

Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут ім. Ігоря Сікорського”
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Лабораторна робота №2

Виконали:
студенти КА-77
Буркацький Микита
Морозов Роман
Котів Сергій

Київ-2020

Завдання

Найпершою і найголовнішою справою буде вибрати назву команди і записатись в табличку в цій же папці. У вас уже є OpenCV і навички роботи з зображеннями і не тільки, далі залишився мізер: Кожен учасник або учасниця команди спершу обирає дескриптор (один із розглянутих у лекціях або ж знайдений окремо) та предмет на прикладі якого відбудуватиметься дослідження. Враховуючи що вас багато, будь ласка обирайте унікальніші предмети за улюблену чашку/телефон/мишку. Маючи те і інше напоготові кожен учасник бригади має зняти не менше сотні фото предмета, варіюючи його розміщення та ракурс в кадрі, освітлення, наявність візуальних перешкод, зашакаленість зображення, фокусну віддаль та тримання рук. Сотня фото обраного предмету на однаковій сцені з однаковою якістю зйомки, але з різних ракурсів на жаль не підійде, постараїтесь наполегливо варіювати сцени і умови зйомки. До цих фото варто додати невелику підбірку зображень, що не містять предмет, або ж містять предмет візуально подібний до вашого, штук 20 повинно вистачити, якщо залишиться натхнення можна й більше. Після чого ми нарешті дійшли до цікавого, а саме до дослідження: Вам потрібно згенерувати обраний дескриптор для обраного предмета, після чого з його допомогою розпізнати об'єкт на всій тестовій вибірці збираючи при цьому такі метрики: відносна кількість правильно суміщених ознак, похибка локалізації (відстань між реальним розміщенням предмета в кадрі та розпізнаним) та відносний час обробки фото в залежності від розміру зображення. Метрики мають зберегтись у файлку для подальших досліджень. Наступним кроком ви обмінюєтесь об'єктом з колегою, і уже маючи готову збиралку метрик, обчислюєте їх для предмета вашого сусіда, таким чином у вас збирається 9 наборів даних, по три на дескриптор. Самою ж ідеєю лаби є дослідити розбіжності у роботі ваших дескрипторів та виконати порівняльний аналіз їх поведінки, сформулювати висновки з викладками і прикладами так аби було зрозуміло вам та, сподіваюсь, усім вашим колегам. Таким чином кінцевим результатом буде від вас гуглдок з описом виняткових особливостей, сильних та слабких сторін дескриптора і обґрунтуванням чому вони поводяться саме так. Бажано аби не було більше однієї групи з однаковою комбінацією дескрипторів, хіба що ви об'єднаєтесь і влаштуєте вашим двом дескрипторам страхітливо прискіпливі дослідження. Сорци, дослідні вибірки фото, зібрани метрики заліяти на Гітхаб та кинути посиланням мені на пошту, в сабж листа копіюстимо "[CVPR 2020]" і далі що хочете, хоч єнота намалюйте. Оберіть у команді одну контактну персону з пошти якої прилетить готова лаба і відбудуватиметься подальша комунікація.

Назва команди

Наша команда називається «Комп'ютерний зір».

Дескриптори

SIFT (Scale-Invariant Feature Transform)

ORB (Oriented FAST and Rotated BRIEF)

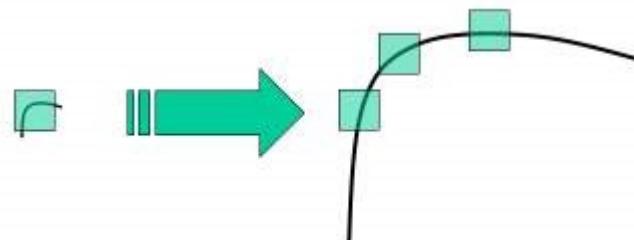
BRIEF (Binary Robust Independent Elementary Features)

Предмети

Книжка1, сірники, книжка2.

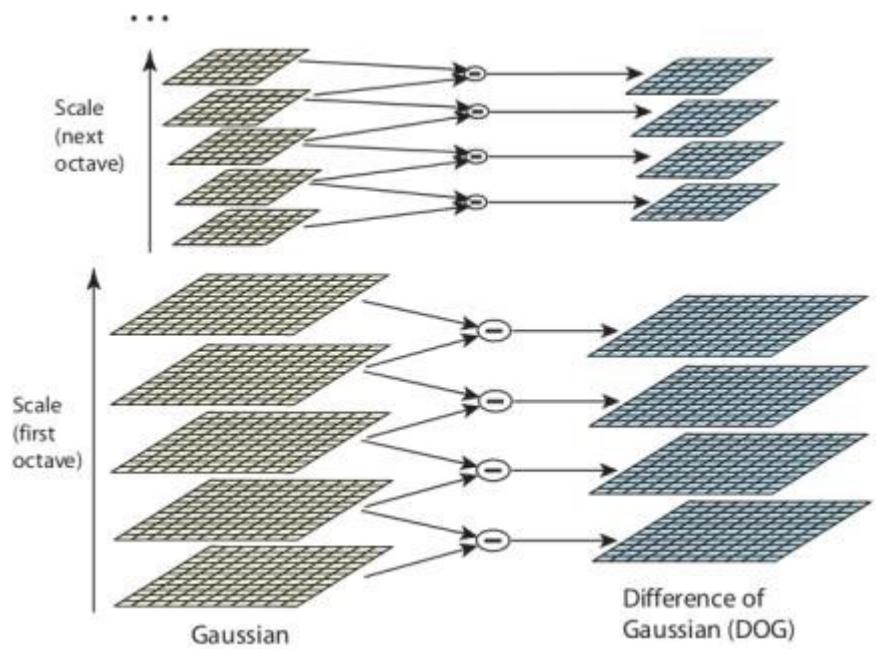
SIFT

Theory. In last couple of chapters, we saw some corner detectors like Harris etc. They are rotation-invariant, which means, even if the image is rotated, we can find the same corners. It is obvious because corners remain corners in rotated image also. But what about scaling? A corner may not be a corner if the image is scaled. For example, check a simple image below. A corner in a small image within a small window is flat when it is zoomed in the same window. So Harris corner is not scale invariant.

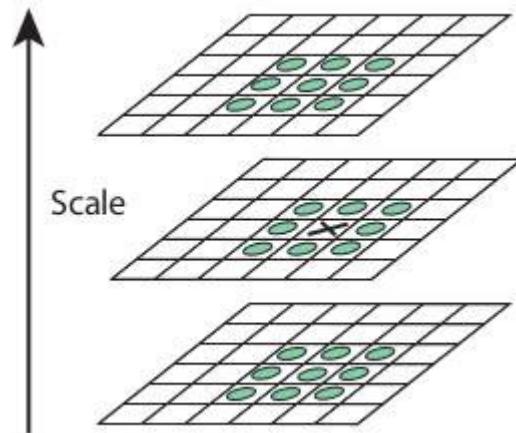


So, in 2004, D.Lowe, University of British Columbia, came up with a new algorithm, Scale Invariant Feature Transform (SIFT) in his paper, Distinctive Image Features from Scale-Invariant Keypoints, which extract keypoints and compute its descriptors.

Scale-space Extrema Detection. From the image above, it is obvious that we can't use the same window to detect keypoints with different scale. It is OK with small corner. But to detect larger corners we need larger windows. For this, scalespace filtering is used. In it, Laplacian of Gaussian is found for the image with various σ values. LoG acts as a blob detector which detects blobs in various sizes due to change in σ . In short, σ acts as a scaling parameter. For eg, in the above image, gaussian kernel with low σ gives high value for small corner while guassian kernel with high σ fits well for larger corner. So, we can find the local maxima across the scale and space which gives us a list of (x, y, σ) values which means there is a potential keypoint at (x,y) at σ scale. But this LoG is a little costly, so SIFT algorithm uses Difference of Gaussians which is an approximation of LoG. Difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different σ , let it be σ_1 and σ_2 . This process is done for different octaves of the image in Gaussian Pyramid. It is represented in below image:



Once this DoG are found, images are searched for local extrema over scale and space. For eg, one pixel in an image is compared with its 8 neighbours as well as 9 pixels in next scale and 9 pixels in previous scales. If it is a local extrema, it is a potential keypoint. It basically means that keypoint is best represented in that scale. It is shown in below image:



Regarding different parameters, the paper gives some empirical data which can be summarized as, number of octaves = 4, number of scale levels = 5, initial $\sigma = 1.6$

, $k = \sqrt{2}$ etc as optimal values.

Keypoint Localization. Once potential keypoints locations are found, they have to be refined to get more accurate results. They used Taylor series expansion of scale space to get more accurate location of extrema, and if the intensity at this extrema is less than a threshold value (0.03 as per the paper), it is rejected. This threshold is called contrast Threshold in OpenCV DoG has higher response for edges, so edges

also need to be removed. For this, a concept similar to Harris corner detector is used. They used a 2×2 Hessian matrix (H) to compute the principal curvature. We know from Harris corner detector that for edges, one eigen value is larger than the other. So here they used a simple function, If this ratio is greater than a threshold, called edgeThreshold in OpenCV, that keypoint is discarded. It is given as 10 in paper. So it eliminates any low-contrast keypoints and edge keypoints and what remains is strong interest points.

Orientation Assignment. Now an orientation is assigned to each keypoint to achieve invariance to image rotation. A neighbourhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created. (It is weighted by gradient magnitude and gaussian-weighted circular window with σ equal to 1.5 times the scale of keypoint. The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation. It creates keypoints with same location and scale, but different directions. It contributes to stability of matching.

Keypoint Descriptor. Now keypoint descriptor is created. A 16×16 neighbourhood around the keypoint is taken. It is divided into 16 sub-blocks of 4×4 size. For each sub-block, 8 bin orientation histogram is created. So a total of 128 bin values are available. It is represented as a vector to form keypoint descriptor. In addition to this, several measures are taken to achieve robustness against illumination changes, rotation etc.

Keypoint Matching. Keypoints between two images are matched by identifying their nearest neighbours. But in some cases, the second closest-match may be very near to the first. It may happen due to noise or some other reasons. In that case, ratio of closest-distance to second-closest distance is taken. If it is greater than 0.8, they are rejected. It eliminates around 90% of false matches while discards only 5% correct matches, as per the paper. So this is a summary of SIFT algorithm. For more details and understanding, reading the original paper is highly recommended. Remember one thing, this algorithm is patented. So this algorithm is included in Non-free module in OpenCV.

ORB

Theory. As an OpenCV enthusiast, the most important thing about the ORB is that it came from "OpenCV Labs". This algorithm was brought up by Ethan Rublee, Vincent Rabaud, Kurt Konolige and Gary R. Bradski in their paper ORB: An efficient alternative to SIFT or SURF in 2011. As the title says, it is a good alternative to SIFT and SURF in computation cost, matching performance and mainly the patents. Yes, SIFT and SURF are patented and you are supposed to pay them for its use. But ORB is not !!!

ORB is basically a fusion of FAST keypoint detector and BRIEF descriptor with many modifications to enhance the performance. First it use FAST to find keypoints, then apply Harris corner measure to find top N points among them. It also use pyramid to produce multiscale-features. But one problem is that, FAST doesn't compute the orientation. So what about rotation invariance? Authors came up with following modification. It computes the intensity weighted centroid of the patch with located corner at center. The direction of the vector from this corner point to centroid gives the orientation. To improve the rotation invariance, moments are computed with x and y which should be in a circular region of radius r , where r is the size of the patch.

Now for descriptors, ORB use BRIEF descriptors. But we have already seen that BRIEF performs poorly with rotation. So what ORB does is to "steer" BRIEF according to the orientation of keypoints. For any feature set of n binary tests at location (x_i, y_i) , define a $2 \times n$ matrix, S which contains the coordinates of these pixels. Then using the orientation of patch, θ , its rotation matrix is found and rotates the S to get steered(rotated) version $S\theta$. ORB discretize the angle to increments of $2\pi/30$ (12 degrees), and construct a lookup table of precomputed BRIEF patterns. As long as the keypoint orientation θ is consistent across views, the correct set of points $S\theta$ will be used to compute its descriptor. BRIEF has an important property that each bit feature has a large variance and a mean near 0.5. But once it is oriented along keypoint direction, it loses this property and become more distributed. High variance makes a feature more discriminative, since it responds differentially to inputs. Another desirable property is to have the tests uncorrelated, since then each test will contribute to the result. To resolve all these, ORB runs a greedy search among all possible binary tests to find the ones that have both high variance and means close to 0.5, as well as being uncorrelated. The result is called rBRIEF.

For descriptor matching, multi-probe LSH which improves on the traditional LSH, is used. The paper says ORB is much faster than SURF and SIFT and ORB descriptor works better than SURF. ORB is a good choice in low-power devices for panorama stitching etc.

BRIEF

Theory. We know SIFT uses 128-dim vector for descriptors. Since it is using floating point numbers, it takes basically 512 bytes. Similarly SURF also takes minimum of 256 bytes (for 64-dim). Creating such a vector for thousands of features takes a lot of memory which are not feasible for resource-constraint applications especially for embedded systems. Larger the memory, longer the time it takes for matching. But all these dimensions may not be needed for actual matching. We can compress it using several methods like PCA, LDA etc. Even other methods like hashing using LSH (Locality Sensitive Hashing) is used to convert these SIFT descriptors in floating point numbers to binary strings. These binary strings are used to match features using Hamming distance. This provides better speed-up because finding hamming distance is just applying XOR and bit count, which are very fast in modern CPUs with SSE instructions. But here, we need to find the descriptors first, then only we can apply hashing, which doesn't solve our initial problem on memory.

BRIEF comes into picture at this moment. It provides a shortcut to find the binary strings directly without finding descriptors. It takes smoothed image patch and selects a set of n_d (x,y) location pairs in a unique way (explained in paper). Then some pixel intensity comparisons are done on these location pairs. For eg, let first location pairs be (x_1, y_1) and (x_2, y_2) . If $|I(x_1, y_1) - I(x_2, y_2)| > \epsilon$, then its result is 1, else it is 0. This is applied for all the location pairs to get a n_d -dimensional bitstring. This n_d can be 128, 256 or 512. OpenCV supports all of these, but by default, it would be 256 (OpenCV represents it in bytes. So the values will be 16, 32 and 64). So once you get this, you can use Hamming Distance to match these descriptors.

One important point is that BRIEF is a feature descriptor, it doesn't provide any method to find the features. So you will have to use any other feature detectors like SIFT, SURF etc. The paper recommends to use CenSurE which is a fast detector and BRIEF works even slightly better for CenSurE points than for SURF points. In short, BRIEF is a faster method for feature descriptor calculation and matching. It also provides high recognition rate unless there is large in-plane rotation.

Особливі характеристики

BRIEF. Після того, як особливі точки знайдені, обчислюють їх дескриптори, тобто набори ознак, що характеризують околиця кожної особливої точки. BRIEF - швидкий евристичний дескриптор, будується з 256 бінарних порівнянь між яркостями пікселів на розмите зображення. Було розглянуто кілька способів вибору точок для бінарних порівнянь. Як виявилося, один з кращих способів - вибирати точки випадковим чином Гауссовським розподілом навколо центрального пікселя. Ця випадкова послідовність точок вибирається один раз і надалі не змінюється. Розмір даної околиці точки дорівнює 31x31 піксель, а апертура розмиття дорівнює 5. Отриманий бінарний дескриптор

виявляється стійкий до змін освітлення, перспективному сптворення, швидко обчислюється і порівнюється, але дуже нестійкий до обертання в площині зображення.

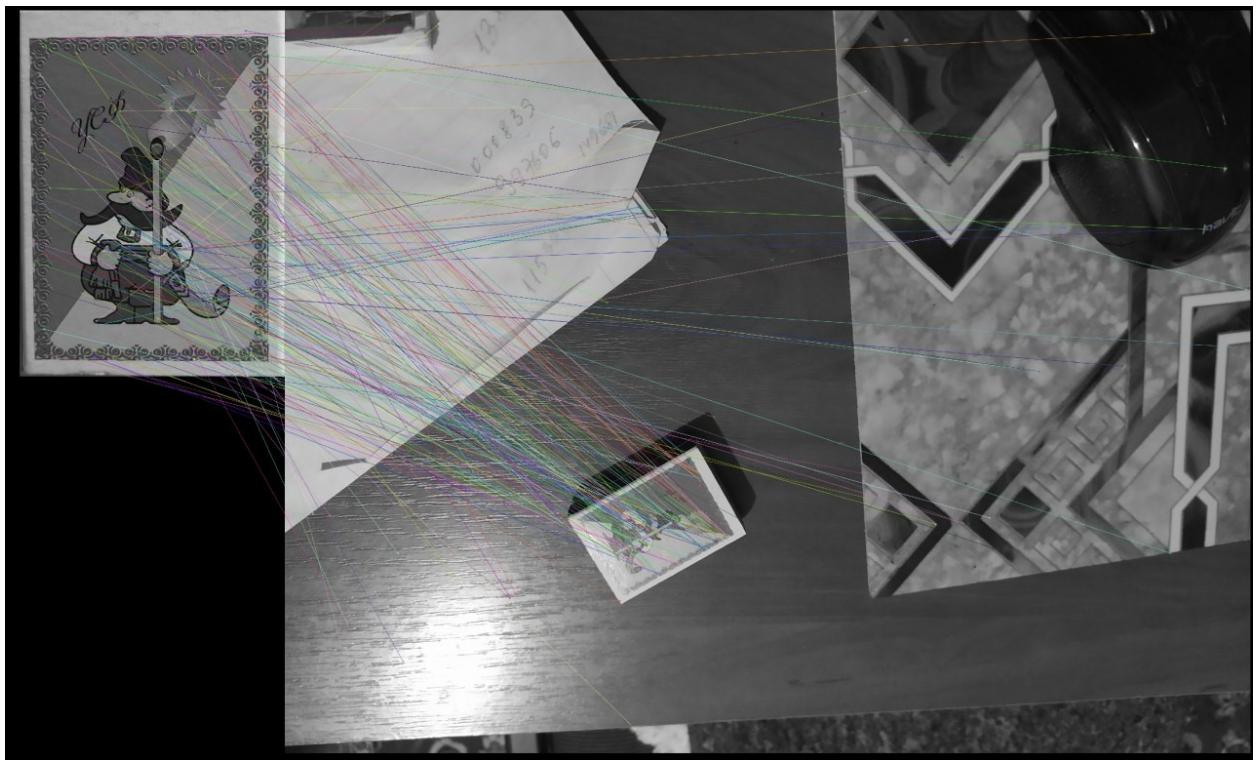
ORB. Швидкий і ефективний. Розвитком всіх цих ідей став алгоритм ORB, в якому зроблена спроба поліпшити проізводітельноть BRIEF при повороті зображення. Запропоновано спочатку обчислювати орієнтацію особливої точки і потім проводити бінарні порівняння вже у відповідність з цією орієнтацією. Працює алгоритм так:

- 1) Особливі точки виявляються за допомогою швидкого деревовидного FAST на оригінальному документі і на кількох зображеннях з піраміди зменшених зображень.
- 2) Для виявленіх точок обчислюється міра Харриса, кандидати з низьким значенням міри Харриса відкидаються.
- 3) Обчислюється кут орієнтації особливої точки. Для цього, спочатку обчислюються моменти яскравості для околиці особливої точки: x , y - піксельні координати, I - яскравість. I потім кут орієнтації особливої точки. У підсумку отримуємо деякий напрямок для околиці особливої точки.
- 4) Маючи кут орієнтації особливої точки, послідовність точок для бінарних порівнянь в дескрипторі BRIEF повертається у відповідність з цим кутом.
- 5) За отриманими точкам обчислюється бінарний дескриптор BRIEF.

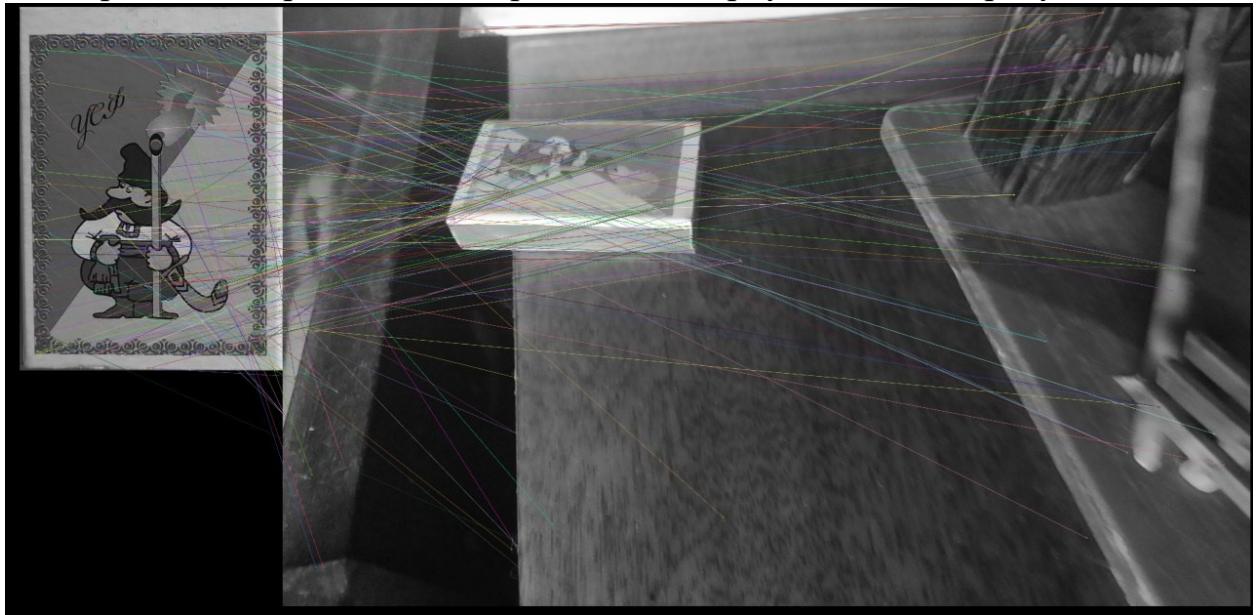
Є в ORB ще одна цікава деталь, яка вимагає окремих пояснень. Справа в тому, що в той момент, коли ми «довертаємо» всі особливі точки до нульового кута, випадковий вибір бінарних порівнянь в дескрипторі перестає бути випадковим. Це призводить до того, що, по-перше, деякі бінарні порівняння виявляються залежними між собою, по-друге, їх середнє вже не дорівнює 0.5 (1 - світліше, 0 - темніше, коли вибір був випадковим, то в середньому було 0.5) і все це істотно зменшує здатність дескриптора розрізняти особливі точки між собою. Рішення - потрібно вибрати «правильні» бінарні тести в процесі навчання.

Висновки

SIFT



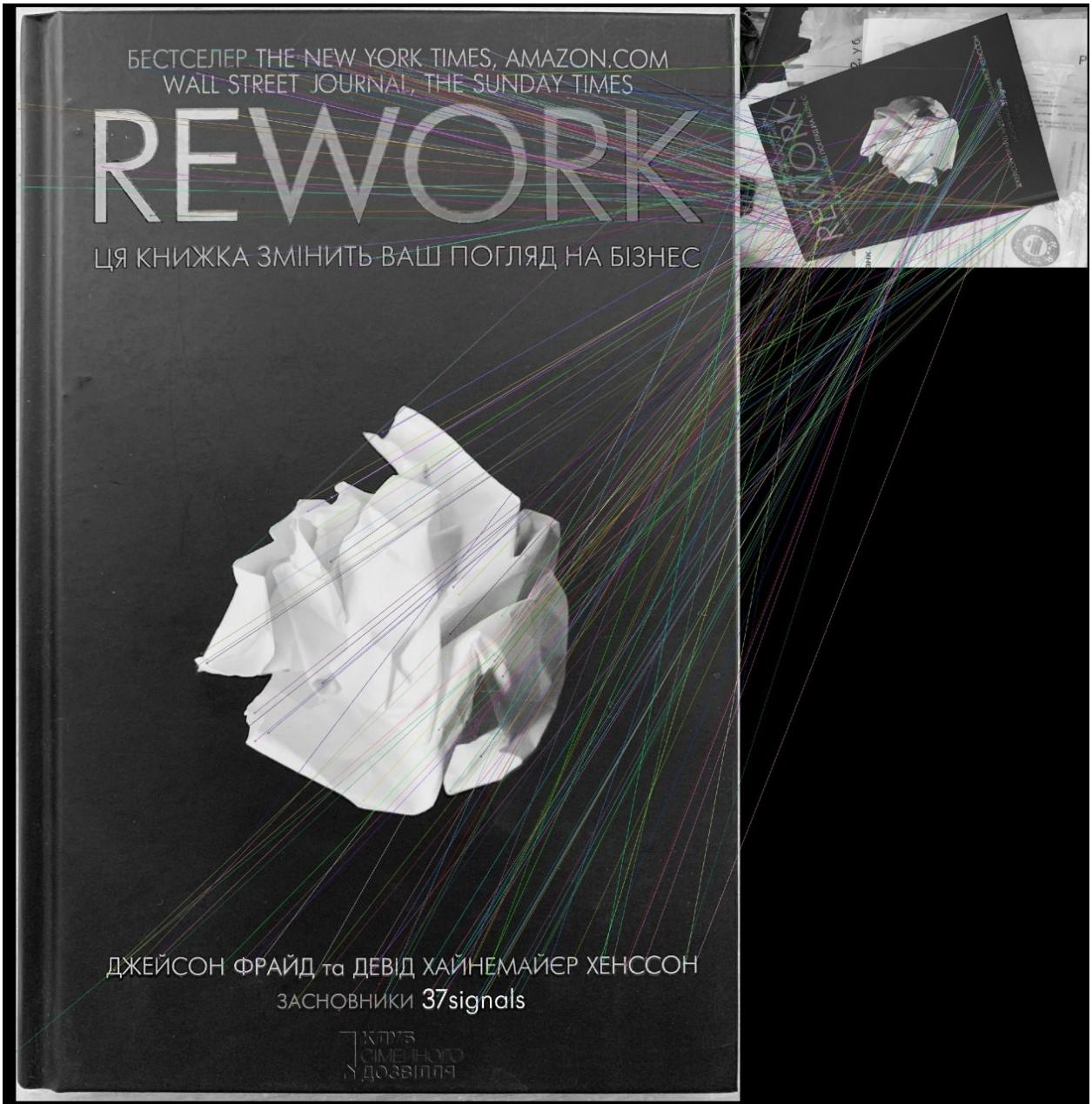
SIFT працює найкраще коли зображення повернуте на 30-45 градусів

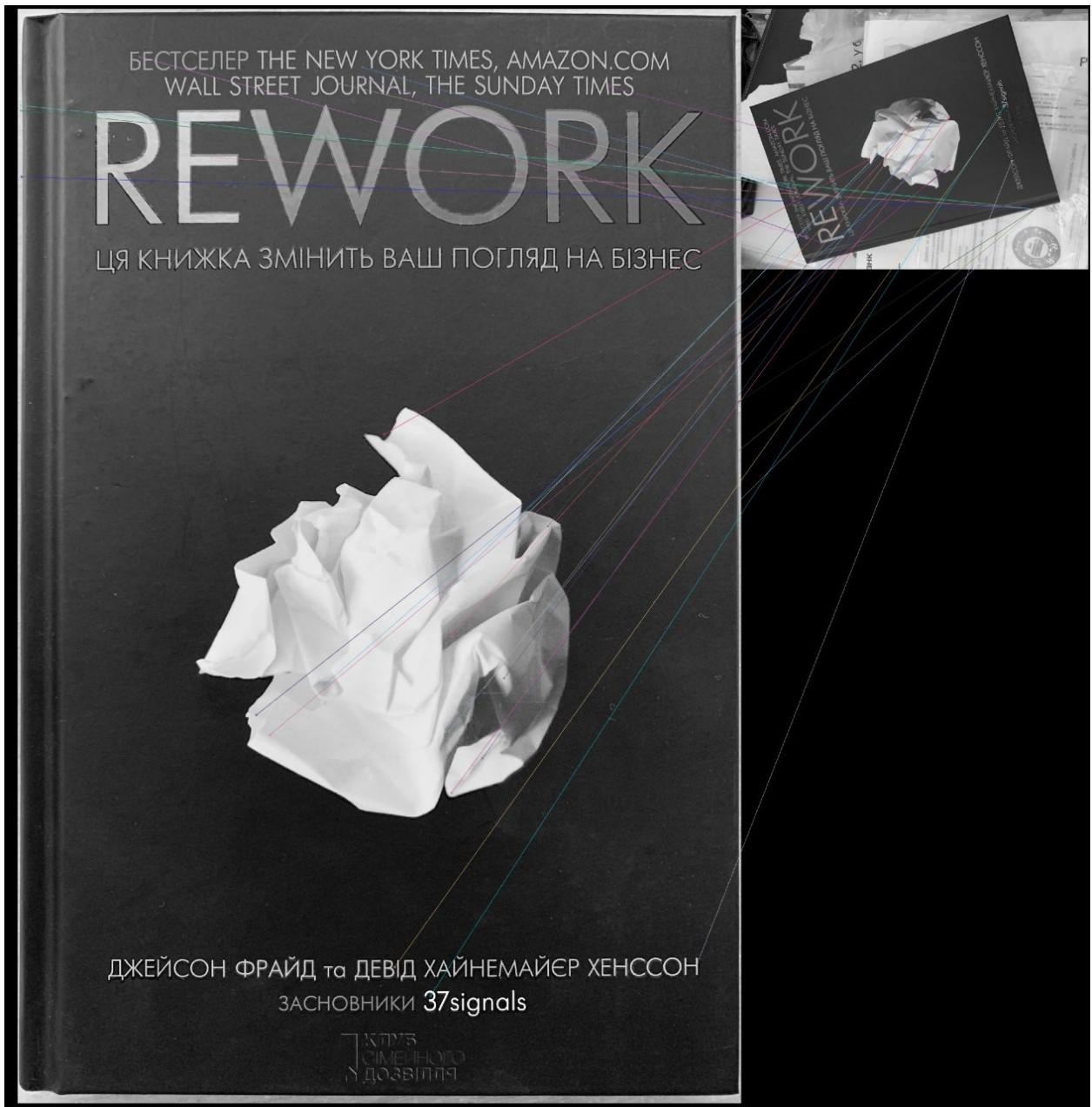


але із більшим поворотом та трептінням камери, зміною світла якість зменшується, хоч і не так суттєво як для інших

Загалом, дескриптор справився добре на фотографіях з книгами, бо в них було мало різноманіття, світло та якість змінюються не сильно

на фотографіях із сірниками виникали проблеми із поворотом, розмиттям та більшим контрастом світла



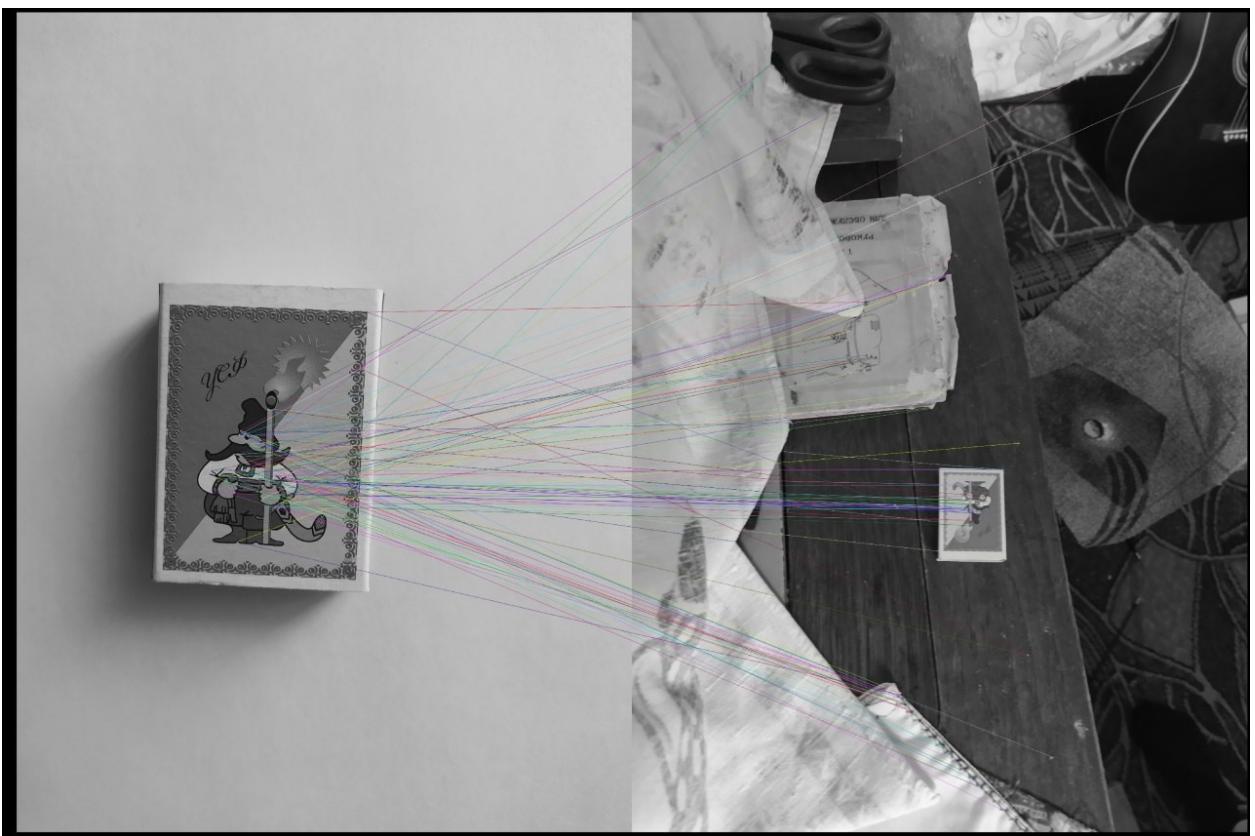


при повороті та трохи гіршому світлі SIFT робить кілька помилок



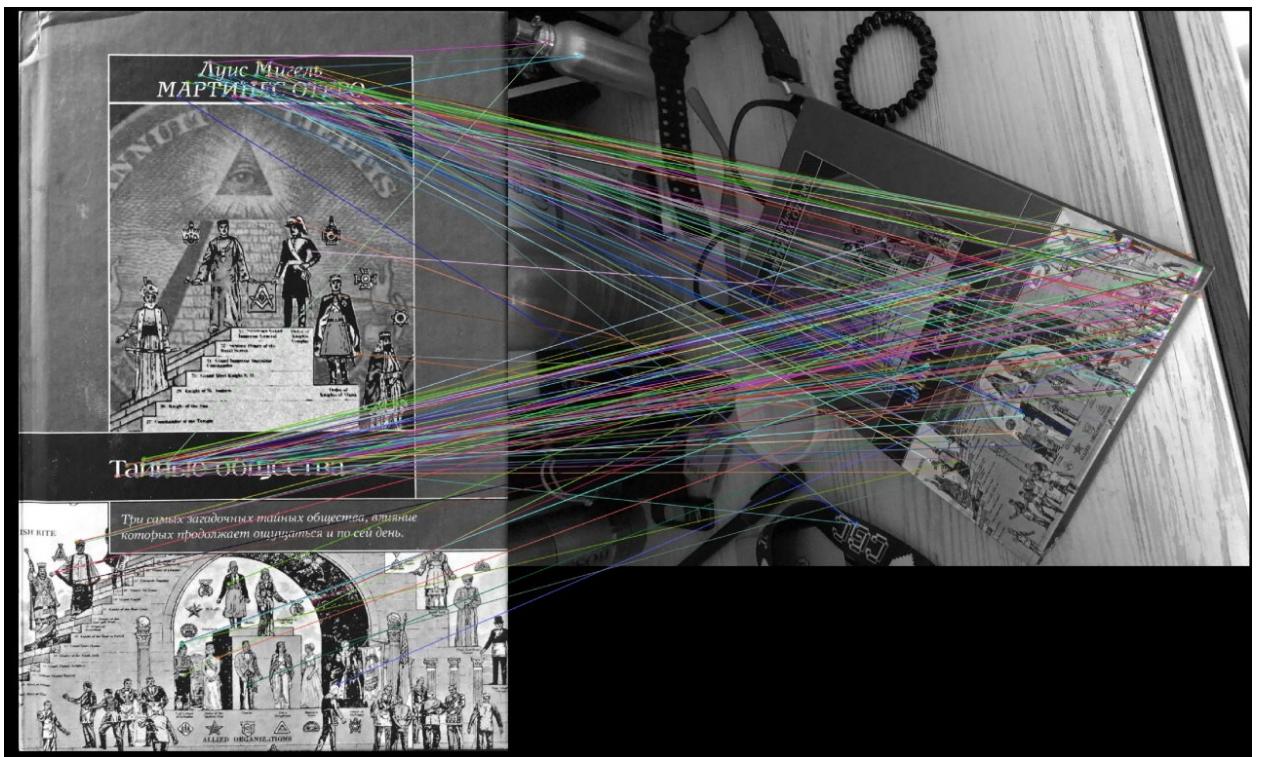
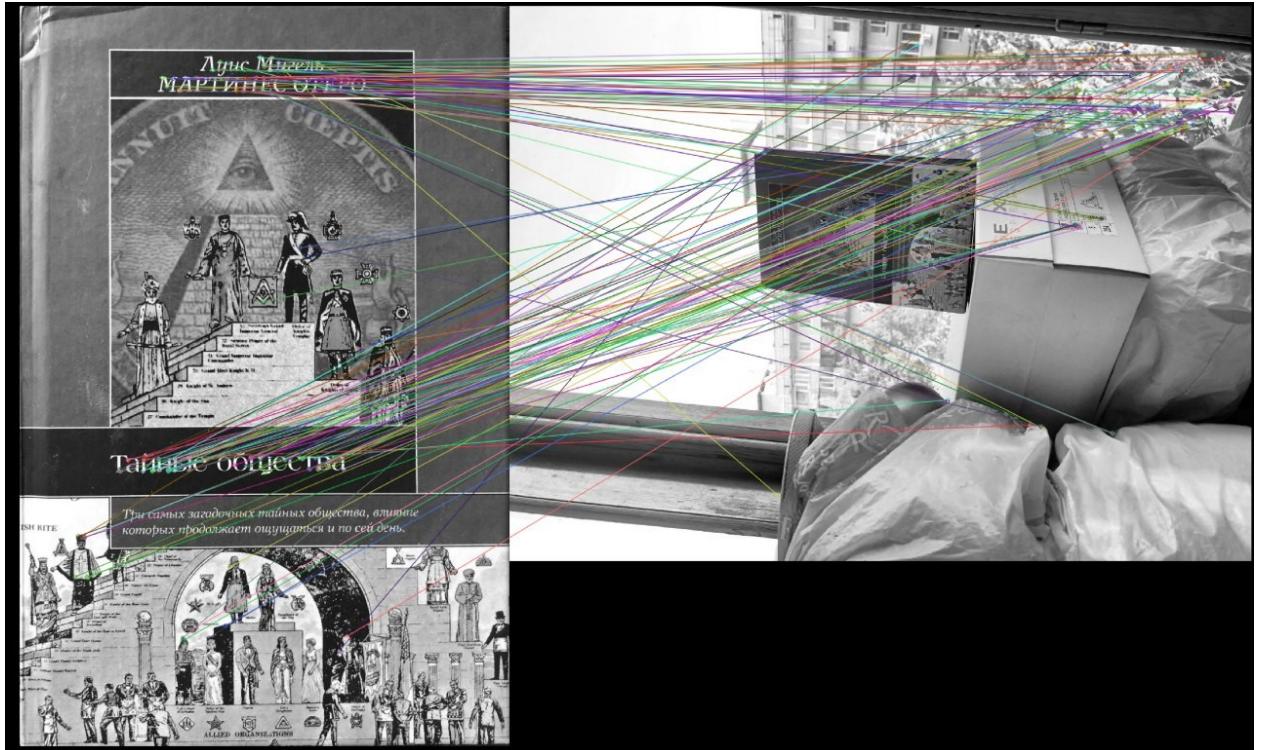
тут SIFT вказує на вазонок декількома метчами

ORB

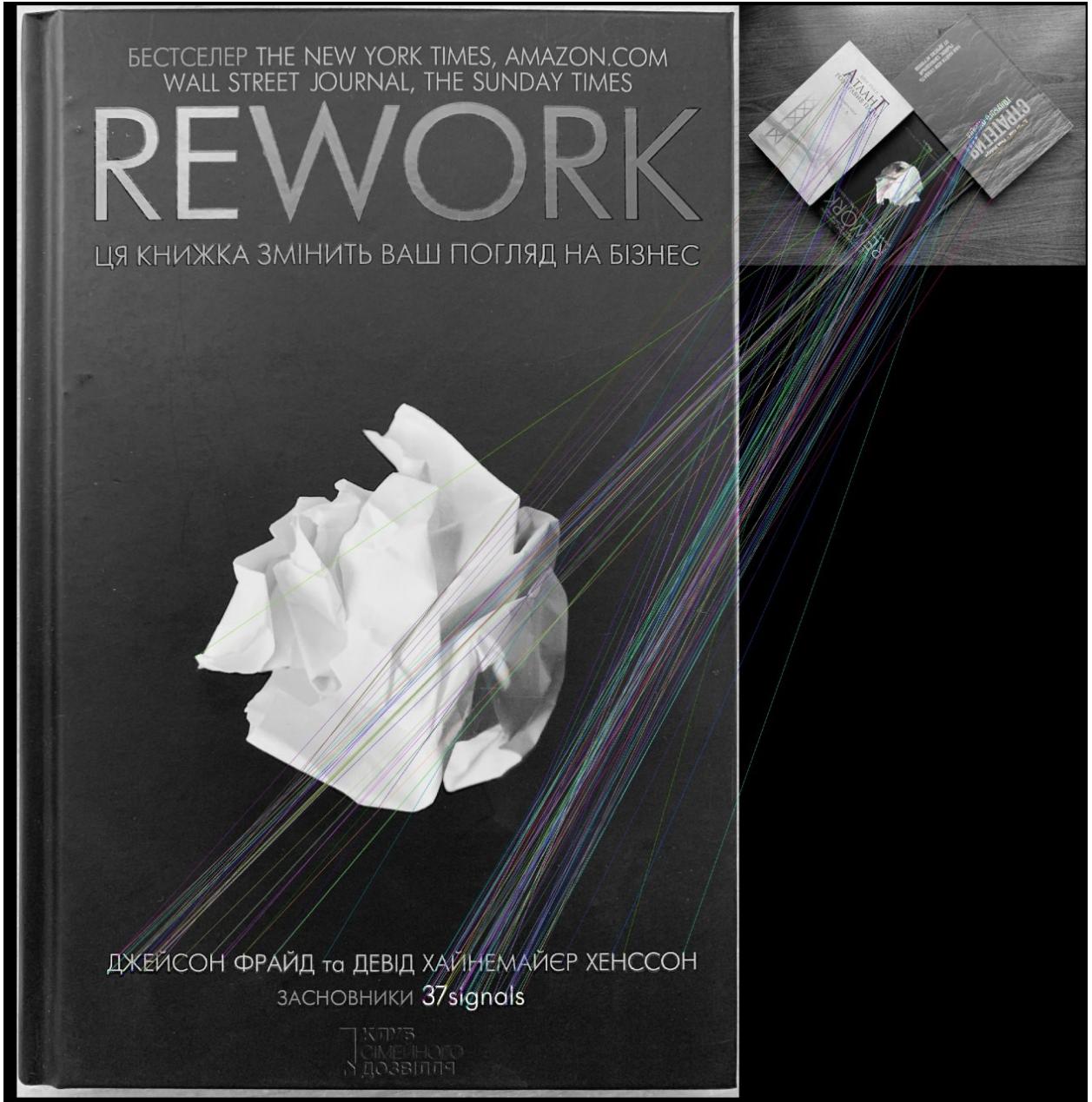


при поворотах, а також при гіршому свіtlі починає працювати погано, і взагалі на датасеті з сірниками показує себе не найкращим чином

Із книжками трохи краща ситуація, але все рівно він дуже сильно вразливий до зміни свіtlа, як ось тут на дуже яскравому фоні результат малий

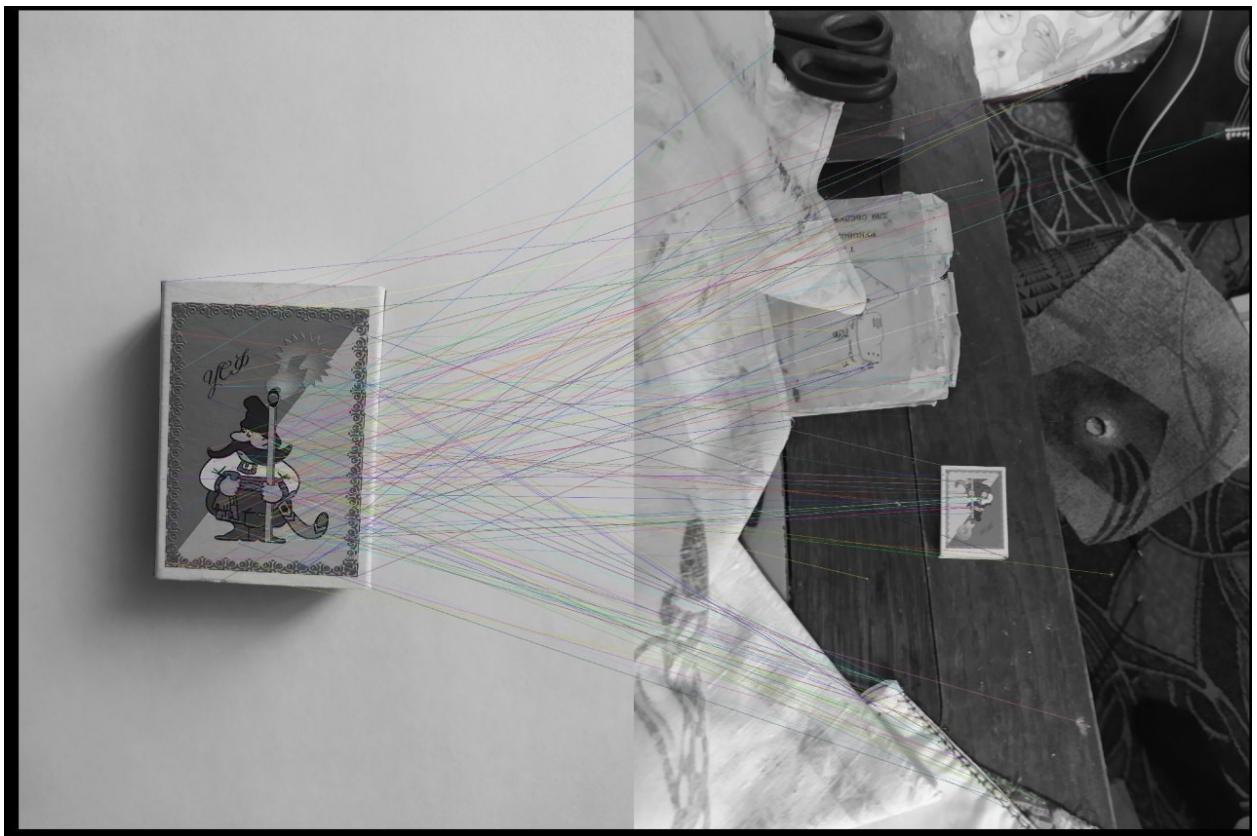


при невеликому повороті та нормальному свіtlі, все ж показує себе непогано

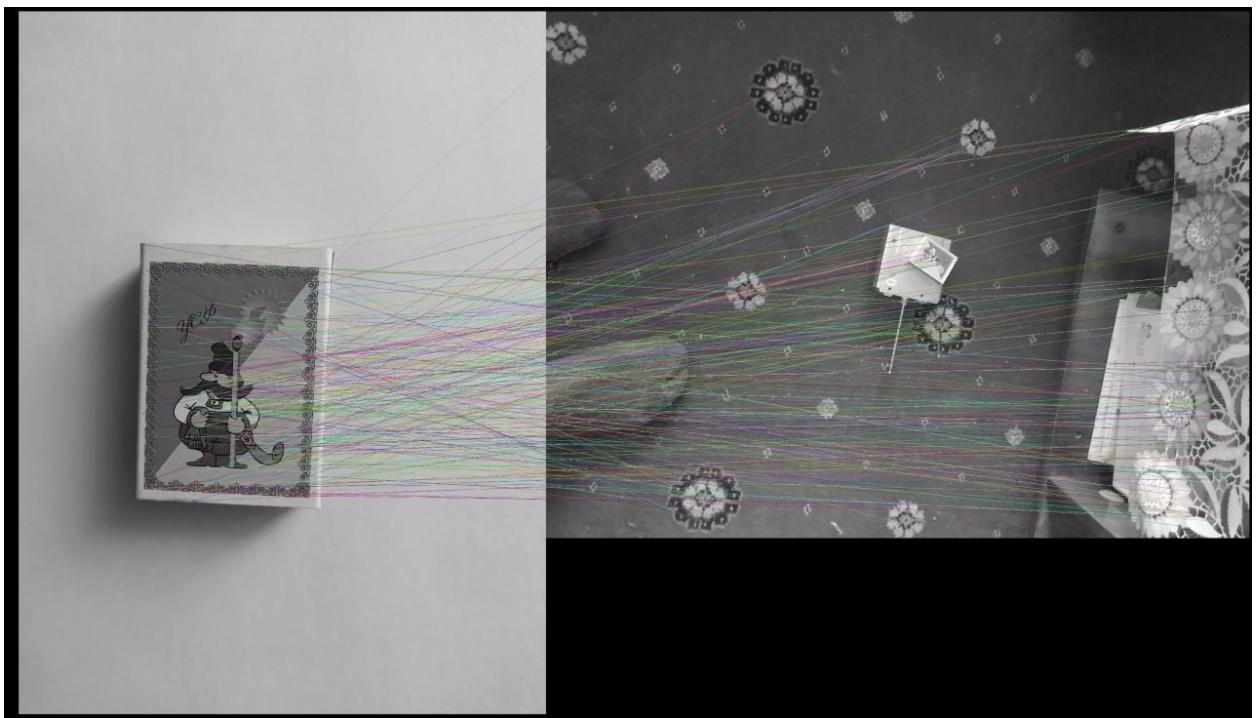


із подібними предметами починає розпізнавати їх на фото(тут літери - основна частка метчів, через це і вказує на подібність)

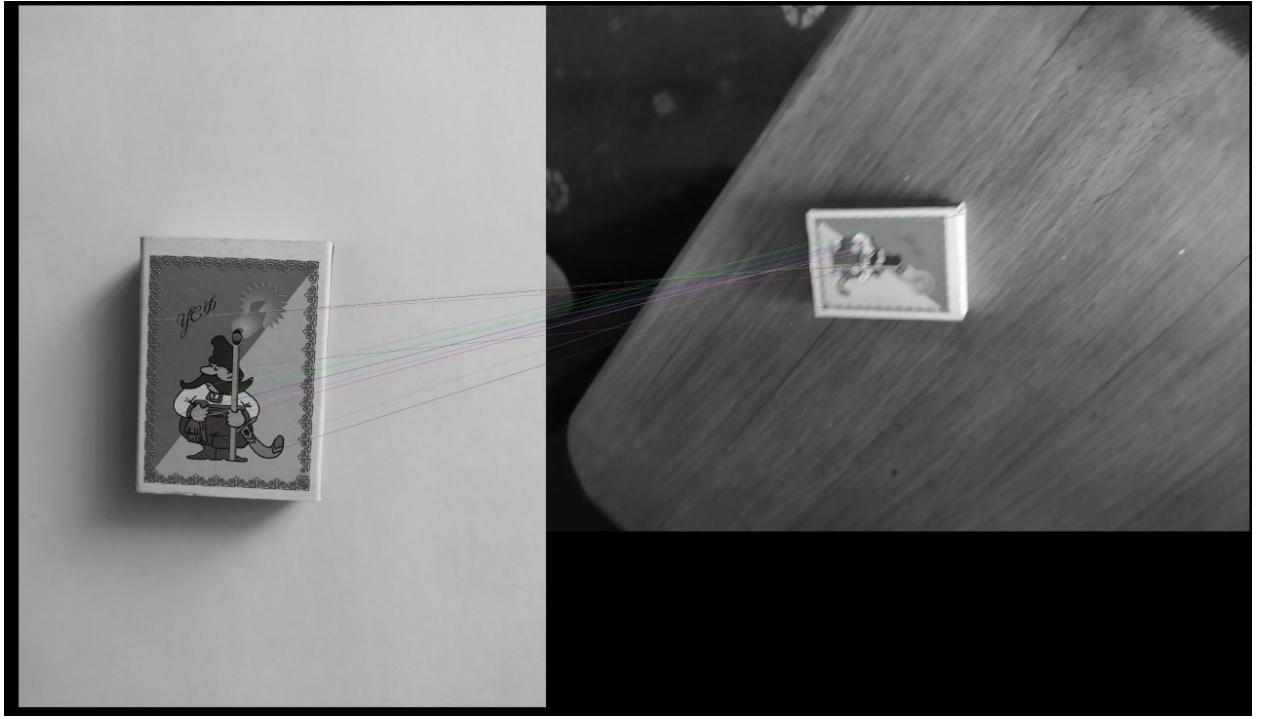
BRIEF



BRIEF загалом справився погано із датасетом із сірниками, особливо на фото де коробка далеко і не в хорошому світлі

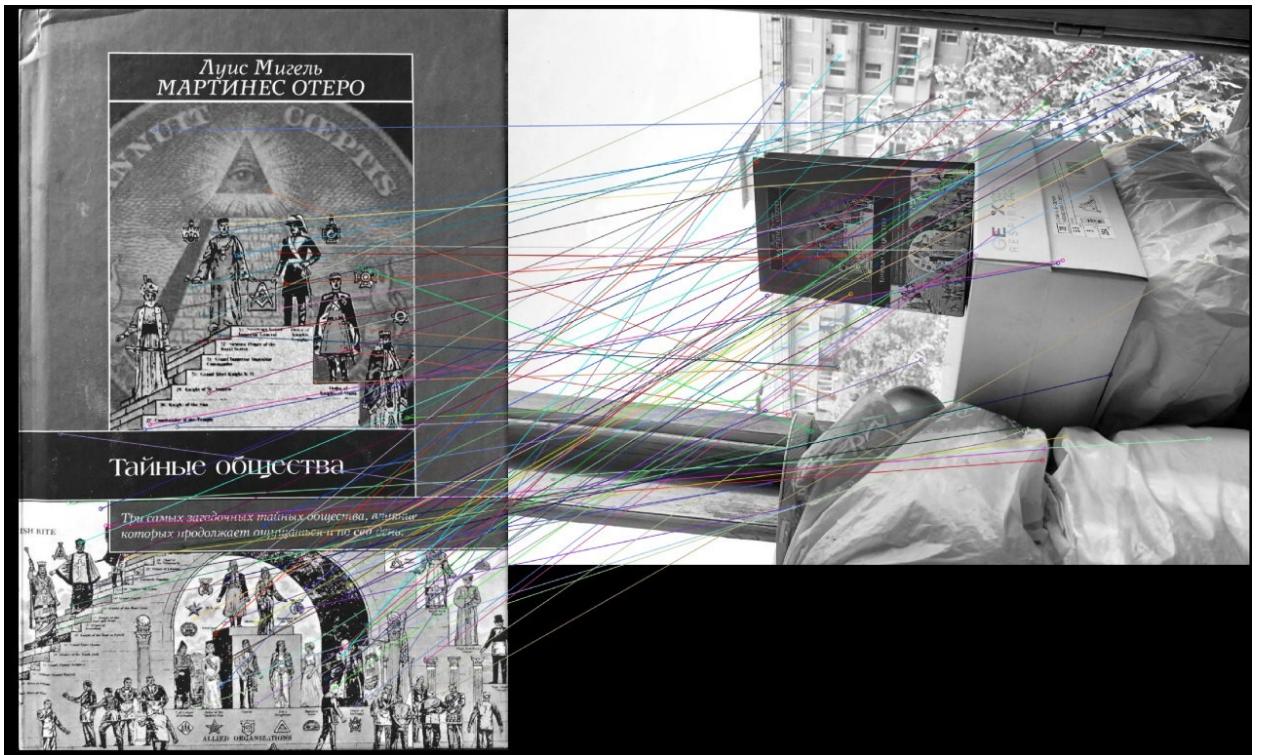


тут, через велику кількість переходів кольору, багато метчів пішли 'в молоко'



але наприклад на цьому розмитому, та відносно великому фото BRIEF правильно розпізнав коробку

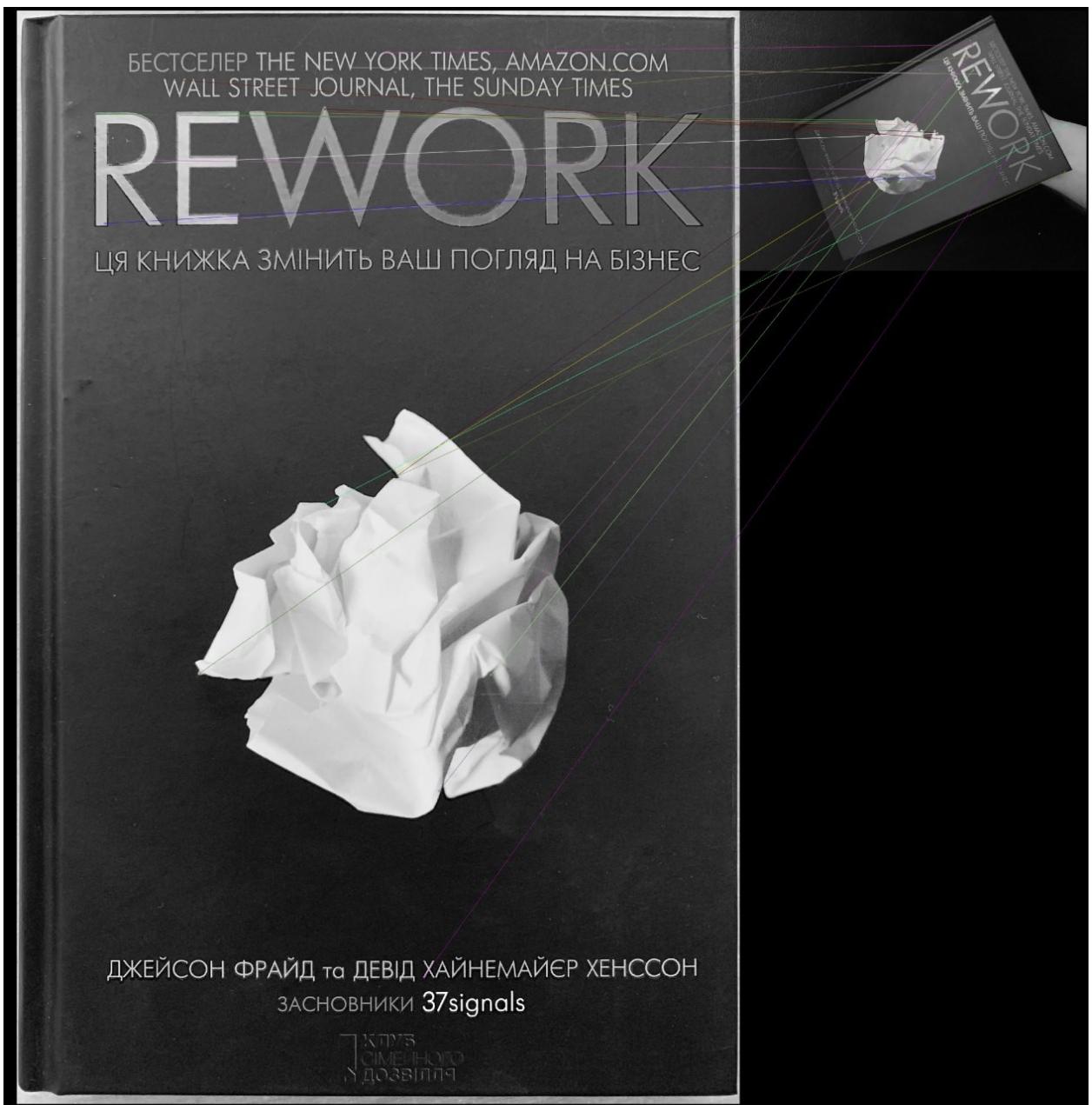
загалом, навіть на датасетах із книгами, що були досить хорошими, BRIEF показав посередні результати через велику кількість поворотів предмету



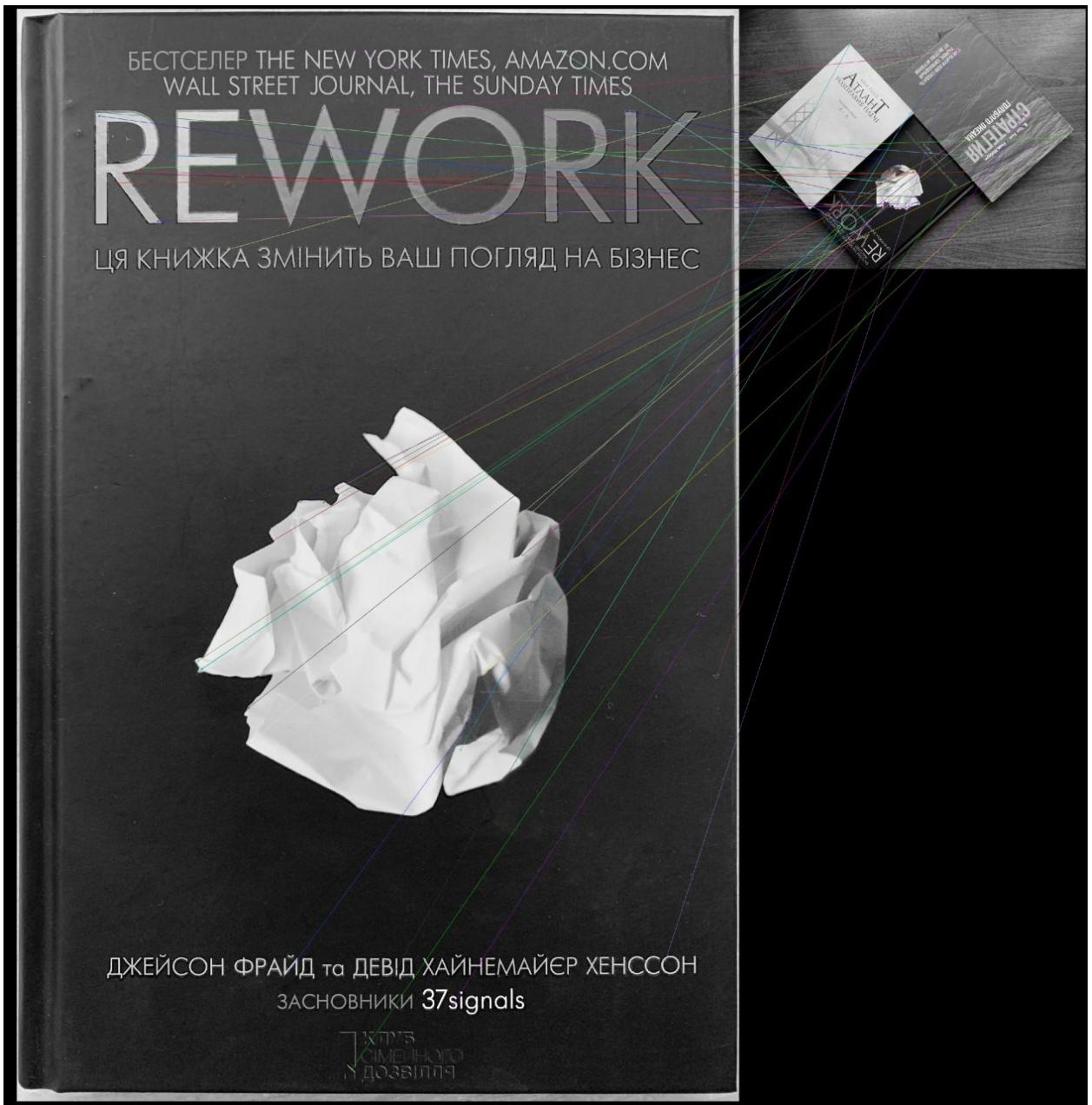
на засвітленій фотографії результати не найкращі



через сильний поворот картинки, тут теж бачимо, що він впорався не найкраще



на простих фото показує непогані результати



із іншими предметами також доволі багато промахів

| localization_error | correct_hit_percents |
|--------------------|----------------------|
| 37.621095941649564 | 0.28767418087910906 |
| 41.04523618895108 | 0.311626746506986 |
| 40.631651693158375 | 0.39578774617067836 |
| 40.31340872374794 | 0.42023082145281737 |
| 39.26230258640518 | 0.1495515848565756 |
| 37.79303482587072 | 0.17378523257824657 |
| 40.09493509798986 | 0.13121597702301038 |
| 37.221061792864134 | 0.2674166020170675 |
| 38.64036746825169 | 0.2181421666863138 |
| 38.2699560025143 | 0.275092936802974 |
| 41.844938271604846 | 0.24910813138147372 |
| 43.20107108081676 | 0.12324123241232413 |
| 40.10889370932737 | 0.30425026399155225 |
| 41.857829010566896 | 0.4579850417949846 |

BRIEF

розділяючи випадкову вибірку конкретних метрик цього дескриптора бачимо його відносно невелику похибку локалізації та відсоток ключових точок, що зметчилися. Це пов'язано із тим, що дескрипторів на фото знаходимо відносно небагато, в кілька, а то і десяток разів менше, ніж SIFT'ом.

| localization_error | correct_hit_percents |
|--------------------|----------------------|
| 189.610790767108 | 0.06456953642384106 |
| 163.2818058549276 | 0.03011847407811411 |
| 230.99802936678358 | 2.4864864864864864 |
| 225.40287470413477 | 0.13765748950069995 |
| 157.34724971776578 | 0.12350563343940828 |
| 185.71825755141384 | 0.07252876687505663 |
| 166.63291983739103 | 0.09196102314250913 |
| 181.50644064849246 | 0.12864077669902912 |
| 198.3336090503041 | 0.4435336976320583 |
| 175.1636938174094 | 0.07809160746260035 |
| 172.8571875752743 | 0.08899548085308809 |
| 150.22449562113468 | 0.06382213812677388 |
| 171.19390446494342 | 0.13303571428571428 |
| 143.53266863249144 | 0.13538184984411425 |
| 178.63622374414203 | 0.14967934035730646 |
| 191.52965933519897 | 0.12434847356664185 |

SIFT

якщо говорити про цей алгоритм, то бачимо, що похибку локалізації він видає високу, а відсоток метчів невисокий. Причина цьому - дуже велика кількість ключових точок на фотографіях, і, як наслідок, більшість із них не знаходять собі "пари", коли шукаємо метчі. Звісно, навіть після метчів дескриптори із

високою похибою локалізації відсіються, тому нехай дані цифри нікого не обманюють.

| localization_error | correct_hit_percents |
|--------------------|----------------------|
| 60.98461538461534 | 0.26 |
| 62.4909090909091 | 0.22 |
| 64.51724137931035 | 0.232 |
| 56.48818897637795 | 0.254 |
| 60.38524590163934 | 0.244 |
| 61.07692307692307 | 0.286 |
| 59.33333333333333 | 0.186 |
| 56.223776223776234 | 0.286 |
| 61.22818791946308 | 0.298 |
| 60.76119402985075 | 0.268 |
| 63.28467153284671 | 0.274 |
| 52.53103448275868 | 0.29 |
| 62.093220338983045 | 0.236 |
| 60.878504672897165 | 0.214 |
| 63.86046511627909 | 0.258 |

ORB

Якщо говорити про цей алгоритм, то його похибка локалізації значно менша, ніж у SIFT`а, але вища ніж у BRIEF`а знову ж таки через кількість ключових точок. Кількість зметчених ключових точок найкраща, і через це результати його роботи краща, ніж у BRIEF

Якщо говорити про швидкодію алгоритмів, то найкращу швидкість показує ORB, обходячи на фотографіях одного розміру інші алгоритми в десятки разів. Алгоритм швидко детектить ключові точки на фото різних розмірів. Далі в швидкодії йде SIFT, в середньому на 20% обходячи BRIEF. Із збільшенням фото, швидкість його роботи починає зростати. BRIEF працював найповільніше.

Проаналізувавши, можемо дійти висновку, що BRIEF і ORB швидші алгоритми. Ми виявили, що SIFT більш масштабно інваріантний, ніж BRIEF і ORB. Тобто, якщо ви збираєтесь використовувати їх для конкретного випадку використання, можна спробувати обидва, щоб побачити, який найкраще відповідає вашим потребам.

Також, ми показали, що ORB - найшвидший алгоритм, в той час як SIFT працює найкраще в більшості сценаріїв. Для особливого випадку, коли кут повороту пропорційний 90 градусам, ORB перевершує SIFT, а на зашумлених зображеннях ORB і SIFT показують майже однакові характеристики.

Results

- <https://github.com/morozov-roman/CVPR>

References

- https://opencv-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_brief/py_brief.html
- https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html
- https://opencv-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html