אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

INTRODUCTION TO DIGITAL IMAGE PROCESSING

361.1.4751

FINAL REPORT

PROJECT NAME:

POP IT

STUDENTS:

Anastasia Smoliakov – 321198202

Chanel Michaeli – 208491787

Noam Atias – 311394357

Eldar Mamedov – 313251043

LECTURER:

Dr. Tammy Riklin Raviv

SUBMITTION DATE:

12.03.2023

תוכן עניינים

תקציר הפרויקט

בפרויקט זה יצרנו משחק המושתת באופיו על עקרונות המשחק "קליעה למטרה". המשחק משלב השתתפות פעילה של מתמודד שמטרתו לזרוק כדור אל עבר מסך שעליו מוקרנים בלונים הנעים מעלה, ולפגוע בבלון. כל פגיעה בבלון מזכה את המתמודד בנקודה וקצב עליית הבלונים גדל כך שרמת הקושי עולה. במהלך המשחק יש לכל מתמודד 6 ניסיונות לפגוע בבלונים.

הקדמה

בפרויקט זה השתמשנו בכלים של עיבוד תמונה, אשר למדנו בקורס, על מנת לבנות משחק אינטראקטיבי של פיצוץ בלונים על ידי זריקת כדור. במסגרת המשחק, אנחנו מקרינים בלונים אשר עולים בהדרגה על המסך כאשר מטרת השחקן היא לפגוע בבלון עם כדור ו – "לפוצץ" אותו.

המערכת המרכיבה את המשחק כוללת 2 מצלמות, מקרן, 6 כדורים ובד עבה וכהה שהודבק לקיר ומשמש כמסך שעליו מוקרן המשחק. נתאר את הרכבת סביבת המערכת בקצרה:

- את מסך הבד העבה הדבקנו על גבי קיר הספרייה אשר עליו מוקרן המשחק שבנינו.
- את המקרן מיקמנו מול המסך המודבק, כאשר המקרן נמצא במרחק של כ – 3.5 מטרים ומונח על כיסא בגובה סטנדרטי של הספרייה.
- שימוש ב – 2 מצלמות:
  - מצלמה ראשונה מוקמה באזור התחתון של המסך אשר עליו הוקרן המשחק, כך שמצלמה זו מצלמת את כל אזור המסך כלפי מעלה ומפתח קטן לפני המסך עצמו (מעט לפני הקיר עצמו).
  - מצלמה שנייה מוקמה על גבי המקרן עצמו והודבקה אליו על מנת למנוע תזוזה שלה כתוצאה של חזרת הכדור הפוגע בקיר (מסך הבד המודבק), כאשר מצלמה זו מצלמת את נקודת מבטו של השחקן שזורק את הכדור (ממוקדמת ממול המסך המודבק).

בנוסף לבלונים המוקרנים על המסך, השחקן יוכל לראות בפינה הימנית העליונה את מספר הזריקות שנותרו לו, ובפינה השמאלית העליונה יוכל לראות את הניקוד שצבר עד כה.

המצלמה הראשית והמקרן

המצלמה שממוקמת על הקיר מתחת למסך

כדור המשחק

במהלך המשחק נצפה לתוצאות הבאות:

- במקרה שבו השחקן מצליח לפגוע עם הכדור בבלון, השחקן צובר נקודה, מונה הזריקות שנותרו מתעדכן והבלון "מתפוצץ", ובנוסף קצב עליית הבלונים יגדל לאחר כל "פיצוץ" של בלון, כך שרמת קושי המשחק עולה.
- במקרה שבו השחקן מפספס את הבלון המערכת תזהה שהייתה פגיעה בקיר ומונה הזריקות שנותרו יתעדכן בהתאם.
- בסופו של כל משחק תוצג טבלה שמציגה את עשרת השחקנים המובילים עד כה, כלומר בעלי הניקוד הגבוה ביותר.

אתגרי הפרויקט

האתגרים איתם התמודדנו במהלך בניית הפרויקט:

- בתחילת הפרויקט חשבנו להשתמש במצלמה אחת מרכזית הממוקמת מול המסך, אך נתקלנו בקושי של זיהוי הרגע של פגיעת הכדור במסך ולכן לא יכולנו לדעת איזו תמונה מייצגת בצורה המדויקת ביותר את רגע הפגיעה בקיר, אשר קריטי עבור זיהוי פגיעה בבלון או החטאה. לכן החלטנו להוסיף מצלמה נוספת שבאמצעותה נקבל אינדיקציה טובה יותר לרגע פגיעת הכדור במסך.
- במהלך הפרויקט גילינו כי בעת זריקת הכדור ופגיעתו בקיר צורתו מעט משתנה ומקבלת עיוות בחלק מן התמונות, ולכן הדבר מקשה על מציאת הכדור על ידי Hough circles. לכן במקרה כזה החלטנו לבצע זיהוי של הכדור על ידי צבע ולא לפי צורה.
- בחלק מן התמונות שהתקבלו על ידי המצלמה הממוקמת מול המסך, הכדור נראה בהיר מידי כתוצאה של אור המקרן ולכן "נבלע" ברקע המסך הלבן, כך שהיה קשה לזהות את הכדור על ידי צבע או צורה, לכן החלטנו בתחילת האלגוריתם לבצע הורדה של רמת הבהירות של התמונה.
- בתחילת הפרויקט ביצענו את כל הפעולות הנדרשות (צילום, עיבוד התמונות ועדכון ממשק המשחק) בצורה סריאלית. במקרה זה המשחק לא פעל בזמן אמת והיו השהיות רבות בתהליך. על מנת להתמודד ולגרום למערכת לפעול בצורה מהירה חילקנו את שלבי הפרויקט ל- threads (תהליכונים) ובכך ביצענו במקביל את הפעולות המרכזיות בפרויקט.

אילוצי הפרויקט

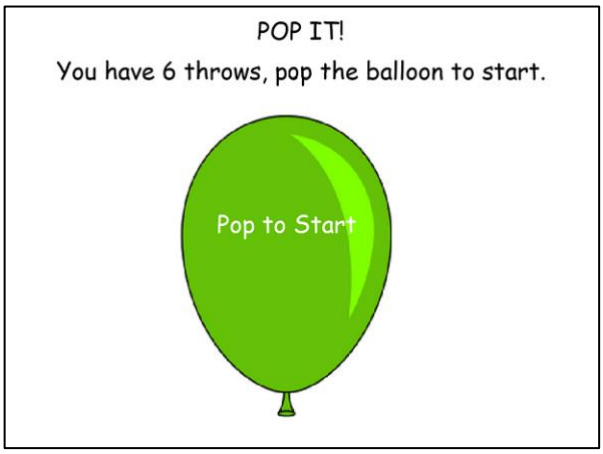
אילוצי הפרויקט איתם התמודדנו:

- מיקום הפרויקט. מכיוון שהמשחק כולל זריקות של כדור היינו חייבים למקם את הפרויקט כך שיוקרן על גבי קיר בספרייה כך שכאשר נזרק כדור נוכל לקבל בבירור אימפקט מהקיר והכדור יוחזר לאחר פגיעה.
- מיקום המצלמה שבאמצעותה ביצענו את זיהוי הפגיעה בקיר. בתור התחלה מיקמנו את המצלמה מהצד של המסך, אך מכיוון שאלגוריתם זיהוי הפגיעה עושה שימוש בשינוי של כיוון תנועה כל תנועה קלה של אנשים ברקע וכו' גרמה להפרעה במהלך המשחק. לאחר מכן שונה המיקום כך שהמצלמה מוקמה מעל המסך כאשר היא מצלמת מטה, אך גם כאן היו בעיות מכיוון שבחלק מהפעמים בהם נזרק כדור לאחר החזרה מהקיר הוא פגע בכיסא או שולחן וחזר אל אזור הקיר והמסך דבר שזוהה כזריקה נוספת וגרם לשגיאות נוספות. לכן הוחלט למקם את המצלמה בחלקו התחתון של המסך.

בניית ממשק המשחק

ממשק המשחק נבנה על ידי ספריית Pygame המשמשת לפיתוח משחקי וידאו ויישומי מולטימדיה. ספרייה זו כוללת מודולים לניהול תמונות, סאונד, התקני קלט, גרפיקה וכו'.

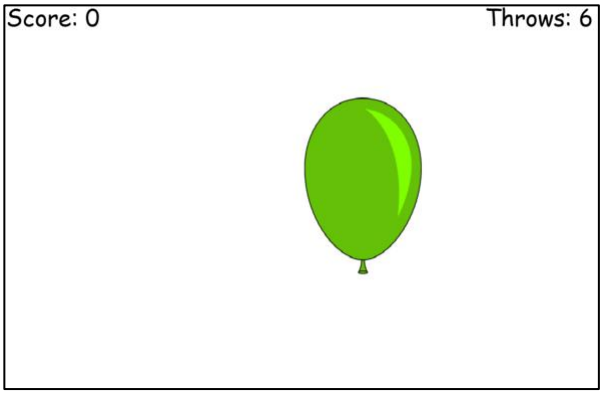
המשחק המוקרן רץ בלולאה כך שבכל איטרציה חלונית המשחק מתעדכנת, כאשר בהתחלה מוגדר גודל החלון של המשחק המוקרן וקצב ריענון מסך ל – 30 תמונות בשנייה ($30\ FPS$) בדומה לתדירות המקסימלית של המצלמות. עליית מסך הפתיחה מותנה באתחול המצלמות ולאחר מכן יעלה המסך הבא:



מסך הפתיחה לפני תחילת כל משחק.

תחילת המשחק עצמו מותנת במשתתף, כאשר המשתתף מוכן נלחץ על הבלון והמשחק יתחיל.

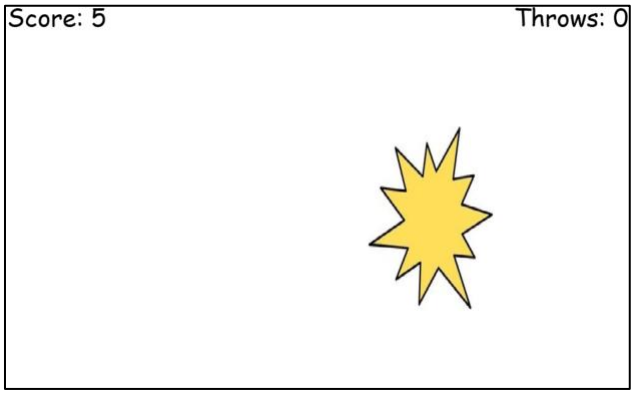
במהלך כל משחק יופיע על גבי המסך בצידו הימני והשמאלי העליונים מספר הניקוד שאותו משתתף צבר עד כה ואת מספר הזריקות שנותרו, כאשר בתחילת המשחק מופיע שנותרו 6 זריקות וניקוד 0.



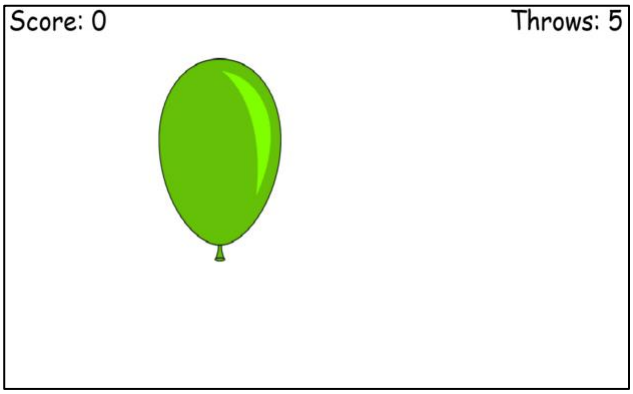
המסך ההתחלתי של תחילת כל משחק הכולל 6 זריקות וניקוד 0.

במהלך המשחק ישנה לולאה מרכזית של המשחק שבודקת בכל איטרציה אם הייתה פגיעה בקיר, ולאחר הפגיעה בקיר נבדוק אם הייתה פגיעה בבלון או פספוס על ידי אלגוריתם לזיהוי פגיעת הכדור בבלון, וכך בכל פעם שנקבל התראה שהיה אירוע ("event") כלשהו נבצע עדכון בהתאמה:

- במידה וקיבלנו התראה ("event") שהייתה פגיעה בקיר (אותה אנחנו מזהים עם אלגוריתם של optical flow) נעדכן את כמות הזריקות שנותרו למשתתף.
- במידה והייתה התראה ("event") שהייתה פגיעה בבלון אז נשמיע פיצוץ, נקרין אנימציה של פיצוץ ונעדכן את הניקוד.

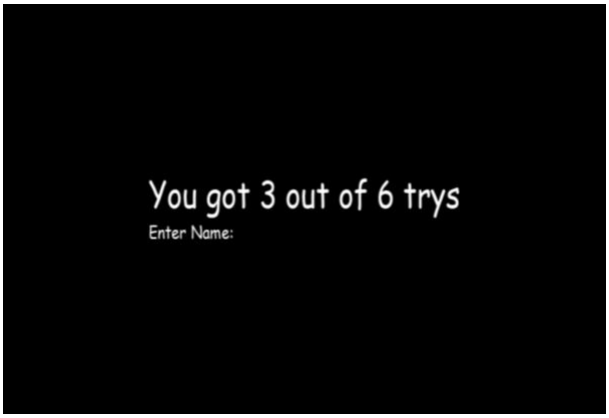


עדכון הזריקות והניקוד לאחר זריקה בה התבצע פגיעה ו"פיצוץ" של הבלון.



עדכון הזריקות והניקוד לאחר זריקה בודדת במקרה של פספוס הבלון.

בסוף המשחק, נעבור למסך שמציג את תוצאת המשחק. כל שחקן יכול להזין את שמו ולאחר מכן, נציג הניקוד של עשרת המתחרים הטובים ביותר.



תוצאת המשחק של המשתתף.

Competitors	Score (%)
Chanel	83
Lea	83
Hadar	83
muhamed	83
Nsatia	83
Olga	66
mila	66
Daniel	66
itkik	66
Eden	66

טבלת דירוג 10 המשתתפים בטובים ביותר.

היתרונות הבולטים של ספריית Pygame הם: קל ללמידה ושימוש ,התמיכה טובה ברשת, הוא קוד פתוח וקיימת תמיכה באנימציות, צלילים ווידיאו.

תיאור אלגוריתמים בעיבוד תמונה

אלגוריתם לזיהוי פגיעת הכדור במסך המשחק – Optical Flow

לצורך זיהוי פגיעת הכדור במסך המשחק, השתמשנו באלגוריתם שנקרא Optical Flow.

Optical flow היא טכניקה לזיהוי תנועה בתמונות. בדרך כלל משתמשים בה עבור רצף של תמונות בנקודות זמן סמוכות.

נגדיר את האובייקט הנע בזמן t עם עוצמה $I(x,y,t)$, אז לאחר זמן dt האובייקט עבר dx, dy. לכן עוצמת האובייקט כעת היא $I(x+dx, y+dy, t+dt)$. אנו מניחים כי עוצמת האובייקט הנע בין התמונות לא משתנה, כלומר:

$$I(x,y,t) - I(x+dx, y+dy, t+dt) = 0$$

נבצע קירוב טיילור למשוואה ונקבל:

$$\frac{dI}{dx}\delta x + \frac{dI}{dy}\delta y + \frac{dI}{dt}\delta t = 0$$

נחלק ב-δt ונקבל :

$$\frac{dI}{dx}u + \frac{dI}{dy}v + \frac{dI}{dt} = 0$$

כאשר $u = \frac{\delta x}{\delta t}$, $v = \frac{\delta y}{\delta t}$ הם רכיבי ה – x, y של המהירות (וקטורי הזרימה האופטית). נגדיר את $I'_x = \frac{dI}{dx}$ להיות גרדיאנט התמונה לאורך הציר האופקי, $I'_y = \frac{dI}{dy}$ להיות גרדיאנט התמונה לאורך הציר האנכי, $I'_t = \frac{dI}{dt}$ להיות גרדיאנט לפי זמן, ונוכל לכתוב:

$$I'_x u + I'_y v = -I'_t$$

לכן בהנחה כי לכל הפיקסלים בתמונה יש את אותו וקטור תנועה (dx, dy) נוכל לכתוב כי ההבדל בין תנועת הפיקסלים בין שני תמונות I_1, I_2 מיוצג כך:

$$I_1 - I_2 \approx I'_x u + I'_y v + I'_t$$

מאחר שקיבלנו משוואה אחת עם שני משתנים u, v, נשתמש בשיטה שנקראת *Gunnar Farneback Method* שמייעלת את אופן חישוב צפיפות הזרימה האופטית.

נגדיר את הקואורדינטות של פיקסל $p_i = (x_i, y_i)$ בחלון בגודל n פיקסלים, כך שנקבל את סט המשוואות הבא:

$$\begin{cases} I'_x(p_1)u + I'_y(p_1)v = -I'_t(p_1) \\ I'_x(p_2)u + I'_y(p_2)v = -I'_t(p_2) \\ \vdots \\ I'_x(p_n)u + I'_y(p_n)v = -I'_t(p_n) \end{cases}$$

נכתוב את מערכת המשוואות בצורה מטריצית:

$$A\gamma = b$$

כאשר:

$$A = \begin{pmatrix} I'_x(p_1) & I'_y(p_1) \\ I'_x(p_2) & I'_y(p_2) \\ \vdots & \vdots \\ I'_x(p_n) & I'_y(p_n) \end{pmatrix}, \qquad \gamma = \begin{pmatrix} u \\ v \end{pmatrix} \ , \ b = \begin{pmatrix} -I'_t(p_1) \\ -I'_t(p_2) \\ \vdots \\ -I'_t(p_n) \end{pmatrix}$$

על ידי שימוש באלגוריתם *Least Squares* נוכל למצוא את הווקטור γ:

$$A^T A\gamma = A^T b \quad \Rightarrow \quad \underbrace{(A^T A)^{-1} A^T A}_{identity\ matrix}\gamma = (A^T A)^{-1} A^T b \quad \Longrightarrow \quad \gamma = (A^T A)^{-1} A^T b$$

שיטת *Farneback* מתבססת על ההנחה שהעוצמה של כל פיקסל בחלון יכולה להיות משוערך על ידי פולינום מסדר גבוה. שיטה זו נקראת הרחבה פולינומיאלית והיא מאפשרת רמת דיוק גבוהה יותר של השערוך מאחר שמשתמשים בקירוב טיילור מסדר גבוה.

השתמשנו בפונקציה *cv2.calcOpticalFlowFarneback()* כדי לחשב את הזרימה האופטית בשיטה זו. הפונקציה מקבלת כקלט שני פריימים ומספר פרמטרים שונים ומחזירה את צפיפות הזרימה, כלומר שדה וקטורי שמגדיר את שינוי המיקום של כל פיקסל בתמונה הנוכחית לעומת התמונה הקודמת על ידי חישוב גודל וכיוון הזרימה האופטית ממערך של וקטורי הזרימה.

נרחיב על אופן החישוב:

כשלב מקדים, הפונקציה יוצרת פירמידה גאוסית על ידי הקטנת גודל התמונות בקנה מידה מסוים. זה נעשה על ידי הפרמטר *'pyr_scale'* המגדיר את קנה המידה בין הרמות העוקבות של הפירמידה (חייב להיות קטן מ-1). קנה מידה סטנדרטי (ערך ברירת מחדל) הוא 0.5, כלומר כל שכבה שמתווספת קטנה פי 2 מקודמתה.

פירמידה גאוסית הינה רצף של תמונות מוקטנות שנוצרות על ידי שימוש חוזר במסנן גאוסי ו – downsampling. הפירמידה הגאוסית מייצגת את התמונה בקני מידה שונים כאשר כל רמה בפירמידה מייצגת תמונה ברזולוציה שונה.

הפרמטר *'levels'* בפונקציה מגדיר את מספר הרמות בפירמידה הגאוסית. ערך ברירת המחדל הוא 3 רמות. ערך גדול יותר יביא לתוצאות מדויקות יותר אך סיבוכיות החישוב תגדל.

בשלב הבא הפונקציה מחשבת את הגרדיאנטים $\frac{dI}{dx}$, $\frac{dI}{dy}$ של כל תמונה בפירמידה על ידי שימוש באופרטור Sobel. אופרטור זה מאפשר חישוב משוערך של הגרדיאנטים של התמונה בכיוון אופקי ואנכי על ידי שני הגרעינים:

$$horizontal_{kernel} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \ , \ vertical_{kernel} = \begin{bmatrix} -1 & -2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

כדי לחשב את גודל וכיוון הגרדיאנט של כל פיקסל נבצע את החישובים הבאים:

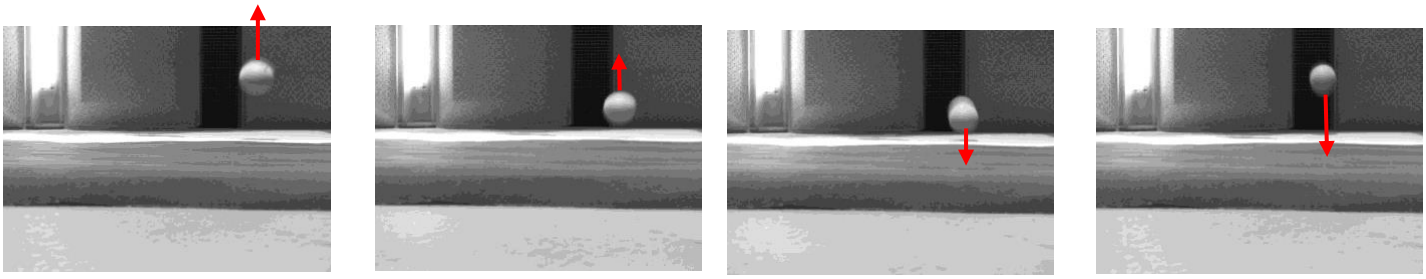
$$magnitude = \sqrt{{G_x}^2 + {G_y}^2} \quad , direction = atan2(G_x, G_y)$$

כאשר G_x הוא הגרדיאנט האופקי ו-G_y הוא הגרדיאנט האנכי של התמונה.

את הגרדיאנט $\frac{dI}{dt}$ נחשב על ידי חיסור שני הפריימים.

בשלב הבא הפונקציה מחשבת את צפיפות הזרימה האופטית של כל רמה בפירמידה על ידי שיטת ההרחבה הפולינומיאלית, כאשר הפרמטר *'win_size'* קובע את גודל החלון שעליו מחושב הצפיפות (ערך ברירת מחדל מוגדר להיות 15) והפרמטר *'poly_n'* מגדיר את המעלה של הפולינום (ערך ברירת מחדל מוגדר להיות 5). בנוסף הפונקציה משתמשת במסנן גאוסי שסטיית התקן שלו מוגדרת על ידי הפרמטר *'poly_sigma'* (ערך ברירת מחדל מוגדר להיות 1.1) על מנת להחליק את השדה המתקבל. לבסוף, הפונקציה מבצעת איטרציות על הרמות של הפירמידה כאשר מספר האיטרציות מוגדר על ידי הפרמטר *'iterations'* (ערך ברירת מחדל מוגדר להיות 3) וזאת על מנת לשפר את ביצועי האלגוריתם.

לאחר שהסברנו על אופן הפעולה של הפונקציה, נתאר את אופן השימוש שלנו באלגוריתם לזיהוי פגיעה של הכדור במסך. האלגוריתם מקבל סך הכל ארבעה פריימים עוקבים ומשתמש פעמיים בפונקציה *cv2.calcOpticalFlowFarneback* כאשר פעם ראשונה הפונקציה מקבלת את זוג הפריימים הראשון ופעם שניה הפונקציה מקבלת את זוג הפריימים השני מתוך ארבעת הפריימים.



דוגמה לארבעה פריימים שמתקבלים באלגוריתם מתוך המצלמה הצמודה לקיר.

בנוסף, הפרמטרים ששני הפונקציות מקבלים הם זהים. הגדרנו כי מספר רמות הפירמידה בכל פונקציה הוא $levels = 3$, קנה המידה של התמונות בפירמידה הוא $pyr\_scale = 0.5$, אלו הם פרמטרים שהגדרנו בערך ברירת המחדל שלהם.

את גודל החלון הגדרנו $win\_size = 20$, כאשר לקחנו בחשבון כי ככל שגודל החלון גדול יותר כך סיבוכיות החישוב גדולה יותר ושדה הזרימה שמתקבל מטושטש יותר אך היתרון הוא שהאלגוריתם פחות רגיש לרעשים ומספק זיהוי מהיר יותר של תנועה.

את מעלת הפולינום הגדרנו $poly\_n = 7$, ככל שפרמטר זה גדול יותר כך שערוך הזרימה יהיה חלק יותר אך יהיה רגיש יותר לרעשים.

את מספר האיטרציות בחרנו להיות 3 ואת סטיית התקן של המסנן הגאוסי להיות $poly\_sigma = 1.5$, ככל שפרמטר זה גדול יותר כך שערוך הזרימה יהיה חלק יותר אך מצד שני יהיה רגיש יותר לרעשים.

נבצע מכפלה סקלרית בין תוצאות שערוך הזרימה שקיבלנו משני האלגוריתמים עבור שני צמדי הפריימים, תוצאת המכפלה תספק לנו מידע לגבי שינוי כיוון התנועה של הכדור.

במקרה שבו בשני הפריימים הראשונים הכדור נע לכיוון המסך ובשני הפריימים השניים הכדור נע נגד כיוון המסך, כפי שניתן לראות בסט התמונות למעלה, נקבל בתוצאת המכפלה מספר שלילי.

נקבע ערך סף של 0.1 שלפיו אם הערך המוחלט של תוצאת המכפלה גדול ממנו וגם תוצאת המכפלה שלילית אז מצאנו כי היה שינוי בכיוון התנועה של הכדור, כלומר הכדור פגע במסך.

אלגוריתם לזיהוי פגיעת הכדור בבלון

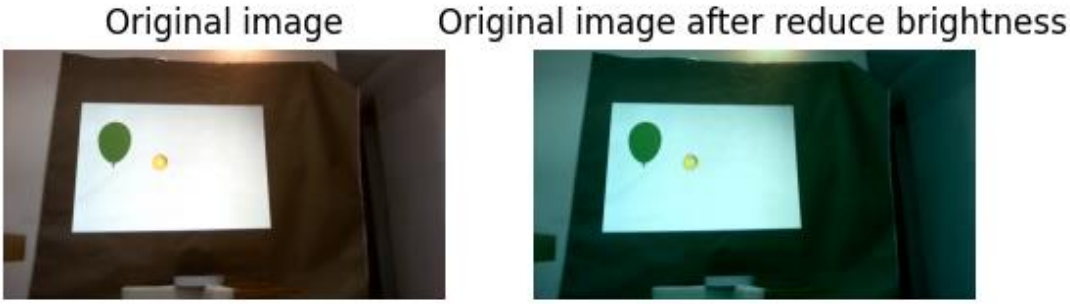
התמונות שמתקבלות לאלגוריתם זה הן התמונות המצולמות על ידי המצלמה שמול המסך. האלגוריתם מתחיל לפעול על התמונה רק כאשר קיבל חיווי שכדור פגע במסך וכעת המטרה היא לבדוק אם הכדור שפגע במסך פגע בבלון.

חלק א – פספוס או חשד לפגיעה בבלון

בחלק הראשון של האלגוריתם נרצה באופן ראשוני לשלול פגיעה של הכדור בבלון.
חלק זה נועד כדי לקצר זמני ריצה מאחר שקל יותר לזהות שלא הייתה פגיעה בבלון מאשר לזהות שהייתה פגיעה בבלון. לכן מטרת חלק זה היא לבדוק אם הכדור פספס את הבלון או שישנו חשד לפגיעה בבלון. חלק זה מוגדר בפונקציה שנקראת *'miss_or_maybe'* ושלבי הפונקציה הם להלן:

<u>שלב 1: הורדה של בהירות התמונה</u>

שמנו לב כי במרבית התמונות, הכדור נראה מאוד בהיר ולכן האלגוריתם לא מצליח לזהות אותו. לכן כדי לפתור בעיה זו הורדנו את רמת הבהירות על ידי שימוש בפונקציה המובנת *'cv2.add'* שמקבלת את התמונה במרחב RGB ואת הערך המוחסר מהפיקסלים בתמונה. אנחנו בחרנו להחסיר את הערך 50 מהפיקסלים של התמונה.



<u>שלב 2 : זיהוי מסך המשחק וחיתוך התמונה לפיו</u>

בשלב זה האלגוריתם מבצע חיתוך של התמונה המתקבלת על ידי המצלמה, כך שהתמונה החתוכה כוללת את מסך המשחק בלבד (של המקרן). זאת עשינו על ידי בניית פונקציה שנקראת *'only_screen'*.
פונקציה זו מקבלת כקלט את התמונה, פקטור שבעזרתו נחתוך את התמונה, ערך סף וערך מקסימלי ומחזירה תמונה חתוכה של מסך המשחק בלבד. נציג את שלבי האלגוריתם:

1. המרה התמונה ל – Grayscale.
2. מסיכה בינארית: יוצרים מסיכה בינארית לתמונה ב – Grayscale, על ידי שימוש בפונקציה מובנת של Python שנקראת *'cv2.threshold'*.

הסבר על הפונקציה המובנית:

'cv2.threshold' מקבלת תמונה ב – Grayscale, ערך סף, ערך מקסימלי ואת המתודה של המיסוך.
אנחנו בחרנו ב – *cv2.THRESH_BINARY* , מתודה זו הופכת פיקסלים שערכם גדול מערך הסף לערך המקסימלי שבחרנו להיות 255 (לבן) ופיקסלים שערכם קטן מערך הסף ל-0 (שחור). את ערך הסף בחרנו להיות 160.

3. פילטר להחלקה: כדי להחליק את המסכה שקיבלנו ולהימנע מרעשים, יצרנו פילטר על ידי שימוש בפונקציה המובנת של Python – *'cv2.morphologyEx'*.

הסבר על הפונקציה המובנית:

'cv2.morphologyEx' מקבלת תמונה בינארית, את מתודת המסנן ואת הגרעין (Kernel).
אנחנו בחרנו במתודה שנקראת *cv2.MORPH_OPEN* שמטרתה להחליק את התמונה ולהסיר רעשים.
אופן הפעולה מבוסס על שני מתודות אחרות:

- הראשונה Erosion שמטרתה לצמצם את עובי האובייקטים הלבנים במסכה
- השנייה dilating שמטרתה להרחיב את עובי האובייקטים שבמסכה. באופן זה ניתן להסיר רעשים בתמונה.

את הגרעין (Kernel) בחרנו להיות מטריצת אחדים בגודל 5×5 לצורך החלקת המסכה.

4. מציאת קונטורים: לאחר החלקת התמונה, נשתמש בפונקציה מובנת *'cv2.findContours'* למציאת הקונטורים בתמונה הבינארית.

הסבר על הפונקציה המובנית:

'cv2.findContours' מקבלת תמונה בינארית לאחר החלקה, את מתודת מציאת הקונטורים ואת האלגוריתם לשרטוט הקונטור.
אנחנו בחרנו במתודה *cv2.RETR_EXTERNAL* שמטרתה למצוא קונטורים ראשיים ולהתעלם מקונטורים פנימיים, כלומר קונטורים שנמצאים בתוך קונטורים. האלגוריתם שבחרנו לשרטוט הקונטורים הוא *cv2.CHAIN_APPROX_SIMPLE* שמטרתו לדחוס קווים אנכיים, אופקיים ואלכסוניים ולהשאיר רק את נקודות הקצה שלהם, כלומר הקודקודים של הקונטור. בצורה כזאת אנו חוסכים במקום בזיכרון וחוסכים בזמן ריצה.

5. חיתוך התמונה למסך המשחק: מתוך כל הקונטורים שנמצאו, נבחר את הקונטור הגדול ביותר שהוא בעצם המסך של המשחק.

לקונטור זה נמצא את ה – Bounding Box בצורה מלבנית על ידי שימוש ב – *'cv2.boundingRect'* שמקבל את הקונטור ומחזיר את האורך h והרוחב w של ה – Bounding Box ואת מיקום הפיקסל בפינה השמאלית העליונה (x, y).

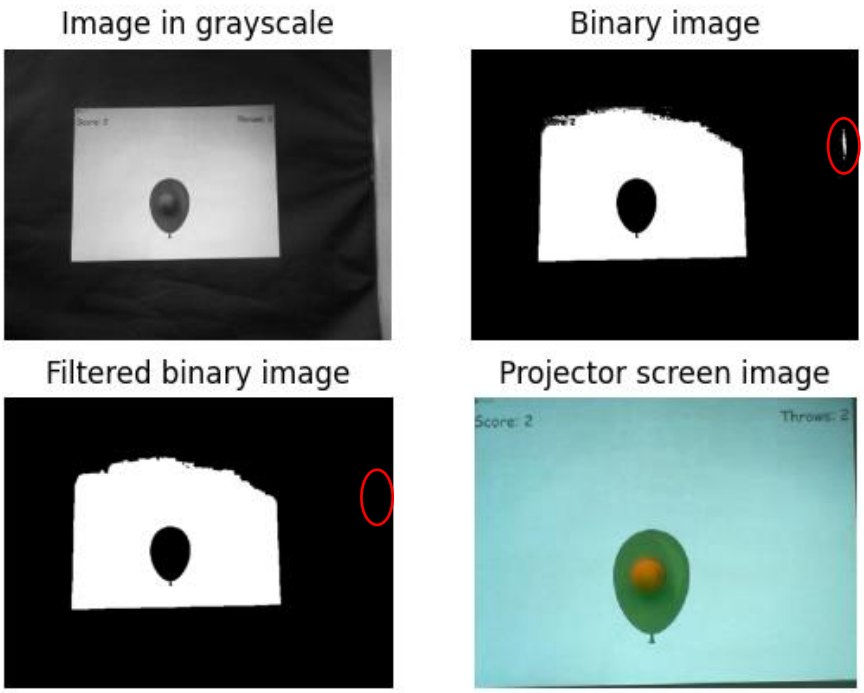
את התמונה נחתוך לפי גבולות הקופסא שמצאנו, ונשתמש ב – factor מסוים שבחרנו על מנת לוודא שכל המסך נמצא בתמונה החתוכה, ללא קצוות מיותרים באופן הבא:

$$x = x + factor$$

$$w = w - 1.5 * factor$$

אנחנו בחרנו פקטור של 25 פיקסלים.

ניתן לראות את השלבים השונים באלגוריתם למציאת מסך המשחק ולבסוף את התמונה החתוכה:



תהליך חיתוך התמונה לפי מסך המקרן מוצג בתמונות לפי השלבים.
ניתן לראות דוגמא להחלקת התמונה ע"י שני העיגולים האדומים (לפני החלקה ואחרי).

<u>שלב 3: בדיקה לצורך הבחנה – האם ניתן לזהות את הכדור על ידי Hough matrix?</u>

כדי לבדוק אם ניתן לזהות את הכדור על ידי Hough matrix בנינו פונקציה שנקראת *'find_circles'*.

פונקציה זו מקבלת:

- תמונה חתוכה של מסך המשחק בלבד (המסך המוקרן).
- מספר הערוץ של התמונה עליו נפעיל את האלגוריתם Hough matrix
- ערך רדיוס מקסימלי וערך רדיוס מינימלי.

הפונקציה מחזירה את העיגולים שהיא מצאה בתחום הרדיוסים הנבחר.

נציג את שלבי הפונקציה:

1. המרה למרחב הצבעים HSV: מצאנו שזיהוי מעגלים על ידי Hough matrix עובד טוב יותר כאשר הקלט שלו הוא ערוץ ה – Value של מרחב הצבעים HSV. נראה כיצד הדבר משפיע ובא לידי ביטוי בשלב האנליזה.

2. שימוש ב – Hough matrix על מנת למצוא מעגלים עבור הערוץ השלישי בתמונת ה – HSV (ערוץ ה – Value) . השתמשנו בפונקציה *'cv2.HoughCircles'* שמקבלת תמונה בערוץ בודד, מתודה למציאת המעגלים, *cv2.HOUGH_GRADIENT* , המרחק המינימלי בין מרכזי מעגלים, ערך הגרדיאנט למציאת קצוות (בחרנו 50 באופן קבוע), פרמטר נוסף שמגדיר רגישות למציאת מעגלים (ככל שיותר גדול כך יותר מעגלים יימצאו) שבחרנו להיות 30 באופן קבוע, וערכי רדיוסים מינימלי ומקסימלי שמגדירים טווח רדיוסים שבו ימצאו המעגלים. בחרנו שהערך המינימלי של הרדיוס הוא 0 והערך המקסימלי ההתחלתי הוא 20. הפונקציה מחזירה מעגלים בהתאם לפרמטרים שהגדרנו.

כפי שלמדנו בהרצאות, על מנת לייצג מעגל אנחנו זקוקים לשלושה פרמטרים (r, a , b):

$$(x-a)^2+(y-b)^2=r^2$$

האלגוריתם הבסיסי של מציאת מעגלים אינו יעיל מכיוון שהוא כרוך בהתמודדות עם מרחב פרמטרים תלת ממדיים ולכן השיטה למציאת המעגלים שנתמכת בפונקציה *'cv2.HoughCircles'* היא *'cv2.HOUGH_GRADIENT'* . כפי שעולה מהשם, שיטה זו לוקחת בחשבון את הגרדיאנט. לפי השיטה הבסיסית עבור כל נקודת קצה (edge point) ניתן לשרטט את המעגלים המתאימים במרחב הפרמטרים שהוגדרו מראש ובכך להגדיל את הצבירה לכל פיקסל. אך בעזרת השיטה *HOUGH_GRADIENT*, במקום לצייר את המעגל במלואו, ניתן להגדיל את כמות הערכים הצוברים בכיוון השיפוע של כל נקודת קצה.

אלגוריתם זה מורכב מ-2 שלבים:

- תחילה, מזוהים כל האופציות האפשריות למרכזי המעגל.
- לאחר מכן, עבור כל מרכז מחושב הרדיוס הטוב ביותר.

הגרדיאנט מחושב עבור כל פיקסל קצה (edge pixel) ונצברים הפיקסלים השוכנים בשני הכיוונים של הגרדיאנט. הפיקסלים בעלי ערכי הצבירה הגדולים ביותר מועמדים להיות מרכזי המעגלים. לאחר מכן, יש למצוא את הרדיוס הטוב ביותר, אך כעת מרחב הפרמטרים מצטמצם למימד יחיד. עבור כל מרכז מעגל אפשרי יחושב המרחק שלו מכל הפיקסלים בקצה (זה בדיוק הרדיוס) .הרדיוס הטוב ביותר יהיה זה שנתמך בצורה הטובה ביותר על ידי פיקסלי הקצה.

השתמשנו בפונקציה *'find_circles'* בלולאה, כאשר בכל פעם הגדלנו את הערך המקסימלי של הרדיוס, עד שהצלחנו למצוא מעגל או עד הערך 30 (ונשמור את הרדיוס המקסימלי הזה להמשך). זאת מאחר שלא ניתן לדעת בדיוק את הרדיוס המתאים של הכדור לכן נבדוק בכל פעם אם ניתן למצוא את הכדור עם רדיוס שונה.

במידה ולא נמצא מעגל כלשהו בסיום האיטרציות, נרצה למצוא את הכדור ע״י צבע. לשם כך נגדיר

משתנה $flag$ שתפקידו לציין לפי איזה שיטה אנחנו מוצאים את הכדור כך שכל שאר הפונקציות יקבלו ויפעלו לפי ערך זה. במידה ומצאנו מעגל $flag = 0$ אחרת $flag = 1$ ואז ננסה למצוא אותו על ידי צבע.

<u>שלב 4: זיהוי הכדור על ידי Hough matrix או על ידי צבע</u>

שלב זה מתחלק לשניים כתלות בערך שה – $flag$ קיבל:

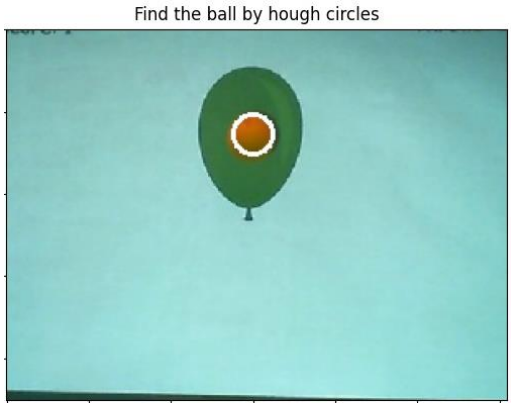
א. כאשר $flag = 0$ – נמצא את הכדור על ידי Hough matrix בעזרת הרדיוס המקסימלי ששמרנו מהחלק הקודם. אופציה זו היא המועדפת עלינו מאחר שבמקרה זה נקבל את מרכז הכדור ואת הרדיוס שלו בצורה מדויקת ללא צורך בשערוך. לצורך כך בנינו פונקציה שנקראת *'Circles'*.

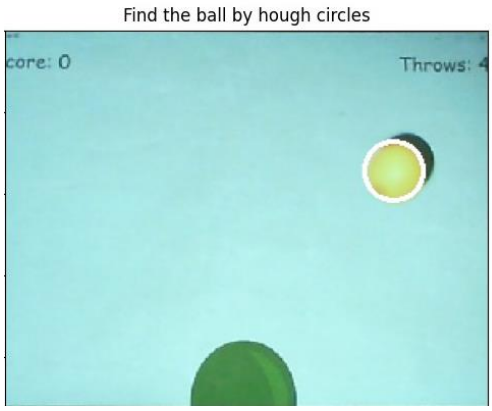
פונקציה זו מקבלת:

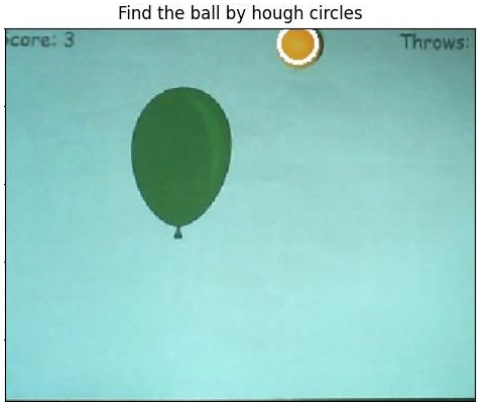
- תמונה חתוכה של מסך המשחק בלבד (המסך המוקרן).
- את הרדיוס המקסימלי שמצאנו בחלק הקודם.

הפונקציה ממירה את התמונה הנתונה למרחב הצבעים HSV ומכניסה לאלגוריתם של Hough matrix את ערוץ ה – Value בלבד, את הרדיוס המקסימלי עם אותם הפרמטרים כמו בפונקציה של *'find_circles'*. הפונקציה מחזירה הרדיוס של כל המעגלים שנמצאו בטווח הרדיוסים הנתונים, ואנחנו נבחר את המעגל בעל הרדיוס הקטן ביותר שהרי הוא הכדור. בנוסף הפונקציה מחזירה את הקואורדינטות של מרכז הכדור ותמונה ב – Grayscale שמכילה את המעגל שנמצא.

נציג כמה דוגמאות של זיהוי כדור בתמונה על ידי Hough circles:







ב. כאשר $flag = 1$ – נמצא את הכדור לפי צבע. לשם כך בנינו פונקציה שנקראת *'find_orange'*.

פונקציה זו מקבלת את התמונה חתוכה של מסך המשחק בלבד (המסך המוקרן), וממירה אותו למרחב הצבע HSV.

שמנו לב כי הכדורים שבהם השתמשנו במשחק, שצבעם כתום, לעיתים נראים בתמונה בצבע צהוב.

לכן, יצרנו מסיכה לפי ערכי הצבעים צהוב וכתום במרחב הצבעים HSV כאשר הגדרנו את גבולות ערכי הצבעים במרחב HSV:

$$yellow_{lower} = (0,50,50)$$
$$yellow_{upper} = (40,255,255)$$

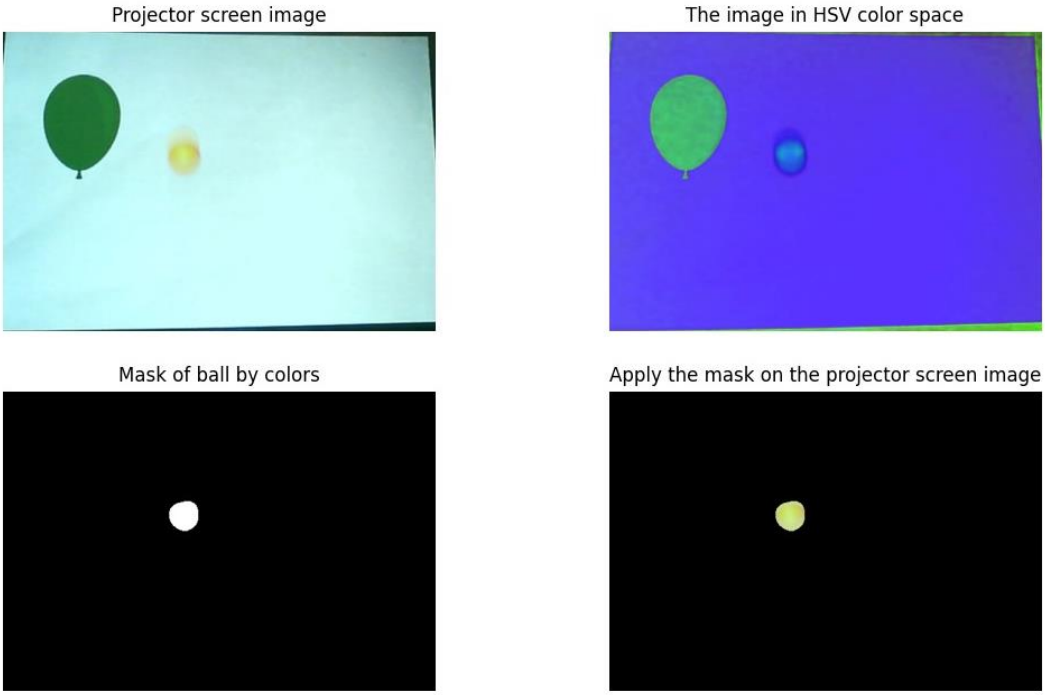
והפעלנו אותה על התמונה כך שקיבלנו תמונה שמכילה רק את הצבעים צהוב וכתום שהיו בתמונה המקורית, שהם יוצרים את הכדור. לאחר מכן החלקנו את התמונה שהתקבלה על ידי שימוש בפילטר חציון עם גודל גרעין (Kernel) 9×9, וזאת כדי להסיר רעשים. כעת אנו רוצים לשערך מתוך התמונה שהתקבלה את הרדיוס של הכדור ומרכז הכדור. כדי לשערך את מרכז הכדור בציר x ביצענו את החישוב הבא:

$$x_{center,estimate} = \frac{x_{max} + x_{min}}{2}$$

כאשר x_{min} היא הקואורדינטה הקטנה ביותר בציר x שבה זוהה שיש צבע כתום או צהוב ו-x_{max} היא הקואורדינטה הקטנה ביותר בציר x שבה זוהה שיש צבע כתום או צהוב. באופן דומה שיערכנו את מרכז הכדור בציר y. כדי לשערך את ביצענו את החישוב הבא:

$$r_{estimate} = \frac{\max\left(x_{max} - x_{min}\ ,\ y_{max} - y_{min}\right)}{2}$$

נציג דוגמה לשלבי מציאת הכדור על ידי צבע:



שלבי מציאת הכדור על ידי צבע בתמונות.

<u>שלב 5: חיתוך התמונה סביב הכדור</u>

כעת לאחר שהצלחנו למצוא את הכדור על ידי אחד מהאלגוריתמים המוצעים לעיל, נחתוך את התמונה סביב הכדור כך שמרכז הכדור יהיה במרכז התמונה החתוכה. לשם כך בנינו את הפונקציה *'ball_area'* פונקציה זו מקבלת:

- תמונה חתוכה של מסך המשחק בלבד (המסך המוקרן).
- $factor$ שמטרתו להגדיר את גודל חיתוך התמונה סביב הכדור.

הפונקציה קוראת לפונקציות מהשלב הקודם שמזהות את הכדור וכך מקבלת את ערכי מרכז הכדור ורדיוס הכדור. הפונקציה מבצעת חיתוך של התמונה סביב הכדור ואת גבולות החיתוך אנו קובעים באופן הבא:

$$left_{boundry} = \max(0, x_{center} - r \cdot factor)$$
$$right_{boundry} = \min(width, x_{center} + r \cdot factor)$$
$$upper_{boundry} = \max(0, y_{center} - r \cdot factor)$$
$$lower_{boundry} = \min(height, y_{center} + r \cdot factor)$$

כאשר $r, x_{center}, y_{center}$ הם הפרמטרים שקיבלנו מהפונקציות של השלב הקודם המגדירים את הכדור (מרכז ורדיוס). את ערך הפקטור נקבע כך:

$$factor = \begin{cases} 1, & flag = 1 \\ 2.5, & flag = 0 \end{cases}$$

הפונקציה מחזירה את התמונה החתוכה סביב הכדור. העקרון המנחה הוא שאם הייתה פגיעה של הכדור בבלון אז בהכרח נוכל למצוא את הבלון בסביבת הכדור. לכן אנו חותכים את התמונה סביב הכדור על מנת שבהמשך נחפש סביב הכדור את הבלון. במידה ולא נמצא, נדע שהיה פספוס וודאי.

<u>שלב 6: זיהוי קצוות של התמונה החתוכה סביב הכדור</u>

נבצע זיהוי קצוות של התמונה החתוכה מהשלב הקודם על ידי שימוש במתודה של Canny.

לצורך כך בנינו פונקציה שנקראת *'Canny'* שמקבלת את התמונה, ומעבירה את התמונה ל – Grayscale. לאחר מכן משתמשים בפונקציה המובנת *'cv2.Canny'* שמקבלת את התמונה, ערך סף נמוך וערך סף גבוה. הפונקציה מוצאת את קצוות התמונה ב – 4 שלבים:

1. סינון רעש עם פילטר גאוסי (5×5 Gaussian filter)
2. מציאת עוצמת הגרדיאנט בכיוון X ובכיוון Y בעזרת Sobel kernel ואז נוכל למצוא את גרדיאנט העוצמה והכיוון לכול פיקסל באופן הבא:

$$\theta = tan\frac{G_y}{G_x} \qquad G = \sqrt{G_y^2 + G_x^2}$$

3. דיכוי לא מקסימלי – אחרי שמצאנו את הגרדיאנט וכיוונו אנחנו רוצים לסנן פיקסלים לא רצויים, שלא באמת נותנים לנו מידע על קצוות התמונה. בעצם נעבור על כל פיקסל ונבדוק אותו עם השכנים שלו ונראה אם נוצר מקסימום מקומי. שלב זה נותן לנו תמונת קצוות דקה.

4. סף היסטרזיס – שלב זה יחליט לנו איזה גרדיאנטים הם באמת קצוות ואיזה נמחק. כל עוצמת הגרדיאנט מעל ערך הסף המקסימלי נחשבת להיות חלק מהקצוות, כמו כן כל עוצמת הגרדיאנט מתחת לערך הסף המינימלי בטוח נמחקת. כל עוצמת הגרדיאנטים שבין ערכי הסף יכולים להיות מוגדרים קצה או לא קצה תלוי בחיבוריות שלהם, אם הם מחוברים לגרדינאטים שעברו את הסף המקסימילי הם יחשבו לקצה אחרת יחשבו לא קצה.

<u>שלב 7: יצירת מסיכה של הכדור בתמונה החתוכה סביב הכדור</u>

לאחר מציאת קצוות התמונה, נרצה ליצור מסיכה לכדור על התמונה החתוכה סביב הכדור.

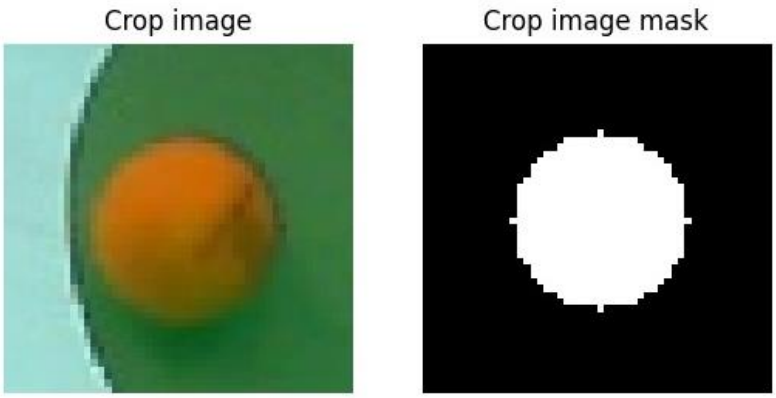
השלב הבא מתחלק לשניים כתלות בערך שה- $flag$ קיבל:

א. כאשר $flag = 1$ – יצרנו פונקציה שנקראת *'ball_mask'* שמקבלת את התמונה החתוכה, ומוצאת את הרדיוס והמרכז של הכדור המשוערך על ידי צבע (משלב 3 על ידי *'find_orange'*).

יצרנו צירים x, y מהתמונה בגודל הרוחב והאורך של התמונה ולאחר מכן יצרנו מסיכה על ידי משוואת מעגל סביב מרכז הכדור:

$$mask = [(x - x_{canter})^2 + (y - y_{canter})^2 = r^2]$$

את המסכה אנו יוצרים בגודל של התמונה החתוכה, כך שבתמונה יש רק את הכדור הממוסך. הפונקציה מחזירה את תמונה זו. נציג דוגמה לתמונה חתוכה סביב הכדור ואת התמונה של מיסוך הכדור כאשר זיהוי הכדור היה לפי צבע:



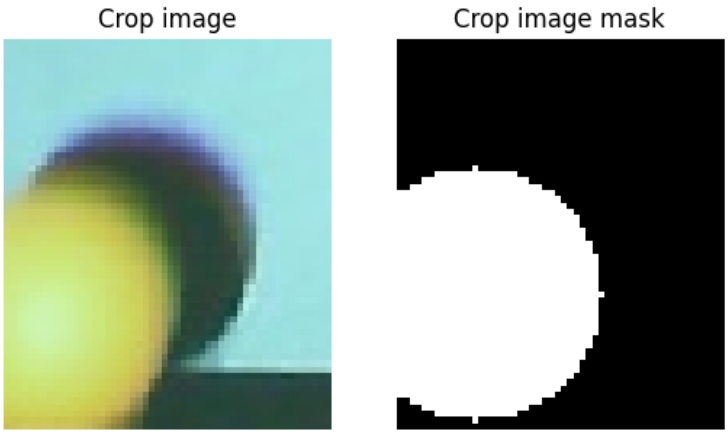
מציאת הכדור לפי צבע וחיתוך שלו (צד שמאל) ויצירת מסכת הכדור (צד ימין)

ב. כאשר $flag = 0$ – מצאנו את הכדור על ידי Hough circles. כעת נרצה למצוא את קצוות הכדור ולמלא אותן. לכן בנינו פונקציה שנקראת *'full_contour'* שמקבלת את תמונת הקצוות משלב קודם של התמונה החתוכה סביב הכדור, וממלאת את הקונטור הגדול ביותר בתמונה, שהרי הוא הכדור ומחזירה את התמונה של קונטור הכדור המלא בלבד. זאת עשינו על ידי שימוש בפונקציה המובנת

cv2.findContours (עם אותם הפרמטרים כמו שכבר הסברנו) כדי למצוא את הקונטורים בתמונה ולאחר מכן השתמשנו ב – *cv2.drawContours* כדי לשרטט את הקונטורים על תמונה ריקה בגודל התמונה החתוכה, ואז ניקח את התמונה שבה יש את הקונטור הגדול ביותר והיא המסכה של הכדור.
בנוסף אנו בודקים אם הכדור נמצא בגבולות התמונה כך שהוא נחתך או במקרה שלא נמצאה מסכה באופן זה, אנו נייצר מסיכה על ידי *'ball_mask'* כאשר במקרים אלה הפונקציה תקבל תוספת לרדיוס הכדור שמצאנו על ידי הפונקציה *'Circles'* , זאת על מנת לוודא שכל הכדור נמצא תחת המסכה.

<u>הערה:</u> שלב זה נועד כי ליצור מסכה לכדור אשר נעשה בה שימוש בהמשך רק עבור בדיקת מקרה קיצון בו הכדור פגע בקצוות מסך המשחק כך שלאחר חיתוך התמונה סביב הכדור, הכדור לא ימצא במרכז התמונה החתוכה אלא בפינות.

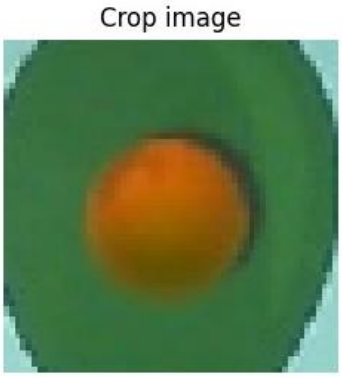
בשלב הבא נסביר למה מקרה קיצון זה בעייתי וכיצד אנחנו מצליחים להתמודד איתו בעזרת המסכות שיצרנו לכדור עבור כל אחד מן המקרים. (בקצרה – נסתכל על מספר הפיקסלים של המסכות ועבור threshold מסוים נדע אם מדובר בכדור בודד או שיש בסביבתו בלון). נציג דוגמא עבור מקרה קיצון בו יש חשיבות ליצירת המסכות בשלב זה עבור השלבים הבאים:



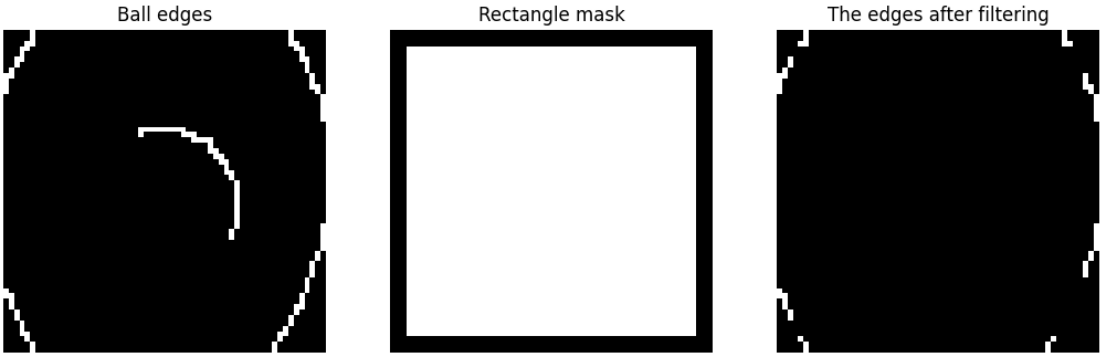
מקרה בו הכדור פגע בקצוות מסך המשחק

<u>שלב 8 – האם הבלון מצוי בסביבת הכדור?</u>

שלב זה נעשה בפונקציה שנקראת *'miss_or_maybe'* שמטרתה לבדוק אם הבלון מצוי בקרבת הכדור.
העקרון המנחה הוא שהכדור נמצא במרכז התמונה החתוכה ואילו הבלון הרבה יותר גדול מהכדור בתמונה, לכן נוכל להניח כי בשולי תמונת הקצוות יהיו קצוות של בלון בלבד ולא של הכדור, כפי שניתן לראות בתמונה הבאה:



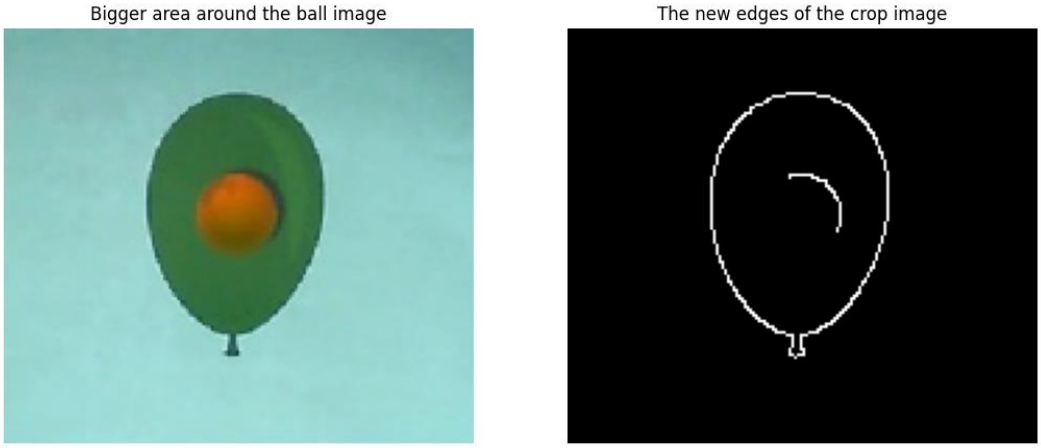
לכן ניצור תמונה שחורה שמסגרתה לבנה בגודל התמונה החתוכה, אשר מהווה מעין מסיכה שנפעיל על תמונת הקצוות משלב 5, כך שבפועל נשאר רק עם הקצוות שנמצאים בשולי התמונה.



קצוות התמונה החתוכה, המסכה הריבועית, והקצוות שנותרו לאחר סינון המסכה על הקצוות התמונה החתוכה.

במקרה שאין קצוות שנותרו בשולי התמונה נוכל לומר בוודאות כי הכדור לא נמצא בסביבת הבלון, כלומר הכדור לא פגע בבלון ולכן נדווח כי היה פספוס ונמשיך לתמונה הבאה.

במקרה שבו ישנם קצוות שנותרו בשולי התמונה (כפי שניתן לראות בתמונה מעלה), אנו יודעים בוודאות שאלו קצוות של בלון ולכן ניתן לומר כי ישנו בלון באזור הכדור ולכן אנו חושדים שהייתה פגיעה של הכדור בבלון. כדי לוודא שהחשד מוצדק, נחתוך את התמונה של מסך המשחק סביב הכדור כך שתהיה גדולה יותר מהתמונה החתוכה הקודמת כך שכעת התמונה תכלול את הכדור ואת הבלון יחדיו כך:



תמונה חתוכה גדולה יותר המכילה את הכדור והבלון יחדיו.

נמצא את תמונת הקצוות של תמונה זו (כפי שמוצג מעלה) ואת הקונטור הגדול ביותר, על ידי *'full_contour'*. כעת אנו חושדים כי הקונטור הגדול ביותר הינו הבלון. בנוסף כדי להתמודד עם מקרה הקיצון מהשלב הקודם נקבע ערך סף של 2000 פיקסלים, כך שאם סכום הפיקסלים הלבנים בתמונת הקונטור קטן מערך הסף אז הקונטור שמצאנו הוא הכדור ולא הבלון, דבר שמעיד שגם לאחר הגדלת התמונה מהכדור (Zoom out) עדיין הקונטור הגדול ביותר הוא הכדור, ולכן הבלון נמצא רחוק מן הכדור כלומר מצב של פספוס הבלון. אם מספר הפיקסלים הלבנים גדול מערך הסף, נמשיך לחשוד שזה הבלון, ואז נחסיר בין סכום הפיקסלים הלבנים שבתמונת הקונטור, שחשודה להיות הקונטור של הבלון, לבין סכום הפיקסלים הלבנים של המסכה של הכדור שמצאנו בשלב 7 . במקרה שההפרש הוא אפס, אנו יכולים לדעת בוודאות כי מדובר באותה תמונה ולכן החשד הופרך. אם ההפרש שונה מאפס, אז החשד ישנו ומכאן נמשיך לאלגוריתם הבא שמטרתו לאשש את החשד שהייתה פגיעה בבלון.

הפונקציה *'miss_or_maybe'* מחזירה:

- ה – $flag$ שלפיו מצאנו את הכדור.
- התמונה החתוכה של הכדור והבלון יחדיו (במקרה של חשד לפגיעה) ותמונה שחורה במקרה של פספוס.
- המסכה של הבלון.
- הרדיוס המקסימלי של הכדור.

חלק ב – אישוש החשד של פגיעת הכדור בבלון

בחלק השני של האלגוריתם נרצה באופן סופי לאשש פגיעה של הכדור בבלון.

בנינו פונקציה שנקראת *'hit_or_miss'* שמטרתה לוודא אם הייתה פגיעה בבלון. פונקציה זו מקבלת הפלט של הפונקציה *'miss_or_maybe'*, כלומר:

- ה – $flag$ שלפיו מצאנו את הכדור.
- התמונה החתוכה של הכדור והבלון יחדיו (במקרה של חשד לפגיעה) ותמונה שחורה במקרה של פספוס.
- המסכה של הבלון.
- הרדיוס המקסימלי של הכדור.

נציג את השלבים של האלגוריתם (בכל השלבים אנו משתמשים בפונקציות שכבר הסברנו לפני כן ולכן לא נחזור על ההסבר):

1. נמצא את הכדור בתמונה על ידי צבע או על ידי Hough circles, כלומר נמצא את רדיוס ומרכז הכדור (כלומר על ידי קריאה לפונקציה *'Circles'* או *'find_orange'* בהתאם לערך של ה - $flag$).
2. נמצא את המסכה של הכדור על ידי *'ball_mask'* או *'fill_contours'* בהתאם לערך של ה - $flag$.
3. נפעיל אופרטור של AND בין המסכה של הכדור למסכה של הבלון שקיבלנו מהפונקציה *'miss_or_maybe'* שייצור תמונה שמכילה את החפיפה בין הכדור לבלון.

4. נקבע ערך סף שערכו יהיה עשירית מסכום הפיקסלים הלבנים במסכה של הבלון.

אם סכום הפיקסלים הלבנים בתמונת החפיפה בין הכדור לבלון גדול מערך הסף שקבענו אז נוכל לומר בוודאות כי הכדור פגע בבלון. אחרת, נדווח כי הכדור לא פגע בבלון.

תרשים זרימה של שלבי האלגוריתם

אנליזה עבור חלקי האלגוריתם

ישנם כמה נושאים שאותם נרצה לחקור ולהציג את האנליזה שלהם.

אנליזה ראשונה – זיהוי הכדור על ידי Hough Circles וצבע

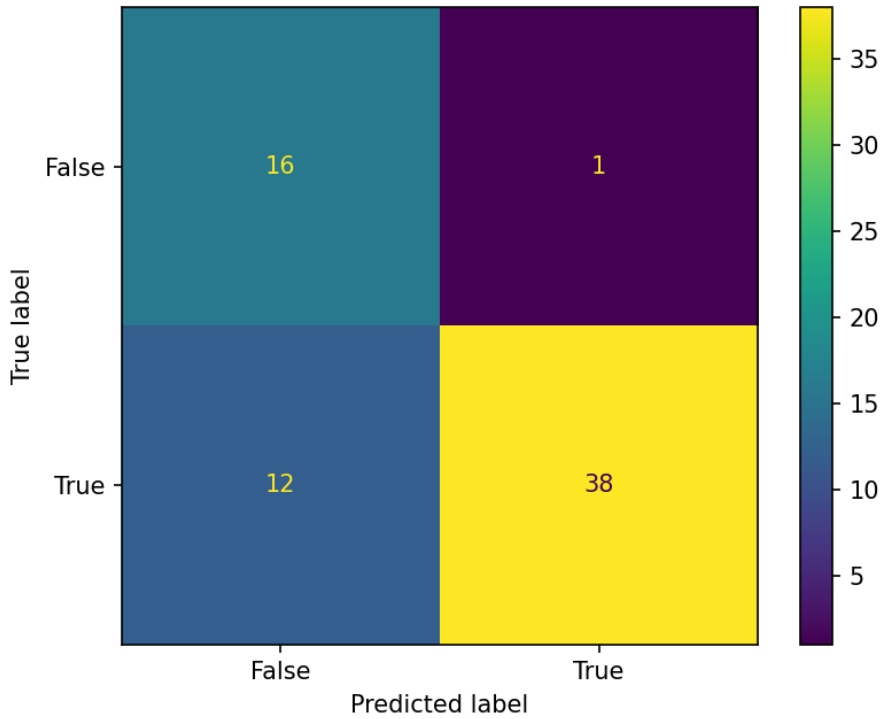
זיהוי הכדור על ידי Hough Circles בלבד:

במהלך בניית האלגוריתם המרכזי אשר מבצע את הזיהוי של הכדור, בלון ומיקום הפגיעה נתקלנו בקושי לזהות את הכדור. הרעיון ההתחלתי היה לזהות את מיקום הכדור (לאחר חיתוך התמונה לפי קצוות המסך) על ידי Hough Circles (כמתואר בהסבר של האלגוריתם) היות והכדור עגול.

בחלק מהמקרים האלגוריתם זה לא הצליח לזהות את הכדור שבתמונה מאחר שבעת הזריקה הוא שינה את צורתו העגולה במעט או שהצל של הכדור על המסך הקשה על זיהויו.

ביצענו ניתוח של האלגוריתם על 67 תמונות שבהם הכדור מצולם בעת פגיעתו במסך, לכל תמונה נתנו label ידנית (True במידה וכל הכדור במלואו נמצא על גבי המסך, False במידה והכדור לא נמצא על גבי המסך או לא נמצא בשלמותו). האלגוריתם מחזיר True במידה והצליח למצוא את הכדור על ידי Hough Circles, אחרת מחזיר False.

ניתן לראות את התוצאות של האלגוריתם ב – confusion matrix הבאה:



confusion matrix עבור מציאת הכדור על ידי Hough matrix בלבד.

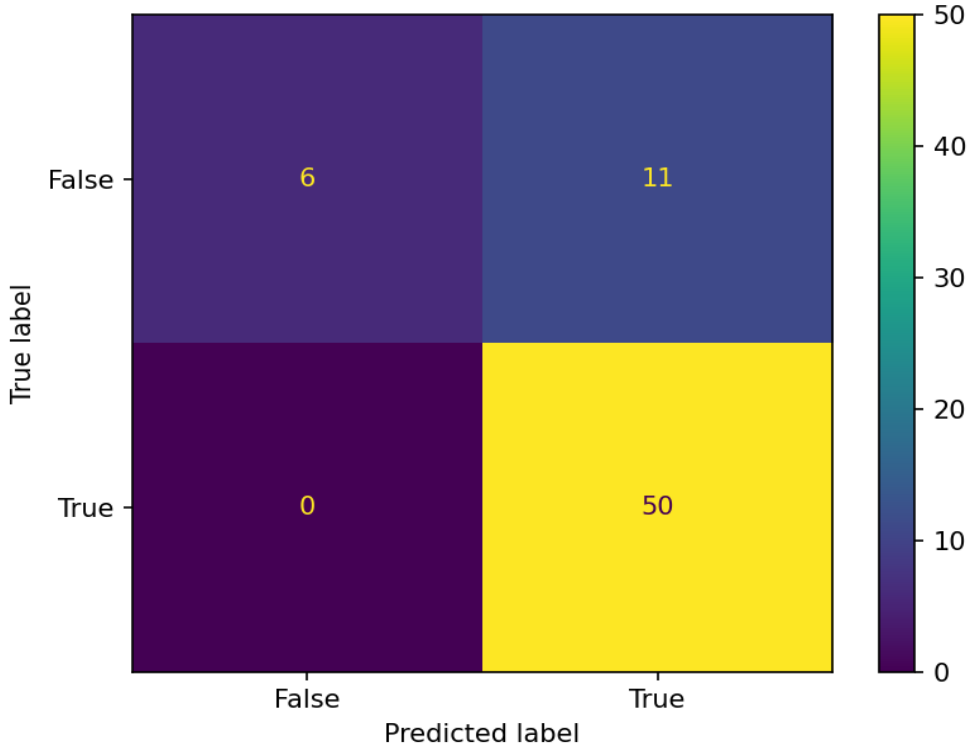
ניתן לראות שברוב המקרים אנחנו מזהים שהיה כדור בתמונה כשאכן היה כדור (צהוב). את התמונות שבהן אין כדור האלגוריתם ניתח נכון ואכן דיווח שאין כדור בתמונה. לפי המטריצה הנתונה האלגוריתם טעה ב- 12 תמונות מתוך 50 תמונות, כלומר דיווח כי לא הייתה פגיעה של הכדור במסך כאשר בפועל הכדור מופיע בתמונה של מסך המשחק, כלומר ישנה טעות של כ-24%. תמונות שבהן הכדור פוגע במלואו בתוך הבלון או ישנו צל רב מאחורי הכדור כך שהכדור מאבד את צורתו העגולה הן התמונות שהאלגוריתם שלנו לא יודע להתמודד איתן בצורה תקינה. כמובן שהתוצאות של האלגוריתם הנ"ל לא מספיק טובות לכן בחרנו בסופו של דבר לבצע זיהוי נוסף על ידי צבע.

זיהוי הכדור על ידי צבע בלבד:

צבעו של הכדור כתום ואינו משתנה ולכן נוכל לבצע את הזיהוי של הכדור בתמונה לפי צבעו. תחילה, הרעיון היה להחליף את האלגוריתם הקודם (זיהוי הכדור לפי Hough Circles) באלגוריתם החדש שמנתח את הצבעים התמונה. יצרנו מסכה לפי ערכי הצבעים צהוב וכתום (תחום הצבעים רחב כדי לתמוך בכמה זוויות צילום וגווניו של הכדור) במרחב הצבעים HSV, שיערכנו את רדיוס הכדור ואת מיקום מרכז הכדור אם הכדור נמצא בתמונה.

נציין כי גם תמונות שבהם רק חלק מהכדור מופיעה על המסך בתמונה מוגדרות כ- False לכן נצפה כי בתמונות אלו נקבל שהאלגוריתם זיהה כדור למרות שמבחינתנו לא היה.

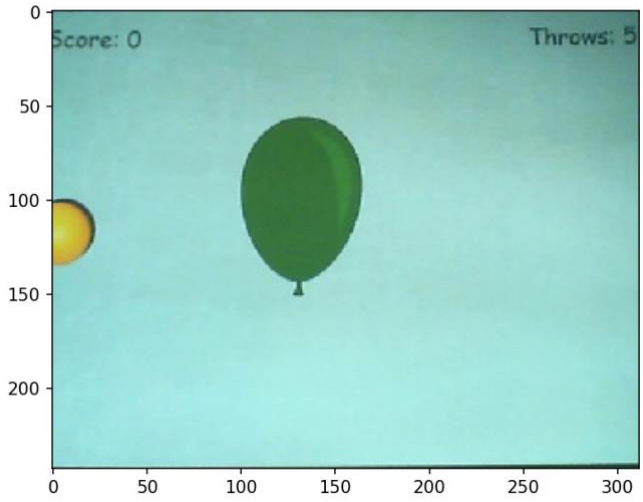
התוצאות של האלגוריתם הנ"ל מוצגות ב – confusion matrix הבאה:



confusion matrix עבור מציאת הכדור על ידי צבע בלבד.

ניתן לראות שאלגוריתם הצבע מזהה את כל התמונות שיש בהן כדור, אך כעת אנחנו מזהים גם תמונות שהכדור בחלקו מחוץ למסך, תמונות שמוגדרות כפספוס.

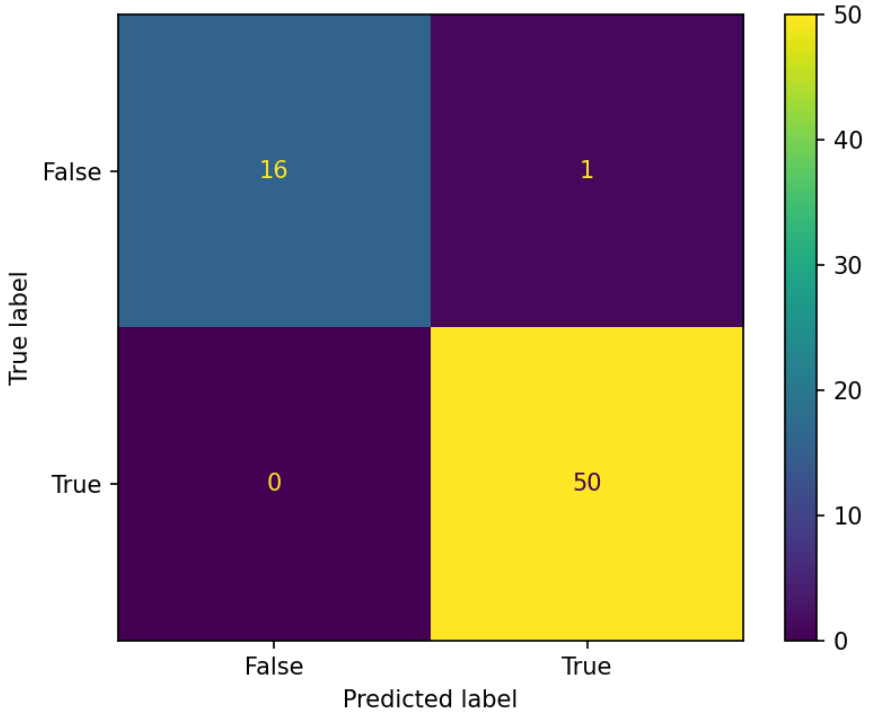
התמונה המוצגת למטה היינה דוגמה לבעיה הנוכחית:



המסקנה שהגענו אליה היא שעל מנת לקבל את התוצאות הטובות ביותר, נצטרך לשלב את האלגוריתמים.

תחילה נבדוק האם אנחנו מזהים מעגלים לפי השיטה הראשונה (Hough Circles) שהוצגה קודם, ובמידה ולא נמצא מעגל ניגש לזיהוי הכדור על ידי אלגוריתם הצבע.

ניתן לראות את התוצאות של האלגוריתם המאוחד ב – confusion matrix הבאה:



confusion matrix עבור מציאת הכדור על ידי צבע ו – Hough Circles.

על מנת להעריך את ביצועי האלגוריתם נשתמש ב – F1 scores אשר מוגדר על ידי הנוסחה הבאה:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

כאשר $precision, recall$ מוגדרים להיות:

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

לפי המטריצה המוצגת למעלה נקבל:

- $true\ positives = 50$
- $false\ negatives = 0$
- $false\ positives = 1$

נחשב:

- $precision = \frac{true\ positives}{true\ positives + false\ positives} = \frac{50}{50+1} = 0.9804$
- $recall = \frac{true\ positives}{true\ positives + false\ negatives} = \frac{50}{50} = 1$

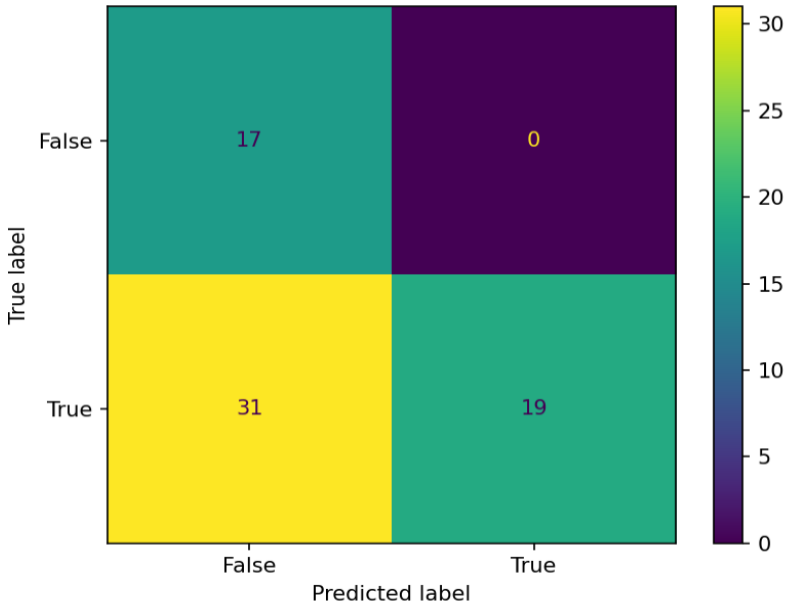
$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} = 2 \cdot \frac{0.9804 \cdot 1}{0.9804 + 1} = 0.9901$$

לסיכום, קיבלנו אלגוריתם אשר מזהה כדור בתמונה בדיוק של כ- 99%.

אנליזה שניה – זיהוי המעגלים ע"י Hough Circles עבור תמונת Grayscale / ערוץ ה Value.

במהלך האלגוריתם של זיהוי הפגיעה של הכדור בבלון השתמשנו רבות באלגוריתם ה – Hough Circles על מנת למצוא את הכדור על ידי זיהוי מעגלים. נבצע השוואה בין התוצאות של האלגוריתם כאשר התמונה שאותה נכניס לאלגוריתם היא ב – Grayscale לבין התוצאות כאשר נכניס את ערוץ ה – Value במרחב הצבעים של HSV. נבצע סימולציה דומה לזו שהוצגה למעלה, ניקח 67 תמונות שעליהן נרצה לבצע את הניתוח -האם ישנו כדור בתמונה (האלגוריתם שעושה שימוש רב ב-Hough Circles).

בשלב ראשון נבצע את הסימולציה כאשר התמונה שנכניס לפונקציה הינה ב- grayscale ונקבל את ה – confusion matrix הבאה:



confusion matrix עבור Hough Circles עבור תמונת Grayscale

על מנת להעריך את ביצועי האלגוריתם נשתמש ב – F1 scores אשר הוגדרה קודם:

- $true\ positives = 19$
- $false\ negatives = 31$
- $false\ positives = 0$

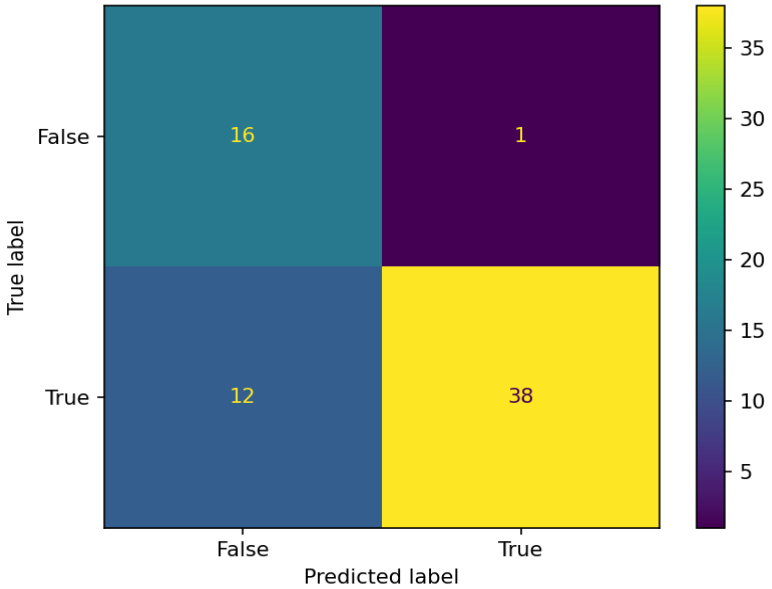
נחשב:

- $precision = \frac{true\ positives}{true\ positives + false\ positives} = \frac{19}{19+0} = 1$

– $recall = \frac{true\ positives}{true\ positives + false\ negatives} = \frac{19}{31+19} = 0.3800$

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} = 2 \cdot \frac{1 \cdot 0.3800}{1 + 0.3800} = 0.5507$$

לסיכום, קיבלנו אלגוריתם אשר מזהה כדור על ידי זיהוי מעגלים בתמונה בדיוק של כ- 55%.
נשים לב כי קיבלנו שברוב המקרים האלגוריתם לא הצליח לזהות מעגלים ולכן קבע שאין כדור בתמונה כאשר בפועל היה כדור בתמונה. בנוסף, ניתן לראות שאת כל התמונות שאין בהן כדור האלגוריתם זיהה בצורה נכונה. כאשר נבצע את אותה הסימולציה על אותן התמונות, אך כעת נכניס ל –Hough Circles את הערוץ ה – Value במרחב ה-HSV ונקבל את ה – confusion matrix שקיבלנו עבור האנליזה הראשונה:



confusion matrix עבור מציאת הכדור על ידי Hough matrix בלבד.

כעת קיבלנו תוצאות טובות יותר. ניתן לראות שהאלגוריתם מצליח לזהות טוב יותר את המעגלים בתמונות ולכן מזהה בצורה נכונה יותר תמונות שבהם יש כדור. ניזכר שערוץ ה – Value מתאר את בהירות הצבע. הערך הנע בין 0 ל-1 כאשר 0 מייצג את הצבע השחור ו-1 הוא הצבע הבהיר ביותר בהתאם לכל גוון. ניתן להסביר את השוני בין שתי הגישות על ידי כך שערוץ ה – Value מתאר את השינויים בצבע בצורה מדויקת יותר. שיטת Hough Gradient משתמשת במידע של הגרדיאנטים של קצוות. כלומר, בערוץ ה – Value יש יותר מידע על הפרשי הצבע בין הקצוות ולכן מזהים בערוץ זה את המעגלים בצורה יעילה יותר.

על מנת להעריך את ביצועי האלגוריתם נשתמש ב – F1 scores אשר הוגדרה קודם:

- $true\ positives = 38$
- $false\ negatives = 12$
- $false\ positives = 1$

נחשב:

- $precision = \frac{true\ positives}{true\ positives + false\ positives} = \frac{38}{38+1} = 0.9744$

- $recall = \frac{true\ positives}{true\ positives + false\ negatives} = \frac{38}{38+12} = 0.7600$

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} = 2 \cdot \frac{0.9744 \cdot 0.7600}{0.9744 + 0.7600} = 0.8539$$

לסיכום, קיבלנו אלגוריתם אשר מזהה כדור על ידי זיהוי מעגלים בתמונה בדיוק של כ- 85%, שיפור משמעות בביצועים ביחס לשיטה הקודמת (כאשר העיבוד נעשה על תמונה ב – grayscale).

חיבוריות המערכת לעבודה בזמן אמת

על מנת לקבל מערכת שפועלת בזמן אמת בצורה מהירה ותקינה מימשנו את הפרויקט ב- python ומקבלנו את תהליכי העיבוד על ידי תהליכונים ("threading"). ישנם סה"כ שלושה תהליכונים:

- תהליכון של המשחק
- תהליכון של צילום התמונות ושמירתן
- תהליכון של עיבוד התמונות

כל התהליכונים רצים במקביל ללא תלוי אחד בשני, לעיתים אנחנו מתנאים חלק מהפעולות באירועים ("events") שתלויים בתהליכונים שונים. בעליית המערכת נפעיל את ריצת כל התהליכונים כאשר כל תהליכון פועל בלולאת מתמדת (לולאת while true).

תהליכון של המצלמות (cameras_function):

תחילה נפעיל את שתי המצלמות, את תדירות הצילום נגדיר להיות 30 FPS כמו הגרפיקה של המשחק. כאשר המצלמות מסיימות את אתחולן אנחנו שולחים event_start ליתר המערכת שאומר כי ניתן להתחיל להריץ את המשחק עצמו.

ישנן שתי מחסניות המוגדרות להיות גלובליות וכל מחסנית שומרת 16 תמונות סה"כ. השמירה של התמונות מתבצעת במידת הצורך ולא תמיד. נבדוק כל תמונה שמתקבלת ממצלמה שמצלמת מלמטה-זיהוי פגיעה בקיר- האם ישנו צבע כדור בפריים הנוכחי (ישנו כדור בפיירם). רק במידה וזיהינו את הצבע הכתום וגווניו בפריים נשמור את התמונה במחסנית המתאימה לאותה המצלמה. את השמירה של התמונות המתאימות למצלמה שממוקמת ממול לקיר-מזהה את הפגיעה בבלון – נבצע במקביל לשמירה של התמונות במחסנית הראשונה.

על מנת לחסוך זמן עיבוד ושהמערכת תעבוד בזמן אמת רצינו לא להריץ את האלגוריתם של עיבוד התמונה על כל פריים שמתקבל מהמצלמות ולכן שמרנו רק את התמונות שמעניינות אותנו (היות וברוב התמונות אין זריקה/פגיעה בקיר/פגיעה בבלון). בגלל שאנחנו יודעים את צבע הכדור ושהוא אינו משתנה במהלך המשחק בחרנו לבצע את הסינון של התמונות לפי צבע.

בגלל המקבול והגישה למחסניות בכמה תהליכונים במקביל השתמשנו בנעילה של המחסניות בעת השימוש בהן. lock_pop ו-lock_hit הינם המפתחות שבעזרתן נעלנו את המחסניות בעת הקריאה מהן וכתיבה אליהן.

תהליכון של המשחק (game_main):

כל ממשק המשחק פועל בתהליכון נפרד, התחלת המשחק מותנה בעליית המצלמות ועל כן הפונקציה שאחראית על המשחק תלויה באירוע – event אשר יגיע מהתהליכון שאחראי על המצלמות (כך בעצם מתבצעת התקשורת בין התהליכונים השונים).

המשחק בכל רגע נתון מעדכן את המסך ואת תזוזת הבלון, במידה וישנו אירוע של פגיעה בקיר (event אשר יגיע מהתליכון המרכזי שמבצע את עיבוד התמונה והזיהוי) המשחק במיידי מעדכן את מספר הזריקות שנותרו לשחקן. ללא תלות בפגיעה בקיר, במידה והיה event של פיצוץ הבלון אשר מוגדר באלגוריתם שמזהה את מיקום הכדור ביחס למיקום הבלון נעדכן את התצוגה-פיצוץ הבלון ונעלה את הניקוד של השחקן. כאשר נסיים את המשחק, כלומר כשהשחקן זרק את כמות הזריקות המוגדרת מראש (המיינה של כמות הזריקות יורדת ל-0) נציג מסך סיום משחק. בתצוגה הסופית נציג את כמות הזריקות המוצלחות שפגעו בבלון מתוך סך הזריקות שהשחקן זרק, בנוסף ניתן אפשרות לשחקן להזין את שמו. לאחר שהשחקן יזין את שמו נציג טבלת סיכום אשר מציגה את עשרת השחקנים הטובים ביותר ואת ניקודם בהתאמה.

על מנת להפסיק את הרצת הקוד- הפסקת הפעילות של כלל התהליכונים, נצא מהתצוגה של המשחק, כלומר נלחץ על האיקס של חלונית המשחק.

תהליכון של עיבוד התמונות (detect_direction_change):

אלגוריתם של עיבוד התמונות משתי המצלמות מתבצע בתהליכון נפרד. בכל רגע נתון נעבד ארבע תמונות מהמחסנית של המצלמה שמזהה את הפגיעה בקיר- במידה ונזהה על ידי אלגוריתם- optical flow שהיה שינוי בכיוון התנועה, ניגש למחסנית השנייה ונעבד את התמונה המתאימה. נבדוק האם

הייתה פגיעה בבלון או שהייתה רק פגיעה בקיר ללא פיצוץ הבלון. במידה ונזהה פגיעה בבלון נדליק את ה- event המתאים (על מנת שממשק המשחק יוכל לעדכן את האנימציה). ההפרדה של עיבוד התמונות לשני אלגוריתמים נפרדים בוצע על מנת למקבל את המערכת ולגרום לכל המשחק לפעול בזמן אמת ולהגיב לשחקן בהתאמה. נשים לב, האלגוריתם המרכזי של עיבוד התמונה מתבצע אך ורק על תמונות שבהן היה שינוי בכיוון התנועה, כלומר הייתה פגיעה בקיר ובכך הצלחנו לצמצם את זמני העיבוד הכוללים של המערכת.

נדגיש שבכל פנייה למחסניות (קריאה או כתיבה) ננעל תחילה את הגישה לתא הרלוונטי על מנת למנוע עיבוד תמונה של רעש. במידה ושני תהליכונים ניגשים במקביל לאותו התא במחסנית עלולה להיווצר התנגשות שלעיתים תשמור רעש או תמונות לא תקינות במחסנית והנעילה של התא המתאים לפני מונע זאת.

סיכום

נראה שהמערכת מתמודדת היטב עם אלגוריתמי עיבוד התמונה בזמן אמת. המערכת משלבת מספר אלגוריתמים כאשר כל אחד מהם בנפרד עובד בצורה יעילה ובשילוב של כלל המערכת הביצועים אינם נפגעים.

ישנם אלגוריתמים יעילים ומדויקים יותר מאלגוריתם optical flow, אך מכיוון שלא רצינו להוסיף אלגוריתמים המתבססים על למידת מכונה בחרנו להתמקד ב – optical flow ולייעל את הפרמטרים על מנת לקבל תוצאות טובות יותר. הקושי היה בבחירת התמונה המדויקת שמייצגת בצורה הטובה ביותר את רגע הפגיעה של הכדור בקיר, ראינו כי בחלק מן הפעמים זה השפיע על יעילות זיהוי פגיעת הכדור בבלון אך באופן כללי, מאחר שהשגיאה בבחירת התמונה הייתה יחסית קטנה, האלגוריתם הצליח להתמודד ולספק תוצאה טובה.

במהלך הפרויקט הכרנו בצורה מעמיקה את האלגוריתמים של עיבוד תמונה, השתדלנו לחשוב בצורה יצירתית על מנת לספק פתרונות למקרי קצה והקפדנו על עבודה בזמן אמת.

הצעות לשיפור המערכת הן:

1. הוספת תמיכה בסוגי זריקות שונות וקיצוניות הן במהירות והן בכיוון הזריקה.
2. ביצוע אופטימיזציה לקוד המשחק לצורך ייעול ומהירות תגובה.
3. ניתן לשדרג את ממשק המשחק ולהוסיף תמיכה במשחק של שני משתתפים.

סרטון של הפרויקט

https://youtu.be/GboBYjXLzr0

ביבליוגרפיה

מציאת מסכה בינארית:

https://www.geeksforgeeks.org/python-thresholding-techniques-using-opencv-set-1-/simple-thresholding

זרימה אופטית:

/https://www.geeksforgeeks.org/opencv-the-gunnar-farneback-optical-flow

/https://learnopencv.com/optical-flow-in-opencv

:Hough Circles

/https://theailearner.com/tag/cv2-houghcircles