עבודה בית 2
ראייה ממוחשבת

מרצה: סימון קורמאן

המגישים:

1) **שם: באסל סגיר**
ת.ז: 316543909

2) **שם: אסיה חטיב**
ת.ז: 206217028

תאריך ההגשה: 07/06/2021

שאלות:

1) Epipolar Geometry :

בשאלה זו ממשנו את האלגוריתם כמו השלבים שמוצעים בתיאור של עבודה הבית :
תיאור קצר על המבנה של הקוד .. לקוד יש UI קטן שהוא פונקציה שמקבלת 2 שמות של תמונות שחיבות להיות באותו PATH של המודל והיא מריצה את הקוד לפי התמונות מכיוון שעלינו לבחור נקודות באופן ידני אז הנקודות ממשות HARDCODED ועבור כל זוג נתון כקלט היא לוקחת את הנקודות המתאמות לזוג ואחרת היא מחזירה שאין תמונה שמורה בשם אחר.

ופונקציה ראשית Q 1 שהיא מחשבת את השלבים הבאים שנסביר בהמשך ההסבר של השאלה .

א. לכל זוג תמונות, סמנו 8 או יותר זוגות של נקודות מתאימות. דאגו לכם שהן מפוזרות היטב במרחב התמונה וכי הן לא יושבות על מישור יחיד בסצנה.

לזוג התמונות הראשונות סמנו את הנקודות הבאות :

```
if image_name1 == "im_courtroom_00086_left.jpg" and image_name2 == "im_courtroom_00089_right.jpg":
    pts_left = np.array(
        [[207, 40], [280, 130], [342, 228], [406, 438], [32, 308], [617, 387],
         [696, 474], [838, 212], [725, 62], [427, 46]]).astype(np.float32)
    pts_right = np.array(
        [[265, 17], [313, 90], [360, 171], [537, 282], [287, 224], [626, 255],
         [661, 304], [732, 141], [653, 36], [405, 13]]).astype(np.float32)
    number_of_points = 10
```

בחרתי את הנקודות מפוזרות במרחב ויושבות על פיקסלים שקיים תנועה שרואים אותה בעיין גם כל כך ברורה בסה"כ 10 נקודות.

לזוג התמונות השני בחרתי 8 נקודות מפוזרות באותה צורה של בחירה לזוג קודם הנקודות :

```
elif image_name1 == "im_family_00084_left.jpg" and image_name2 == "im_family_00100_right.jpg":
    pts_left = np.array(
        [[857, 210], [869, 314], [234, 328], [321, 288], [124, 51], [57, 253],
         [470, 261], [218, 195]]).astype(np.float32)
    pts_right = np.array(
        [[806, 211], [862, 417], [126, 332], [324, 294], [579, 32], [534, 235],
         [736, 270], [751, 185]]).astype(np.float32)
    number_of_points = 8
else:
    print("There is no saved image with this name, ERROR!")
    return
```

ב. חשבו את ה matrix fundamental בעזרת 2 שיטות estimation מתוך האפשרויות הבאות ניתן להשתמש במימושים קיימים.

בחרתי בשתי שיטות ראשונות

- 7-point algorithm
- 8-point algorithm

קראתי להם עם אותה מתודה אבל עם פרמטר שקוראים לו METHOD שהוא 1 עבור השיטה הראשונה ו 2 עבור השנייה.

```
Q1(im1, im2, pts_left, pts_right, 1)
cv.destroyAllWindows()
print("#######")
Q1(im1, im2, pts_left, pts_right, 2)
cv.destroyAllWindows()
```

והשתמשתי בקוד קיים שממומש בספריה CV שהוא מחזיר על ידי מתודה למצוא את המטריצה הפונדמנטלית והשיטה מועברת כפרמטר :

```
if method == 1:
    print("7 Point Algorithm: ")
    F1, mask1 = cv.findFundamentalMat(pts_left, pts_right, cv.FM_7POINT)
if method == 2:
    print("8 Point Algorithm: ")
    F1, mask1 = cv.findFundamentalMat(pts_left, pts_right, cv.FM_8POINT)
```

ג. לכל אחת מהשיטות, הציגו את זוגות הנקודות ואת הישרים האפיפולריים המתאימים לכל נקודה בתמודה שמנגד. השתמשו בקוד משלכם להפעלת ה)matrix fundamental ע"י כפל מטריצה .)ניתן להשתמש בקוד קיים לציור הקווים האפיפולריים שקיבלתם)במימוש שלכם יתכן ותצטרכו לחשב את קצוות הקווים שהם נקודות החיתוך עם מסגרת התמונה(. ראו דוגמא מצורפת.

```
for point1 in pts_right:
    np_temp = np.array(point1)
    vec = np.append(np_temp, 1)
    pts2.append(vec)
    value = (F1.T).dot(vec)
    lines1.append(value)

for point2 in pts_left:
    vec = np.array(point2)
    vec = np.append(vec, 1)
    pts1.append(vec)
    value = (F1).dot(vec)
    lines2.append(value)
```

הקוד שכתבנו לחשב את הישרים .

ד. לכל אחת מהשיטות, חשבו את מדדי השגיאה הבאים)ע"י קוד שלכם(, כממוצע על זוגות ההתאמות שיצרתם:

ממשנו שתי מתודות עזר שמחשבות את המרחקים המבוקשים בתרגיל על ידי המשוואות שלהם :

- מרחק אלגבראי: $x'^T F x$

```
def Algebraic_Distance(pts1, pts2, F1):
    algebraic_distance = 0
    for i in range(len(pts1)):
        algebraic_distance += (pts2[i].T.dot(F1)).dot(pts1[i])
    algebraic_distance = algebraic_distance / (len(pts1))
    return algebraic_distance
```

- מרחק אפיפולרי (סימטרי): $d(x', Fx)^2 + d(x, F^T x')^2$, כאשר אם נסמן ב $l = (a, b, c)^T$ את הישר Fx, אז $d(x', Fx) = x'^T F x / \sqrt{a^2 + b^2}$ (שימו לב שאתם מבינים למה זהו המרחק האוקלידי בין כל נקודה והישר המתאים, שאמור להיות אפס בתנאים מושלמים).

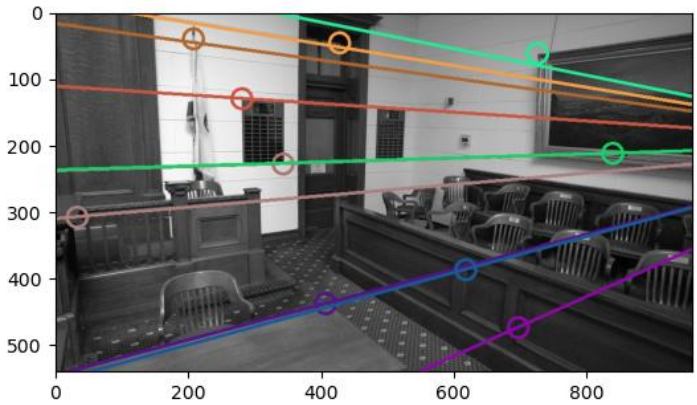
```
def Epipolar_Distance(pts1, pts2, lines1, lines2, F1):
    epipolar_distance = 0
    for i in range(len(pts1)):
        x1 = (pts2[i].T.dot(F1)).dot(pts1[i]) / (math.sqrt(lines1[i][0] ** 2 + lines1[i][1] ** 2))
        x2 = (pts1[i].T.dot(F1.T)).dot(pts2[i]) / (math.sqrt(lines2[i][0] ** 2 + lines2[i][1] ** 2))
        epipolar_distance += ((x1 ** 2) + (x2 ** 2))
    return epipolar_distance
```

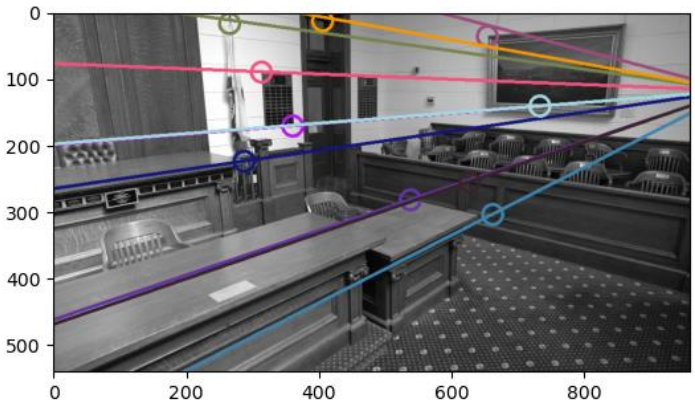
ואת המתודה שבוחרת צבע באופן רנדומאלי שצובעת את הכו ואת הנקודה בעיגול.

```python
def drawlines(img1, img2, lines, pts1, pts2):
    """ ... """
    r, c = img1.shape
    img1 = cv.cvtColor(img1, cv.COLOR_GRAY2BGR)
    img2 = cv.cvtColor(img2, cv.COLOR_GRAY2BGR)
    for r, pt1, pt2 in zip(lines, pts1, pts2):
        random_number1 = random.randint(0, 255)
        random_number2 = random.randint(0, 255)
        random_number3 = random.randint(0, 255)
        color = (random_number1, random_number2, random_number3)
        x0, y0 = map(int, [0, - r[2] / r[1]])
        x1, y1 = map(int, [c, - (r[2] + r[0] * c) / r[1]])
        img1 = cv.line(img1, (x0, y0), (x1, y1), color, 4)
        img1 = cv.circle(img1, tuple(pt1), 15, color, 3)
        img2 = cv.circle(img2, tuple(pt2), 15, color, 3)
    return img1, img2
```

תוצאות החלק הזה :

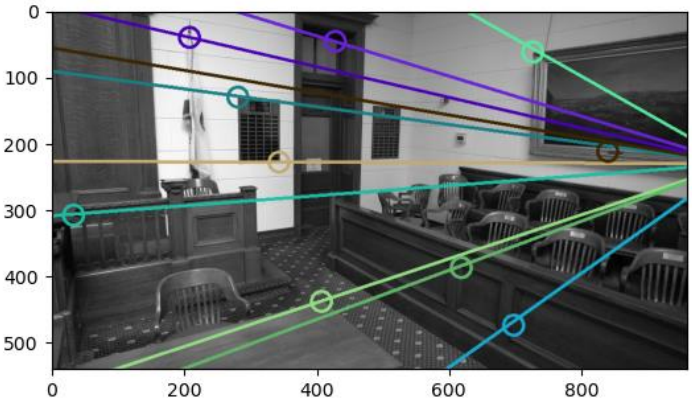
הזוג הראשון עם שיטה 7 POINT ALGORITHM

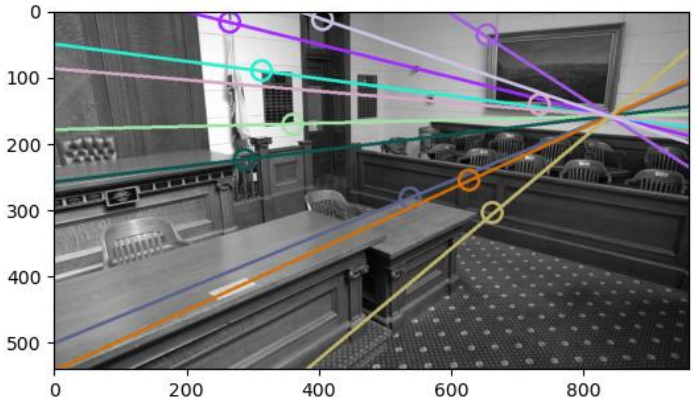





```
7 Point Algorithm:
The Algebraic Distance is : 0.04367785254269496
The (Symmetric) Epipolar Distance is : 70.85045847338378
```

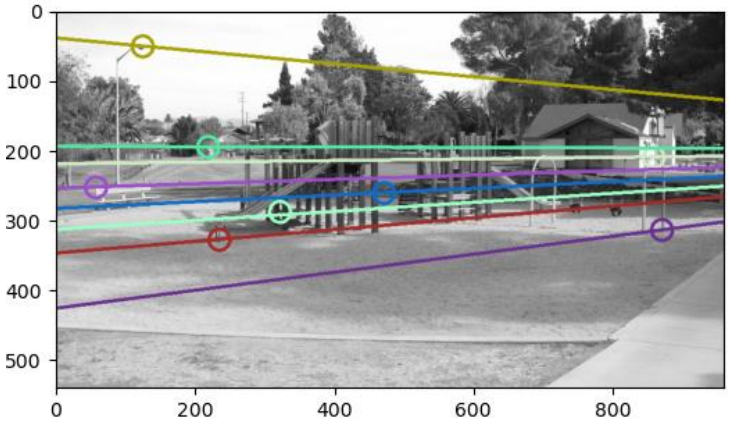
הזוג הראשון עם שיטה 8 POINT ALGORITHM

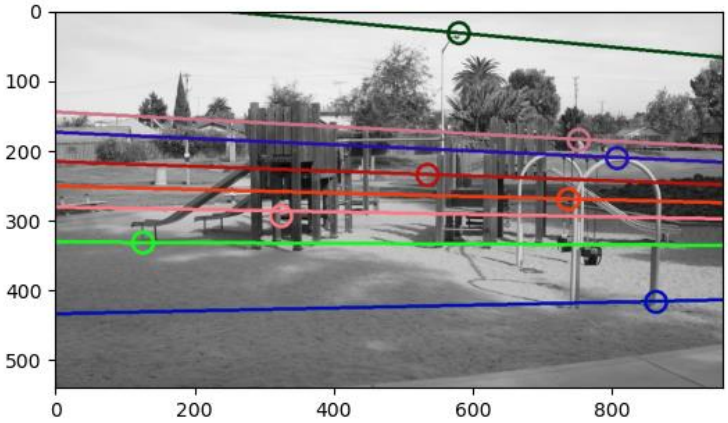





```
8 Point Algorithm:
The Algebraic Distance is : -0.014349296253851434
The (Symmetric) Epipolar Distance is : 34.07849187063302
```

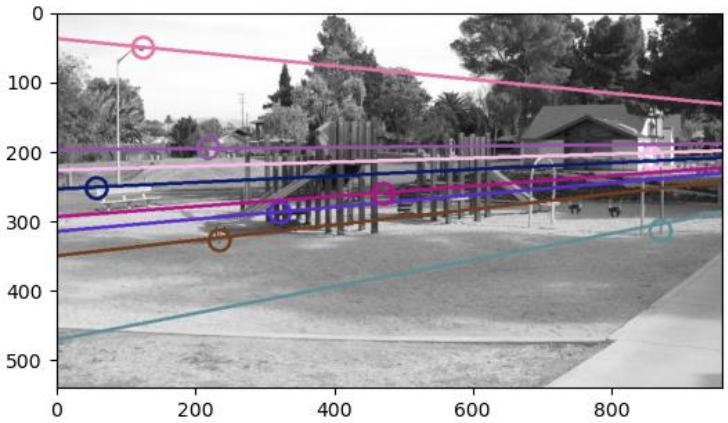
הזוג השני עם שיטה 7 POINT ALGORITHM

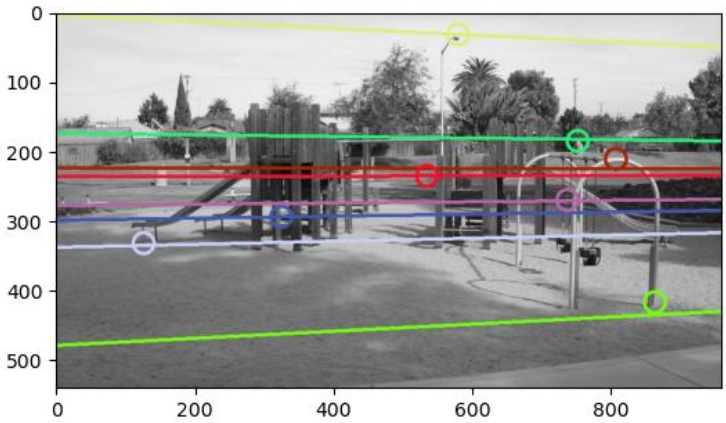





```
7 Point Algorithm:
The Algebraic Distance is : 0.010266458136210785
The (Symmetric) Epipolar Distance is : 9.3489185134044
#######
```

הזוג השני עם שיטה 8 POINT ALGORITHM






```
8 Point Algorithm:
The Algebraic Distance is : -0.06244053234262205
The (Symmetric) Epipolar Distance is : 91.08017568480047
```

להרצה ולקבל תוצאות אפשר להריץ את המודל q1_python אבל חייב ש 4 התמונות יהיו באותו מיקום של המודל כלומר נמצאות ב PATH של המודל עצמו.

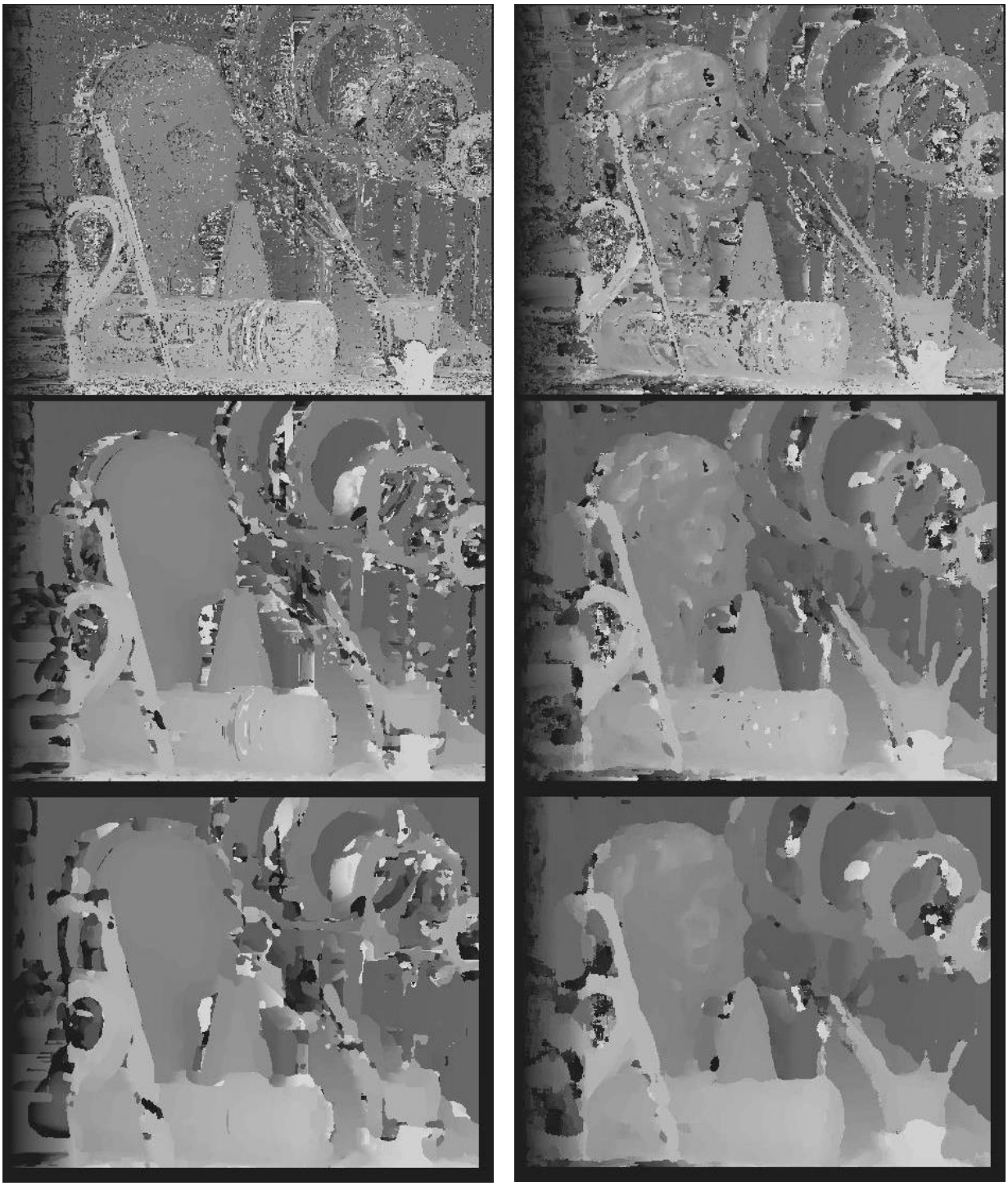
2) Photometric Stereo :

בשאלה זו חישבנו את מפת העומקים (disparities) המתאימה לתמונה השמאלית של rectified stereo pair בעזרת שני אלגוריתמים SSD ו NCC.

<u>תוצאות :</u>

נתחיל עם **Art Pair** –

SSD **NCC**



Art_Pair, Algorithm = SSD, window=15, time=2.5802 minutes, AvgErr=13.2482, medErr=2.0000, Bad05=74.4837% Bad4=45.7358%	Art_Pair, Algorithm = SSD, window=9, time=2.5434 minutes, AvgErr=11.5504, medErr=1.6667, Bad05=69.9971% Bad4=41.8849%	Art_Pair, Algorithm = SSD, window=3, time=2.8043 minutes AvgErr=11.7961 medErr=3.3333 Bad05=74.3693 % Bad4=46.8922%

Art_Pair, Algorithm = NCC, window =15, time=7.4062 minutes, AvgErr=15.1664, medErr=4.3333, Bad05=75.5286% Bad4=50.1051 %	Art_Pair, Algorithm = NCC, window =9, time=6.8114 minutes, AvgErr=12.4909, medErr=1.3333, Bad05=65.2531% Bad4=42.4575 %	Art_Pair, Algorithm = NCC, window=3, time=6.9436 minutes, AvgErr=11.7897, medErr=1.3333, Bad05=62.5025% Bad4=44.1375 %

מבחינת איכות התמונות בזוג Art התמונות של SSD בשלושת החלונות יותר טובות כי יש בהן פחות רעש ועומק כל פריט נראה יותר נקי ולכן העין רואה העומקים בצורה יותר טובה .

מבחינת מדדי השגיאה :

עבור חלון 3 , השגיאה AvgErr קרובה מאוד ב SSD ו NCC אבל שאר הmeasurements שהם medErr, Bad05, Bad4 יותר טובות ב NCC אבל NCC לוקח יותר זמן .

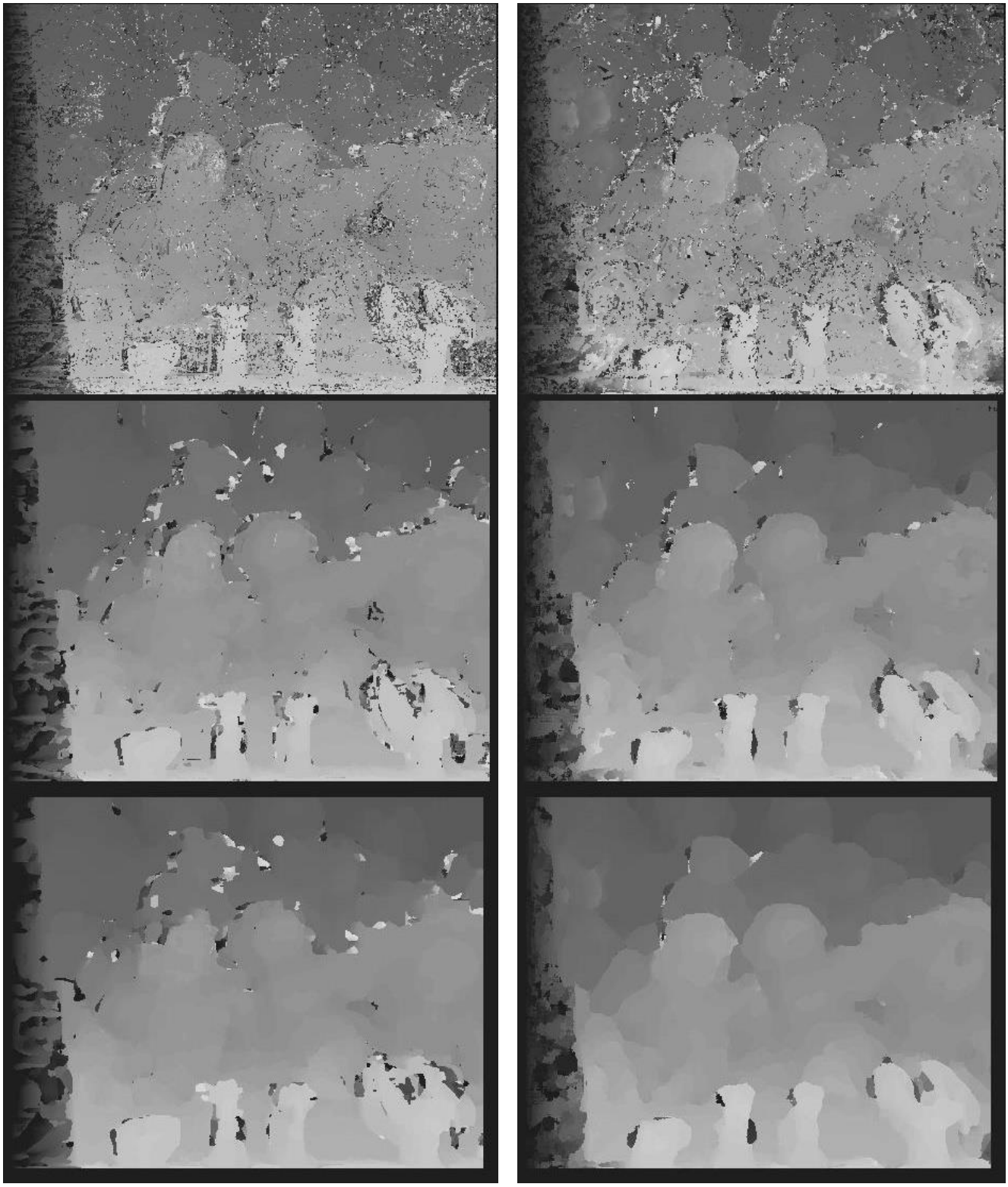
בחלונות 9 ו 15 השגיאות קרובות אבל NCC לוקח יותר זמן , כמו כן , בחלונות 9 ו 15 התמונות קרובות יותר ל ground_truth

בשיטת SSD ככל שהחלון גדול יותר התמונה נמרחת יותר ורואים העומקים בצורה יותר טובה . כנ"ל עבור NCC

שיטת NCC לקחה יותר זמן בכל הזוגות ובכל החלונות .

Dolls Pair

NCC **SSD**



Dolls_Pair, Algorithm = SSD, window=15, time=2.5580 minutes, AvgErr=9.6753, medErr=1.0000, Bad05=71.5989% Bad4=30.6801 %	Dolls_Pair, Algorithm = SSD, window=9, time=2.5144 minutes, AvgErr=7.8152, medErr=0.6667, Bad05=62.8527% Bad4=24.4849 %	Dolls_Pair, Algorithm = SSD, window=3, time=2.6026 minutes, AvgErr=8.5113, medErr=1.0000, Bad05=62.3040% Bad4=31.9053 %

Dolls_Pair, Algorithm = NCC, window =15, time=8.1893 minutes, AvgErr=10.3308, medErr=1.3333, Bad05=71.6835% Bad4=33.0074 %	Dolls_Pair, Algorithm = NCC, window =9, time=6.7796 minutes, AvgErr=8.2538, medErr=0.6667, Bad05=60.0321% Bad4=25.6722 %	Dolls_Pair, Algorithm = NCC, window =3, time=6.8144 minutes, AvgErr=8.3760, medErr=0.6667, Bad05=51.5621% Bad4=29.7309 %

בזוג Dolls_Pair מבחינת איכות , בחלון 3 שתי המפות קרובות בשתי השיטות SSD ו NCC . שתי המפות מכילות הרבה רעש אבל אם בוהים יותר בשתי המפות רואים שמפת SSD יותר טובה .

וככל שהחלון גדול יותר המפות נראות יותר טובות .

בחלונות 9, 15 מפת SSD יותר טובה מבחינת איכות ומבחינת זמן מאשר חלון 3.

מבחינת מדדי השגיאה חלון 9 , SSD השגיאות AvgErr, medErr, Bad4 יותר קטנות מאשר חלונות 3 ו 15 באותה שיטה (SSD) למעט השגיאה Bad05 אבל היא לא גדולה בהרבה .

חלון 9, שיטת NCC ארבע השגיאות יותר קטנות משאר החלונות .

כל השגיאות בשתי השיטות בכל החלונות מאוד קרובות ולא רואים הבדל משמעותי בין המפות בשתי השיטות עבור החלון המתאים .

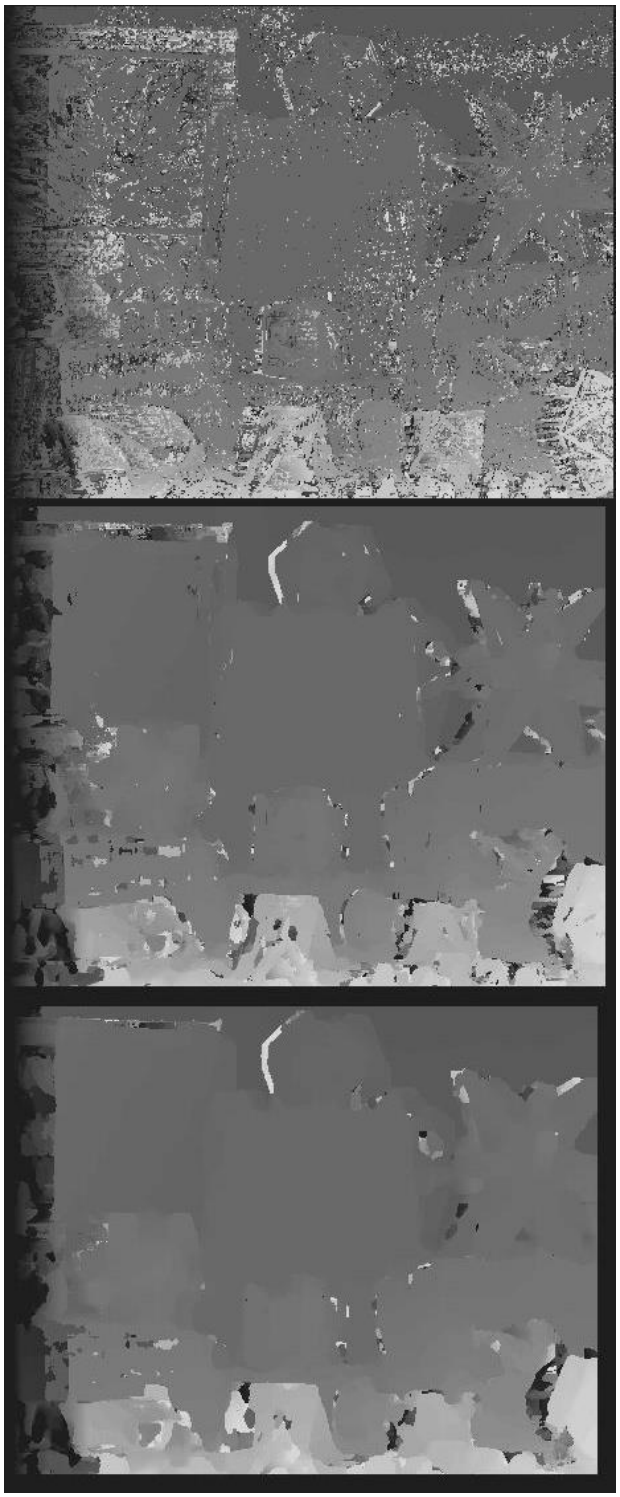
NCC לוקח יותר זמן .

הזמן של SSD בחלונות 9 ו 15 קרוב אבל יותר גדול בחלון 3 .

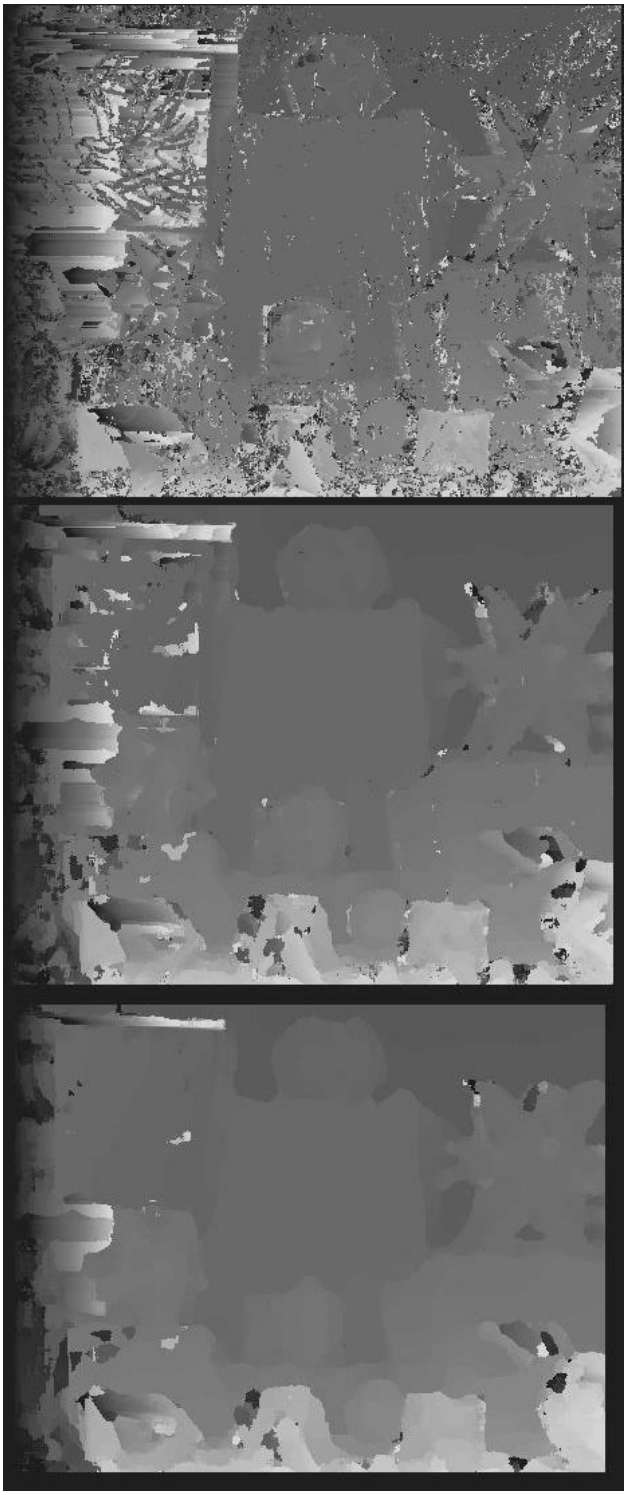
ב NCC לחלון 15 לקח זמן יותר מחלון 3 ו 9 .

Moebius Pair

NCC



SSD



Moebius_Pair, Algorithm = SSD, window=15, time=2.5174 minutes, AvgErr=9.1550, medErr=1.0000, Bad05=60.3683% Bad4=34.4113 %	Moebius_Pair, Algorithm = SSD, window=9, time=2.4830 minutes, AvgErr=8.0391, medErr=0.6667, Bad05=54.5543% Bad4=31.1996 %	Moebius_Pair, Algorithm = SSD, window=3, time=2.5443 minutes, AvgErr=9.4719, medErr=1.3333, Bad05=60.9643% Bad4=40.4436 %

Moebius_Pair, Algorithm = NCC, window=15, time=7.8247 minutes, AvgErr=9.2238, medErr=1.0000, Bad05=61.3117% Bad4=34.1089 %	Moebius_Pair, Algorithm = NCC, window=9, time=6.6867 minutes, AvgErr=7.5245, medErr=0.6667, Bad05=51.3875% Bad4=27.9260 %	Moebius_Pair, Algorithm = NCC, window=3, time=6.6925 minutes, AvgErr=8.6157, medErr=0.6667, Bad05=53.2969% Bad4=36.5630 %

בזוג Moebius Pair איכות המפות בSSD נראות יותר טובות וחלקות יותר למרות ששיטת SSD לוקחת פחות זמן מ NCC .

מבחינת מדדי השגיאה :

בשיטת SSD ארבע השגיאות יותר טובות בחלון 9 מאשר חלון 3 ו 15 וכנ"ל עבור חלון 9 בשיטת NCC .

<u>סיכום :</u> המפה המועדפת היא מפה בעלת חלון 9 בשיטת SSD .

בחלון 15 בNCC ו SSD השגיאות יותר גדולות כי הגדלנו חלון ההסתכלות שלנו .

3) New View Synthesis :

יצירת view synthesis של תמונת alley_2 לקח זמן 28.6 דקות .

יצירת view synthesis של תמונת ambush_6 לקח זמן 31.73 דקות .

```
alley_2 time=28.6402 minutes,
ambush_6 time=31.7314 minutes,

Process finished with exit code 0
```

2) בסדרת התמונות שיצרנו שמנו לב שההזזה נותן לצופה מידע על המבנה התלת ממדי של הסצנה .

<u>הסבר:</u> רואים/מרגישים שהנקודות הקרובות בתמונה בעלות עומק קטן יותר מוזזות יותר משאר הנקודות הרחוקות בעלות עומק גדול יותר וכך נוצר depth כאשר רואים סדרת התמונות ברצף .

3) א)חורים מסוג קווים דקים כמו רשת נוצרים בגלל חוסר מידע.

חורים משמעותיים יותר נוצרים בגלל forward mapping , אובייקטים שהם רחוקים יותר מאובייקטים אחרים ימופו לאותו פיקסל ומופיע הפיקסל הקרוב יותר ומאחריו הפיקסל הרחוק יותר .

ב) החורים המשמעותיים הם מסוג אינהרנטי ,

החורים הנוצרים מ forward warp mapping אפשר לפתור אותם דרך לעשות backward warp mapping .

הסבר לסעיף א :

חשבו reprojection של כל קואורדינטות התמונה (פיקסלים) אל המרחב התלת-ממדי.

השתמשתי בנוסחא הזאת :

$$u = \frac{1}{s_x} f \frac{X}{Z} + o_x \qquad v = \frac{1}{s_y} f \frac{Y}{Z} + o_y$$

כאשר $s_x = 1, o_x = u_0,\ o_y = v_0$

מהמשוואה בצד ימין נקבל :

$$y = \left(\frac{v - v_0}{f_y}\right) * z$$

מהמשוואה בצד שמאל נקבל :

$$x = \left(\frac{u - u_0}{f_x}\right) * z$$

מימוש :

```
def back_projection(k, depth):
    fx, fy, u0, v0 = k[0, 0], k[1, 1], k[0, 2], k[1, 2]
    height = depth.shape[0]
    width = depth.shape[1]
    cam_points = np.zeros((height * width, 3))
    new_cam_points = np.zeros((height * width, 4))
    i = 0
    # Loop through each pixel in the image
    for v in range(height):
        for u in range(width):
            x = (u - u0) * depth[v, u] / fx
            y = (v - v0) * depth[v, u] / fy
            z = depth[v, u]
            cam_points[i] = [x, y, z]
            new_cam_points[i] = [x, y, z, 1]

            i += 1
    return cam_points, new_cam_points  # 2d to 3d
```

הסבר לסעיף ב :

בסעיף זה אנחנו צריכים לחזור ממרחב 3D למרחב 2D.

נכפיל שלושת המטריצות - מטריצת K שהיא מטריצת ה intrinsic במטריצת $[I|0]$ כפול המטריצה שמכילה הקואורדינטות x,y,z ועוד ממד של אחדים .

אחרי הכפלת שלושת המטריצות נקבל מטריצה שאחד הממדים שלה הוא 3 והוא מכיל קואורדינטות x,y,z . נחלק x ב z ונחלק y ב z ונמפה בחזרה ערך התמונה RGB למיקום x,y שקיבלנו .

מימוש :

```
def back_to_2d(intrinsic, depth, image, extrinsic):
    """ P = K[I|0]"""
    'x = M*N*X'
    Im = np.zeros(image.shape)
    height = image.shape[0]
    width = image.shape[1]
    cam_points, new_cam_points = back_projection(intrinsic, depth)
    I_0 = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0]])
    i = 0
    for v in range(height):
        for u in range(width):
            # coor_2d = intrinsic.dot(I_0.dot(new_cam_points[i]))
            coor_2d = intrinsic.dot(extrinsic.dot(new_cam_points[i]))
            z = (coor_2d[2])
            x = int((coor_2d[0]) / z)
            y = int((coor_2d[1]) / z)

            if 0 <= y < height and 0 <= x < width:
                Im[y, x] = image[v, u]
            i += 1
    return Im
```