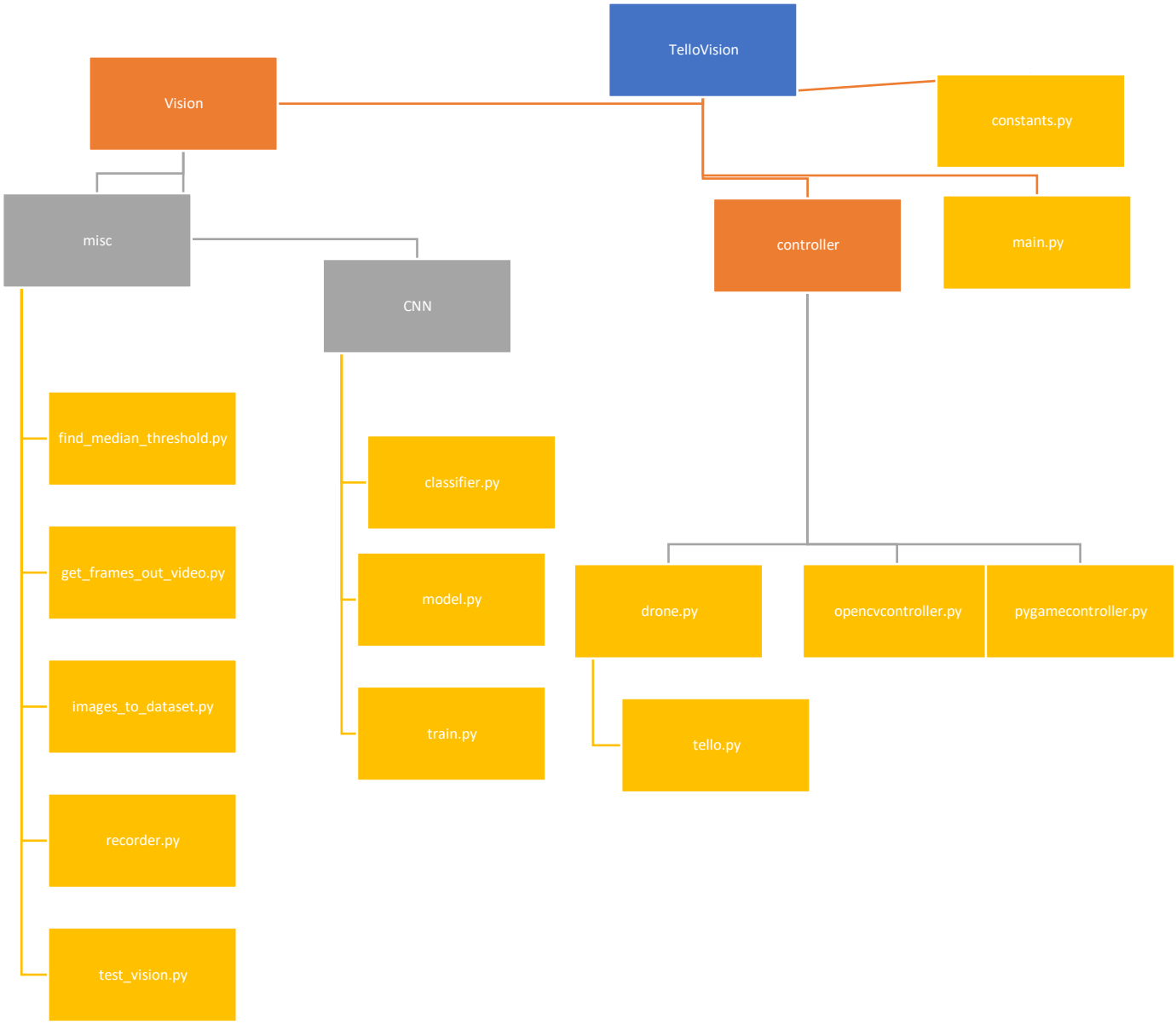
Keras

קרס הינה ספרייה אשר נכללת בתוך Tensorflow אשר מייעלת את הבנייה של מודלי למידת מכונה באמצעות ממשק High level אשר נועד לפשט תהליכים כדי לתת לחוקרים אפשרות לחקור ולא לתכנת. היא נכתבה על ידי מהנדס גוגל בתור ספריה נפרדת מTensorflow ולבסוף נכללה בתור submodule בתוך tensorflow מבית גוגל.

Numpy

Numpy הינה ספריה אשר מאפשרת גישה לכלים של אנליזה נומרית ואלגברה לינארית בפרט, בנוסף לכפל מטריצות וחישובים אשר לא היו קיימים לפני בפייתון. היא בנוסף כוללת אובייקט array אשר נותן גישה ל Array C כמו בשפות תכנות אחרות, דבר אשר לא קיים בפייתון. רוב הספריות הגדולות משתמשות כיום בNumpy לחישובים מתמטיים ובמיוחד באובייקט הArray (לדוגמה: פריים של OpenCV מיוצג באמצעות מטריצה שמיוצגת באמצעות Array של numpy).

מבנה הפרויקט



מבנה הקוד

מבנה הקוד מחולק בעיקרו לmodules שתלויים אחד בשני ומאפשרים תכנות יחסית מובן על פי הפעולות שאנחנו מבצעים (תכנות פונקציונלי), אסביר על כל Module בנפרד ולמה כל class משמש בתוך הmodule.

Controller module

מודל זה מורכב מ4 קלאסים,

Drone.py

מחלקה אבסטרקית אשר נועדה לתאר אילו פונקציות ותכונות יש ליישם ברחפן

```
1.     def move_up(self, x: int):
2.         return self.move("up", x)
3.
4.     def move_down(self, x: int):
5.         return self.move("down", x)
6.
7.     def move_right(self, x: int):
8.         return self.move("right", x)
9.
10.    def move_left(self, x: int):
11.        return self.move("left", x)
12.
13.    def move_forward(self, x: int):
14.        return self.move("forward", x)
15.
16.    def move_backward(self, x: int):
17.        return self.move("back", x)
```

אלו חלק מהפונקציות הלא אבסטרטקיות שdrone מממש וכולן מפנות לפונקציה האבסטרקטית move שמיושמת במחלקה היורשת.

Tello.py

מחלקה זאת כוללת 2 אובייקטים מרכזיים:

- TelloController – מחלקה אשר יורשת מDrone ומטרתה לממש את המחלקות של Drone אשר כוללות שליטה על הרחפן באמצעות ממשק על גבי הWifi ובאמצעות שימוש בספריית הסוקטים ופתיחה של thread נפרד לקבלת התשובה מהרחפן ובדיקה האם הפקודה התקבלה או לא. מחלקה זו מבוססת על הSDK הרשמי של Tello ומהווה מימוש ישיר שלה. ניתן למצוא אותה כאן:https://dl-cdn.ryzerobotics.com/downloads/tello/20180910/Tello%20SDK%20Documentation%20EN_1.3.pdf
- TelloVideoReceiver – מחלקה היורשת מcv2.VideoCapture ופותחת שרת Udp על המחשב אשר קולט את השידור וידאו מן הרחפן.

ישנן כמה פונקציות בסיס בשתי האובייקטים שהייתי רוצה להתעכב עליהן:

TelloController

```
1.  def send_command(self, command: str, timeout: float) -> bool:
2.      """
3.      Send command to Tello and wait for response
4.      Args:
5.          timeout (float): Command timeout
6.          command (bool): Command to send
7.
```

```
    Returns:
         (bool): True if command succeeded, False if not
    """
    self.error_flag = False
    timer = threading.Timer(timeout, self._set_error_flag)
    self.socket.sendto(command.encode('utf-8'), self.drone_address)

    timer.start()
    while self.response is None:
        if self.error_flag:
            raise RuntimeError("command timed out")
    timer.cancel()

    response = self.response.decode('utf-8')
    self.response = None
    if response == 'ok':
        return True
    elif response == 'error':
        return False
    return False
```

פונקציה זו אשר ממומשת בTelloController הינה הפונקציה המרכזית של המחלקה בשל כך שכל הפונקציות האחרות (מלבד get_state) מבוססות עליה ובעצם מהוות ממשק שהבסיס שלהם זה פונקציה זו. היא בעצם אחראית על שליחה של הפקודה על פי מה שהSDK קבע, ובמידה שאנו לא מקבלים תשובה מהרחפן היא הופכת את הדגל של error לTrue וזה מעלה exception של Timeout.

עוד פונקציה שהייתי רוצה לספר עליה בTelloController הינה get_state()

```
def get_state(self):
    """
    Get tello state from tello and place it as a dict in local var state

    """
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.bind(('0.0.0.0', 8890))
    while True:
        state_string = sock.recv(1024).decode('utf-8')
        list_states = state_string.split(';')[:-1]
        self.state = dict([i.split(':') for i in list_states])
```

פונקציה זו מקבלת מן הרחפן את המצב שלו כרגע והופכת את זה ל dictionary באמצעות list comprehension שמפריד בין המצבים שאנו מקבלים מהרחפן.

TelloVideoReceiver

```
class TelloVideoReceiver(cv2.VideoCapture):
    def __init__(self):
        self.address = 'udp://0.0.0.0:11111'
        super().__init__(self.address)

        self.frame = None
        self.status = False

        self.thread = threading.Thread(target=self._thread_handler, daemon=True)  # start new thread for video receiving
        self.thread.start()

    def read(self, image=None):  # reimplementing read function so it would not be accessed by thread
        return self.status, self.frame

```

```
15.     def _thread_handler(self):
16.         while True:
17.             if self.isOpened():
18.                 self.status, self.frame = super().read()
19.             else:
20.                 self.status, self.frame = (False, None)
```

מחלקה זו בעצם יורשת מcv2.VideoCapture וברגע יצירתה היא מפעילה שרת UDP על המחשב שמפעיל את תוכנת ffmpeg שממיר את הפקטות אשר אנו מקבלים מהרחפן לסטרים של וידאו. על מנת לא לתקוע את התוכנה ולגרום לרציפות התוכנה או לRace condition בלי תקלות ועם מינימום של Frame drops, אני מימשתי את Read מחדש ככה שיהווה רק ממשק לקבלה של Frame, status וככה אני מונע בעצם Race condition ששתי אובייקטים יבקשו בו זמנית את אותו הפריים.

2 המחלקות אשר נותרו במודל זה הינם Controllers אשר נועדו לממש את הממשק בין המשתמש לרחפן/בין הרשת נוירוים לרחפן באמצעות 2 Frameworks שונים. שתי המחלקות דומות מאוד בפעולות אשר הן ממשות ככה שאני אנסה לתמצת כמה שניתן.

- Opencvcontroller.py – מחלקה אשר ממשמת את הממשק הגרפי של המצלמה באמצעות פונקציית imshow של OpenCV ואת הקלט מן המקלדת באמצעות ספריה חיצונית בשם Pynput אשר מפעילה בת'רד נפרד keyboard listener.
- PygameController.py – מחלקה אשר מממשת גם את הממשק הגרפי וגם את קבלת הקלט מהמקלדת באמצעות ספריית Pygame. קבלת הקלט מהמקלדת בנוסף לעדכון התנועה של הרחפן נעשה באמצעות תור איוונטים של Pygame אשר מקליט איוונטים במחשב ובמידה ואחד מהם הינו הקלדה הוא מתנהג לפיו.

OpenCVController.py

```
1.  def set_interval(func, sec):
2.      def func_wrapper():
3.          set_interval(func, sec)
4.          func()
5.      t = threading.Timer(sec, func_wrapper)
6.      t.start()
7.      return t
```

זוהי פונקציה אשר נועדה לחקות את set_interval מJS ומפעילה בכל זמן מסויים את הפונקציה מחדש.

```
1.  def update(self):
2.      if self.send_rc_control:
3.          self.tello.set_rc(self.left_right_velocity, self.for_back_velocity,

4.                            self.up_down_velocity, self.yaw_velocity)
```

פונקציה אשר מעדכנת את מנועי הרחפן על פי קריאות שset_interval עושה בכל כמה רגעים.

```
1.  def keydown(self, key):
2.          if key == keyboard.Key.up:  # set forward velocity
3.              self.for_back_velocity = SPEED
4.          elif key == keyboard.Key.down:  # set backward velocity
5.              self.for_back_velocity = -SPEED
6.          elif key == keyboard.Key.left:  # set left velocity
7.              self.left_right_velocity = -SPEED
8.          elif key == keyboard.Key.right:  # set right velocity
9.              self.left_right_velocity = SPEED
10.         elif key == keyboard.KeyCode.from_char('w'):  # set up velocity
```

```
            self.up_down_velocity = SPEED
        elif key == keyboard.KeyCode.from_char('s'):  # set down velocity
            self.up_down_velocity = -SPEED
        elif key == keyboard.KeyCode.from_char('a'):  # set yaw counter clockwise velocity
            self.yaw_velocity = -SPEED
        elif key == keyboard.KeyCode.from_char('d'):  # set yaw clockwise velocity
            self.yaw_velocity = SPEED
        elif key == keyboard.Key.esc:
            self.should_stop = True
        elif key == keyboard.Key.enter:
            self.tello.emergency()
        elif key == keyboard.Key.space:
            self.is_vision_control = not self.is_vision_control  # Switch between vision and manual controller

    def keyup(self, key):
        if key == keyboard.Key.down or key == keyboard.Key.up:  # set zero forward/backward velocity
            self.for_back_velocity = 0
        elif key == keyboard.Key.right or key == keyboard.Key.left:  # set zero left/right velocity
            self.left_right_velocity = 0
        elif key == keyboard.KeyCode.from_char('w') or key == keyboard.KeyCode.from_char(
                's'):  # set zero up/down velocity
            self.up_down_velocity = 0
        elif key == keyboard.KeyCode.from_char('a') or key == keyboard.KeyCode.from_char('d'):  # set zero yaw velocity
            self.yaw_velocity = 0
        elif key == keyboard.KeyCode.from_char('t'):  # takeoff
            if not self.tello.takeoff():
                print("[ERROR] Takeoff failed")
            else:
                self.send_rc_control = True
        elif key == keyboard.KeyCode.from_char('l'):  # land
            self.tello.land()
            self.send_rc_control = False
```

keydown ו keyup אלו פונקציות אשר מופעלות על ידי הת'רד של pynput ואומרות מה הן הכפתורים של שליטה עצמית על הרחפן.

פריסת המקלדת והפקודות.

```
def run(self):
        """
        Runs the program based on vision or manual control.
        always starting with manual control.
        Manual control keys:
```

```
                    w - up
         a - yaw left  s - down  d - yaw right
                    ↑ - forward
         ← - left  ↓ - back  → - right
         l - land
         enter - emergency
         escape - quit
         space - switch between vision and manual control

        Returns:

        """
        listener = keyboard.Listener(
            on_press=self.keydown,
            on_release=self.keyup
        )
        listener.start()

        self.tello.stop_stream()
        self.tello.start_stream()

        if not self.video.isOpened():
            open_video = self.video.open(self.video.address)
            time.sleep(5)
            if not open_video:
                return False

        status, frame = self.video.read()
        set_interval(self.update, ((1000 // FPS) / 1000))
        #################### Manual Control ####################
        while True:
            while not self.should_stop and not self.is_vision_control:
                if not status:
                    break
                status, frame = self.video.read()
                text = "Battery: {}%".format(self.tello.state['bat'])
                cv2.putText(frame, text, (5, 720 - 5),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

                cv2.imshow('Video', frame)
                cv2.waitKey(1)

            #################### Vision Control ####################
            classifier = self.classifier
            direction = None
            engine = None  # To be in control of which direction we need to make speed 0

            while not self.should_stop and self.is_vision_control:

                # Screen output and reading from stream
                status, frame = self.video.read()
                copy_frame = frame.copy()
                frame = process_image_gaussian(frame)

                text = "Battery: {}%".format(self.tello.state['bat'])
                cv2.putText(copy_frame, text, (5, 720 - 5),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
                label = "{}: {:.2f}%".format(direction, classifier.prob * 100)
                cv2.putText(copy_frame, label, (10, 25), cv2.FONT_HERSHEY_SIMPLEX,
                            0.7, (0, 255, 0), 2)

                # handle the engines so we know what was last direction
```

```
68.                 if direction == 'up' or direction == 'down':
69.                     engine = 'up_down'
70.                 elif direction == 'left' or 'right':
71.                     engine = 'left_right'
72.
73.                 # Vision movement
74.                 direction = classifier.classify(frame)
75.                 #direction = classifier.classify(frame) if classifier.prob >
    40.0 else 'background'
76.
77.                 if direction == 'up':
78.                     if engine == 'left_right':
79.                         self.left_right_velocity = 0
80.                     self.up_down_velocity = SPEED
81.                     print("UP")
82.
83.                 elif direction == 'down':
84.                     if engine == 'left_right':
85.                         self.left_right_velocity = 0
86.                     self.up_down_velocity = -SPEED
87.                     print("DOWN")
88.
89.                 elif direction == 'left':
90.                     if engine == 'up_down':
91.                         self.up_down_velocity = 0
92.                     self.left_right_velocity = -SPEED
93.                     print("LEFT")
94.
95.                 elif direction == 'right':
96.                     if engine == 'up_down':
97.                         self.up_down_velocity = 0
98.                     self.left_right_velocity = SPEED
99.                     print("RIGHT")
100.
101.                        elif direction == 'random':
102.                            self.left_right_velocity = 0
103.                            self.up_down_velocity = 0
104.
105.                        cv2.imshow('After vision', copy_frame)
106.                        cv2.waitKey(1)
107.
108.                        if not status:
109.                            break
110.                    if self.should_stop:
111.                        break
112.                self.tello.land()
113.                self.tello.stop_stream()
114.                self.video.release()
```

זו הפונקציית ריצה הראשית, היא מחולקת ל3 חלקים: איתחול, שליטה ידנית, שליטה ויזן.

השליטה הידנית מציגה בפונקציית run את המסך של הסרטון מהרחפן. הויזן לעומת זאת עובד ככה:

- הצג את הוידאו ורשום עליו את ההסתברות של החץ והבטריה
- זוז על פי כיוון החץ
- במידה והחלפת מנוע, עצור את המנוע השני
- במידע ואתה רואה דברים רנדומליים, עצור את המנוע.

PygameController.py

מחלקה זאת בניגוד לOpenCVController מחולקת ל3 אובייקטים אשר יורשים ממחלקה אבסטרקטית אשר מגדירה את הפעולות שעליהם לממש:

- ManualController
- VisionController
- MainController

ManualController

אחראי על התזוזה הידנית והוא מקבל אובייקט של telloController וtelloVideoReceiver:

```
1.  class ManualController(Controller):
2.
3.      def __init__(self, tello: TelloController, stream: TelloVideoReceiver, s
    creen):
4.          super().__init__()
5.          pygame.display.set_caption("Tello video stream")
6.          self.screen = screen
7.
8.          self.tello = tello
9.          self.video = stream
10.         self.event = None
11.         self.in_control = True
12.         self.send_rc_control = False
13.
14.         self.for_back_velocity = 0
15.         self.left_right_velocity = 0
16.         self.up_down_velocity = 0
17.         self.yaw_velocity = 0
18.
19.         pygame.time.set_timer(pygame.USEREVENT + 1, 1000 // FPS)  # Update T
    imer
20.
21.     def run(self):
22.         self.tello.stop_stream()
23.         self.tello.start_stream()
24.
25.         if not self.video.isOpened():
26.             open_video = self.video.open(self.video.address)
27.             time.sleep(5)
28.             if not open_video:
29.                 return False
30.
31.         status, frame = self.video.read()
32.
33.         should_stop = False
34.         while not should_stop and self.in_control:
35.             for event in pygame.event.get():
36.                 if event.type == pygame.USEREVENT + 1:
37.                     self.update()
38.                 elif event.type == pygame.QUIT:
39.                     should_stop = True
40.                 elif event.type == pygame.KEYDOWN:
41.                     self.event = event
42.                     if event.key == pygame.K_ESCAPE:
43.                         should_stop = True
44.                     else:
45.                         self.keydown(event.key)
46.                 elif event.type == pygame.KEYUP:
47.                     self.keyup(event.key)
48.
```

```
            if not status:
                break

            self.screen.fill([0, 0, 0])

            status, frame = self.video.read()
            text = "Battery: {}%".format(self.tello.state['bat'])
            cv2.putText(frame, text, (5, 720 - 5),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            frame = np.rot90(frame)
            frame = np.flipud(frame)

            background = pygame.surfarray.make_surface(frame)
            self.screen.blit(background, (0, 0))
            pygame.display.update()

            time.sleep(1 / FPS)
        if self.in_control:
            self.tello.land()
            self.tello.stop_stream()
            self.video.release()
```

הוא מסתכל על הqueue של events שpygame מוציא ובמידה אם הוא מקבל סיגנל של quit אז מבצע יציאה בטוחה, ובמידה והוא מקבל סיגנל של keyDown הוא פונה לפונקציה של keyDown ומבצע את הפעולות שם על פי ההגדרה של הkey. ובמידה והkeyUp הוא פונה לפונקציה של keyUp.

הוא גם מעדכן את ההתמונה באמצעות עדכון של הsurface של pygame.

VisionController

אחראי על תנועה באמצעות הויזן ומקבל גם כן את TelloController ו TelloVideoReceiver.

```
class VisionController(Controller):
    def __init__(self, tello: TelloController, video: TelloVideoReceiver, screen):
        super().__init__()
        self.tello = tello
        self.video = video
        self.classifier = Classifier(MODEL_PATH, LABELS_PATH)
        self.in_control = False
        self.screen = screen
        self.result = None
        self.event = None

        self.yaw_velocity = None
        self.left_right_velocity = None
        self.for_back_velocity = None
        self.up_down_velocity = None

    def run(self):
        should_stop = False
        classifier = self.classifier
        direction = None
        engine = None  # To be in control of which direction we need to make speed 0

        while not should_stop and self.in_control:
            for event in pygame.event.get():
                # Pygame events
                if event.type == pygame.USEREVENT + 1:
                    self.update()
                elif event.type == pygame.QUIT:
                    should_stop = True
```

```
30.                 elif event.type == pygame.KEYDOWN:
31.                     self.event = event
32.                     if event.key == pygame.K_ESCAPE:
33.                         should_stop = True
34.
35.             self.screen.fill([0, 0, 0])
36.
37.             # Screen output and reading from stream
38.             status, frame = self.video.read()
39.             frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
40.             frame = np.rot90(frame)
41.             frame = np.flipud(frame)
42.             copy_frame = frame.copy()
43.             frame = process_image_gaussian(frame)
44.
45.             text = "Battery: {}%".format(self.tello.state['bat'])
46.             cv2.putText(copy_frame, text, (5, 720 - 5),
47.                         cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
48.             label = "{}: {:.2f}%".format(direction, classifier.prob * 100)
49.             cv2.putText(copy_frame, label, (10, 25), cv2.FONT_HERSHEY_SIMPLE
    X,
50.                         0.7, (0, 255, 0), 2)
51.
52.             # handle the engines so we know what was last direction
53.             if direction == 'up' or direction == 'down':
54.                 engine = 'up_down'
55.             elif direction == 'left' or 'right':
56.                 engine = 'left_right'
57.
58.             # Vision movement
59.             direction = classifier.classify(frame) if classifier.prob > 70.0
     else 'background'
60.
61.             if direction == 'up':
62.                 if engine == 'left_right':
63.                     self.left_right_velocity = 0
64.                 self.up_down_velocity = SPEED
65.                 print("UP")
66.
67.             elif direction == 'down':
68.                 if engine == 'left_right':
69.                     self.left_right_velocity = 0
70.                 self.up_down_velocity = -SPEED
71.                 print("DOWN")
72.
73.             elif direction == 'left':
74.                 if engine == 'up_down':
75.                     self.up_down_velocity = 0
76.                 self.left_right_velocity = -SPEED
77.                 print("LEFT")
78.
79.             elif direction == 'right':
80.                 if engine == 'up_down':
81.                     self.up_down_velocity = 0
82.                 self.left_right_velocity = SPEED
83.                 print("RIGHT")
84.
85.             elif direction == 'background':
86.                 self.left_right_velocity = 0
87.                 self.up_down_velocity = 0
88.
89.             background = pygame.surfarray.make_surface(frame)
90.             self.screen.blit(background, (0, 0))
91.             pygame.display.update()
92.
93.             if not status:
```

```
                break

        if self.in_control:
            self.tello.land()
            self.tello.stop_stream()
```

הבקר הזה בדומה לבקר הידני ולבקר בOpenCvController שולט על הרחפן באמצעות עדכון של המנועים, לעומת אחרים אבל הוא יוצר אובייקט של Classifier אשר מזהה את החצים באמצעות רשת הנוירונים והמודל שאומן.

MainController

אחראי על סנכרון בין 2 הבקרים, אחראי גם על אתחול אובייקט הרחפן וקבלת הוידאו מהרחפן בנוסף לקבלת קלט מהמקלדת על בסיס קבוע על מנת להחליף בין הבקרים.

```
class MainController:
    def __init__(self):
        self.tello = TelloController()
        self.video = TelloVideoReceiver()

        self.screen = pygame.display.set_mode([960, 720])

        self.manual_controller = ManualController(self.tello, self.video, self.screen)
        self.vision_controller = VisionController(self.tello, self.video, self.screen)

        thread = threading.Thread(target=self.check_and_switch_controllers, daemon=True)
        thread.start()

    def check_and_switch_controllers(self):
        while True:
            # check who's controlling the drone
            control = self.manual_controller if self.manual_controller.in_control else self.vision_controller
            if control.event is not None:
                if control.event.type == pygame.KEYDOWN:
                    if control.event.key == pygame.K_SPACE:
                        self.manual_controller.in_control = not self.manual_controller.in_control
                        self.vision_controller.in_control = not self.vision_controller.in_control
                    elif control.event.key == pygame.K_RETURN:
                        self.tello.emergency()

    def run(self):
        self.manual_controller.run()
        self.vision_controller.run()
```

פה אנחנו מאתחלים את הרחפן ונותנים אותו ואת המסך של pygame כפרמטר לבקרים, יוצרים Thread חדש אשר אחראי על החלפת הבקרים ובמקרה חירום גם על השבתת הרחפן.

Vision Module

מבנה מודל זה מורכב מClass בשם Vision.py ועוד 2 submodules:

- Misc – כולל כלים אשר נועדו לפשט תהליכים בקוד ועושים אוטומציה לדברים.
- CNN – כל הקוד אשר נוגע לרשת נוירונים ולזיהוי תמונה באמצעותה.

Vision.py

מחלקה אשר כוללת 3 פעולות עיקריות ו2 מחלקות עזר:

```
1.  @gbv.PipeLine
2.  def contours_to_polygons(cnts):
3.      """
4.      performs approxPolyDP algorithm on a list of contours
5.
6.      :param cnts: the list of contours
7.      :return: a list of polygons from the contours
8.      """
9.      arc_lengths = map(lambda cnt: 0.03 * cv2.arcLength(cnt, True), cnts)
10.     return list(map(lambda cnt: cv2.approxPolyDP(cnt, next(arc_lengths), Tru
    e), cnts))
```

פונקציה זו ממירה אובייקטים מסוג גבולות למצולעים על פי הזוויות של הקווים.

```
1.  def process_image_gaussian(frame):
2.      grayscaled = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
3.      blur = cv2.GaussianBlur(grayscaled, (5, 5), 0)
4.      th = cv2.threshold(blur, 120, 255, cv2.THRESH_BINARY)[1]
5.      th = np.bitwise_not(th)
6.      pipe = gbv.EMPTY_PIPELINE + gbv.find_contours + contours_to_polygons + g
    bv.sort_polygons
7.      list_polygons = pipe(th)
8.      list_polygons_filtered = []
9.      if not len(list_polygons) == 0:
10.         list_polygons = list_polygons[:min(len(list_polygons), 4)]
11.         for poly in list_polygons:
12.             if check_arrow(poly):
13.                 list_polygons_filtered = [poly]
14.                 break
15.     drawn = gbv.draw_contours(np.zeros(frame.shape, np.uint8), list_polygons
    _filtered, (255, 255, 255),
16.                               thickness=cv2.FILLED)
17.     img = cv2.cvtColor(drawn, cv2.COLOR_BGR2GRAY)
18.     return img
```

פונקציה אשר מקבלת frame ומפעילה עליו את הפעולות הבאות:

1. ממירה את התמונה מצבע לשחור לבן
2. מפעילה את אלגוריתם gaussian_blur על התמונה על מנת לטשטש את הפרטים הקטנים.
3. מפעילה Mean Threshold על הפריים.
4. מוצאת את הגבולות של האובייקטים, ממירה אותם למצולעים וממינת אותם על פי גודל.
5. על 4 המצולעים הגדולים ביותר (במידה ויש) בודקת מי מהם בעל 7 נקודות ולכן מוגדר כחץ ולוקחת את הגדול ביותר מהם.
6. מציירת את האובייקט על המסך כחץ בעל מילוי בצבע לבן.
7. מחזירה את התמונה

```
def process_image_gaussian_adaptive(frame):
    grayscaled = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # blur = cv2.GaussianBlur(grayscaled, (5, 5), 0)
    th = cv2.adaptiveThreshold(grayscaled, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 205, 1)
    th = np.bitwise_not(th)
    pipe = gbv.EMPTY_PIPELINE + gbv.find_contours + contours_to_polygons + gbv.sort_polygons
    list_polygons = pipe(th)
    list_polygons_filtered = []
    if not len(list_polygons) == 0:
        list_polygons = list_polygons[:min(len(list_polygons), 4)]
        for poly in list_polygons:
            if check_arrow(poly):
                list_polygons_filtered = [poly]
                break
    drawn = gbv.draw_contours(np.zeros(frame.shape, np.uint8), list_polygons_filtered, (255, 255, 255),
                              thickness=cv2.FILLED)
    img = cv2.cvtColor(drawn, cv2.COLOR_BGR2GRAY)
    return img
```

פונקציה אשר מבצעת את אותם צעדים כמו בפונקציה הקודמת, אך בצעדים 2,3 היא מפעילה Gaussian adaptive threshold במקום על התמונה.

```
def process_image_color(frame):
    th = THRESHOLD(frame)
    pipe = gbv.EMPTY_PIPELINE + gbv.find_contours + contours_to_polygons + gbv.sort_polygons
    list_polygons = pipe(th)
    list_polygons_filtered = []
    if not len(list_polygons) == 0:
        list_polygons = list_polygons[:min(len(list_polygons), 4)]
        for poly in list_polygons:
            if check_arrow(poly):
                list_polygons_filtered = [poly]
                break
    drawn = gbv.draw_contours(np.zeros(frame.shape, np.uint8), list_polygons_filtered, (255, 255, 255),
                              thickness=cv2.FILLED)
    img = cv2.cvtColor(drawn, cv2.COLOR_BGR2GRAY)
    return img
```

פונקציה אשר מבצעת את אותן פעולות כמו בפונקציות הקודמות אך משתמשת בThreshold שמוגדר על פי טווח של צבעים מוגדרים מראש.

```
def check_arrow(polygon):
    return polygon.shape[0] == 7
```

פונקציה אשר מקבלת אובייקט מסוג מצולע, ומחזירה האם המספר נקודות שלו שווה ל7 (ולכן מקיים את התנאי של חץ).

CNN submodule

מודל אשר מתאר את כל הפעולות לאימון ובניית מודל הCNN.

Model.py

```
1.  from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Activ
    ation, Dropout
2.  from tensorflow.keras.models import Sequential
3.
4.
5.  class CNN:
6.      @staticmethod
7.      def build(width, height, depth, num_of_outputs):
8.          """
9.          Defines custom made CNN that aims to classify between 4 arrows (up,
    down, left, right)
10.         Args:
11.             width: Width of img
12.             height: Height of image
13.             depth: How many depth does the image have (1 for greyscale, 3 fo
    r RGB)
14.             num_of_outputs: how many outputs there is
15.         """
16.         input_shape = (width, height, depth)
17.
18.         model = Sequential()
19.
20.         # First set of layers (CONV => relu => MaxPool)
21.         model.add(Conv2D(32, (3, 3), input_shape=input_shape))
22.         model.add(Activation('relu'))
23.         model.add(MaxPool2D(pool_size=(3, 3)))
24.         model.add(Dropout(0.25))  # Dropout forth of the neurons
25.
26.         # Second set of Layers ((CONV => RELU) *2 => MaxPool)
27.         model.add(Conv2D(64, (3, 3)))
28.         model.add(Activation('relu'))
29.         model.add(Conv2D(64, (3, 3)))
30.         model.add(Activation('relu'))
31.         model.add(MaxPool2D(pool_size=(2, 2)))
32.         model.add(Dropout(0.25))
33.
34.         # Flatten the model
35.         model.add(Flatten())
36.         model.add(Activation("relu"))
37.         model.add(Dropout(0.5))
38.
39.         # Final output and applying softmax filter
40.         model.add(Dense(num_of_outputs))
41.         model.add(Activation("softmax"))
42.
43.         return model
```

בניית המודל באמצעות ספריית keras והוספת השכבות של רשת הנוירונים למודל הכולל.

Train.py

```
1. # Arguments parser
2. parser = argparse.ArgumentParser()
3. parser.add_argument("-d", "--dataset", help="Path for dataset")
4. parser.add_argument("-m", "--model", help="The path for saving the model")
5. parser.add_argument("-l", "--
   label", help="Path for saving the label binary")
6. args = parser.parse_args()
```

חלק זה בקוד אחראי על הבנת הפרמטרים של הפונקציה בזמן ריצה למקום שמירת המודל והתוויות..

```
1. # Load Dataset
2. dataset = ArrowsDataset(args.dataset, IMAGE_DIMS)
3. (x_train, y_train), (x_test, y_test) = dataset.load_data()
```

פה אנו טוענים את הDataset עם התמונות על פי התמונות של האימון והתמונות הרנדומליות של הבדיקת accuracy של המודל.

```
1. model = CNN().build(IMAGE_DIMS[0], IMAGE_DIMS[1], 1, 5)
2. opt = Adam(learning_rate=LR)
3. model.compile(loss=LOSS, optimizer=opt, metrics=['accuracy'])
4.
5.
6. # Train the model
7. output = model.fit(x_train, y_train, validation_data=(x_test, y_test), batch
   _size=BATCH_SIZE, epochs=EPOCHS)
```

בחלק זה אנחנו בונים את המודל באמצעות המחלקה של CNN, מציינים איזה optimizer אנחנו משתמשים (ADAM) ומקמפלים אותו עם הפרמטר של הLoss, האופטימיזר, ומה אנחנו רוצים לראות באימון (Accuracy).

לאחר מכן אנחנו מפעילים את הפעולה fit על המודל אשר מאמנת אותו באמצעות הdata של האימון והפרמטרים אשר אנחנו קבענו מראש.

```
1. try:
2.     with open(args.label, "wb") as f:
3.         print("[INFO] Saving labels bin...")
4.         f.write(pickle.dumps(dataset.lb))
5.         f.close()
6. except Exception as e:
7.     print(f"[ERROR] Saving labels failed: {e}")
8. model.save(args.model)
```

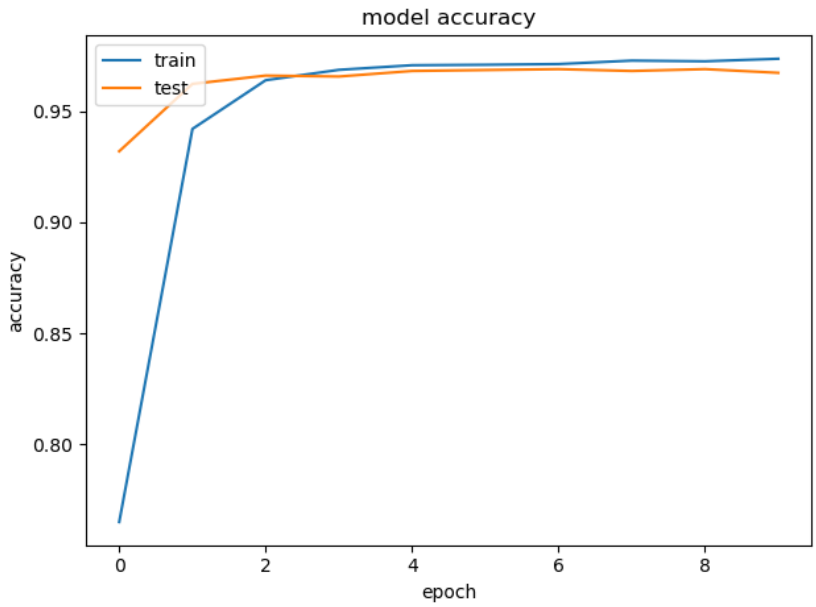
לאחר שאימנו את המודל, אנחנו שומרים אותו ואת התוויות שלו לשימוש עתידי על מנת שלא נצטרך לאמן את המודל כל פעם מחדש באמצעות פעולת ()save של המודל.

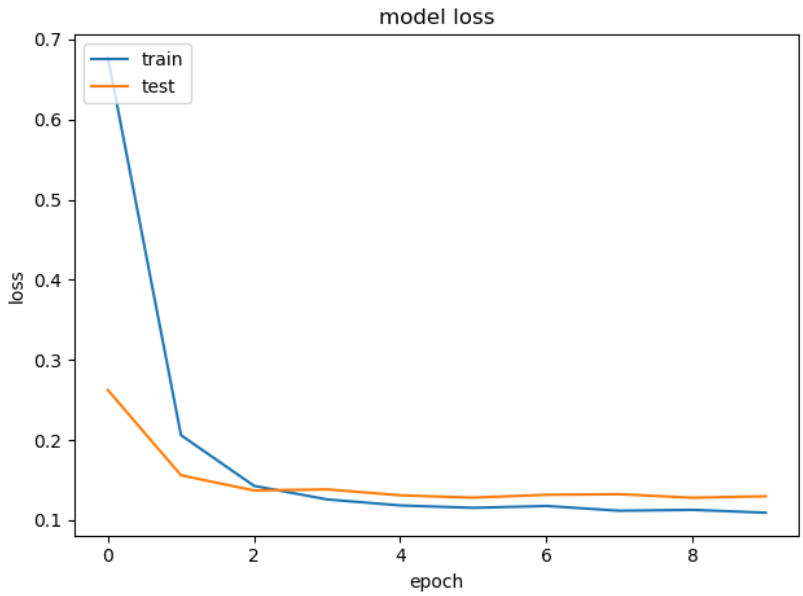
```
1. plt.plot(output.history['accuracy'])
2. plt.plot(output.history['val_accuracy'])
3. plt.title('model accuracy')
4. plt.ylabel('accuracy')
5. plt.xlabel('epoch')
6. plt.legend(['train', 'test'], loc='upper left')
7. plt.show()
8.
9. plt.plot(output.history['loss'])
10. plt.plot(output.history['val_loss'])
11. plt.title('model loss')
```

```
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

לאחר כל פעולות אלו, אנו רוצים לראות בצורה גרפית מה היה הloss שלנו והaccuracy שלנו לכל אורך האימון והEpochs.

דוגמה לגרפים כאלו:





Classifier.py

```
1.  class Classifier:
2.      def __init__(self, model_path, labels_path):
3.          print("[INFO] loading model")
4.          self.model = load_model(model_path)
5.          self.labels = pickle.loads(open(labels_path, "rb").read())
6.          self.prob = 0.0
7.          self.deque = deque(maxlen=50)
8.
9.      def classify(self, frame):
10.         # Resize frame for classification
11.         frame = cv2.resize(frame, (64, 64)) / 255.0
12.         frame = img_to_array(frame)
13.         frame = np.expand_dims(frame, axis=0)
14.
15.         # Classify the image
16.         probabilities = self.model.predict(frame)[0]
17.         self.deque.append(probabilities)
18.         result = np.array(self.deque).mean(axis=0)
19.
20.         index = np.argmax(result)
21.         self.prob = probabilities[index]
22.         result = self.labels.classes_[index]
23.
24.         return result
```

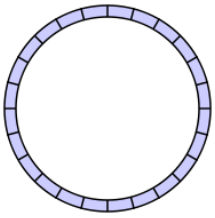
קלאס זה מתאר אובייקט בשם Classifier, אשר כולל מספר תכונות

- המודל המאומן
- תווית האימון
- הסתברות אשר משתנה אחרי כל פריים
- תור מעגלי בגודל קבוע של 50

בפעולה בשם classify מתבצעים מספר פעולות אשר אני אתאר את סדר פעילותם:

1. משנה את הפריים לגודל אשר מתאים למודל.
2. חוזה את הפלט של המודל באמצעות פעולת predict של המודל.
3. מוסיפה את ההסתברות לתור מעגלי.
4. מחשבת את הממוצע של התור המעגלי, ועל פי הממוצע מוציאה תוצאה ממוצעת אשר היא תהיה הprediction הסופי.

הסיבה שבגללה יישמתי תור מעגלי הינה בשל כך שרשתות נוירונים מסוג זה נוטות להיות לא יציבות מבחינת התוצאות ולכן על מנת מלהמנע מאי יציבות אנו מיישמים תור אשר מחשב את הממוצע הכולל.



תמונה אשר מתארת את סוג התור שאנו מיישמים פה.

Misc Pacakge

בתוך חבילה זאת יש מספר מחלקות אשר כוללות בעיקר כלים קטנים אשר נועדו לעזור להוציא מידע ולהמירו.

Get_frames_out_video.py

```
1.  def get_frames_out(video_path, output_path):
2.      cap = cv2.VideoCapture(video_path)
3.      i = 0
4.      while cap.isOpened():
5.          ret, frame = cap.read()
6.          # extract the frame and apply filter
7.          if ret is True:
8.              # Apply contours filter
9.              frame = process_image_gaussian(frame)
10.
11.             cv2.imwrite(f"{output_path}\\{i}.jpg", frame)
12.         else:
13.             cap.release()
14.         i += 1
```

מחלקה הכוללת את הפונקציה הנלווית הנל, אשר מקבלת סרטון שצולם מראש ועל כל פריים בסרטון מפעילה את אפקט עיבוד התמונה ושומרת את התמונה כjpg.

Images_to_dataset.py

```
1.  class ArrowsDataset:
2.      def __init__(self, path_to_dataset, img_dims):
3.          """
4.
5.          Args:
6.              path_to_dataset: Path of dataset
7.              img_dims: Dimensions of image
8.          """
9.          self.path = path_to_dataset
10.         self.img_dims = img_dims
11.         self.data = None
12.         self.labels = None
13.         self.lb = None
14.
15.     def load_data(self):
16.         """
17.         Returns: Tuple of data splitted to train and test
18.         """
19.         data = []
20.         labels = []
21.         print("[INFO] Loading images...")
22.         images_paths = sorted(list(list_images(self.path)))
23.
24.         loop = tqdm(total=len(images_paths), position=0, leave=False)
25.
26.         random.seed(69)
27.         random.shuffle(images_paths)
28.
29.         for i, imagePath in enumerate(images_paths):
30.             img = cv2.imread(imagePath)
31.             img = cv2.resize(img, self.img_dims)
32.             img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
33.             img = img_to_array(img)
34.             data.append(img)
35.
36.             label = imagePath.split(os.path.sep)[-
    2]  # get the label according to folder name
```

```
            labels.append(label)

            loop.set_description("loading...".format(i))
            loop.update(1)

        # Make the images range from 0 to 1
        data = np.array(data, dtype=float) / 255.0

        # This will make the labels in binary form, [up,down,left,right] => array([[0, 0, 0, 1],
        #
        [1, 0, 0, 0],
        #
        [0, 1, 0, 0],
        #
        [0, 0, 1, 0]])
        lb = LabelBinarizer()
        labels = lb.fit_transform(labels)

        self.data = data
        self.labels = labels
        self.lb = lb

        x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=69)
        print("[INFO] Finished loading images")

        return (x_train, y_train), (x_test, y_test)
```

במחלקה זאת קוראים מספר דברים, אנחנו יוצרים אובייקט חדש בשם ArrowDataset אשר כולל את התכונה של הנתיב לספריה, וגודל התמונה.

הפונקציה load data פותחת את התמונות מהתקייה אשר מאחסנת את התמונות, משתמש בספריה בשם tqdm אשר עושה לי קו טעינה שמציין את ההתקדמות של הטעינת תמונות, ועל כל תמונה לוקח אותה ומקטין אותה לגודל אשר מתאים לרשת נוירונים ומכניס אותה לרשימה אשר כוללת את כל התמונות. לאחר מכן אנו ממירים את הרשימה לאובייקט מסוג ndarray של numpy ומחלקים ב255 על מנת שהערך של כל פיקסל יהיה בין 0 ל 1. לאחר שעשינו את כל זה אנחנו מעבדים את התווית באמצעות LabelBinarizer של scipy, ואז מחלקים את הדאטה באופן רנדומלי לtest data ו train data. ומחזירים את זה כ tuple שכולל את זה ככה.

Recorder.py

מחלקה זאת אחראית על הקלטת הסרטונים מהרחפן ושמירתם במחשב.

```
1.  class Recorder:
2.      def __init__(self, video_path, fps, video_time):
3.          self.is_timer_over = False
4.          self.recorder = gbv.OpenCVRecorder(video_path, fps)
5.          self.video_time = video_time
6.
7.      def change_timer_status(self):
8.          self.is_timer_over = not self.is_timer_over
9.
10.     def run(self):
11.         tello = TelloController()
12.         tello.start_stream()
13.         video = TelloVideoReceiver()
14.         time.sleep(5)
15.         timer = threading.Timer(self.video_time, self.change_timer_status)
16.         while True:
17.             if video.isOpened():
18.                 timer.start()
19.                 break
20.         while True:
21.             result, frame = video.read()
22.             if result and not self.is_timer_over:
23.                 cv2.imshow('title', frame)
24.                 cv2.waitKey(1)
25.                 self.recorder.record(frame)
26.             if self.is_timer_over:
27.                 video.release()
28.                 break
29.
30.
31. recorder = Recorder(os.path.join(DATASET_PATH, 'left.avi'), 25, 15)
32. recorder.run()
```

באמצעות מחלקה זאת אנחנו מפעילים את הרחפן באמצעות TelloController ומפעילים את הסטרים, ואומרים לתוכנה את הדבר הבא: יש טיימר, כל עוד הטיימר פועל תיקח את הפריים שמגיע מהמצלמה ותכניס אותו לתוך סרטון, ברגע שהטיימר נגמר הוא שומר את הסרטון ומשחרר את המצלמה.

Test_vision.py

```
1.  while True:
2.      status, frame = receiver.read()
3.      copy_frame = process_image_gaussian(frame)
4.      label = classifier.classify(copy_frame)
5.
6.      # build the label and draw the label on the image
7.      label = "{}: {:.2f}%".format(label, classifier.prob * 100)
8.      frame = imutils.resize(frame, width=400)
9.      cv2.putText(frame, label, (10, 25), cv2.FONT_HERSHEY_SIMPLEX,
10.                 0.7, (0, 255, 0), 2)
11.     cv2.imshow("Output", frame)
12.     cv2.imshow("cam", copy_frame)
13.     cv2.waitKey(1)
```

מחלקה אשר עיקרה הינו לבדוק את תקינות הויזן באמצעות קבלת הוידאו מהמצלמה ונסיון זיהוי התמונה באמצעות הפעולה process_image_gaussian ועליה כתיבת הכיוון והסבירות שזהו הכיוון.

הקוד המלא

הקוד

Main.py

```
from controller.opencvcontrol import MainController


def main():
    controller = MainController()
    controller.run()


if __name__ == '__main__':
    main()
```

constants.py

```
from gbvision import ColorThreshold
import os

THRESHOLD = ColorThreshold([[169, 189], [84, 255], [42, 242]], 'HSV')

MODEL_PATH = os.path.abspath('C:\\Users\\Ofek\\Desktop\\coding\\TelloVision'
)
LABELS_PATH = os.path.abspath('C:\\Users\\Ofek\\Desktop\\coding\\TelloVision
\\label.pickle')
DATASET_PATH = os.path.abspath('dataset')

IMAGE_DIMS = (64, 64)
BATCH_SIZE = 32
LR = 1e-3
EPOCHS = 10
LOSS = "categorical_crossentropy"

HORIZONTAL_SPEED = 35
VERTICAL_SPEED = 28
FPS = 120
```

Controller/__init__.py

```
from .drone import Drone
from .tello import TelloController, TelloVideoReceiver
```

Controller/drone.py

```
import abc


class Drone(abc.ABC):

    @abc.abstractmethod
    def _connect(self) -> bool:
        """
        connecting the drone

        Returns:
            True if responded, false if not
        """
        pass

    @abc.abstractmethod
```

```
17.     def send_command(self, command: str, timeout: float) -> bool:
18.         """Abstract method for command sending to the drone
19.         Args:
20.             command : the command you would like to send
21.             timeout : Command timeout
22.
23.         Returns:
24.             Response from the drone
25.
26.
27.         """
28.         pass
29.
30.     @abc.abstractmethod
31.     def takeoff(self):
32.         """
33.         Initiate takeoff
34.         """
35.         pass
36.
37.     @abc.abstractmethod
38.     def land(self):
39.         pass
40.
41.     @abc.abstractmethod
42.     def move(self, direction: str, x: int):
43.         pass
44.
45.     @abc.abstractmethod
46.     def emergency(self):
47.         pass
48.
49.     @abc.abstractmethod
50.     def stop(self):
51.         pass
52.
53.     def move_up(self, x: int):
54.         return self.move("up", x)
55.
56.     def move_down(self, x: int):
57.         return self.move("down", x)
58.
59.     def move_right(self, x: int):
60.         return self.move("right", x)
61.
62.     def move_left(self, x: int):
63.         return self.move("left", x)
64.
65.     def move_forward(self, x: int):
66.         return self.move("forward", x)
67.
68.     def move_backward(self, x: int):
69.         return self.move("back", x)
70.
71.     @abc.abstractmethod
72.     def rotate_clockwise(self, x: int):
73.         pass
74.
75.     @abc.abstractmethod
76.     def rotate_counter_clockwise(self, x: int):
77.         pass
78.
79.     @abc.abstractmethod
80.     def go_to(self, x: int, y: int, z: int, speed: int):
81.         pass
82.
```

```
    @abc.abstractmethod
    def go_to_curve(self, x1: int, x2: int, y1: int, y2: int, z1: int, z2: int, speed: int):
        pass
```

Controller/opencvcontroller.py

```
from pynput import keyboard
from controller.tello import TelloVideoReceiver, TelloController
from vision.CNN.clasifier import Classifier
import cv2
import time
import threading
from constants import MODEL_PATH, LABELS_PATH, FPS, HORIZONTAL_SPEED, VERTICAL_SPEED
from vision.vision import process_image_gaussian


def set_interval(func, sec):
    def func_wrapper():
        set_interval(func, sec)
        func()
    t = threading.Timer(sec, func_wrapper)
    t.start()
    return t


class MainController:
    def __init__(self):
        self.tello = TelloController()
        self.video = TelloVideoReceiver()
        self.classifier = Classifier(MODEL_PATH, LABELS_PATH)

        self.is_vision_control = False
        self.should_stop = False
        self.send_rc_control = False

        self.for_back_velocity = 0
        self.left_right_velocity = 0
        self.up_down_velocity = 0
        self.yaw_velocity = 0

    def run(self):
        """
        Runs the program based on vision or manual control.
        always starting with manual control.
        Manual control keys:
                    w - up
         a - yaw left  s - down  d - yaw right
                    ↑ - forward
         ← - left  ↓ - back  → - right
         l - land
         enter - emergency
         escape - quit
         space - switch between vision and manual control

        Returns:

        """
        listener = keyboard.Listener(
            on_press=self.keydown,
            on_release=self.keyup
        )
        listener.start()
```

```
57.
58.         self.tello.stop_stream()
59.         self.tello.start_stream()
60.
61.         if not self.video.isOpened():
62.             open_video = self.video.open(self.video.address)
63.             time.sleep(5)
64.             if not open_video:
65.                 return False
66.
67.         status, frame = self.video.read()
68.         set_interval(self.update, ((1000 // FPS) / 1000))
69.         #################### Manual Control ####################
70.         while True:
71.             while not self.should_stop and not self.is_vision_control:
72.                 if not status:
73.                     break
74.                 status, frame = self.video.read()
75.                 text = "Battery: {}%".format(self.tello.state['bat'])
76.                 cv2.putText(frame, text, (5, 720 - 5),
77.                             cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
78.
79.                 cv2.imshow('Video', frame)
80.                 cv2.waitKey(1)
81.
82.             #################### Vision Control ####################
83.             classifier = self.classifier
84.             direction = None
85.             engine = None  # To be in control of which direction we need to
    make speed 0
86.
87.             while not self.should_stop and self.is_vision_control:
88.
89.                 # Screen output and reading from stream
90.                 status, frame = self.video.read()
91.                 copy_frame = frame.copy()
92.                 frame = process_image_gaussian(frame)
93.
94.                 text = "Battery: {}%".format(self.tello.state['bat'])
95.                 cv2.putText(copy_frame, text, (5, 720 - 5),
96.                             cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
97.                 label = "{}: {:.2f}%".format(direction, classifier.prob * 10
    0)
98.                 cv2.putText(copy_frame, label, (10, 25), cv2.FONT_HERSHEY_SI
    MPLEX,
99.                             0.7, (0, 255, 0), 2)
100.
101.                        # handle the engines so we know what was last directi
    on
102.                        if direction == 'up' or direction == 'down':
103.                            engine = 'up_down'
104.                        elif direction == 'left' or 'right':
105.                            engine = 'left_right'
106.
107.                        # Vision movement
108.                        direction = classifier.classify(frame)
109.                        # direction = classifier.classify(frame) if classifie
    r.prob > 40.0 else 'random'
110.
111.                        if direction == 'up':
112.                            if engine == 'left_right':
113.                                self.left_right_velocity = 0
114.                            self.up_down_velocity = HORIZONTAL_SPEED
115.                            print("UP")
116.
117.                        elif direction == 'down':
```

```
118.                            if engine == 'left_right':
119.                                self.left_right_velocity = 0
120.                            self.up_down_velocity = -HORIZONTAL_SPEED
121.                            print("DOWN")
122.
123.                        elif direction == 'left':
124.                            if engine == 'up_down':
125.                                self.up_down_velocity = 0
126.                            self.left_right_velocity = -VERTICAL_SPEED
127.                            print("LEFT")
128.
129.                        elif direction == 'right':
130.                            if engine == 'up_down':
131.                                self.up_down_velocity = 0
132.                            self.left_right_velocity = VERTICAL_SPEED
133.                            print("RIGHT")
134.
135.                        elif direction == 'random':
136.                            self.left_right_velocity = 0
137.                            self.up_down_velocity = 0
138.
139.                        cv2.imshow('After vision', copy_frame)
140.                        cv2.waitKey(1)
141.
142.                        if not status:
143.                            break
144.                    if self.should_stop:
145.                        break
146.                self.tello.land()
147.                self.tello.stop_stream()
148.                self.video.release()
149.
150.            def update(self):
151.                if self.send_rc_control:
152.                    self.tello.set_rc(self.left_right_velocity, self.for_back
    _velocity,
153.                                      self.up_down_velocity, self.yaw_velocit
    y)
154.
155.            def keydown(self, key):
156.                if key == keyboard.Key.up:  # set forward velocity
157.                    self.for_back_velocity = HORIZONTAL_SPEED
158.                elif key == keyboard.Key.down:  # set backward velocity
159.                    self.for_back_velocity = -HORIZONTAL_SPEED
160.                elif key == keyboard.Key.left:  # set left velocity
161.                    self.left_right_velocity = -VERTICAL_SPEED
162.                elif key == keyboard.Key.right:  # set right velocity
163.                    self.left_right_velocity = VERTICAL_SPEED
164.                elif key == keyboard.KeyCode.from_char('w'):  # set up veloci
    ty
165.                    self.up_down_velocity = HORIZONTAL_SPEED
166.                elif key == keyboard.KeyCode.from_char('s'):  # set down velo
    city
167.                    self.up_down_velocity = -HORIZONTAL_SPEED
168.                elif key == keyboard.KeyCode.from_char('a'):  # set yaw count
    er clockwise velocity
169.                    self.yaw_velocity = -HORIZONTAL_SPEED
170.                elif key == keyboard.KeyCode.from_char('d'):  # set yaw clock
    wise velocity
171.                    self.yaw_velocity = HORIZONTAL_SPEED
172.                elif key == keyboard.Key.esc:
173.                    self.should_stop = True
174.                elif key == keyboard.Key.enter:
175.                    self.tello.emergency()
176.                elif key == keyboard.Key.space:
```

```
                    self.is_vision_control = not self.is_vision_control  # Switch between vision and manual controller

            def keyup(self, key):
                if key == keyboard.Key.down or key == keyboard.Key.up:  # set zero forward/backward velocity
                    self.for_back_velocity = 0
                elif key == keyboard.Key.right or key == keyboard.Key.left:  # set zero left/right velocity
                    self.left_right_velocity = 0
                elif key == keyboard.KeyCode.from_char('w') or key == keyboard.KeyCode.from_char(
                        's'):  # set zero up/down velocity
                    self.up_down_velocity = 0
                elif key == keyboard.KeyCode.from_char('a') or key == keyboard.KeyCode.from_char('d'):  # set zero yaw velocity
                    self.yaw_velocity = 0
                elif key == keyboard.KeyCode.from_char('t'):  # takeoff
                    if not self.tello.takeoff():
                        print("[ERROR] Takeoff failed")
                    else:
                        self.send_rc_control = True
                elif key == keyboard.KeyCode.from_char('l'):  # land
                    self.tello.land()
                    self.send_rc_control = False
```

Controller/pygamecontroller.py

```
from controller.tello import TelloVideoReceiver, TelloController
from vision.CNN.clasifier import Classifier
import pygame
import cv2
import numpy as np
import time
import threading
import abc
from constants import MODEL_PATH, LABELS_PATH, FPS, HORIZONTAL_SPEED
from vision.vision import process_image_gaussian


class Controller(abc.ABC):
    @abc.abstractmethod
    def run(self):
        pass


class ManualController(Controller):

    def __init__(self, tello: TelloController, stream: TelloVideoReceiver, screen):
        super().__init__()
        pygame.display.set_caption("Tello video stream")
        self.screen = screen

        self.tello = tello
        self.video = stream
        self.event = None
        self.in_control = True
        self.send_rc_control = False

        self.for_back_velocity = 0
        self.left_right_velocity = 0
        self.up_down_velocity = 0
        self.yaw_velocity = 0
```

```
36.
37.         pygame.time.set_timer(pygame.USEREVENT + 1, 1000 // FPS)  # Update T
    imer
38.
39.     def run(self):
40.         self.tello.stop_stream()
41.         self.tello.start_stream()
42.
43.         if not self.video.isOpened():
44.             open_video = self.video.open(self.video.address)
45.             time.sleep(5)
46.             if not open_video:
47.                 return False
48.
49.         status, frame = self.video.read()
50.
51.         should_stop = False
52.         while not should_stop and self.in_control:
53.             for event in pygame.event.get():
54.                 if event.type == pygame.USEREVENT + 1:
55.                     self.update()
56.                 elif event.type == pygame.QUIT:
57.                     should_stop = True
58.                 elif event.type == pygame.KEYDOWN:
59.                     self.event = event
60.                     if event.key == pygame.K_ESCAPE:
61.                         should_stop = True
62.                     else:
63.                         self.keydown(event.key)
64.                 elif event.type == pygame.KEYUP:
65.                     self.keyup(event.key)
66.
67.             if not status:
68.                 break
69.
70.             self.screen.fill([0, 0, 0])
71.
72.             status, frame = self.video.read()
73.             text = "Battery: {}%".format(self.tello.state['bat'])
74.             cv2.putText(frame, text, (5, 720 - 5),
75.                         cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
76.             frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
77.             frame = np.rot90(frame)
78.             frame = np.flipud(frame)
79.
80.             background = pygame.surfarray.make_surface(frame)
81.             self.screen.blit(background, (0, 0))
82.             pygame.display.update()
83.
84.             time.sleep(1 / FPS)
85.         if self.in_control:
86.             self.tello.land()
87.             self.tello.stop_stream()
88.             self.video.release()
89.
90.     def update(self):
91.         if self.in_control:  # If in control
92.             if self.send_rc_control:
93.                 self.tello.set_rc(self.left_right_velocity, self.for_back_ve
    locity,
94.                                   self.up_down_velocity, self.yaw_velocity)
95.
96.     def keydown(self, key_event):
97.         if key_event == pygame.K_UP:  # set forward velocity
98.             self.for_back_velocity = HORIZONTAL_SPEED
```

```
99.         elif key_event == pygame.K_DOWN:  # set backward velocity
100.                    self.for_back_velocity = -HORIZONTAL_SPEED
101.                elif key_event == pygame.K_LEFT:  # set left velocity
102.                    self.left_right_velocity = -HORIZONTAL_SPEED
103.                elif key_event == pygame.K_RIGHT:  # set right velocity
104.                    self.left_right_velocity = HORIZONTAL_SPEED
105.                elif key_event == pygame.K_w:  # set up velocity
106.                    self.up_down_velocity = HORIZONTAL_SPEED
107.                elif key_event == pygame.K_s:  # set down velocity
108.                    self.up_down_velocity = -HORIZONTAL_SPEED
109.                elif key_event == pygame.K_a:  # set yaw counter clockwise ve
    locity
110.                    self.yaw_velocity = -HORIZONTAL_SPEED
111.                elif key_event == pygame.K_d:  # set yaw clockwise velocity
112.                    self.yaw_velocity = HORIZONTAL_SPEED
113.
114.            def keyup(self, key_event):
115.
116.                if key_event == pygame.K_UP or key_event == pygame.K_DOWN:  #
     set zero forward/backward velocity
117.                    self.for_back_velocity = 0
118.                elif key_event == pygame.K_LEFT or key_event == pygame.K_RIGH
    T:  # set zero left/right velocity
119.                    self.left_right_velocity = 0
120.                elif key_event == pygame.K_w or key_event == pygame.K_s:  # s
    et zero up/down velocity
121.                    self.up_down_velocity = 0
122.                elif key_event == pygame.K_a or key_event == pygame.K_d:  # s
    et zero yaw velocity
123.                    self.yaw_velocity = 0
124.                elif key_event == pygame.K_t:  # takeoff
125.                    if not self.tello.takeoff():
126.                        print("[ERROR] Takeoff failed")
127.                    else:
128.                        self.send_rc_control = True
129.                elif key_event == pygame.K_l:  # land
130.                    not self.tello.land()
131.                    self.send_rc_control = False
132.
133.
134.        class VisionController(Controller):
135.            def __init__(self, tello: TelloController, video: TelloVideoRecei
    ver, screen):
136.                super().__init__()
137.                self.tello = tello
138.                self.video = video
139.                self.classifier = Classifier(MODEL_PATH, LABELS_PATH)
140.                self.in_control = False
141.                self.screen = screen
142.                self.result = None
143.                self.event = None
144.
145.                self.yaw_velocity = None
146.                self.left_right_velocity = None
147.                self.for_back_velocity = None
148.                self.up_down_velocity = None
149.
150.            def run(self):
151.                should_stop = False
152.                classifier = self.classifier
153.                direction = None
154.                engine = None  # To be in control of which direction we need
    to make speed 0
155.
156.                while not should_stop and self.in_control:
157.                    for event in pygame.event.get():
```

```
158.                        # Pygame events
159.                        if event.type == pygame.USEREVENT + 1:
160.                            self.update()
161.                        elif event.type == pygame.QUIT:
162.                            should_stop = True
163.                        elif event.type == pygame.KEYDOWN:
164.                            self.event = event
165.                            if event.key == pygame.K_ESCAPE:
166.                                should_stop = True
167.
168.                    self.screen.fill([0, 0, 0])
169.
170.                    # Screen output and reading from stream
171.                    status, frame = self.video.read()
172.                    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
173.                    frame = np.rot90(frame)
174.                    frame = np.flipud(frame)
175.                    copy_frame = frame.copy()
176.                    frame = process_image_gaussian(frame)
177.
178.                    text = "Battery: {}%".format(self.tello.state['bat'])
179.                    cv2.putText(copy_frame, text, (5, 720 - 5),
180.                                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

181.                    label = "{}: {:.2f}%".format(direction, classifier.prob *
    100)
182.                    cv2.putText(copy_frame, label, (10, 25), cv2.FONT_HERSHEY
    _SIMPLEX,
183.                                0.7, (0, 255, 0), 2)
184.
185.                    # handle the engines so we know what was last direction
186.                    if direction == 'up' or direction == 'down':
187.                        engine = 'up_down'
188.                    elif direction == 'left' or 'right':
189.                        engine = 'left_right'
190.
191.                    # Vision movement
192.                    direction = classifier.classify(frame) if classifier.prob
    > 70.0 else 'background'
193.
194.                    if direction == 'up':
195.                        if engine == 'left_right':
196.                            self.left_right_velocity = 0
197.                        self.up_down_velocity = HORIZONTAL_SPEED
198.                        print("UP")
199.
200.                    elif direction == 'down':
201.                        if engine == 'left_right':
202.                            self.left_right_velocity = 0
203.                        self.up_down_velocity = -HORIZONTAL_SPEED
204.                        print("DOWN")
205.
206.                    elif direction == 'left':
207.                        if engine == 'up_down':
208.                            self.up_down_velocity = 0
209.                        self.left_right_velocity = -HORIZONTAL_SPEED
210.                        print("LEFT")
211.
212.                    elif direction == 'right':
213.                        if engine == 'up_down':
214.                            self.up_down_velocity = 0
215.                        self.left_right_velocity = HORIZONTAL_SPEED
216.                        print("RIGHT")
217.
218.                    elif direction == 'background':
219.                        self.left_right_velocity = 0
```

```
                        self.up_down_velocity = 0

                    background = pygame.surfarray.make_surface(frame)
                    self.screen.blit(background, (0, 0))
                    pygame.display.update()

                    if not status:
                        break

                if self.in_control:
                    self.tello.land()
                    self.tello.stop_stream()

            def update(self):
                if self.in_control:  # If in control
                    self.tello.set_rc(self.left_right_velocity, self.for_back_velocity,
                                      self.up_down_velocity, self.yaw_velocity)


        class MainController:
            def __init__(self):
                self.tello = TelloController()
                self.video = TelloVideoReceiver()

                self.screen = pygame.display.set_mode([960, 720])

                self.manual_controller = ManualController(self.tello, self.video, self.screen)
                self.vision_controller = VisionController(self.tello, self.video, self.screen)

                thread = threading.Thread(target=self.check_and_switch_controllers, daemon=True)
                thread.start()

            def check_and_switch_controllers(self):
                while True:
                    # check who's controlling the drone
                    control = self.manual_controller if self.manual_controller.in_control else self.vision_controller
                    if control.event is not None:
                        if control.event.type == pygame.KEYDOWN:
                            if control.event.key == pygame.K_SPACE:
                                self.manual_controller.in_control = not self.manual_controller.in_control
                                self.vision_controller.in_control = not self.vision_controller.in_control
                            elif control.event.key == pygame.K_RETURN:
                                self.tello.emergency()

            def run(self):
                self.manual_controller.run()
                self.vision_controller.run()

```

Controller/tello.py

```
import socket
import threading
from controller.drone import Drone
```

```
4.  import cv2
5.  from typing import Dict
6.  import time
7.
8.
9.  class TelloController(Drone):
10.     """
11.     Interact simply with tello drone
12.     """
13.
14.     def __init__(self, drone_ip="192.168.10.1", drone_port=8889, local_ip='0
    .0.0.0', local_port=5809,
15.                  command_timeout=30, control_timeout=5):
16.         self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
17.         self.socket.bind((local_ip, local_port))
18.         self.drone_address = (drone_ip, drone_port)
19.         self.response = None
20.         self.state = None
21.
22.         self.receive_thread = threading.Thread(target=self.__thread_handler)

23.         self.receive_thread.daemon = True
24.         self.receive_thread.start()
25.
26.         self.state_recv = threading.Thread(target=self.get_state, daemon=Tru
    e)
27.         self.state_recv.start()
28.
29.         self.command_timeout = command_timeout
30.         self.control_timeout = control_timeout
31.
32.         self.error_flag = False
33.
34.         if not self._connect():
35.             raise RuntimeError("Drone hasn't returned a response")
36.
37.     def __thread_handler(self):
38.         """
39.         Receive responses from drone and places them in self.response
40.
41.         Returns: None
42.         """
43.         while True:
44.             try:
45.                 self.response = self.socket.recv(256)
46.             except Exception:
47.                 break
48.
49.     def _set_error_flag(self):
50.         self.error_flag = not self.error_flag
51.
52.     def send_command(self, command: str, timeout: float) -> bool:
53.         """
54.         Send command to Tello and wait for response
55.         Args:
56.             timeout (float): Command timeout
57.             command (bool): Command to send
58.
59.         Returns:
60.              (bool): True if command succeeded, False if not
61.         """
62.         self.error_flag = False
63.         timer = threading.Timer(timeout, self._set_error_flag)
64.         self.socket.sendto(command.encode('utf-8'), self.drone_address)
65.
66.         timer.start()
```

```
67.         while self.response is None:
68.             if self.error_flag:
69.                 raise RuntimeError("command timed out")
70.         timer.cancel()
71.
72.         response = self.response.decode('utf-8')
73.         self.response = None
74.         if response == 'ok':
75.             return True
76.         elif response == 'error':
77.             return False
78.         return False
79.
80.     def send_command_without_response(self, command: str):
81.         """
82.             Send command to Tello and wait for response
83.             Args:
84.                 command (bool): Command to send
85.         """
86.         self.socket.sendto(command.encode('utf-8'), self.drone_address)
87.
88.     def _connect(self) -> bool:
89.         """
90.         Connect to tello sdk
91.
92.         Returns(bool): True if connected, False if failed
93.         """
94.         return self.send_command("command", self.command_timeout)
95.
96.     def takeoff(self):
97.         return self.send_command("takeoff", self.command_timeout)
98.
99.     def land(self):
100.                return self.send_command("land", self.command_timeout)
101.
102.            def emergency(self):
103.                return self.send_command("emergency", self.command_timeout)
104.
105.            def stop(self):
106.                return self.send_command("stop", self.control_timeout)
107.
108.            def start_stream(self):
109.                return self.send_command("streamon", self.command_timeout)
110.
111.            def stop_stream(self):
112.                return self.send_command("streamoff", self.command_timeout)
113.
114.            def move(self, direction: str, x: int):
115.                return self.send_command(f"{direction} {x}", self.control_tim
    eout)
116.
117.            def rotate_clockwise(self, x: int):
118.                return self.send_command(f"cw {x}", self.control_timeout)
119.
120.            def rotate_counter_clockwise(self, x: int):
121.                return self.send_command(f"ccw {x}", self.control_timeout)
122.
123.            def go_to(self, x: int, y: int, z: int, speed: int):
124.                return self.send_command(f"go {x} {y} {z} {speed}", self.cont
    rol_timeout)
125.
126.            def go_to_curve(self, x1: int, x2: int, y1: int, y2: int, z1: int
    , z2: int, speed: int):
127.                return self.send_command(f"curve {x1} {y1} {z1} {x2} {y2} {z2
    } {speed}", self.control_timeout)
128.
```

```python
        def flip(self, direction: str):
            return self.send_command(f"flip {direction}", self.control_timeout)

        def get_state(self):
            """
            Get tello state from tello and place it as a dict in local var state

            """
            sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            sock.bind(('0.0.0.0', 8890))
            while True:
                state_string = sock.recv(1024).decode('utf-8')
                list_states = state_string.split(';')[:-1]
                self.state = dict([i.split(':') for i in list_states])

        def set_rc(self, x: int, y: int, z: int, yaw: int):
            return self.send_command_without_response(f"rc {x} {y} {z} {yaw}")


    class TelloVideoReceiver(cv2.VideoCapture):
        def __init__(self):
            self.address = 'udp://0.0.0.0:11111'
            super().__init__(self.address)

            self.frame = None
            self.status = False

            self.thread = threading.Thread(target=self._thread_handler, daemon=True)  # start new thread for video receiving
            self.thread.start()

        def read(self, image=None):  # reimplementing read function so it would not be accessed by thread
            return self.status, self.frame

        def _thread_handler(self):
            while True:
                if self.isOpened():
                    self.status, self.frame = super().read()
                else:
                    self.status, self.frame = (False, None)
```

Vision/__init__.py

```python
from vision.CNN import CNN
from vision.misc import ArrowsDataset
```

Vision/vision.py

```python
import gbvision as gbv
import cv2
import numpy as np
from constants import THRESHOLD


@gbv.PipeLine
def contours_to_polygons(cnts):
    """
    performs approxPolyDP algorithm on a list of contours

```

```
12.     :param cnts: the list of contours
13.     :return: a list of polygons from the contours
14.     """
15.     arc_lengths = map(lambda cnt: 0.03 * cv2.arcLength(cnt, True), cnts)
16.     return list(map(lambda cnt: cv2.approxPolyDP(cnt, next(arc_lengths), Tru
    e), cnts))
17.
18.
19. def process_image_gaussian(frame):
20.     grayscaled = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
21.     blur = cv2.GaussianBlur(grayscaled, (5, 5), 0)
22.     th = cv2.threshold(blur, 120, 255, cv2.THRESH_BINARY)[1]
23.     th = np.bitwise_not(th)
24.     pipe = gbv.EMPTY_PIPELINE + gbv.find_contours + contours_to_polygons + g
    bv.sort_polygons
25.     list_polygons = pipe(th)
26.     list_polygons_filtered = []
27.     if not len(list_polygons) == 0:
28.         list_polygons = list_polygons[:min(len(list_polygons), 4)]
29.         for poly in list_polygons:
30.             if check_arrow(poly):
31.                 list_polygons_filtered = [poly]
32.                 break
33.     drawn = gbv.draw_contours(np.zeros(frame.shape, np.uint8), list_polygons
    _filtered, (255, 255, 255),
34.                               thickness=cv2.FILLED)
35.     img = cv2.cvtColor(drawn, cv2.COLOR_BGR2GRAY)
36.     return img
37.
38.
39. def process_image_gaussian_adaptive(frame):
40.     grayscaled = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
41.     # blur = cv2.GaussianBlur(grayscaled, (5, 5), 0)
42.     th = cv2.adaptiveThreshold(grayscaled, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN
    _C, cv2.THRESH_BINARY, 205, 1)
43.     th = np.bitwise_not(th)
44.     pipe = gbv.EMPTY_PIPELINE + gbv.find_contours + contours_to_polygons + g
    bv.sort_polygons
45.     list_polygons = pipe(th)
46.     list_polygons_filtered = []
47.     if not len(list_polygons) == 0:
48.         list_polygons = list_polygons[:min(len(list_polygons), 4)]
49.         for poly in list_polygons:
50.             if check_arrow(poly):
51.                 list_polygons_filtered = [poly]
52.                 break
53.     drawn = gbv.draw_contours(np.zeros(frame.shape, np.uint8), list_polygons
    _filtered, (255, 255, 255),
54.                               thickness=cv2.FILLED)
55.     img = cv2.cvtColor(drawn, cv2.COLOR_BGR2GRAY)
56.     return img
57.
58.
59. def process_image_color(frame):
60.     th = THRESHOLD(frame)
61.     pipe = gbv.EMPTY_PIPELINE + gbv.find_contours + contours_to_polygons + g
    bv.sort_polygons
62.     list_polygons = pipe(th)
63.     list_polygons_filtered = []
64.     if not len(list_polygons) == 0:
65.         list_polygons = list_polygons[:min(len(list_polygons), 4)]
66.         for poly in list_polygons:
67.             if check_arrow(poly):
68.                 list_polygons_filtered = [poly]
69.                 break
```

```
    drawn = gbv.draw_contours(np.zeros(frame.shape, np.uint8), list_polygons_filtered, (255, 255, 255),
                              thickness=cv2.FILLED)
    img = cv2.cvtColor(drawn, cv2.COLOR_BGR2GRAY)
    return img


def check_arrow(polygon):
    return polygon.shape[0] == 7
```

Vision/CNN/__init__.py

```
from .model import CNN
```

Vision/CNN/model.py

```
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Activation, Dropout
from tensorflow.keras.models import Sequential


class CNN:
    @staticmethod
    def build(width, height, depth, num_of_outputs):
        """
        Defines custom made CNN that aims to classify between 4 arrows (up, down, left, right)
        Args:
            width: Width of img
            height: Height of image
            depth: How many depth does the image have (1 for greyscale, 3 for RGB)
            num_of_outputs: how many outputs there is
        """
        input_shape = (width, height, depth)

        model = Sequential()

        # First set of layers (CONV => relu => MaxPool)
        model.add(Conv2D(32, (3, 3), input_shape=input_shape))
        model.add(Activation('relu'))
        model.add(MaxPool2D(pool_size=(3, 3)))
        model.add(Dropout(0.25))  # Dropout forth of the neurons

        # Second set of Layers ((CONV => RELU) *2 => MaxPool)
        model.add(Conv2D(64, (3, 3)))
        model.add(Activation('relu'))
        model.add(Conv2D(64, (3, 3)))
        model.add(Activation('relu'))
        model.add(MaxPool2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        # Flatten the model
        model.add(Flatten())
        model.add(Activation("relu"))
        model.add(Dropout(0.5))

        # Final output and applying softmax filter
        model.add(Dense(num_of_outputs))
        model.add(Activation("softmax"))

        return model

```

Vision/CNN/classifier.py

```
1.  from tensorflow.keras.models import load_model
2.  from tensorflow.keras.preprocessing.image import img_to_array
3.  import pickle
4.  from collections import deque
5.  import cv2
6.  import numpy as np
7.
8.
9.  class Classifier:
10.     def __init__(self, model_path, labels_path):
11.         print("[INFO] loading model")
12.         self.model = load_model(model_path)
13.         self.labels = pickle.loads(open(labels_path, "rb").read())
14.         self.prob = 0.0
15.         self.deque = deque(maxlen=50)
16.
17.     def classify(self, frame):
18.         # Resize frame for classification
19.         frame = cv2.resize(frame, (64, 64)) / 255.0
20.         frame = img_to_array(frame)
21.         frame = np.expand_dims(frame, axis=0)
22.
23.         # Classify the image
24.         probabilities = self.model.predict(frame)[0]
25.         self.deque.append(probabilities)
26.         result = np.array(self.deque).mean(axis=0)
27.
28.         index = np.argmax(result)
29.         self.prob = probabilities[index]
30.         result = self.labels.classes_[index]
31.
32.         return result
```

Vision/CNN/train.py

```
1.  from tensorflow.keras.optimizers import Adam
2.  import matplotlib.pyplot as plt
3.  import argparse
4.  import pickle
5.  from vision import CNN
6.  from vision import ArrowsDataset
7.  from constants import IMAGE_DIMS, BATCH_SIZE, EPOCHS, LOSS, LR
8.
9.  # Arguments parser
10. parser = argparse.ArgumentParser()
11. parser.add_argument("-d", "--dataset", help="Path for dataset")
12. parser.add_argument("-m", "--model", help="The path for saving the model")
13. parser.add_argument("-l", "--
    label", help="Path for saving the label binary")
14. args = parser.parse_args()
15.
16.
17.
18. # Load Dataset
19. dataset = ArrowsDataset(args.dataset, IMAGE_DIMS)
20. (x_train, y_train), (x_test, y_test) = dataset.load_data()
21.
```

```
# Init Model and compile
model = CNN().build(IMAGE_DIMS[0], IMAGE_DIMS[1], 1, 5)
opt = Adam(learning_rate=LR)
model.compile(loss=LOSS, optimizer=opt, metrics=['accuracy'])


# Train the model
output = model.fit(x_train, y_train, validation_data=(x_test, y_test), batch_size=BATCH_SIZE, epochs=EPOCHS)

try:
    with open(args.label, "wb") as f:
        print("[INFO] Saving labels bin...")
        f.write(pickle.dumps(dataset.lb))
        f.close()
except Exception as e:
    print(f"[ERROR] Saving labels failed: {e}")
model.save(args.model)

# Plotting
plt.plot(output.history['accuracy'])
plt.plot(output.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

plt.plot(output.history['loss'])
plt.plot(output.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Vision/misc/__init__.py

```
from .images_to_dataset import ArrowsDataset
```

Vision/misc/find_median_threshold.py

```
import cv2
import numpy as np
from constants import THRESHOLD
import gbvision as gbv
from controller.tello import TelloVideoReceiver, TelloController

stdv = np.array([20, 200, 200])


def main():
    # start stream
    drone = TelloController()
    drone.start_stream()
    receive = TelloVideoReceiver()
    window = gbv.StreamWindow('feed', receive)
    while True:
        frame = window.show_and_get_frame()
        k = window.last_key_pressed
        if k == 'r':
            bbox = cv2.selectROI('feed', frame)
```

```
21.             thr1 = gbv.median_threshold(frame, stdv, bbox, gbv.ColorThreshol
    d.THRESH_TYPE_HSV)
22.             thr = gbv.EMPTY_PIPELINE + thr1 \
23.                   + gbv.Dilate(6) + gbv.Erode(2) + gbv.find_contours + gbv.c
    ontours_to_polygons + gbv.sort_polygons
24.             thr2 = gbv.EMPTY_PIPELINE + thr1
25.             break
26.     cv2.destroyAllWindows()
27.
28.     threshold = gbv.StreamWindow('threshold', receive)
29.     threshold.open()
30.     threshold2 = gbv.StreamWindow('thr2', receive)
31.     threshold2.open()
32.
33.     while True:
34.         status, frame = receive.read()
35.         list_contours = thr(frame)
36.         if not len(list_contours) == 0:
37.             list_contours = [list_contours[0]]
38.         drawn = gbv.draw_contours(np.zeros(frame.shape, np.uint8), list_cont
    ours, (255, 255, 255))
39.         if not window.show_frame(frame):
40.             break
41.         if not threshold.show_frame(drawn):
42.             break
43.         if not threshold2.show_frame(thr2(frame)):
44.             break
45.
46.     window.close()
47.     threshold.close()
48.     drone.stop_stream()
49.
50.
51. if __name__ == '__main__':
52.     main()
```

Vision/misc/get_frames_out_video.py

```
1.  import cv2
2.  import argparse
3.  import os
4.  # from constants import THRESHOLD
5.  from vision.vision import process_image_gaussian
6.
7.
8.  def get_frames_out(video_path, output_path):
9.      cap = cv2.VideoCapture(video_path)
10.     i = 0
11.     while cap.isOpened():
12.         ret, frame = cap.read()
13.         # extract the frame and apply filter
14.         if ret is True:
15.             # Apply contours filter
16.             frame = process_image_gaussian(frame)
17.
18.             cv2.imwrite(f"{output_path}\\{i}.jpg", frame)
19.         else:
20.             cap.release()
21.         i += 1
22.
23.
24. if __name__ == '__main__':
```

```
    automate = True
    if automate:
        directions = ['up', 'down', 'right', 'left', 'random']
        for direction in directions:
            get_frames_out(os.path.join(os.getcwd(), f'dataset\\{direction}.avi'),
                           os.path.join(os.getcwd(), f'dataset\\{direction}'))
    else:
        # Argument parser
        parser = argparse.ArgumentParser()
        parser.add_argument("file", help="The path for the file")
        parser.add_argument("-o", "--output", type=str, help="The output directory for the dataset")
        args = parser.parse_args()

        output_path = os.path.join(os.getcwd(), args.output)
        file_path = os.path.join(os.getcwd(), args.file)
        get_frames_out(file_path, output_path)
```

Vision/misc/images_to_dataset.py

```
from imutils.paths import list_images
import cv2
import os
import numpy as np
import random

from sklearn.preprocessing import LabelBinarizer
from tensorflow.keras.preprocessing.image import img_to_array
from sklearn.model_selection import train_test_split
from tqdm import tqdm

"""
Im not sure if i wanna use it because after I programmed this I found out Keras has
tf.keras.preprocessing.image_dataset_from_directory() function who do basically the same
"""


class ArrowsDataset:
    def __init__(self, path_to_dataset, img_dims):
        """

        Args:
            path_to_dataset: Path of dataset
            img_dims: Dimensions of image
        """
        self.path = path_to_dataset
        self.img_dims = img_dims
        self.data = None
        self.labels = None
        self.lb = None

    def load_data(self):
        """
        Returns: Tuple of data splitted to train and test
        """
        data = []
        labels = []
        print("[INFO] Loading images...")
```

```
39.         images_paths = sorted(list(list_images(self.path)))
40.
41.         loop = tqdm(total=len(images_paths), position=0, leave=False)
42.
43.         random.seed(69)
44.         random.shuffle(images_paths)
45.
46.         for i, imagePath in enumerate(images_paths):
47.             img = cv2.imread(imagePath)
48.             img = cv2.resize(img, self.img_dims)
49.             img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
50.             img = img_to_array(img)
51.             data.append(img)
52.
53.             label = imagePath.split(os.path.sep)[-
    2]  # get the label according to folder name
54.             labels.append(label)
55.
56.             loop.set_description("loading...".format(i))
57.             loop.update(1)
58.
59.         # Make the images range from 0 to 1
60.         data = np.array(data, dtype=float) / 255.0
61.
62.         # This will make the labels in binary form, [up,down,left,right] =>
    array([[0, 0, 0, 1],
63.         #
           [1, 0, 0, 0],
64.         #
           [0, 1, 0, 0],
65.         #
           [0, 0, 1, 0]])
66.         lb = LabelBinarizer()
67.         labels = lb.fit_transform(labels)
68.
69.         self.data = data
70.         self.labels = labels
71.         self.lb = lb
72.
73.         x_train, x_test, y_train, y_test = train_test_split(data, labels, te
    st_size=0.2, random_state=69)
74.         print("[INFO] Finished loading images")
75.
76.         return (x_train, y_train), (x_test, y_test)
```

Vision/misc/recorder.py

```
1.  import gbvision as gbv
2.  from constants import DATASET_PATH, FPS
3.  from controller import TelloVideoReceiver, TelloController
4.  import threading
5.  import cv2
6.  import os
7.  import time
8.
9.
10. class Recorder:
11.     def __init__(self, video_path, fps, video_time):
12.         self.is_timer_over = False
13.         self.recorder = gbv.OpenCVRecorder(video_path, fps)
14.         self.video_time = video_time
15.
```

```
    def change_timer_status(self):
        self.is_timer_over = not self.is_timer_over

    def run(self):
        tello = TelloController()
        tello.start_stream()
        video = TelloVideoReceiver()
        time.sleep(5)
        timer = threading.Timer(self.video_time, self.change_timer_status)
        while True:
            if video.isOpened():
                timer.start()
                break
        while True:
            result, frame = video.read()
            if result and not self.is_timer_over:
                cv2.imshow('title', frame)
                cv2.waitKey(1)
                self.recorder.record(frame)
            if self.is_timer_over:
                video.release()
                break


recorder = Recorder(os.path.join(DATASET_PATH, 'left.avi'), 25, 15)
recorder.run()
```

Vision/misc/test_vision.py

```
from tensorflow.keras.models import load_model
from controller.tello import TelloVideoReceiver, TelloController
import time
import imutils
import cv2
from vision.vision import process_image_gaussian, process_image_gaussian_adaptive, process_image_color
from vision.CNN.clasifier import Classifier
from constants import MODEL_PATH, LABELS_PATH

tello = TelloController()
tello.start_stream()
receiver = TelloVideoReceiver()
time.sleep(5)
classifier = Classifier(MODEL_PATH, LABELS_PATH)

while True:
    status, frame = receiver.read()
    copy_frame = process_image_gaussian(frame)
    label = classifier.classify(copy_frame)

    # build the label and draw the label on the image
    label = "{}: {:.2f}%".format(label, classifier.prob * 100)
    frame = imutils.resize(frame, width=400)
    cv2.putText(frame, label, (10, 25), cv2.FONT_HERSHEY_SIMPLEX,
                0.7, (0, 255, 0), 2)
    cv2.imshow("Output", frame)
    cv2.imshow("cam", copy_frame)
    cv2.waitKey(1)
```