עיבוד תמונה - קובץ הסברים לפרויקט סוף

מגישים:
אביב זבולוני ת.ז. 211313333
אליה אטלן ת.ז. 318757200

הרצה:
בשביל להריץ את הקוד פשוט יש להריץ את הקובץ coins_summation.py באופן הבא:
python coins_summation.py

בשביל להריץ על תמונה אחרת שלא קיימת במאגר תמונות ששלחת לנו. ניתן להוסיף אותה לתיקייה imgs ולקרוא לה בשם 00 כדי שתופיע ראשונה בריצה.

אסטרטגיה:
נפעל ב 4 שלבים על מנת לזהות את הסכום:

1) זיהוי המטבעות ובידודם מהרקע
2) לכידת כל מטבע בנפרד
3) מציאת "דוגמאות" - מטבעות בסיס שאותן נשווה לכל מטבע בתמונה מסוימת
4) השוואה של כל מטבע שנמצא מול ה"דוגמאות" וקביעת ערך המטבע לפי מירב ההתאמות בעזרת stitching (דורש בידוד ותיוג של מטבע בודד מכל סוג שיהווה דוגמה)

זיהוי המטבעות ובידודם מהרקע

הפונקציה **`circle_edges`** תפקידה למצוא את קצוות האוביקטים (=המטבעות):
התחלנו בפילטר גאוסייאני גדול, רצינו לנקות את הרעש הכיתוב והציורים של המטבעות, מכיוון שהם לא מעניינים אותנו בשלב זה.
לקבלת הקצוות אנחנו מנפחים את התמונה(dilate), ומחסרים את התמונה המקורית מהמנופחת ומקבלים את הקצוות.
הפעלנו threshold עם ערכים שניסינו עד שקיבלנו עיגולים "נקיים" עבור רוב התמונות.
לפני החזרת התוצאה "כירסמנו" בתמונה על מנת שהמעגלים יהיו דקים יותר.

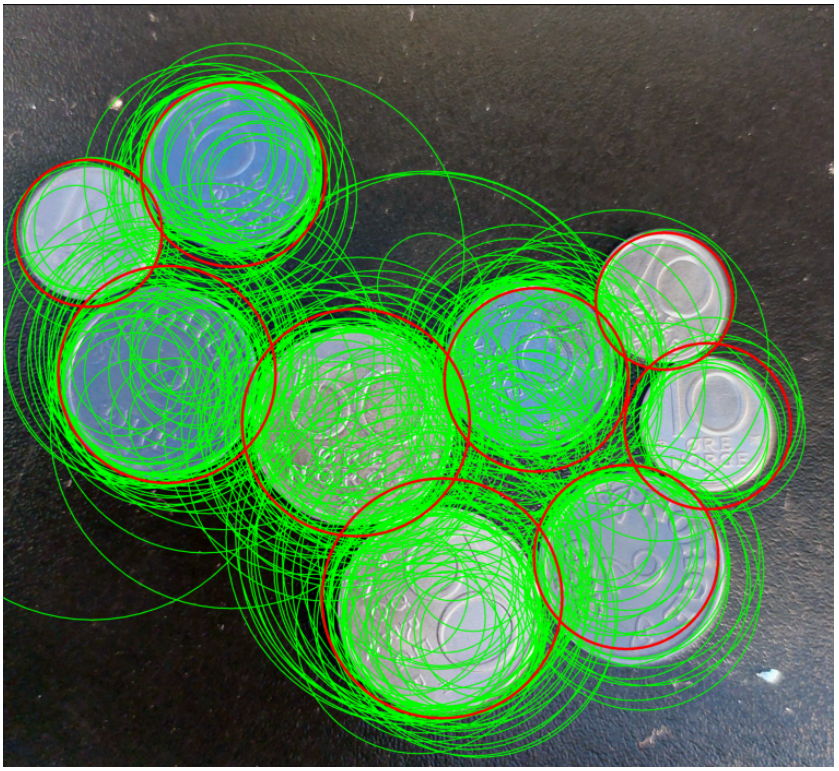
לכידת כל מטבע בנפרד

הפונקציה **`find_coins`** מקבלת את התמונה המקורית, threshold (נסביר את משמעותו בהמשך).
נתחיל בלקבל את קצוות התמונה בעזרת **`circle_edges`.**
נמצא מעגלים בתמונה הקצוות ע"י **`cv2.HoughCircles`** (הפונקציה של OpenCV המוצאת מעגלים בתמונות בעזרת hough transform).

קיווינו שהפונקציה תמצא מעגל אחד לכל מטבע, בפועל הפונקציה החזירה מעגלים רבים לכל מטבע... ולכן החלטנו לחלק את המעגלים לקבוצות לפי מרחק המרכזים שלהם, כאן ה threshold בא לידי ביטוי:

כל המעגלים הנמצאים עד מרחק ה threshold יהיו באותה קבוצה.
לכל קבוצה (בתנאי שיש בה לפחות 3 מעגלים, למנוע false positive על אזורים שהם לא מטבעות), הגדרנו "מעגל מייצג" שמרכזו ממוצע מרכז המעגלים של הקבוצה, ורדיוסו ממוצע הרדיוסים של הקבוצה.
הפונקציה תחזיר רשימה של המעגלים המייצגים, כך נוכל להשוות כל מטבע בנפרד.

לדוגמא, נסתכל על התמונה הבאה:



בירוק: כלל העיגולים שנמצאו על ידי cv2.HoughCircles
באדום: העיגולים המייצגים כל מטבע, ניתן לראות שהם תואמים די במדויק לכל מטבע.

מציאת "דוגמאות"

על מנת שנוכל לערוך השוואות ולקבוע עבור כל מטבע מה ערכו, אנו זקוקים לדוגמא למטבע. עבור כל מטבע, חיפשנו בתמונות היכן הוא מופיע בצורה הברורה ביותר בשביל שישמש כדוגמה. חתכנו את המטבע בלבד בצורה מדויקת - משהו כזה:

```
base_images = {
    2: cv2.imread("imgs/62.jpg")[3235:4370, 540:1688],
    5: cv2.imread("imgs/5.jpg")[2025:2925, 1175:2075],
    10: cv2.imread("imgs/67.jpg")[1489:2281, 187:1000],
    50: cv2.imread("imgs/50.jpg")[2235:3320, 865:2000],
}
```

הסרנו את הרקע שלו על מנת שלא יווצרו התאמות שווא בין רקעים למרות שהמטבעות שונים.

כל זה קורה בקובץ coin_radius_finder.py
הפונקציה **mask_coin** מקבלת תמונה חתוכה של המטבע לדוגמה.
מוצאת את המעגלים בתמונה בעזרת **cv2.HoughCircles**, גם פה קיבלנו הרבה מאוד מעגלים, על מנת לקבוע מהו המעגל שמהווה את גבולות המטבע הגדרנו את פונקציית **circle_score** שמביאה ציון לכל מעגל לפי הרדיוס שלו וקרבת מרכזו למרכז התמונה.
בעזרת שקלול שני הפרמטרים הללו הצלחנו למצוא את גבולות המטבע, לאחר מכן השתמשנו בפונקציה **mask_circular_objects** שמשתמשת ב**mask_coin** על מנת להסיר את הרקע של הדוגמה.

לבסוף יצרנ כל ו פונקציה ששמה **create_masked_coins** (ונמצאת בקובץ **coins_summation.py**) שמפעילה את **mask_coin** עלה **base_coins** (התמונות הבודדות של המטבעות 2,5,10,50 שלהן קראנו "דוגמאות")

התאמות, ובעיות

לאחר ששמרנו את הדוגמאות, נוכל למצוא כמה התאמות יש בין כל דוגמה לכל מטבע שמצאנו בתמונת ה target, ולבדוק לאיזה מטבע יש הכי הרבה התאמות איתו, ולבסוף לסכום את כל המטבעות.

זוהי פעולתה של הפונקציה **sum_coins**, היא עוברת על כל המטבעות בתמונת target מסויימת, ועבור כל אחת מפעילה פונקצייה אחרת ששמה **classify_from_image**, על מנת לזהות את המטבע ולהוסיף אותו לסכום.

הפונקצייה **classify_from_image** מקבלת מטבע שאותו היא צריכה לזהות. היא עוברת על כל תמונות הבסיס\הדוגמאות, ועל כל תמונת בסיס מפעילה את הפונקציה ששמה **count_matches** על מנת לספור כמה התאמות היו לתמונת ה target עם תמונת הבסיס. לבסוף, היא מחזירה את הערך עבור תמונת הבסיס שהכי התאימה לתמונת המטבע שנרצה לזהות

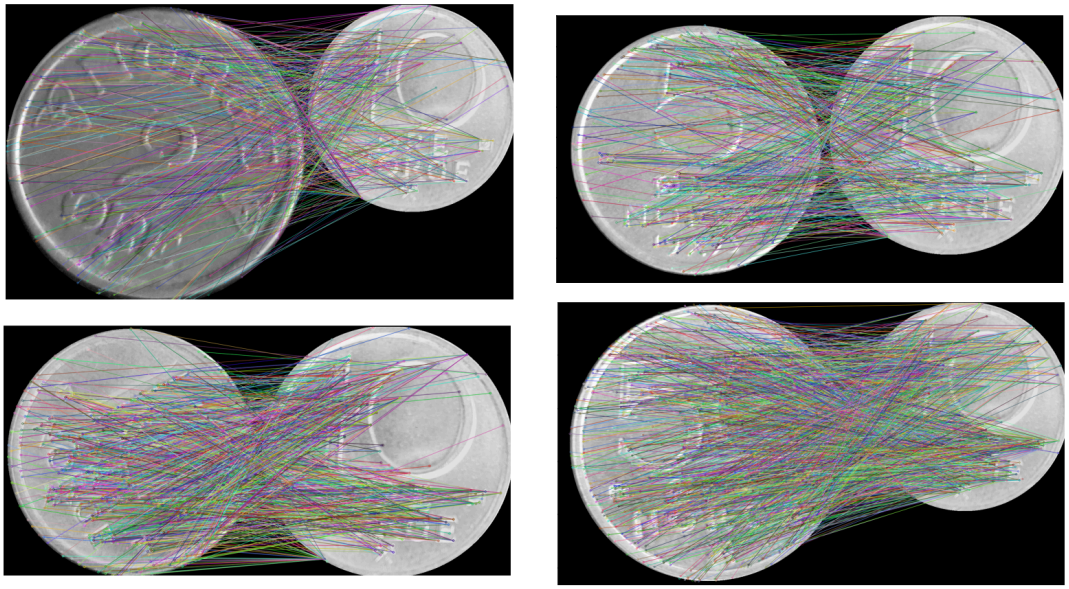
הפונקציה **count_matches** מקבלת שתי תמונות בנוסף ל threshold, ומשתמשת באלגוריתם ה SIFT על מנת למצוא keypoints בין התמונה **template_image** לתמונה **target_image**, לבסוף בהתאם לגודל ה threshold, היא מעיפה כמות מסוימת של keypoints מכיוון שרוב ההתאמות הן לא באמת טובות, ומחזירה את מספר ה keypoints שהיא מצאה.

אבל, לצערנו, אלגוריתם ה SIFT לא פועל טוב בתמונות של המטבעות. ישנם יותר מידי התאמות בין מטבעות שונים. לדוגמא:

נסתכל על התמונה:

וננסה לזהות את הערך של המטבע 10.

אם נצייר את ההתאמות בין המטבע 10 למטבעות הבסיס, נקבל:



משמאל ניתן לראות את מטבעות הבסיס, ומימין את המטבע 10 שאנחנו רוצים לקטלג.
נשים לב שיש הרבה יותר מידי התאמות בין **כל** מטבעות הבסיס למטבע 10, ואפילו ישנם הרבה יותר התאמות למטבע 50 עם המטבע 10 מאשר המטבע 10 עם עצמו.
בנוסף לכך, הרוב המוחלט של ההתאמות אפילו בין מטבע הבסיס 10 למטבע 10 הזה הן התאמות שגויות.

בעיה זו מהווה גורם עיקרי בתוצאות שלנו. ניסינו לשחק בהמון פרמטרים, ולנסות להפעיל את ה SIFT על גרסאות שונות של התמונה (למשל להפעיל עליה equalizeHist, או להעביר רק את ה edges של התמונה לפונקציית ה SIFT, וכולי..), אבל לצערנו אלגוריתם ה SIFT לא מזהה טוב את ה keypoints השונים.

מכיוון שכל דוגמה הביאה מספר שונה של התאמות (אפילו עם עצמה), נדרשנו לנרמל את מספר ההתאמות על מנת לקבל תוצאות רצויות (ללא נרמול המטבע 50 בעל מספר ההתאמות הרב ביותר לכל מטבע בפער), פקטורי הנרמול לכל אחד מהמטבעות מוחזרים מהפונקציה **`ratio`**.