פרויקט "דיוק מלבנים" בקורס עיבוד תמונה ספרתי

מרצה: ג'יהאד אל-סאנע

מגישים: יהונתן פרידמן ועמית בינפלד

תיאור הבעיה ומטרת הפרויקט:

- נתונה תמונה כלשהי של צילום המכיל אובייקט מלבני, ונתונות גם 4 נקודות שמקרבות את 4 הפינות של המלבן, אשר אדם מיקם אותן כשהתבקש לסמן את 4 הפינות של המלבן. האדם התבקש לעבור על סט גדול של תמונות בזמן מוגבל ולכן הוא היה יחסית פזיז ולא מדויק. מטרת הפרויקט לקחת את התמונה ואת 4 הנקודות הנתונות ולדייק את המלבן כמה שניתן. הפלט של התוכנית הוא 4 נקודות חדשות, שאמורות לייצג בצורה יותר מדויקת את המלבן שצולם בתמונה.

הצעה לדרך פתרון:

- דרך הפתרון שאנחנו מציעים מתחלקת ל-2 שלבים, אשר מטרותיהם מתוארות בתרשים הבא:

איתור פינות

- בשלב הראשון המטרה היא לאתר כמה שיותר פיקסלים המהווים פינות איכותיות בתמונה, במטרה למצוא קבוצה יחסית מצומצמת של פינות שבהן נמצאות גם הפינות האמיתיות של האובייקט המלבני.

קביעת הפינות המתאימות

- לאחר שלב איתור הפינות, אנחנו מנסים לאתר בחוכמה, ליד כל נקודה (מתוך 4 הנקודות הקלט שהמשתמש נתן), נקודה מתאימה מבין אלו שנמצאו בשלב הקודם, במטרה למצוא את 4 הפינות האמיתיות של האובייקט המלבני.

שלב 1 - איתור פינות:

על מנת לתפוס בסיכויים גבוהים את הפינות הרצויות, בשלב זה צריך לקחת בחשבון שהפעלת אלגוריתם סטנדרטי של harris לאיתור פינות לבדו עלול לא להספיק. התמונות שאנחנו מקבלים הן צילומים מורכבים ולא כל הפינות בהם נתפסות בקלות באלגוריתם. לכן החלטנו בשלב זה להפעיל מספר אלגוריתמי איתור פינות על מגוון וריאציות של התמונה המקורית ועם מגוון פרמטרים, ולאחד את כל הפינות שנמצאו, במטרה לקבל סט פינות שבסבירות גבוהה מכיל בין היתר את הפינות האמתיות של האובייקט המלבני המבוקש.

<u>איך השלב הזה מתבצע?</u>

אנחנו מפעילים את אלגוריתם harris לאיתור פינות על התמונה המקורית מספר פעמים, כל פעם עם פרמטרים שונים. בנוסף אנחנו מפעילים גם אלגוריתם לאיתור פינות בשם Shi-Tomasi, גם אותו על התמונה המקורית עם מספר פרמטרים שונים. כמו כן, אנחנו מבצעים לתמונה המקורית canny edge detection עם מגוון פרמטרים, ועל כל תוצאה מפעילים שוב את שני האלגוריתמים הנ"ל לאיתור פינות עם מגוון פרמטרים. בנוסף, אנחנו לוקחים את התמונה המקורית, ופעם אחת מחדדים אותה, ופעם שנייה מחליקים אותה, וחוזרים שוב על השלבים שתוארו, הפעם עם התמונות האלה.

<u>יש כאן trade-off:</u>

מצד אחד, אם אנחנו נפעיל כמה שיותר פעמים את אלגוריתמי איתור הפינות, אנחנו נקבל סט גדול יותר של נקודות, ובסבירות גבוהה הנקודות האלו יכילו את 4 הפינות האמתיות של האובייקט המלבני. זה כמובן דבר רצוי, שכן זוהי בדיוק הגדרת ההצלחה של שלב 1. מצד שני, אם האלגוריתמים ימצאו יחד יותר מדי נקודות בקרבת הפינות של האובייקט המלבני, זה עלול לסבך את שלב 2, ויהיה צורך בזמן חישוב משמעותית גדול יותר על מנת להביא להצלחת שלב 2.

לכן, ניסינו למצוא את נקודת האיזון לאחר עשרות בדיקות שביצענו, וכך בחרנו בקפידה את האלגוריתמים והפרמטרים בשלב זה לאיתור הפינות.

דוגמה לקלט:

תוצאת הפעלת שלב 1 לאיתור פינות:

שלב 2 - קביעת הפינות המתאימות:

בשלב זה של האלגוריתם, מסתכלים על כל נקודה מתוך 4 הנקודות של הקלט, ומוצאים בקרבתה את n הנקודות הקרובות ביותר אליה, מבין הנקודות שנמצאו בשלב הקודם. n הוא פרמטר בקוד, בדרך כלל יהיה בין 5 ל- 15. אנחנו נראה מיד שהוא משפיע מאוד על זמן הריצה של האלגוריתם.

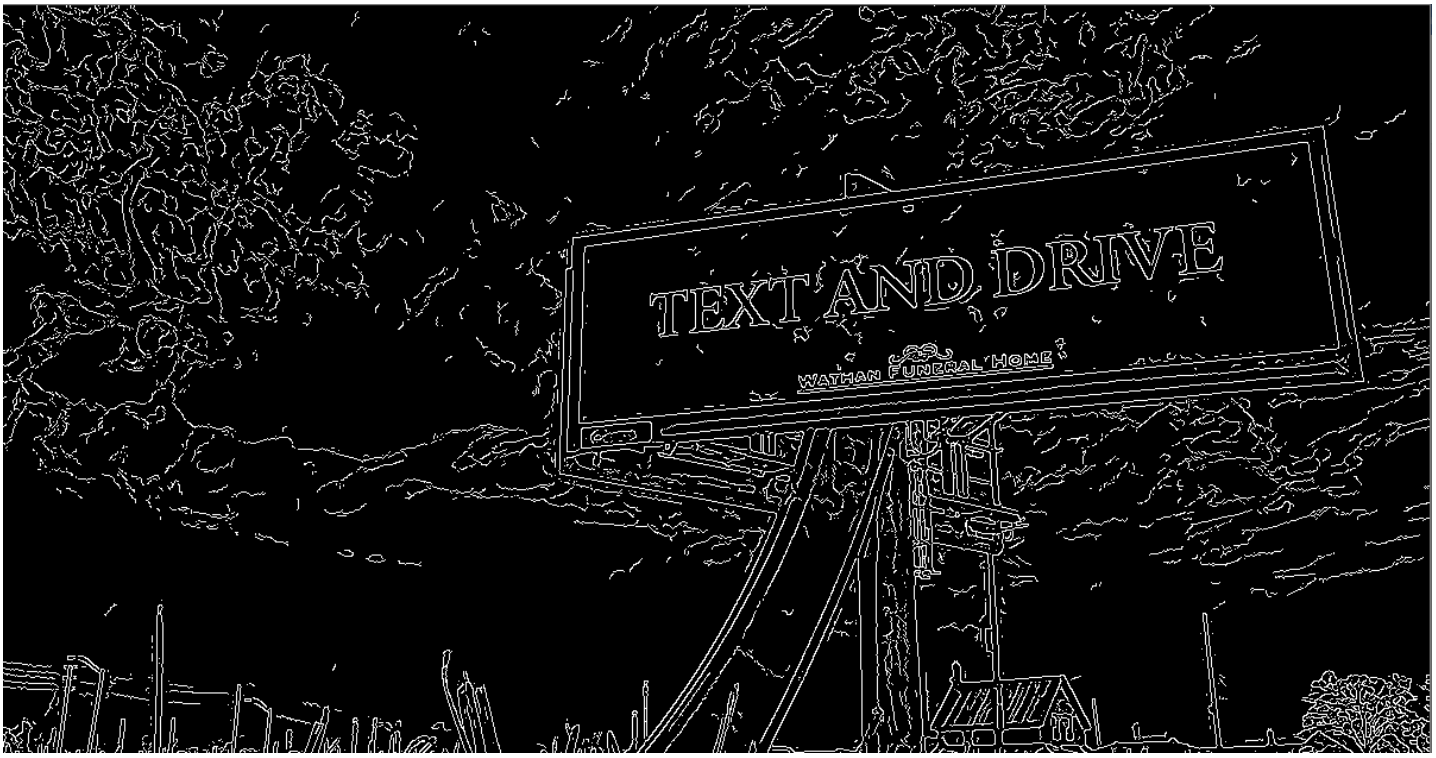
<u>הערה</u>: אנחנו בוחרים להסתכל סביב כל נקודה על חלון בגודל 50x50 סביבה, ולכן אם יש בסביבה הזאת פחות מ- n נקודות, אז נבחר עבורה מספר קטן מ- n של נקודות. לצורך ניתוח הסיבוכיות של האלגוריתם, אנחנו נניח את המקרה הגרוע ביותר, שסביב כל נקודת קלט יש n נקודות בדיוק שנמצאו מהשלב הקודם.

כעת, אנחנו עוברים על כל הקומבינציות האפשריות של בחירת נקודה לכל נקודת קלט, מבין הקבוצה שהגדרנו לה. מדובר ב- n^4 קומבינציות אפשריות. כאשר $n = 5$ מדובר ב- $625 = 5^4$ אפשרויות שצריך לבדוק, וכאשר $n = 15$ מדובר ב- $50{,}625 = 15^4$ אפשרויות, מה שמשמעותית מאריך את זמן הריצה.

עבור כל קומבינציה j כזאת, אנחנו מנסים להעריך כמה טוב הפתרון יהיה לקחת את 4 הנקודות הללו כפינות האמתיות של האובייקט המלבני. בסוף, אנחנו בוחרים את 4 הפינות שמקבלות את הציון הטוב ביותר.

<u>כיצד נותנים ציון עבור 4 פינות פוטנציאליות של קומבינציה j?</u>

אנחנו מבצעים edge detection לאחר שאנחנו מחליקים את התמונה, ומקבלים תמונה (שנקרא לה $edge\_for\_score$):



כעת אנחנו עוברים על 4 הצלעות של המלבן, $l^{1^j}, l^{2^j}, l^{3^j}, l^{4^j}$, הנוצרות מ-4 הנקודות האלו. עבור כל צלע l^{i^j} אנחנו בודקים "כמה" היא יושבת על edge. אנחנו מודדים זאת באחוזים, למשל 90% מהצלע יושבת על edge. אבל, המדידה קצת פחות נאיבית ממה שנדמה. נסביר כיצד המדידה מתבצעת:

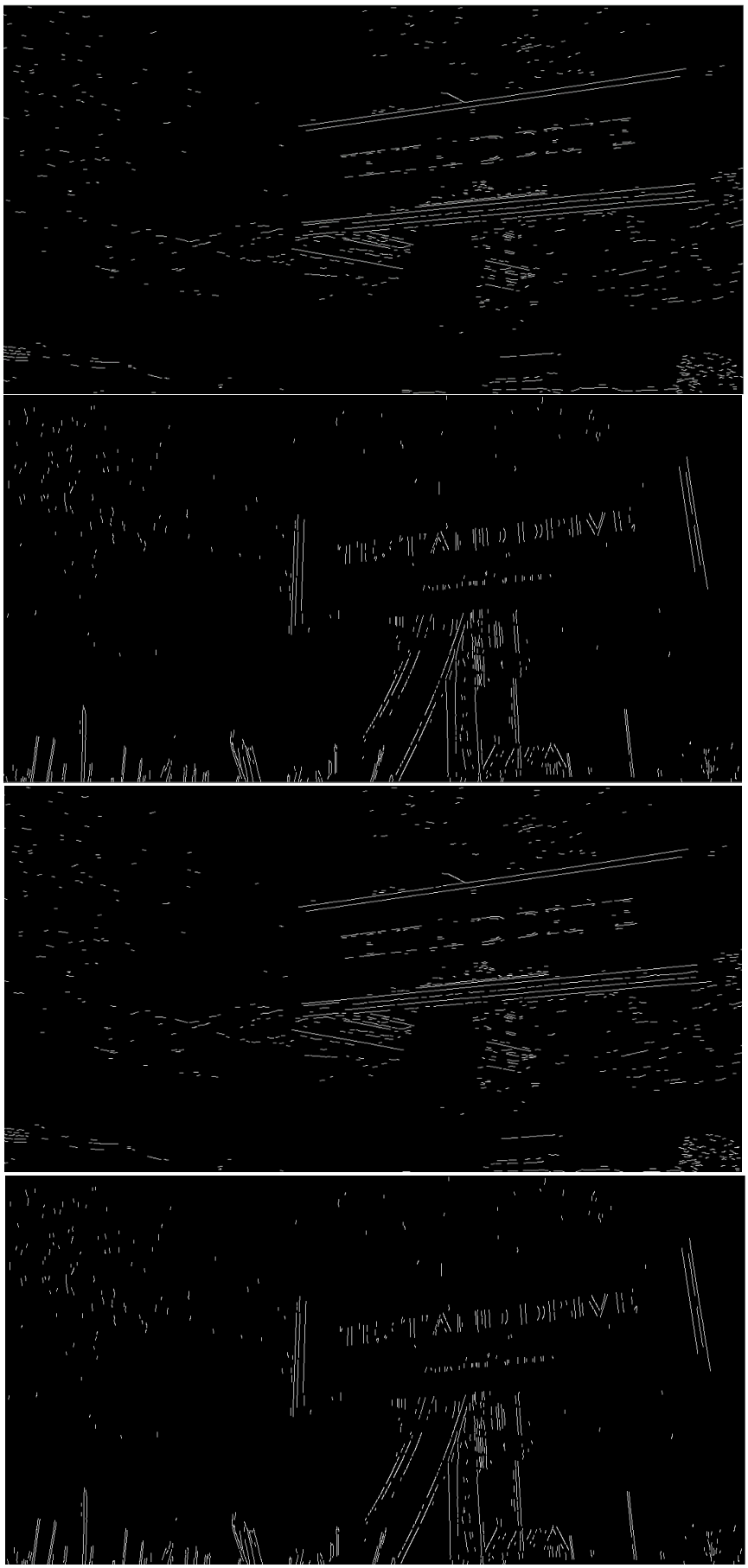
<u>שלבי המדידה</u>:

- מראש, אנחנו מחשבים עבור 4 נקודות הקלט, את משוואות הצלעות שהן יוצרות, k^1, k^2, k^3, k^4 , ומחשבים את השיפועים של הצלעות, m^1, m^2, m^3, m^4. זה הזמן להזכיר את ההנחה שלנו, שהאדם שסימן את 4 הנקודות הללו ניסה להיות מדויק באופן יחסי, ולכן הצלעות האלו מייצגות פחות או יותר את הצלעות של האובייקט המלבני המבוקש.
- בהתאם לכל שיפוע m^i שמצאנו, אנחנו מבצעים $opening$ ל-$edge\_for\_score$ על מנת לנקות רעשים (בעיקר edges שנמצאים בסביבה, שהם לא קשורים לצלע שאנחנו מנסים לתפוס). ה- kernel של ה-

$opening$ תלוי בשיפוע. אנחנו מקבלים תמונת edges מעובדת שנקרא לה $edge\_for\_score\_opening^i$.

- כעת, עבור כל אחד מהישרים l^{i^j}, אנחנו עוברים על כל הפיקסלים שעל הישר ובודקים האם הוא, או אחד השכנים שלו (ב- connectivity 8), מקבל edge בתמונה $edge\_for\_score\_opening^i$. אם כן, מגדילים מונה ב- 1. בסופו של דבר בודקים מהו היחס בין גודל המונה לבין מספר הפיקסלים על הישר, וזה הציון שנותנים לישר. כמובן, ככל שהציון יותר גדול, זה אומר שהצלע יותר מתאימה.
- עבור קומבינציה $\boldsymbol{j}$, הציון שנותנים לה הוא הציון המינימלי מבין 4 הציונים של הישרים $l^{1^j}, l^{2^j}, l^{3^j}, l^{4^j}$. ככה אנחנו מבטיחים שאם צלע שמצאנו לא מייצגת כראוי את הצלע של המלבן האמתי, הפתרון כולו יקבל ציון נמוך.

תוצאות של $edge\_for\_score\_opening^i$:



התוצאה הסופית:

הקומבינציה הטובה ביותר קיבלה ציון של 83.33%. אפשר לקרוא את הציון הזה כציון של האלגוריתם. האלגוריתם מחזיר את התוצאה הטובה ביותר שהוא מוצא, וגם מכמת את אחוז הביטחון שלו באיזשהו אופן...

<u>הערה:</u>

האלגוריתם עשוי להתקשות עם תמונות גדולות, משום שכאשר התמונה גדולה, המשתמש עשוי לפספס את הפינות האמתיות של המלבן בצורה גסה יותר, ושלב 1 של האלגוריתם לא מודע לזה. בשלב 1 נקבל מספר corners בצפיפות גבוהה ולכן נדרש ל n הרבה יותר גדול כדי לתפוס את הפינות האמתיות, וזה יפגע משמעותית בזמן הריצה.

<u>הפתרון שלנו לבעיה:</u>

כל תמונה שמכילה מספר פיקסלים מסוים או יותר (1,000,000 תעבור scale down והאלגוריתם יתבצע על התמונה המוקטנת. בסופו של דבר, התוצאה תומר בחזרה לתמונה המקורית.

הפתרון נובע מההנחה, ומהתצפיות שלנו, ששינוי גודל התמונה, לא פוגע משמעותית ב- edges וב- corners של התמונה.

הבחירה של 1,000,000 היא די שרירותית, ניתן לשנות אותה בתוך הקוד.

החיסרון בשיטה הזאת הוא שכשאנחנו מוצאים בתמונה המוקטנת את הפינות האמתיות של האובייקט המלבני, אנחנו עושים לנקודות scale up חזרה לתמונה המקורית, וכאן עשויים להיות אי דיוקים מינוריים. לכן מי שלא רוצה לוותר רמת הדיוק, שיעבד את התמונה המקורית בגודל המקורי, וישלם על כך בזמן ריצה.

<u>יתרונות של האלגוריתם:</u>

- <u>אם יש לך יותר זמן, תוכל לקבל תוצאות טובות יותר</u>: על ידי הגדלת הפרמטר n, ניתן לשפר את שלב 1 ולהבטיח בסיכויים גבוהים יותר, שלא נפספס את הפינות האמתיות בשלב 1. נסביר שוב את הקשר בין שני הדברים: שיפור שלב 1, ומציאת יותר נקודות מועמדות, עשוי לגרום לכך שסביב כל נקודת קלט תהיה כמות גדולה של נקודות מועמדות, וכדי שלא נפספס ביניהן את הנקודה האמתית בשלב 2, כדאי לבחור n יחסית גדול.
- <u>מתאים למקבול</u>: בהמשך ליתרון קודם, האלגוריתם ניתן למקבול בקלות. רוב העבודה מתבצעת במעבר על כל n^4 הקומבינציות. האיטרציות לא תלויות ולכן ניתנות למקבול באופן טריוויאלי.
- <u>מתאים להכללה</u>: ניתן להכליל את האלגוריתם בקלות לדייק גם פוליגונים עם יותר מ- 4 צלעות.