

مقرر أنظمة رؤية الآلة

الوظيفة 3

Stereo Vision

إعداد :

يزن معلا

إشراف :

د.م. إياد حاتم

2020 April 20

1 المقدمة:

الكاميرا تعد من أهم مصادر الإدخال في مختلف أنظمة التحكم والمعلومات، والصور هي عبارة عن الشكل الأولى للمعلومات التي تقدمها الكاميرا بمختلف أنواعها. والصور بوصفها إسقاطاً ثنائياً بعد مشاهد حقيقية فإنها تفتقر إلى تقديم معلومات عن العمق (البعد الناظمي عن مستوى الصورة) والذي قد يكون معلومة مهمة في بعض التطبيقات. لذلك طُبقت مجموعة من الخوارزميات والآليات للحصول على معلومات عن البعد (أي الحصول على صور ثلاثة بعد) وليس المقصود بالصور ثلاثة بعد أن تكون صوراً مجسمة إسقاطياً، وإنما القدرة على تحديد الإحداثيات الديكارتية على المحاور الثلاث لأي نقطة من نقاط المشهد المدروس.

الفكرة الأساسية في تحصيل المعلومات ثلاثة بعد هي الاعتماد على صورتين ثانية بعد لنفس المشهد، وذلك ممكن بعدة طرق:

«تحريك الكاميرا والتقط صورتين من منظوريين مختلفين لنفس المشهد».

«التقط صور لجسم متحرك من كاميرا ثابتة».

«اعتماد كاميرتين لالتقط نفس المشهد وهو مبدأ Stereo Vision».

2 الملاخص :

الرؤية المجسمة Stereo Vision وتُشخص بوصفها عملية تجميع معلومات من صور ثنائية بعد لتكون تصوّر فراغي للمشهد المتقطط. وتنقسم هذه العملية إلى عدة مراحل هي:

1- استخلاص المعلم المراد اعتمادها في عملية الربط بين الصور Features Extract . واعتمدت خوارزمية SIFT لكشف المعلم المتمثّلة بالزوايا.

2- مطابقة المعلم Matching ، وتحديد النقاط المترابطة بين الصور المختلفة، وذلك عبر خوارزمية الجiran الأقرب KNN

3- حساب المصفوفة الأساسية Fundamental matrix والتي تعبر عن مصفوفة التحويل بين النقاط المقابلة في الصورتين.

4- تحديد موضع الكاميرتين النسيي اعتماداً على المصفوفة الأساسية.

5- التثليث، أي الحصول على الإحداثيات ثلاثة بعد.

6- تقويم الصور لجعل المعلم واقعة على خطوط أفقية متوازية بهدف تسهيل عملية البحث والمطابقة.

7- حساب العمق Compute Depth وذلك انطلاقاً من حساب التفاوت Disparity بين النقاط المقابلة، وكتنبوتية يتم الحصول على خريطة التفاوت والتي تعبر عن عمق العناصر الموجودة في الصورة.

3 استخلاص المعلم : Feature Extraction

استُخدمَت خوارزمية SIFT لأجل تحديد النقاط المهمة المراد اعتمادها في عملية المقارنة والمطابقة، إذ أنه من المكلف برمجياً وزمنياً إجراء عملية المطابقة على كامل عناصر الصورة، بل يجب تحديد عدد من النقاط الحاوية على أكبر قدر من المعلومات الواقعة للصورة.

خوارزمية طورها البروفيسور الكندي David Lowe في العام 1999 ومن أهم ميزات هذه الطريقة عن غيرها من خوارزميات الكشف أنها لا تتأثر باختلاف التباين والإضاءة، بالإضافة إلى أهم خاصية وهي عدم التأثر بحجم المعلم المراد كشفها، وال نقاط المهمة المراد كشفها في الصور هي الزوايا.
وتتألف هذه الخوارزمية كا وصفها Lowe في بحثه من مجموعة مراحل هي باختصار:

1- توليد عدة نسخ لنفس الصورة مع تصغير الحجم Downsampling ، وتطبيق مرشح الكشف على كل نسخة والذي هو مرشح الفرق الغاوسي DoG بقيم مختلفة للتباين σ والناتج هو هرم من نتائج المرشح DoG على عدة نسخ متباينة الحجم من الصورة.

2- النقاط المميزة هي النقاط ذات القيم الحدية محلياً Extrema ، تدرس كل نقطة في جوار ثلاثة المستويات، وتحدد النقاط المهمة مبدئياً.

3- باعتبار أن السوية اللونية تتأثر بتغيرات الإضاءة والتباين ، فالتركيز سيكون على اتجاه النقاط Orientation وليس الطويلة، ومن أجل تحديد الأتجاه الرئيسي لكل نقطة هامة (إذ قد توجد بعض الاتجاهات الثانوية الناتجة عن أحد أشكال الضجيج) يُشكل توزع إحصائي (هستوغرام) لقيمة الاتجاه مثلاً بقيم الطويلة المقابلة لكل اتجاه، وذلك في كل نقطة هامة ضمن الصورة الكلية.

4- تشكيل واصف Descriptor لكل نقطة هامة، وهو شعاع مؤلف من 128 قيمة تصف قيم الاتجاه لجوار حول كل نقطة مهمة بحجم 16×16 [1] .
مكتبة OpenCV توفر جاهزة تجري عملية الكشف وفق : SIFT

```
1 >>> sift = cv2.xfeatures2d.SIFT_create()  
2 >>> Key_points , descriptor = sift.detectAndCompute(image , None)
```

التعليمية الأولى تنشئ بنية خوارزمية SIFT ، والتعليمية الثانية تأخذ بارامتر دخل هو الصورة الرمادية.
والخرج:

Key_points هي إحداثيات النقاط المميزة المكتشفة
descriptor شعاع الواصف بحجم $n \times 128$ حيث توصف كل نقطة من خلال 128 قيمة في الواصف.
ويجب كشف المعلم في كلا الصورتين. [6]

4 مطابقة النتائج : Matching

بعد تحديد المعلم المميزة في كل صورة، يجب مطابقة هذه المعلم لتحديد العناصر المشتركة بين الصورتين. استُخدمت خوارزمية الجيران الأقرب K-nearest Neighbours (KNN) من أجل مطابقة النقاط المعبرة عن المعلم في الصورتين مع بعضها البعض.

تعد من أبسط الخوارزميات المستخدمة في التعلم غير المشرف عليه Unsupervised ، الفكرة العامة هي أن تُدرب الخوارزمية على مجموعة أولى من العينات، ومن ثم تطبيقها على مجموعة ثانية من العينات، و تقوم بحساب المسافة الإقليدية بين بين كل عنصر من المجموعة الثانية مع جميع عناصر المجموعة الأولى ومن ثم تحديد البعد الأقل والذي يقابل العنصر الأقرب. [2]

وبذلك يربط كل عنصر من المجموعة الثانية بعنصر واحد فقط من المجموعة الأولى، ولكن قد يرتبط عنصر من المجموعة الأولى بعدة عناصر من المجموعة الثانية. وهذا يعني خلاً في المطابقة والتعرف، إذ أنه يجب أن يوجد لكل نقطة مقابل واحد فقط، وحل هذه المشكلة نعيد تطبيق نفس الخوارزمية ولكن مع التبديل بين عناصر التدريب والتطبيق. بذلك نحصل على مجموعة جديدة من التوافقات، يجب إجراء مطابقة بين ناتجي عملية التوافق السابقتين و اختيار فقط النقاط المتماثلة في الاتجاهين، بحيث نحصل على ثنائيات من النقاط؛ كل نقطة من المجموعة الأولى تقابل نقطة وحيدة من المجموعة الثانية وبالعكس، وهذا ما يسمى باختبار الاتساق الثنائي Consistency . Bidirectional .

بعد ذلك تطبق تصفية للتطابقات من خلال اختبار النسبة Ratio Test ، وذلك بتحديد عتبة مؤوية معينة لفلترة نقاط التوافق التي من المحتمل أن تكون أكثر فائدة من غيرها في وصف المعلم. ناتج عملية المطابقة عبر KNN يعيد بارامتر المسافة، والذي يعبر عن أقرب توافق للقيمة المعطاة بالإضافة إلى ثاني أقرب قيمة. فإن كانت المسافتان متقاربتين فهذا يعني أن النقطة قد يكون لها أكثر من شيء موافق، وليس نقطة متميزة واضحة الاختلاف، فنطبق عملية فلترة بحيث تبقى فقط النقاط الأكثر تميزاً.

```
1  >>> sift = cv2.xfeatures2d.SIFT_create()
2  >>> im, x = [], []
3  >>> im.append(img1)
4  >>> im.append(img2)
5  >>> for j in range(2):
6      tem_kp, tem_des = sift.detectAndCompute(im[j], None)
7      tar_kp, tar_des = sift.detectAndCompute(im[1-j], None)
8
9  >>> model = NearestNeighbors(n_neighbors=2).fit(tar_des)
10 >>> dist, indices = model.kneighbors(tem_des)
11 >>> u, v = [], []
12 >>> for i in range(len(tem_kp)):
13     point1 = tem_kp[i].pt
14     point2 = tar_kp[indices[i][0]].pt
15     d1, d2 = dist[i]
```

```

16  >>>             if (d1 / d2) <= 0.7:
17  >>>                 u.append(point1)
18  >>>                 v.append(point2)
19  >>>             u,v = np.asarray(u), np.asarray(v)
20  >>>             x.append(u)
21  >>>             x.append(v)
22  >>> x1_fo, x2_fo, x1_ba, x2_ba =x[0],x[1],x[2],x[3]
23  >>> f = {}
24  >>> for x1, x2 in zip(x1_fo, x2_fo):
25  >>>     f[tuple(x1)] = tuple(x2)
26  >>>
27  >>> b = {}
28  >>> for x1, x2 in zip(x2_ba, x1_ba):
29  >>>     b[tuple(x2)] = tuple(x1)
30  >>>
31  >>> x1_final, x2_final = [], []
32  >>> for x1, x2 in zip(x1_fo, x2_fo):
33  >>>     try:
34  >>>         if b[f[tuple(x1)]] == tuple(x1):
35  >>>             x1_final.append(x1)
36  >>>             x2_final.append(x2)
37  >>>     except KeyError:
38  >>>         pass

```

السطر الأول لتشكيل بنية SIFT ، ومن ثم توليد حلقة لتطبيق الخوارزمية مرتين ولكن مع التبديل بين الصورتين الأولى والثانية. و ضمن الحلقة أولاًً إيجاد نقاط التوافق والواصف لكل صورة على حدى ، ومن ثم تكون بنية KNN وتدريرها على قيم واصف صورة المدف (الصورة الثانية) وبعد ذلك تطبيقها على واصف الصورة الأولى .
القيم المعادة من التعليمية في السطر 10 هي :

مصفوفة بحجم $n \times 2$ تحوي المسافة بين عنصر من أحد المجموعتين عن أقرب عنصرين (جارين) في المجموعة الثانية.

مصفوفة بحجم $n \times 2$ تحوي دليل (ترتيب) العناصر المقابلة للمسافات في المصفوفة السابقة. ثم المرور بحلقة على كامل النقاط من المجموعة الأولى، مع مقابلتها (جيرانها الأقرب) من المجموعة الثانية مع تطبيق اختبار النسبة وتحديد نسبة مئوية قيمتها 70% من أجل قبول النقاط الحقيقة لهذه النسبة.

وبعد ذلك تطبيق اختبار الاتساق الثنائي، في الأسطر 23 حتى 29 رُبطت احداثيات النقاط الهامة كثنائيات (x, y) وذلك لكلا عملية الكشف الأمامية والعكسية.

ومن ثم مقارنة النتائج فإن كانت النقطة متواجدة في التحويلين الأمامي والعكسي معاً، فيجب الإبقاء عليها كنقطة نهائية.

بجمع التعليمات السابقة ضمن تابع مستقل تحت اسم Find_match يكون دخله صورتين رماديتين، وخرجته نقاط التوافق بين الصورتين.

نتائج تطبيق التابع:

بعد تطبيق خوارزمية SIFT كان عدد العناصر الناتجة 4064 في الاتجاه الأمامي (من الصورة الأولى إلى الثانية) و 3945 في الاتجاه العكسي .

ولكن بعد تطبيق اختبار النسبة انخفض عدد العناصر إلى 402 في الاتجاه الأمامي، و 381 في الاتجاه العكسي.

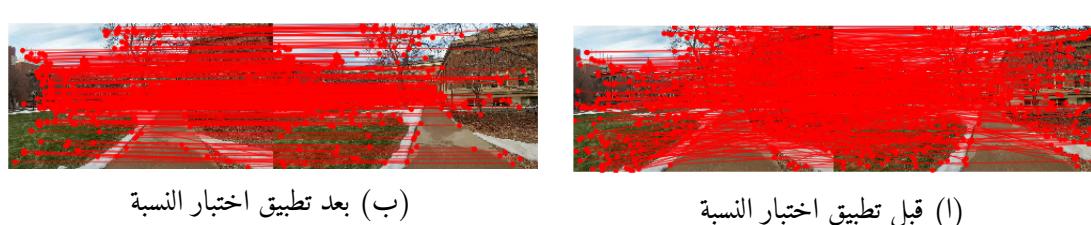
وأخيرا بعد تطبيق اختبار الاتساق الثنائي الاتجاه انخفض عدد المعلم إلى 318 عنصراً بين الصورتين.



شكل 1: الصور الأصلية



شكل 2: الاتجاه الأمامي



شكل 3: الاتجاه العكسي



شكل 4: بعد تطبيق اختبار الاتساق الثاني

5 حساب المصفوفة الأساسية :

لتكن img_1, img_2 صورتان مختلفتان تم التقاطهما لنفس المشهد من مواضع مختلفة، ولتكن p_1 نقطة في صورة img_1 ، إحداثياتها $[x_1 \ y_1 \ 1]$ ، ونقطة في صورة img_2 ، إحداثياتها $[x_2 \ y_2 \ 1]$ ، والنقطتان p_1, p_2 متقابلتان، أي تعبان عن نفس العنصر في الصورتين. فتكون المصفوفة الأساسية Matrix Fundamental هي مصفوفة بحجم 3×3 تربط بين إحداثيات النقطتين المتواقتين وفق العلاقة:

$$P_1^T \cdot F \cdot P_2 = 0 \quad (1)$$

تحسب المصفوفة الأساسية من خلال خوارزمية النقاط ثنائية، والتي تحتاج إلى إحداثيات ثمان نقاط مع مقابلاتها لأجل تحديد قيم المصفوفة الأساسية.[5]

$$\begin{vmatrix} u_0 & v_0 & 1 \end{vmatrix} \cdot \begin{vmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{vmatrix} \cdot \begin{vmatrix} x_1 \\ y_1 \\ 1 \end{vmatrix} = 0 \quad (2)$$

وبحل العلاقة السابقة نحصل على:

$$\begin{vmatrix} F_{11} \cdot u_0 \cdot v_0 & F_{12} \cdot u_1 \cdot v_0 & F_{13} \cdot v_0 \\ F_{21} \cdot u_0 \cdot v_1 & F_{22} \cdot u_1 \cdot v_1 & F_{23} \cdot v_1 \\ F_{31} \cdot u_0 & F_{32} \cdot u_1 & F_{33} \end{vmatrix} = 0 \quad (3)$$

ولكن في حالة المطابقة يوجد عدد كبير من النقاط المتقابلة (مايزيد على 300 نقطة) فإن اعتماد ثمان نقاط فقط قد يؤدي إلى الحصول على نتائج غير دقيقة قد لا تتوافق باقي النقاط. فالمطلوب هو تحديد أفضل قيم للمصفوفة الأساسية، وذلك من خلال تحليل الانحدار Regression . توجد عدة طرق لتحقيق أفضل قيم للمصفوفة الأساسية ومن ضمنها خوارزمية RANSAC .

RANSAC (RANDOM SAmples Consensus) يتم ضمنها تحديد ثمان نقاط عشوائياً ويستخدم هذه النقاط يتم حساب قيمة F ، بعدها يتم توليد نقاط P'_1 اعتماداً على P_0 و مقارنة P'_1 مع P_1 وحساب مجموع مربع الأخطاء الناتجة من النقاط الثمانية.

وتكرار العملية لأجل ثمان نقاط عشوائية أخرى، وهكذا حتى تصبح قيمة الخطأ أصغر من عتبة معينة، أو تكرار العملية لعدد محدد من المرات ولتكن 1000 مرة ومن ثم اختيار قيمة F الموافقة لأصغر خطأ ناتج.[3]

مصفوفة التحويل الأساسية F ناتجة عن جداء مصفوفة معايرة الكاميرا K بالمصفوفة الرئيسية E (والتي تحوي ضمناً مصفوفة الانتقال T والدوران R) . وبما أن مصفوفة الانتقال من نوع rigid تكون برتبة 2 rank=2 فإن أي مصفوفة مكونة منها ستكون لها نفس الرتبة.

و يجعل المصفوفة F من الرتبة 2 وهي بحجم 3×3 ، تُحلل المصفوفة وفق SVD والتي من خلالها يتم الحصول على القيم المميزة eigen values ، ومن ثم جعل أصغر القيم المميزة قيمتها صفر، وبالتالي تقليل رتبة المصفوفة من 3 إلى 2 . [7]

```

1  >>> n, nn = pts1.shape
2  >>> indices = np.arange(n)
3  >>> for nn in range(1000):                      ## RANSAC iterations
4  >>>     np.random.shuffle(indices)             ## random shuffling
5  >>>     first_eight_indices = indices[:8]    ## take random 8
6  >>>     # Compute tentative F using null space of 8 points matrix
7  >>>     ps1,ps2= pts1[first_eight_indices], pts2[
8  >>>         first_eight_indices]
9  >>>     assert ps1.shape == ps2.shape == (8, 2)
10 >>>     A = np.zeros((8, 9))
11 >>>     for i, (u, v) in enumerate(zip(ps1, ps2)):
12 >>>         A[i, 0], A[i, 1], A[i, 2], A[i, 3],= u[0] * v[0], u
13 >>>             [1] * v[0], v[0], u[0] * v[1]
14 >>>             A[i, 4] , A[i, 5], A[i, 6], A[i, 7], A[i, 8]= u[1]
15 >>>             * v[1], v[1], u[0], u[1], 1
16 >>>     F = null_space(A)
17 >>>     F = F[:, 0]
18 >>>     F_tentative = F.reshape(3, 3)
```

```

16  >>>     u, d, vt = np.linalg.svd(F_tentative)      ## Do SVD
17  cleanup
17  >>>     d[2] = 0                                ##make rank=2 by deleting one of
18  the eigen_values
18  >>>     F_cleaned = np.dot(u * d, vt)    ## remake the matrix with
rank=2
19  >>>     loss = []
20  >>>     for pt1, pt2 in zip(pts1, pts2):    ### rebuilding p1
depending on F and p0
21  >>>         u1 = np.asarray([pt1[0], pt1[1], 1])
22  >>>         v1 = np.asarray([pt2[0], pt2[1], 1])
23  >>>         per_point_loss = np.dot(np.matmul(v1, F_cleaned), u1)
24  >>>         loss.append(per_point_loss)
25  >>>     loss = np.asarray(loss)
26  >>>     loss = np.sum(loss ** 2)    ## mean square error
27  >>>     if min_loss is None or loss < min_loss:
28  >>>         min_loss = loss      ## chose the minimum loss
29  >>>         best_F = F_cleaned    ## the final F that makes the
min loss

```

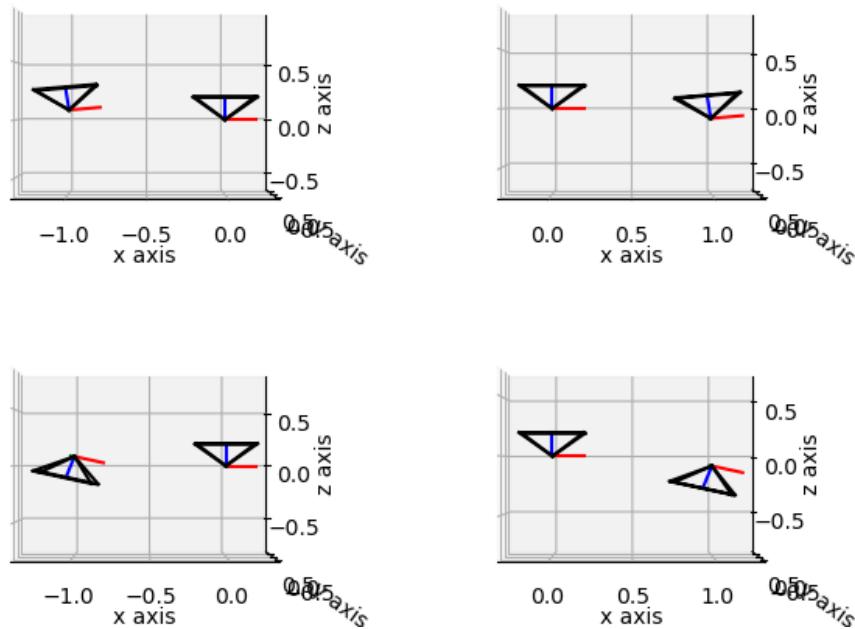
وبعد الحصول على المصفوفة الأساسية F يمكن رسم خطوط epipolar وهي الخطوط الناتجة من تقاطع مستوى epipolar (وهو المستوى المشكل من مركزي الكامرتين و النقطة المدرستة) مع مستوى الصورة .



شكل 5 : خطوط Epipolar

6 إعادة البناء : Reconstruction

بمعرفة المصفوفة الأساسية F ومصفوفة بارامترات الكاميرا K (المعطاة) يمكن تحديد مواضع الكاميرتين والتي تكون ضمن أربع مجموعات تحقق نفس القيم لكل من F و K .
الشكل (6) يعرض التكوينات الأربع الممكنة.



شكل 6: التكوينات الأربع لتوضع الكاميرا

الثلث 7 : Triangulation

ويعني الحصول على الإحداثيات ثلاثية البعد لل نقاط الموجودة في الصورتين، وذلك اعتماداً على إحداثيات نقاط التوافق، ومصفوفتي إسقاط الكاميرتين. وفق العلاقة التالية:

$$\begin{aligned} x_1 \times P_1 \cdot P_{3d} &= 0 \\ x_2 \times P_2 \cdot P_{3d} &= 0 \end{aligned}$$

$$\left| \begin{array}{c} u_1 \\ v_1 \\ 1 \end{array} \right| \times P_1 \cdot \left| \begin{array}{c} P_{3d} \\ 1 \end{array} \right| = 0 \quad (4)$$

$$\left| \begin{array}{c} u_2 \\ v_2 \\ 1 \end{array} \right| \times P_2 \cdot \left| \begin{array}{c} P_{3d} \\ 1 \end{array} \right| = 0$$

حيث x_1, x_2 هي إحداثيات نقطتين متقابلتين من الصورتين، P_1, P_2 هي مصفوفتي إسقاط الكاميرتين، P_{3d} هي إحداثيات ثلاثية البعد الناتجة. لإجراء عملية الجداء الشعاعي يجب تحويل الشعاع الأول إلى الشكل

$$\begin{vmatrix} u_1 \\ v_1 \\ 1 \end{vmatrix} \times P_1 \implies \begin{vmatrix} 0 & -1 & v_1 \\ 1 & 0 & -u_1 \\ -v_1 & u_1 & 0 \end{vmatrix} \cdot P_1 \quad (5)$$

وبحل المعادلة (4) يتم الحصول على الإحداثيات ثلاثية البعد.

```

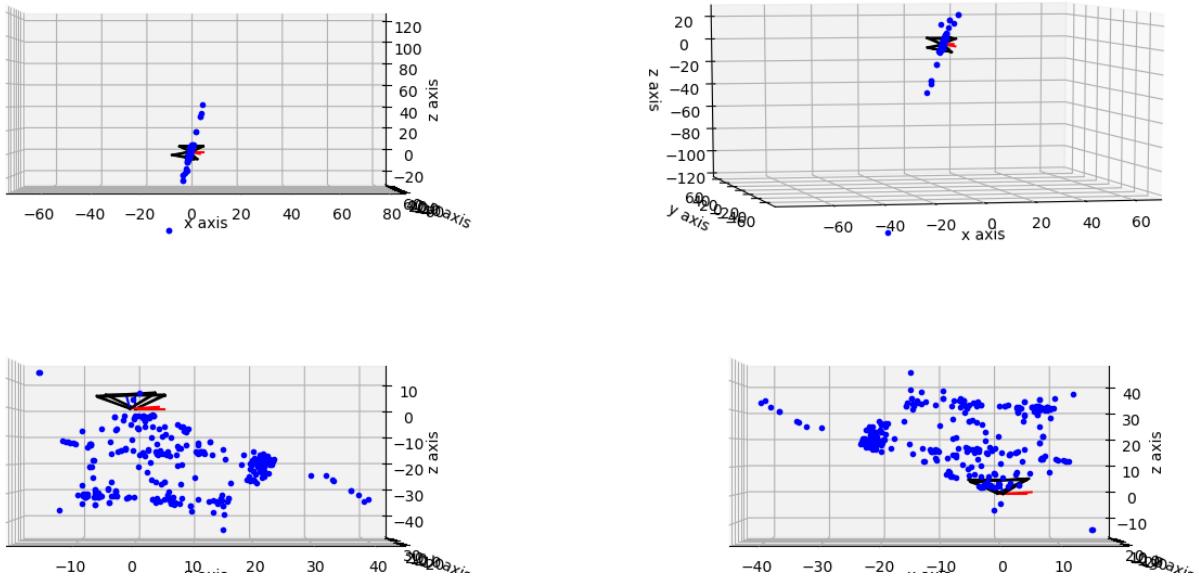
1  >>> for p1, p2 in zip(x1, x2):
2    >>> p1_3d = list(p1) + [1]      ### add z=1
3    >>> p2_3d = list(p2) + [1]
4    >>> p1_skew =np.asarray([
5        >>>                 ## make skew matrix
6        [0, -p1_3d[2], p1_3d[1]],
7        [p1_3d[2], 0, -p1_3d[0]],
8        [-p1_3d[1], p1_3d[0], 0],])
9    >>>
10   >>> p2_skew =np.asarray([
11     [0, -p2_3d[2], p2_3d[1]],
12     [p2_3d[2], 0, -p2_3d[0]],
13     [-p2_3d[1], p2_3d[0], 0],])
14   >>> p1_cross_P1 =np.dot(p1_skew, P1)
15   >>> p2_cross_P2 =np.dot(p2_skew, P2)
16   >>> A = np.vstack((p1_cross_P1[:2], p2_cross_P2[:2])) ## stack results
17   >>> X = null_space(A, rcond=1e-1) ## solve
18   >>> X[:, 0]/ X[3,0]
19   >>> pts3D.append(X[:3])
  >>> pts3D = np.asarray(pts3D)

```

يمكن رسم مواضع الكامرتين مع الغيصة النقطية المعبرة عن النقاط ثلاثية البعد المعاد بناؤها كما في الشكل (7)

8 تصحيح الموضع : Pose Disambiguation

بعد الحصول على التكوينات الأربع لموضع الكامرتين النسبي والنقاط ثلاثية البعد المعاد بناؤها يجب تحديد أي تكوين من هذه التكوينات يحقق أفضل نتيجة. ويتم تقييم التكوينات من خلال حساب عدد النقاط المتوضعة أمام الكامرتين، فالتوسيع الأفضل هو التوسيع الذي يحقق أكبر عدد من النقاط الواقعة أمام الكامرتين معاً. وذلك اعتماداً على تكوينات مصفوفتي الدوران R وموقع مرکزي الكامرتين C والنقاط ثلاثية البعد المعاد بنائهما pts3d ، و اختيار التكوين الأفضل من بينها.



شكل 7: تكوينات توضع الكاميرا مع قيمة النقاط ثلاثة البعد

```

1  >>> best_i = 0
2  >>> bestValid = 0
3  >>> for i, (r, c, pt_3d) in enumerate(zip(Rs, Cs, Pts_3d)):
4    >>> nValid = 0
5    >>> c = c.reshape(-1)
6    >>> r3 = r[2, :]
7    >>> for x in pt_3d:
8      >>>         view_1 = (x - c)[2]
9      >>>         view_2 = np.dot(x - c, r3)
10     >>>        if view_1 > 0 and view_2 > 0:
11       >>>          # Both cameras looks towards this point
12       >>>          nValid += 1
13     >>>        if nValid > bestValid:
14       >>>          bestValid = nValid
15     >>>          best_i = i
16   >>> Rs, Cs, Pts_3d = Rs[best_i], Cs[best_i], Pts_3d[best_i]
```

وفي الشكل (7) في التكوين العلويين عدد النقاط المتوفرة معدومة $nValid = 0$ وفي التكوين السفلي الأيمن $nValid = 4$ وفي التكوين السفلي الأيسر $nValid = 314$.
بالتالي التكوين الرابع هو التكوين الأفضل لموضع الكاميرتين والنقاط ثلاثة البعد.

التقويم 9 : Rectification

الهدف من هذه العملية هو جعل مستوى الصورتين متوازيين ، أي جعل الكامرتين في وضع co_planar ، وبالتالي تصبح خطوط البحث عن التوافقات Epipolar أفقية، ونقطة تقاطعها (EpipolePoints) تقع في اللانهاية. وذلك من خلال تحديد مصفوفتين لتصحيح الصورتين تدعيان بمصفوفة التجانس Homographies.

نقطة epipole تعبّر عن مسقط مركز كل كاميرا على مستوى صورة الكاميرا الثانية. فالنقطة e هي مسقط مركز الكاميرا الثانية o' وفق مصفوفة الإسقاط الخاصة بالكاميرا الأولى M . أي:

$$e = M \begin{vmatrix} o' \\ 1 \end{vmatrix} = \begin{vmatrix} R^T \cdot T \\ 1 \end{vmatrix} = K[I \quad | \quad O] \begin{vmatrix} R^T \cdot T \\ 1 \end{vmatrix} = K \cdot R^T \cdot T \quad (6)$$

ومصفوفة الدوران المصححة R_{rect} وهي من الشكل $R_{rect} = [r_x^T \quad r_y^T \quad r_z^T]$ والتي يجب أن تتحقق الشروط التالية:[4]

1- محور x الجديد يوازي خط الأفق (الواصل بين مركزي الكامرتين) أي:

$$r_x = \frac{c_1 - c_2}{|c_1 - c_2|} \quad (7)$$

2- محور z الجديد يعادل المستوى المشكّل من المحور x الجديد وشعاع عشوائي k (وليكن المحور z القديم). أي:

$$r_z = k \wedge r_x \Rightarrow r_z = \frac{k - r_x * (k \cdot r_x)}{|k - r_x * (k \cdot r_x)|} \quad (8)$$

3- المحور y يعادل المستوى المتشكل من المحورين x, z الجديدين. أي:

$$r_y = r_x \wedge r_z \quad (9)$$

فتعطى مصفوفة التجانس لكل صورة H بالعلاقة:

$$H = K \cdot R_{rect} \cdot R^T \cdot K^{-1} \quad (10)$$

حيث أن K هي مصفوفة المعايرة للكاميرا، و R مصفوفة الدوران الخاصة بالكاميرا، و R_{rect} هي مصفوفة الدوران المصححة.

```

1  >>> C1 = np.zeros(3)
2  >>> R1 = np.identity(3)    ### no rotation for the 1st camera
3  >>> C2 = C.reshape(-1)    ### c is the position of camera's centre
4  >>> R2 = R
5  >>> RrectX = C2 / np.linalg.norm(C2)  ### rotation around x axis
6  >>> R1Z = R1[2, :]      ### arbitrary k is the old z-axis
7  >>> RrectZ = R1Z - RrectX * np.dot(R1Z, RrectX)
8  >>> RrectZ = RrectZ / np.linalg.norm(RrectZ)
9  >>> RrectY = np.cross(RrectZ, RrectX)
10 >>> Rrect = np.asarray([RrectX, RrectY, RrectZ])
11 >>> H1= np.dot(dot(K, Rrect),np.linalg.inv(K))
12 >>> H2= np.dot(dot(K, Rrect),np.dot(R2.T,np.linalg.inv(K)))

```

بعد تطبيق دوران على كل صورة من الصورتين وفق مصفوفة التجانس H الموافقة يتم الحصول على الصور المقومة،
الشكل (8)

وبإعادة إجراء عملية مطابقة المعالم تصبح خطوط المطابقة أفقية، بالإضافة إلى خطوط epipolar أيضاً كما يظهر

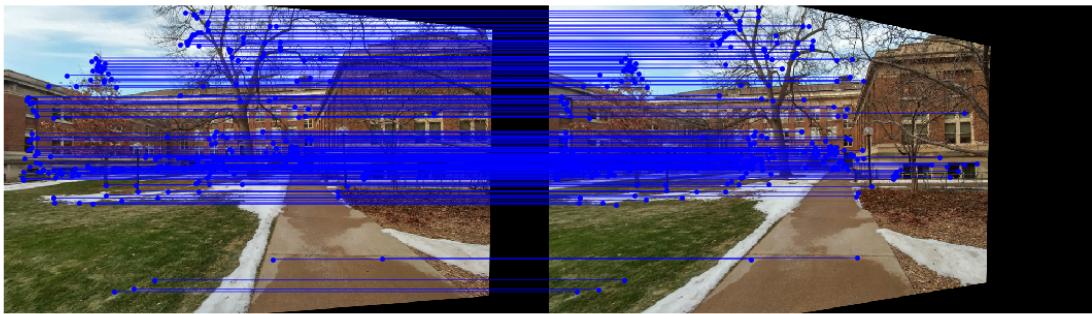


شكل 8: الصور المقومة

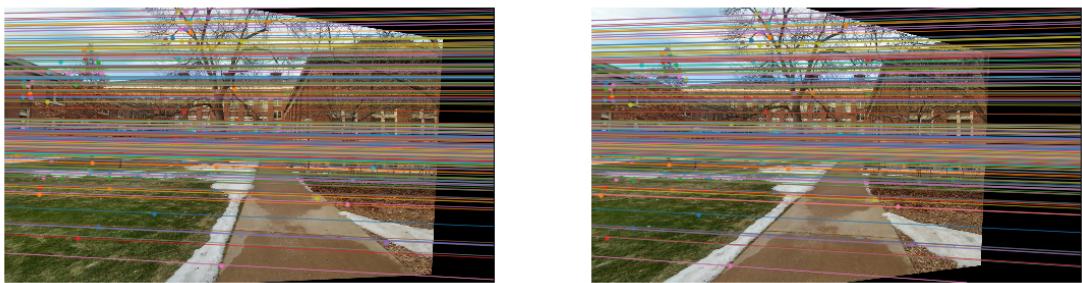
في الشكل (9)

10 حساب التفاوت : Disparity Computing

بعد إجراء عملية تقويم للصورتين وجعل خطوط epipolar أفقية، أي أن كل نقطة من الصورة الأولى تكون مقابلتها في الصورة الأولى واقعة على خط أفقي محدد.



(ا) عملية ربط المعلم



(ب) خطوط Epipolar

شكل 9: بعد تطبيق عملية التقويم

خوارزمية الحل تنص على تحديد الخط الأفقي المقابل في الصورة الثانية، والانزلاق عليه وتحديد النقطة التي تتحقق أعلى تطابق. في تقييم التطابق يتم الاعتماد على كشف SIFT المكثف، والذي يفرق عن SIFT العادي في أن المكثف يوجد الواسط لكل نقاط الصورة، أما العادي فيوجد الواسط فقط للنقطات المحددة وفق خوارزمية Lowe.

فأولاً توصف جميع نقاط الصورة كنقط اهتمام رئيسية مع إهمال نقاط الخلفية الناتجة عن تقويم الصورة وتدويرها، ومن ثم تكوين واسط (من 128 قيمة) لكل نقطة من هذه النقاط. ويتم إجراء المطابقة بحساب البعد الديكارتي بين شعاعي الواسط للنقاطين وفق 128 بُعداً، والنقطة التي تحقق البعد الأدنى هي النقطة المطابقة، ويكون التفاوت هو الفرق بين إحداثي النقطة الأصلية ومطابقتها. أي وفق العلاقة:

$$disparity(i, j) = \operatorname{argmin}_j \{ ||descriptor_1(i, j) - descriptor_2(i, j+k)|| \} \quad (11)$$

where $k = 0, 1, \dots, j$

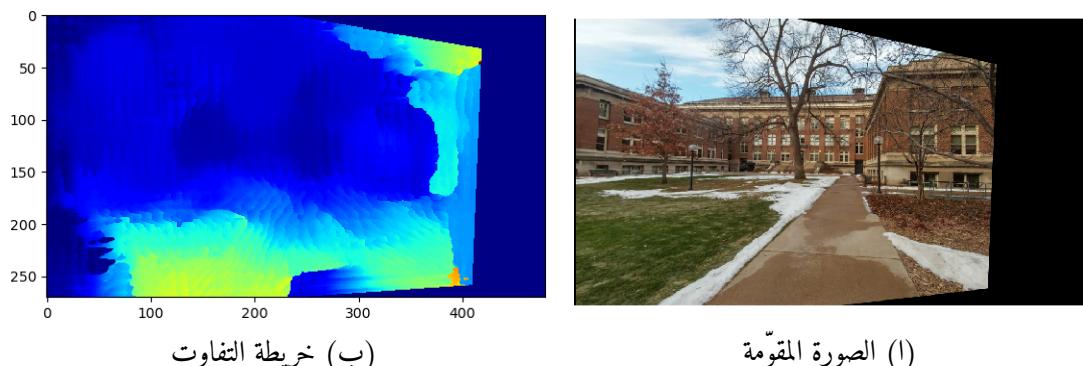
حيث (i, j) هي التفاوت في النقطة (i, j) disparity(i, j) و $descriptor_1(i, j)$ شعاع الواصل للنقطة (i, j) في الصورة الأولى. و $descriptor_2(i, j + k)$ شعاع الواصل للنقاط الواقعة على السطر ذو الرقم i في الصورة الثانية.

```

1  >>> assert img1.shape == img2.shape
2  >>> sift = cv2.xfeatures2d.SIFT_create()
3  >>> im, desc = [], []
4  >>> im.append(img1)
5  >>> im.append(img2)
6  >>> for b in range(2):
7  >>>     h, w = im[b].shape
8  >>>     kp = []
9  >>>     for i in range(h):
10 >>>         for j in range(w):
11 >>>             kp.append(cv2.KeyPoint(x=j, y=i, _size=7)) ## make pixels as key-points
12 >>>     kps, des = sift.compute(im[b], kp)      ## compute descriptors
13 >>>     des = np.asarray(des).reshape((h, w, 128))
14 >>>     desc.append(des)
15 >>>
16 >>> dense1, dense2 = desc[0], desc[1]
17 >>> disparity = np.ones(img1.shape) ## initializing disparity
18 >>> h, w = img1.shape
19 >>> for i in range(h):
20 >>>     for j in range(w):
21 >>>         if img1[i, j] == 0:
22 >>>             continue ##ignoring background
23 >>>         d1_d2_dists = []
24 >>>         d1 = dense1[i, j] ## 1st point's descriptor
25 >>>         for k in range(0, j + 1): ##slipping on a row
26 >>>             d2 = dense2[i, k]
27 >>>             d1_d2_dists.append(np.linalg.norm(d1 - d2))
28 >>>         disparity[i, j] = np.abs(np.argmin(d1_d2_dists) - j)

```

في الشكل (10) بمقارنة الصورة المقومة مع خريطة التفاوت الناتجة، يظهر أن الألوان الناتجة تتغير حسب بعد بالتدرج من الأحمر (الأقرب) فالأخضر فالأزرق وأخيراً الأزرق الداكن للنقاط بعيدة. وبما أن الصورة المستهدفة لا تحوي عناصر أمامية واضحة، بل أن أغلب عناصر الصورة توجد في الخلفية لذلك فإن اللون الأزرق الداكن يطغى على خريطة التفاوت، مع بعض مناطق ملونة بالأخضر تقابل الأرضية القرية من الكاميرا، وغضن الشجرة الذي في أعلى يمين الصورة.



شكل 10: حساب التفاوت

(ا) الصورة المقومة

(ب) خريطة التفاوت

11 المراجع :

- [1] Lowe, David. 2004, Distinctive Image Features From Scale-invariant Keypoints, International Jornal of Computer Vision.
- [2] Kozma, Laszlo.2008, k Nearest Neighbors Algorithm kNN, Helsinki University of Technology.
- [3] Collins, Robert.2009, Robust Estimation RANSAC, Penn State University, Pennsylvania,USA.
- [4] Kitani,Kris . 2012, Stereo Vision Rectification, Carnegie Mellon University.
- [5] Forsyth,David.Ponce,Jean. and 2012,COMPUTER VISION A MODERN APPROACH second edition , University of Illinois at Urbana-Champaign .
- [6] Open Source Computer Vision official site. Introduction to SIFT (Scale-Invariant Feature Transform)
https://docs.opencv.org/trunk/da/df5/tutorial_py_sift_intro.html Accessed on April 12, 2020.
- [7] Brahmbhatt ,Samarth . 2014, Why is the fundamental matrix in computer vision rank 2?
<https://www.quora.com/Why-is-the-fundamental-matrix-in-computer-vision-rank-2>
Accessed on April 17, 2020.