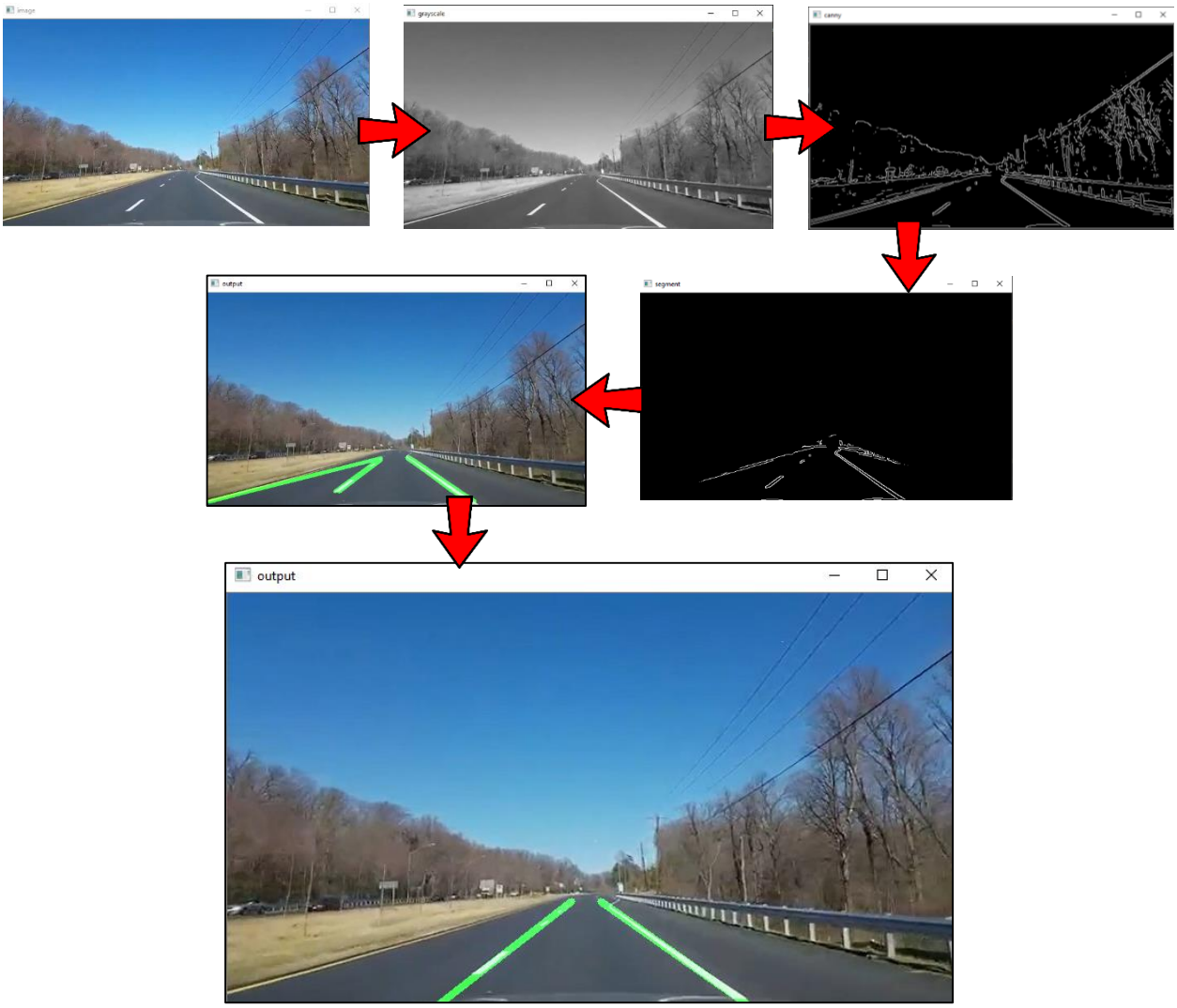
Lane Detector

Chen Asaraf and Or Avital

https://github.com/chenAsaraf/Lane-detection-.git

Hough transform approach

implementation by python with ‘numpy’ library



הקדמה/Introduction

על מנת לנהוג בצורה תקינה בכביש סטנדרטי על הנהג להיות ער לתנאי הדרך, לתמרורים וסימונים שונים על הכביש, ולפעול על פיהם. בעולם המתקדם יותר מתמיד לכיוון של מערכות נהיגה אוטונומיות יש חשיבות רבה לפיתוח אלגוריתמים המסוגלים לבצע את פעולת העיבוד של תנאי הדרך והסימונים ולקבל החלטות.

אחד הסימונים הבסיסיים והחשובים ביותר בכביש אלו הם הקווים המסמנים את הנתיבים, הם מדריכים את הנהג על מנת שיישאר בנתיבו, שידע לאן לכוון את ההגה (בהתאם לעיקולים או פניות בכביש) וכן מהווה בסיס לתקשורת בין הנהגים השונים בכביש. לכן אין זה מפתיע כי אחד השלבים הראשוניים וההכרחיים בפיתוח רכבים ללא נהג, הוא לזהות נתיבים ולעקוב אחריהם .

בפרויקט זה ניסינו לחכות את שלבי עיבוד המידע המתבצעים בצורה טבעית במוחו של נהג, ובאמצעות טכניקות של עיבוד תמונה וראייה ממוחשבת ליצור אלגוריתם המזהה ועוקב אחר הנתיבים בכביש.

ניתן לחלק את שלבים אלו לכמה חלוקות גסות :

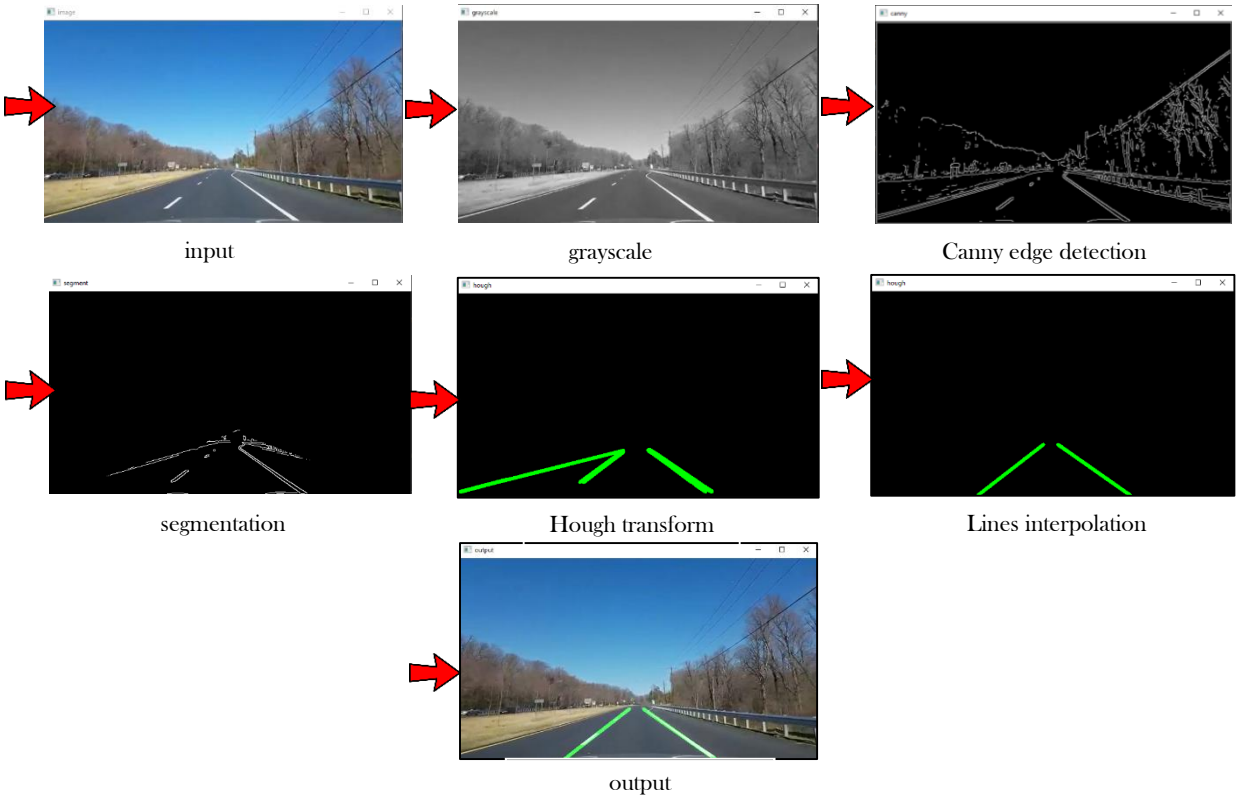
1. קבלת תמונה ויזואלית
2. עיבוד התמונה – ניתוח התמונה על מנת להכין את הקרקע להסקת מסקנות ממנה
3. קבלת החלטות - זיהוי הקווים עצמם

אם נפרט את השלבים הללו כאלגוריתם לזיהוי הנתיבים :

קלט הבעיה : רצף תמונות כווידאו

1. המרת התמונה למרחב צבע אפור
2. טשטוש התמונה באמצעות Gaussian kernel
3. מציאת קווי מתאר של צורות בתמונה
4. בחירת חלק רלוונטי לתמונה כדי לבצע עליו את עיבוד התמונה (טרפז תחתון בתמונה)
5. הפעלת אלגוריתם של Hough transform לזיהוי קווים
6. שילוב תוצאה זו בתמונת הקלט לתצוגה

להלן תצוגה כללית של השלבים בביצוע :



Approach and Method/ שיטה ומימוש

מרבית הנתיבים מיועדים להיות פשוטים יחסית לא רק כדי לעודד את הסדר, אלא גם להקל על הנהגים האנושיים לנווט רכבים במהירות קבועה. לפיכך, הגישה האינטואיטיבית שלנו עשויה להיות לאתר לראשונה קווים ישרים בולטים דרך איתור קווי מתאר וטכניקות שונות לחילוץ תכונות נוספות מהתמונה.

אנו עושים שימוש ב- OpenCV, ספריית קוד פתוח של אלגוריתמים לראיית מחשבים, ליישום.

האלגוריתם המוצע מתבסס על שני אלגוריתמים מרכזיים אותם נפרט בהמשך:

1. Canny edge detector
2. Hough transform

להלן פירוט השלבים ביצירת מזהה נתיבים:

I. קבלת קלט: וידאו ממצלמת dashboard

סרטון הדגימה שלנו יוזן לאלגוריתם כסדרה של תמונות רצופות (frames) בפרקי זמן של 10 אלפיות שנייה.
ניתן לפרוש מהתוכנית בקול עת על ידי לחיצה על מקש 'q'.

II. הפעלת Canny edge detector:

ראשית, נפרט על אלגוריתם זה ונבין את דרך פעולתו על מנת שהמימוש יהיה יותר בהיר.
Canny edge detector הוא אלגוריתם רב שלבי האופטימלי לניתוח בזמן אמת של קווי מתאר בתמונה.
המטרה המרכזית של האלגוריתם היא זיהוי שינויים חדים בבהירות התמונה (מעבר של פיקסלים צמודים משחור ללבן לדוגמא), והגדרתם כ'צלעות', בהינתן סף מסוים של הגבלות אותן השינויים החדים צריכים לקיים.

האלגוריתם מורכב מחמישה שלבים:

1. הפחתת רעשים בתמונה כהכנה לשלב הבא
2. חישוב הנגזרות החלקיות (Gradient calculation)
3. 'דיכוי לא מרבי' – חידוד של הצלעות על ידי בחירת מרכז הצלע והשחרת הפיקסלים שבקצוות. (Non-maximum suppression)
4. סינון של תוצאות הנגזרות על ידי סף כלשהו (Threshold) על תמונת הנגזרות
5. דילול של אזורים גדולים בעלי שינוי נגזרות ('רכסים') לאזור ברוחב פיקסל אחד. (Hysteresis)
6. לבסוף- לחבר את הקווים האלו כדי שיהיו באמת קווי מתאר ברורים.

פתרון זה מתבסס על ההנחה כי כל קו מתאר שחשוב יכיל פיקסלים חזקים מספיק כדי לעבור את הסינון הראשוני. נשים לב בנוסף שהאלגוריתם רגיש לרעשים בהתאם לגודל הGaussian kernel שנבחר.

Pseudo code:

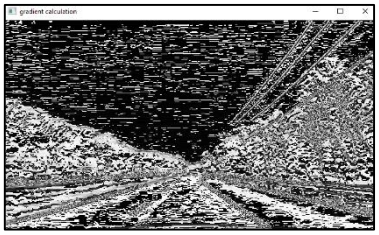
קלט : תמונת RGB

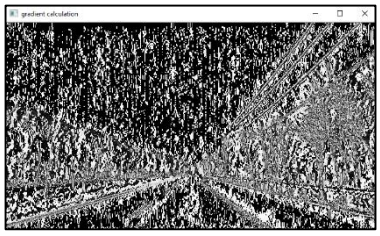
פלט : תמונה בינארית בה הפיקסלים הלבנים הם קווי המתאר

1. העברת התמונה למרחב צבעים אפור – grayscale image

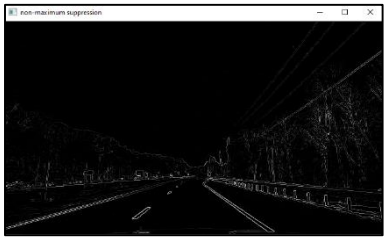
2. טשטוש התמונה באמצעות קונבולוציה (convolution) עם Gaussian kernel .

3. מציאת הגודל והכיוון של הנגזרות החלקיות :
באמצעות Sobel kernel לאורך ציר ה-x וציר ה-y של התמונה
דוגמא לתמונות הנגזרות – (באלגוריתם לא משתמשים בתמונה עצמה אלא בגודל ובכיוון)



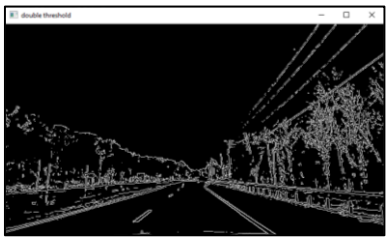


4. 'הצרה' של הצלעות : עבור כל פיקסל נבדוק בכיוון הצלע שעליו הוא נמצא את שני הפיקסלים שמצדדיו (הגרדיאנט נורמל לכיוון הצלע). אם הוא מקסימום מקומי – הפיקסל יעבור לשלב הבא בו אולי יעבור 'דיכוי לא מרבי'. אחרת- ערך הפיקסל נקבע להיות 0 (שחור).



5. Threshold : העברת תמונת הצלעות בשני ספים. כאשר הרף גבוה- מנסים לזהות את הצלעות החזקות (כל מה שגדול מהסף המסוים). כאשר הרף נמוך- מנסים לזהות את הצלעות הלא רלוונטיות (כל מה שקטן מסף מסוים).

6. סימון הצלעות בקו רציף על ידי זיהוי 'רכסים' בשינויי הבהירות בתמונה : עבור כל פיקסל בדוק אם בחלון הפיקסלים סביבו לפחות פיקסל אחד עובר את הסף הגבוה (כלומר הוא חלק מצלע חזקה).



III. בחירת מקטע רלוונטי מהתמונה:

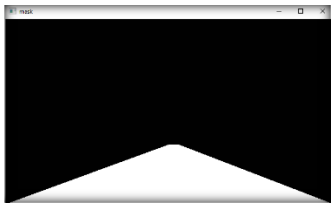
גם לאחר יישום אופרטור canny, ישנם עדיין קצוות רבים שמתגלים שאינם נתיבים. אזור העניין שלנו הוא טרפז בתחתית התמונה. ההנחה כאן שהמצלמה נשארת במקום קבוע, הנתיבים שטוחים וכמעט לרוב השליש העליון של התמונה הוא שמיים, והכביש עצמו יוצר צורת טרפז בתחתית התמונה. כך נוכל "לנחש" את אזור העניין בתמונה.
נתחשב בכך שראשית הצירים בתמונה במחשב היא הפינה השמאלית העליונה של התמונה (פיקסל מספר (0,0)), קואורדינטות השורות גדלות מלמעלה למטה (ציר ה-y) וקואורדינטות העמודות גדלות בכיוון אליו אנו רגילים משמאל-לימין (ציר ה-x).

Pseudo code:

קלט : תמונת 'צלעות' החוזרת מאופרטור canny edge

פלט : תמונה בינארית בה הפיקסלים הלבנים הם הצלעות שנמצאות במקטע המבוקש

- יצירת תמונת mask- מטריצה שמגדירה טרפז, שקודקודי הבסיס שלו נמצאים בפינות התמונה התחתונות, והצלע האופקית העליונה באזור שליש תחתון של התמונה (כלומר $\frac{2}{3} \cdot height$), הפיקסלים שבתך הטרפז מוגדרים להיות בהירות '1' (לבנים) ושאר התמונה '0' (שחורים).



תמונת הmask לדוגמא

- מבצעים פעולת and על הביטים של תמונת ה'צלעות' ותמונת הmask כדי להשאיר רק את הצלעות שממלאות את הטרפז.

IV. Hough transform- מציאת קווים:

טרנספורמציית Hough היא טכניקה למציאת קווים בתמונת 'צלעות' (תמונת קווי המתאר שיצאה לדוגמא מאלגוריתם canny). כדי להתאים קווים לנקודות שמופיעות בתמונה זו משתמשים בטכניקת **הצבעה – voting**, טכניקה כללית שבה נותנים לפיצ'רים ל'הצביע' לכל המודלים שהם יכולים להתאים להם. באמצעות טכניקה זו האלגוריתם מוצא קווים:

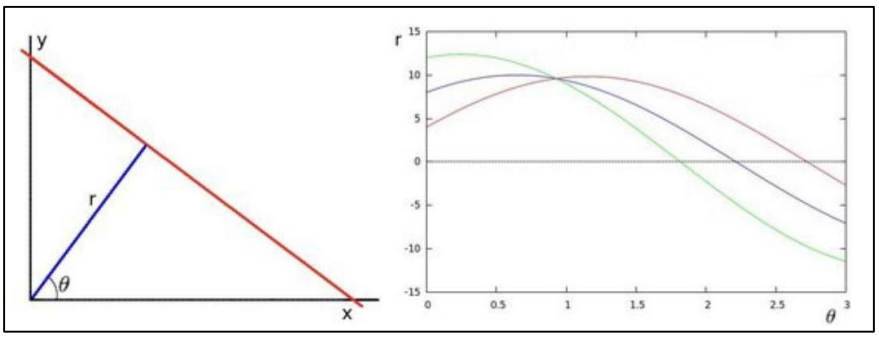
1. כל נקודה מצביעה לקו אפשרי.
2. נחפש את הקווים שקיבלו הכי הרבה הצבעות.

למה זה יעבוד?

← גם אם יהיו הרבה רעשים בתמונה שיצביעו לקווים לא נכונים- ההנחה היא שטעויות מתפרסות על פני הרבה קווים ולכן עדיין הקווים שיקבלו את רוב ה'הצבעות' יהיו הנכונים.

← גם אם יש נקודות שנפספס בגלל שנפלו בתהליך זיהוי הצלעות- זה בסדר כי נקודות אחרות שתפסנו יצביעו לקו הזה.

את 'ההצבעה' מבצעים באמצעות ייצוג של כל קו במרחב הקרטזי -כנקודה במרחב פרמטרים, וכל נקודה במרחב הקרטזי- כקו/כגל במרחב הפרמטרים (תלוי בדרך שבה בוחרים להציג את הקו- בהצגה קרטזית או בהצגה פולארית). אם מספר קווים/גלים במרחב הפרמטרים חותכים את אותה הנקודה, ניתן להניח כי כל הנקודות הללו נמצאות על אותו קו במרחב הקרטזי.



קו בתצוגה פולארית במרחב הקרטזי, וגלים המייצגים נקודות במרחב הפרמטרים כאשר נקודת החיתוך מייצגת את הקו
(התמונה נלקחה ממאמר של nachiket tanksale באתר https://medium.com/)

כדי להבין לעומק את טרנספורמציית Hough יש צורך לצלול למעט מתמטיקה, המפורטת בנספח Hough transform.

Pseudo code:

קלט: תמונת 'צלעות' לאחר חיתוך המקטע הרלוונטי בו נמצא הכביש.

פלט: מטריצה שמספר השורות בה כמספר הקווים שנמצאו ומספר העמודות בה 4 – נקודת מקסימום ונקודת מינימום עבור כל קו שנמצא.

← **פרמטרים חשובים:**

המרחק בין ראשית הצירים למיקום הקו מסומן כ ρ הנקראת בקוד $\boldsymbol{rho}$. טווח הערכים: מהערך השלילי של 'diag_len' עד לערכו החיובי, כאשר 'diag_len' מייצג את אורך האלכסון של קלט התמונה.

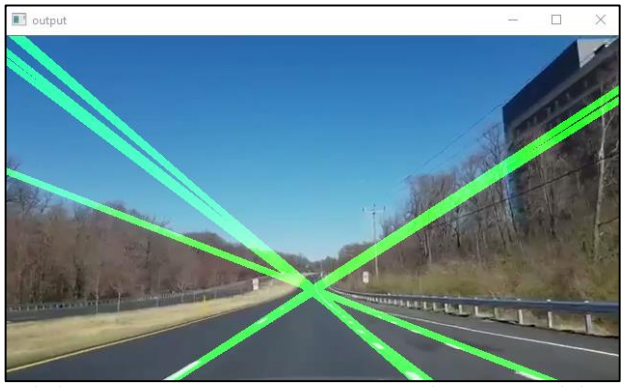
הזווית בין הצד החיובי של ציר ה x ל rho מסומן כ θ הנקראת בקוד $\boldsymbol{theta}$. טווח הערכים: $-90°$ to $90°$.

← ניצור מטריצה דו-ממדית- '**accumulator**' המייצגת את מרחב Hough (מרחב הפרמטרים), בה מספר השורות מייצגות את מספר הערכים האפשריים ל ρ ומספר העמודות מספר הערכים האפשריים ל θ.

← **ההצבעה:**
עבור כל נקודה שהיא צלע בתמונת הקלט (כל נקודה שאינה מסומנת בשחור),
ועבור כל ערך הזווית theta,
מצא את הערך של המרחק rho המתאים לנקודה ולזווית בהתאם למשוואה:
$\rho = x \cos\theta + y \sin\theta$
הגדל את הכניסה במטריצת accumulator במיקום (theta, rho) ב -1.

← **מציאת מקסימום מקומי:** מצא את המיקומים במטריצה accumulator בהם הערך נחשב מקסימלי-עובר סף מסוים.

← **נקודה חשובה:** במימוש רגיל של טרנספורמציית Hough האלגוריתם פשוט מחזיר מטריצה של פרמטרי-הקווים או של שתי נקודות אקראיות על כל קו (בדרך כלל נקודות מספיק רחוקות אחת מהשנייה כך שהקו יראה בתמונה אינסופי). במימוש שלנו הוספנו חישוב של נקודות המקסימום והמינימום של כל קו על מנת שלא ייוצרו קווים אינסופיים בסוף התהליך:



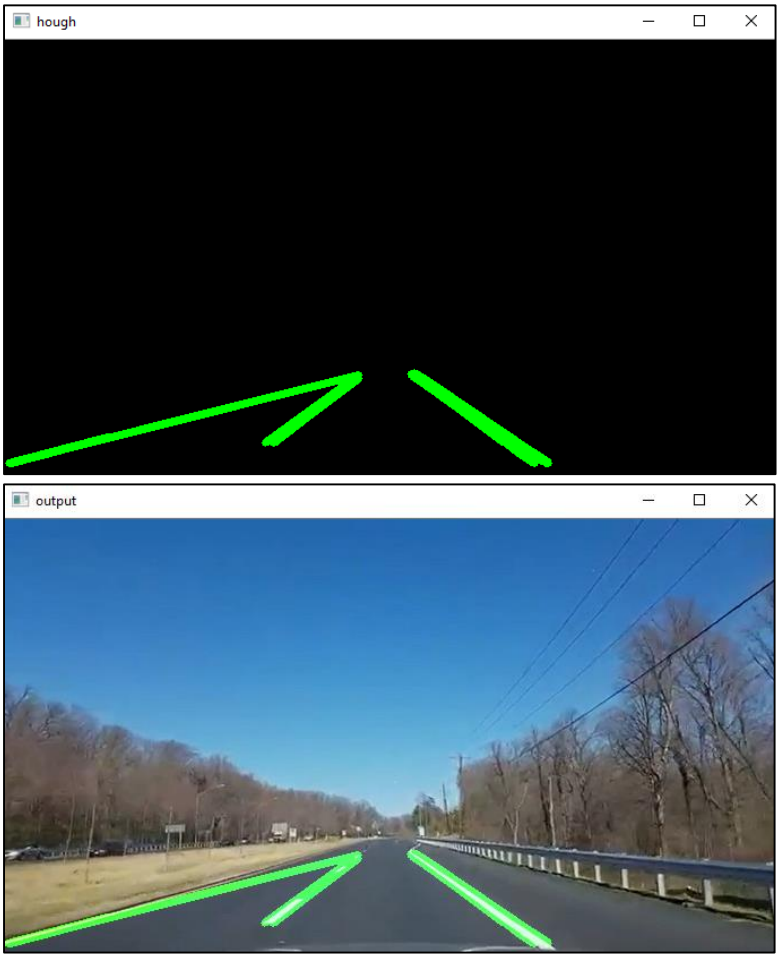
דוגמא לפלט היוצא כאשר לא שומרים את נקודות המקסימום והמינימום של כל מקטע

```
r = rhos[i]
theta = thetas[j]
a = np.cos(theta)
b = np.sin(theta)
x0 = a * r
y0 = b * r
x1 = int(x0 + 1000 * (-b))
y1 = int(y0 + 1000 * (a))
x2 = int(x0 - 1000 * (-b))
y2 = int(y0 - 1000 * (a))
```

קוד השומר שתי נקודות אקראיות, מקור: https://www.geeksforgeeks.org/line-detection-python-opencv-houghline-method/

← **על מנת לשמור עבור כל קו את מינימום ומקסימום:**
ניצור מטריצה בגודל הaccumulator עבור ערכי ה-x המינימליים, ערכי ה-y המינימליים, ערכי ה-x המקסימליים וערכי ה-y המקסימליים.
במהלך המעבר על כל edge point נשמור עבור כל תא המקבל 'הצבעה' את ערך הנקודה המצביעה אליו רק אם היא קטנה יותר מערך המינימום השמור עבור תא זה או גדולה מערך המקסימום השמור עבור תא זה.

← האלגוריתם מחזיר את הנקודות מינימום ומקסימום עבור כל התאים ב הaccumulator שעברו את הסף שקבענו.



הדגמה של שרטוט כל הקווים שנמצאו בטרנספורמציית Hough לפי מערך הנקודות לכל קו

V. <u>איחוד ואינטרפולציה של הקווים:</u>

על מנת למצוא סימוני נתיבים ברורים וליצור תמונת כביש ברורה, (שכן אלגוריתם Hough יכול למצוא מספר קווים צמודים זה לזה כפי שניתן לראות בתמונה האחרונה), הדבר הראשון שעלינו לעשות זה להבדיל בין קו המתאר נתיב ימני לקו המתאר נתיב שמאלי.

יש דרך קלה לזהות את הנתיבים מכל צד:

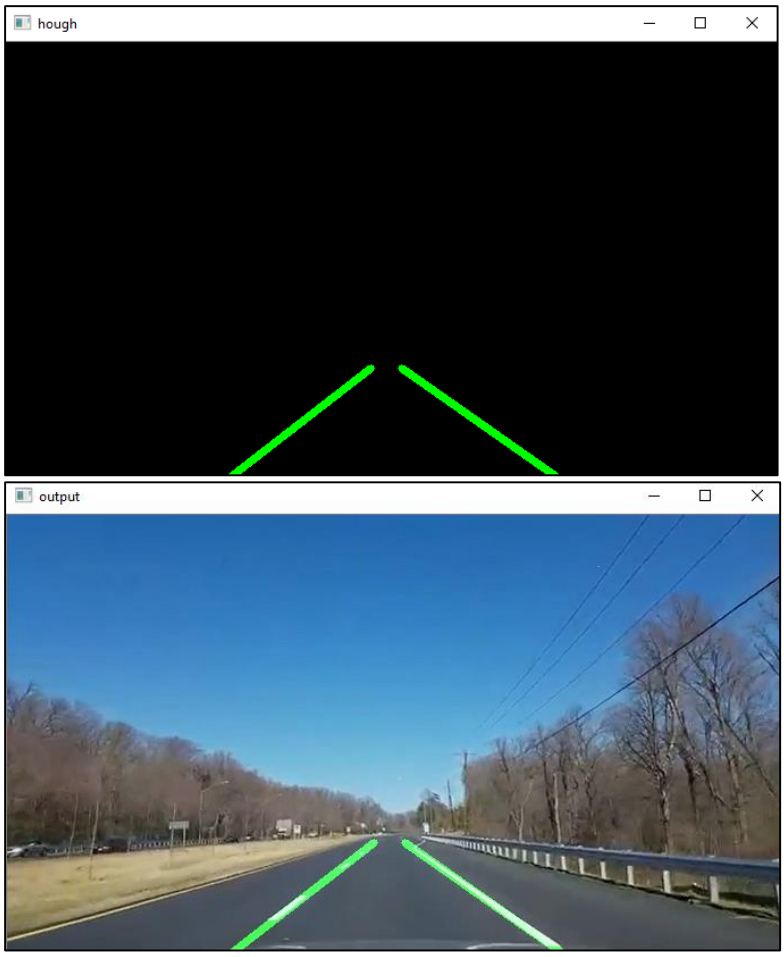
עבור הנתיב השמאלי- השיפוע של הקו יהיה שלילי (ככל שערכי העמודות גדלים, ערכי הקואורדינטות של השורות יורדים. נזכיר שוב שערכי הקואורדינטות של ציר הy הפוכים- הפיקסלים העליונים ביותר בתמונה הם בעלי הערכים הנמוכים ביותר).

עבור הנתיב הימני- השיפוע של הקו יהיה חיובי (ככל שערכי העמודות גדלים, ערכי השורות עולים).

נתעלם מכל הקווים האופקיים- על ידי התעלמות מכל הקווים שהשיפוע שלהם קטן מ0.5 וגדול מ0.5-.

לאחר שהבדלנו בין הקווים השייכים לנתיב השמאלי ולקווים השייכים לנתיב הימני נאמוד את ממוצע הקווים שיתאר קו אחד יחד לכל נתיב.

Results/ תוצאות



לאחר ביצוע אינטרפולציה על הקווים