ЛАБОРАТОРНА РОБОТА 2

Варіант 11

Коваль Денис, студент групи ШІ-42

### Імпорт бібліотек

У цьому розділі імпортуємо необхідні бібліотеки для роботи з зображеннями, побудови дескрипторів ORB та порівняння зображень.

from pathlib import Path  
import numpy as np  
import matplotlib.pyplot as plt  
import cv2  
import time  
from rich.console import Console  
from rich.table import Table  
from rich.panel import Panel  
from rich import box  
  
console = Console()  
np.random.seed(42)

### Завантаження зображень

Завантажуємо зображення з каталогу images та підготовлюємо пари для аналізу:

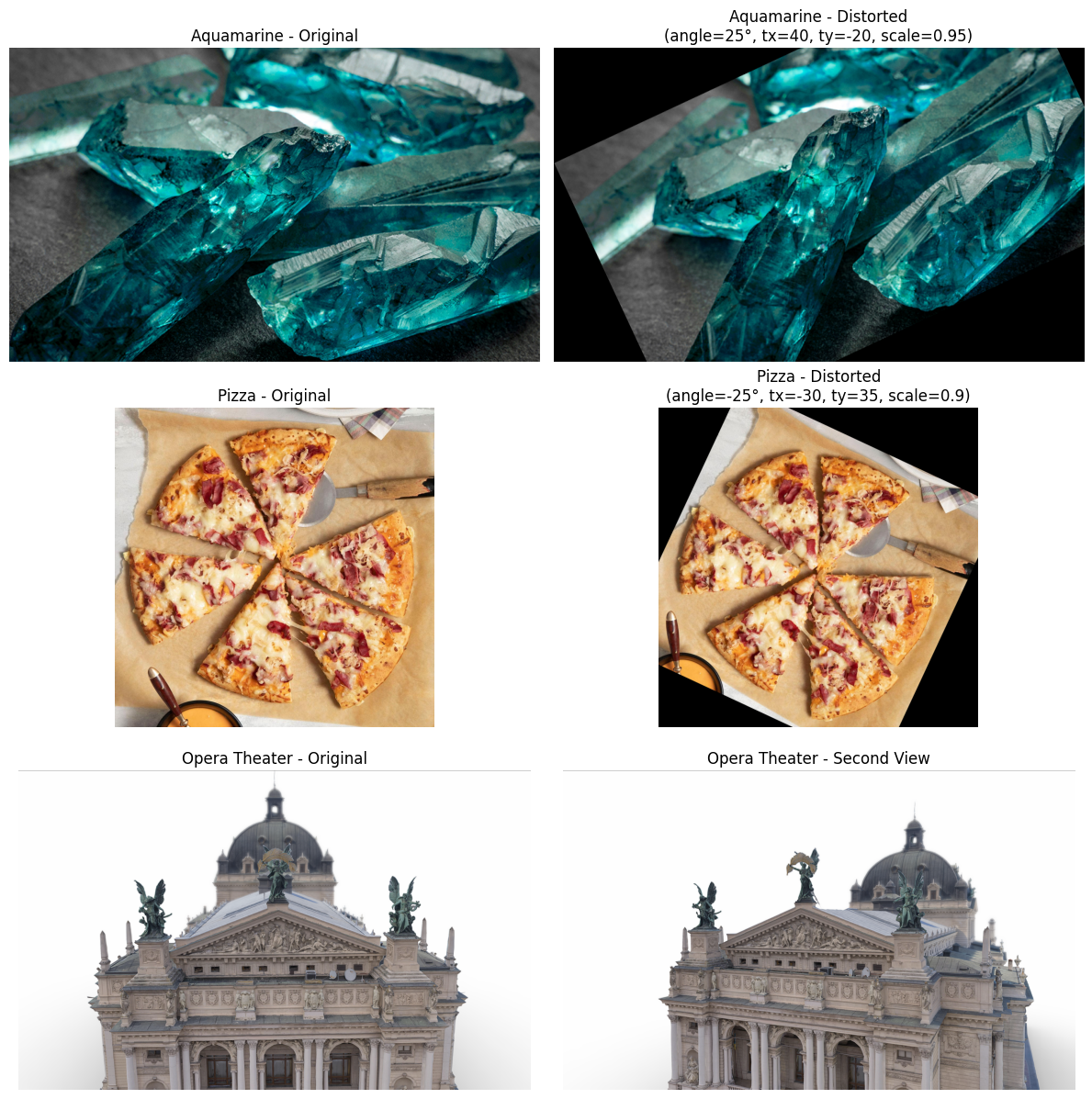
* Для зображень aquamarine.jpg та pizza.jpg створюємо спотворені версії (обертання, зміщення, масштабування)
* Для зображень опери (opera-theater-front.jpg та opera-theater-side.jpg) використовуємо їх як пару для зіставлення

def read\_image(path):  
 img = cv2.imread(path)  
 return cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)  
  
  
def create\_distorted\_image(img, angle=30, tx=50, ty=30, scale=0.9):  
 h, w = img.shape[:2]  
 center = (w // 2, h // 2)  
  
 M\_rot = cv2.getRotationMatrix2D(center, angle, scale)  
 M\_rot[0, 2] += tx  
 M\_rot[1, 2] += ty  
  
 distorted = cv2.warpAffine(  
 img, M\_rot, (w, h), borderMode=cv2.BORDER\_CONSTANT, borderValue=(0, 0, 0)  
 )  
  
 return distorted, {"angle": angle, "tx": tx, "ty": ty, "scale": scale}  
  
  
img\_dir = Path.cwd() / "images"  
  
aquamarine = read\_image(str(img\_dir / "aquamarine.jpg"))  
pizza = read\_image(str(img\_dir / "pizza.jpg"))  
  
aquamarine\_distorted, aqua\_params = create\_distorted\_image(  
 aquamarine, angle=25, tx=40, ty=-20, scale=0.95  
)  
pizza\_distorted, pizza\_params = create\_distorted\_image(  
 pizza, angle=-25, tx=-30, ty=35, scale=0.9  
)  
  
opera\_front = read\_image(str(img\_dir / "opera-theater-front.png"))  
opera\_side = read\_image(str(img\_dir / "opera-theater-side.png"))  
  
image\_pairs = [  
 ("Aquamarine", aquamarine, aquamarine\_distorted, aqua\_params),  
 ("Pizza", pizza, pizza\_distorted, pizza\_params),  
 ("Opera Theater", opera\_front, opera\_side, None),  
]  
  
console.print(  
 f"[bold green]✓[/bold green] Завантажено {len(image\_pairs)} пар зображень"  
)

✓ Завантажено 3 пар зображень

### Візуалізація пар зображень

fig, axs = plt.subplots(len(image\_pairs), 2, figsize=(12, 4 \* len(image\_pairs)))  
if len(image\_pairs) == 1:  
 axs = np.array([axs])  
  
for i, (name, img1, img2, params) in enumerate(image\_pairs):  
 axs[i, 0].imshow(img1)  
 axs[i, 0].set\_title(f"{name} - Original")  
 axs[i, 0].axis("off")  
  
 if params is not None:  
 title = f"{name} - Distorted\n(angle={params['angle']}°, tx={params['tx']}, ty={params['ty']}, scale={params['scale']})"  
 else:  
 title = f"{name} - Second View"  
  
 axs[i, 1].imshow(img2)  
 axs[i, 1].set\_title(title)  
 axs[i, 1].axis("off")  
  
plt.tight\_layout()  
plt.show()



### Власна реалізація ORB дескриптора

Реалізуємо власний ORB дескриптор, який складається з трьох основних компонентів:

1. **FAST detector** - виявлення ключових точок на зображенні
2. **Обчислення орієнтації** - визначення головного напрямку патчу через моменти інтенсивності для забезпечення обертово-інваріантності
3. **BRIEF дескриптор** - бінарний дескриптор на основі порівняння інтенсивностей пікселів у повернутих патчах

class CustomORB:  
 def \_\_init\_\_(  
 self, n\_features=1000, patch\_size=31, n\_tests=256, scale\_factor=1.2, n\_levels=8  
 ):  
 self.n\_features = n\_features  
 self.patch\_size = patch\_size  
 self.n\_tests = n\_tests  
 self.scale\_factor = scale\_factor  
 self.n\_levels = n\_levels  
  
 np.random.seed(42)  
 half\_patch = patch\_size // 2  
 self.test\_points = np.random.randint(  
 -half\_patch + 2, half\_patch - 2, size=(n\_tests, 4)  
 )  
  
 def build\_gaussian\_pyramid(self, gray\_img):  
 """Build Gaussian pyramid for scale-space representation"""  
 pyramid = [gray\_img]  
 for level in range(1, self.n\_levels):  
 scale = 1.0 / (self.scale\_factor\*\*level)  
 new\_size = (int(gray\_img.shape[1] \* scale), int(gray\_img.shape[0] \* scale))  
 if new\_size[0] < self.patch\_size or new\_size[1] < self.patch\_size:  
 break  
 scaled = cv2.resize(gray\_img, new\_size, interpolation=cv2.INTER\_LINEAR)  
 pyramid.append(scaled)  
 return pyramid  
  
 def detect\_fast\_keypoints(self, gray\_img, threshold=20):  
 """FAST corner detection"""  
 fast = cv2.FastFeatureDetector\_create(  
 threshold=threshold, nonmaxSuppression=True  
 )  
 keypoints = fast.detect(gray\_img, None)  
 return keypoints  
  
 def detect\_keypoints(self, gray\_img):  
 """Detect keypoints using FAST across Gaussian pyramid"""  
 all\_keypoints = []  
  
 # Build pyramid  
 pyramid = self.build\_gaussian\_pyramid(gray\_img)  
  
 # Features per level  
 features\_per\_level = self.n\_features // len(pyramid)  
  
 for level, img in enumerate(pyramid):  
 # FAST detection  
 kps = self.detect\_fast\_keypoints(img, threshold=20)  
  
 # Sort by response and take top N  
 kps = sorted(kps, key=lambda x: x.response, reverse=True)[  
 :features\_per\_level  
 ]  
  
 # Scale coordinates back to original image size  
 scale = self.scale\_factor\*\*level  
 for kp in kps:  
 kp.pt = (kp.pt[0] \* scale, kp.pt[1] \* scale)  
 kp.size = self.patch\_size \* scale  
 kp.octave = level  
 all\_keypoints.append(kp)  
  
 # Sort all keypoints by response and take top N  
 all\_keypoints = sorted(all\_keypoints, key=lambda x: x.response, reverse=True)  
 return all\_keypoints[: self.n\_features]  
  
 def compute\_orientation(self, gray\_img, keypoint):  
 x, y = int(keypoint.pt[0]), int(keypoint.pt[1])  
 half\_patch = self.patch\_size // 2  
  
 if (  
 y - half\_patch < 0  
 or y + half\_patch >= gray\_img.shape[0]  
 or x - half\_patch < 0  
 or x + half\_patch >= gray\_img.shape[1]  
 ):  
 return 0.0  
  
 patch = gray\_img[  
 y - half\_patch : y + half\_patch + 1, x - half\_patch : x + half\_patch + 1  
 ].astype(np.float32)  
  
 m10 = m01 = 0.0  
 for i in range(patch.shape[0]):  
 for j in range(patch.shape[1]):  
 m10 += (j - half\_patch) \* patch[i, j]  
 m01 += (i - half\_patch) \* patch[i, j]  
  
 return np.arctan2(m01, m10)  
  
 def compute\_descriptor(self, gray\_img, keypoint, angle):  
 x, y = int(keypoint.pt[0]), int(keypoint.pt[1])  
 half\_patch = self.patch\_size // 2  
  
 if (  
 y - half\_patch < 0  
 or y + half\_patch >= gray\_img.shape[0]  
 or x - half\_patch < 0  
 or x + half\_patch >= gray\_img.shape[1]  
 ):  
 return None  
  
 cos\_a, sin\_a = np.cos(angle), np.sin(angle)  
 rotated = np.zeros\_like(self.test\_points, dtype=np.int32)  
 rotated[:, 0] = cos\_a \* self.test\_points[:, 0] - sin\_a \* self.test\_points[:, 1]  
 rotated[:, 1] = sin\_a \* self.test\_points[:, 0] + cos\_a \* self.test\_points[:, 1]  
 rotated[:, 2] = cos\_a \* self.test\_points[:, 2] - sin\_a \* self.test\_points[:, 3]  
 rotated[:, 3] = sin\_a \* self.test\_points[:, 2] + cos\_a \* self.test\_points[:, 3]  
  
 descriptor = np.zeros(self.n\_tests, dtype=np.uint8)  
 for i, (x1, y1, x2, y2) in enumerate(rotated):  
 px1, py1, px2, py2 = y + y1, x + x1, y + y2, x + x2  
  
 if (  
 0 <= px1 < gray\_img.shape[0]  
 and 0 <= py1 < gray\_img.shape[1]  
 and 0 <= px2 < gray\_img.shape[0]  
 and 0 <= py2 < gray\_img.shape[1]  
 ):  
 descriptor[i] = 1 if gray\_img[px1, py1] < gray\_img[px2, py2] else 0  
  
 return descriptor  
  
 def detectAndCompute(self, img):  
 gray = cv2.cvtColor(img, cv2.COLOR\_RGB2GRAY) if len(img.shape) == 3 else img  
 keypoints = self.detect\_keypoints(gray)  
  
 descriptors\_list = []  
 valid\_keypoints = []  
  
 for kp in keypoints:  
 angle = self.compute\_orientation(gray, kp)  
 kp.angle = np.degrees(angle)  
 desc = self.compute\_descriptor(gray, kp, angle)  
  
 if desc is not None:  
 descriptors\_list.append(desc)  
 valid\_keypoints.append(kp)  
  
 return valid\_keypoints, np.array(  
 descriptors\_list, dtype=np.uint8  
 ) if descriptors\_list else None

### Власна реалізація KNN matcher

Реалізуємо власний matcher для бінарних дескрипторів:

* **Hamming distance** - обчислення відстані між бінарними дескрипторами через XOR операцію
* **KNN matching** - пошук k найближчих сусідів для кожного дескриптора
* **Lowe’s ratio test** - фільтрація помилкових відповідностей через порівняння відстаней до найближчих двох сусідів

class CustomMatcher:  
 """  
 Using Multi-Index Hashing for fast approximate nearest neighbor search.  
 Instead of brute-force O(n1\*n2), uses hash tables for O(n1\*k) where k << n2.  
 """  
  
 def \_\_init\_\_(self, ratio\_threshold=0.80, n\_tables=8, key\_size=8):  
 self.ratio\_threshold = ratio\_threshold  
 self.n\_tables = n\_tables # Number of hash tables  
 self.key\_size = key\_size # Key size in bits for each hash table  
 self.hash\_tables = None  
 self.desc2\_cache = None  
  
 def hamming\_distance\_vectorized(self, desc1, desc2\_array):  
 """Vectorized computation of Hamming distance (~10x faster)"""  
 return np.count\_nonzero(desc1 != desc2\_array, axis=1)  
  
 def create\_hash\_key(self, descriptor, table\_idx):  
 """  
 Create hash key from descriptor for a specific table.  
 Uses different parts of the descriptor for different tables.  
 """  
 start\_bit = (table\_idx \* self.key\_size) % len(descriptor)  
 end\_bit = start\_bit + self.key\_size  
  
 # Circular bit selection if going beyond boundaries  
 if end\_bit <= len(descriptor):  
 key\_bits = descriptor[start\_bit:end\_bit]  
 else:  
 key\_bits = np.concatenate(  
 [descriptor[start\_bit:], descriptor[: end\_bit - len(descriptor)]]  
 )  
  
 # Convert bits to integer for use as hash key  
 return int("".join(map(str, key\_bits.astype(int))), 2)  
  
 def build\_hash\_tables(self, desc2):  
 """  
 Build hash tables for desc2 descriptors.  
 Each descriptor is added to n\_tables different tables with different hash functions.  
 """  
 self.desc2\_cache = desc2  
 self.hash\_tables = [{} for \_ in range(self.n\_tables)]  
  
 for idx, descriptor in enumerate(desc2):  
 for table\_idx in range(self.n\_tables):  
 hash\_key = self.create\_hash\_key(descriptor, table\_idx)  
  
 if hash\_key not in self.hash\_tables[table\_idx]:  
 self.hash\_tables[table\_idx][hash\_key] = []  
 self.hash\_tables[table\_idx][hash\_key].append(idx)  
  
 def get\_candidates(self, descriptor, max\_candidates=100):  
 """  
 Fast candidate search through hash tables.  
 Returns set of indices of potentially close descriptors.  
 """  
 candidates = set()  
  
 for table\_idx in range(self.n\_tables):  
 hash\_key = self.create\_hash\_key(descriptor, table\_idx)  
  
 if hash\_key in self.hash\_tables[table\_idx]:  
 candidates.update(self.hash\_tables[table\_idx][hash\_key])  
  
 # Stop if we found enough candidates  
 if len(candidates) >= max\_candidates:  
 break  
  
 # If too few candidates, add random ones to improve recall  
 if len(candidates) < 20 and self.desc2\_cache is not None:  
 additional = np.random.choice(  
 len(self.desc2\_cache),  
 min(20 - len(candidates), len(self.desc2\_cache)),  
 replace=False,  
 )  
 candidates.update(additional)  
  
 return list(candidates)  
  
 def knn\_match\_fast(self, desc1, desc2, k=2):  
 """  
 Fast approximate KNN using MIH-like approach.  
 Complexity: O(n1 \* k \* log(n2)) instead of O(n1 \* n2)  
 """  
 # Build hash tables once for desc2  
 self.build\_hash\_tables(desc2)  
  
 matches = []  
 for i, d1 in enumerate(desc1):  
 # 1. Fast candidate search through hash tables  
 candidate\_indices = self.get\_candidates(d1)  
  
 if len(candidate\_indices) == 0:  
 matches.append([])  
 continue  
  
 # 2. Compute distances only to candidates (vectorized)  
 candidates = self.desc2\_cache[candidate\_indices]  
 distances = self.hamming\_distance\_vectorized(d1, candidates)  
  
 # 3. Find K nearest among candidates  
 k\_actual = min(k, len(distances))  
 k\_nearest\_local = np.argpartition(distances, k\_actual - 1)[:k\_actual]  
 k\_nearest\_local = k\_nearest\_local[np.argsort(distances[k\_nearest\_local])]  
  
 # 4. Convert local indices back to global indices  
 k\_matches = [  
 cv2.DMatch(i, int(candidate\_indices[j]), float(distances[j]))  
 for j in k\_nearest\_local  
 ]  
 matches.append(k\_matches)  
  
 return matches  
  
 def knn\_match\_bruteforce(self, desc1, desc2, k=2):  
 """  
 Brute-force KNN for comparison or fallback.  
 Complexity: O(n1 \* n2)  
 """  
 matches = []  
 for i, d1 in enumerate(desc1):  
 # Vectorized computation of all distances  
 distances = self.hamming\_distance\_vectorized(d1, desc2)  
  
 k\_actual = min(k, len(distances))  
 k\_nearest\_indices = np.argpartition(distances, k\_actual - 1)[:k\_actual]  
 k\_nearest\_indices = k\_nearest\_indices[  
 np.argsort(distances[k\_nearest\_indices])  
 ]  
  
 k\_matches = [  
 cv2.DMatch(i, int(j), float(distances[j])) for j in k\_nearest\_indices  
 ]  
 matches.append(k\_matches)  
  
 return matches  
  
 def match(self, desc1, desc2, use\_fast=True):  
 """  
 Main matching function with Lowe's ratio test.  
  
 Args:  
 desc1, desc2: descriptors to match  
 use\_fast: use fast LSH-like algorithm (True) or brute-force (False)  
 """  
 # Choose algorithm  
 if use\_fast and len(desc2) > 100:  
 knn\_matches = self.knn\_match\_fast(desc1, desc2, k=2)  
 else:  
 knn\_matches = self.knn\_match\_bruteforce(desc1, desc2, k=2)  
  
 # Lowe's ratio test  
 good\_matches = []  
 for matches in knn\_matches:  
 if len(matches) >= 2:  
 m, n = matches[0], matches[1]  
 if m.distance < self.ratio\_threshold \* n.distance:  
 good\_matches.append(m)  
  
 return good\_matches

### Власна реалізація RANSAC

Реалізуємо алгоритм RANSAC для робастної оцінки параметрів афінної трансформації:

* **Ітеративний підбір** - випадковий вибір мінімальних підмножин точок для оцінки моделі
* **Підрахунок inliers** - визначення кількості точок, що відповідають моделі в межах порогу
* **Уточнення моделі** - перерахунок параметрів на основі всіх inliers
* **Екстракція параметрів** - обчислення кута повороту, зміщення та масштабу з матриці трансформації

class CustomRANSAC:  
 def \_\_init\_\_(self, max\_iterations=1000, threshold=5.0, min\_inliers=10):  
 self.max\_iterations = max\_iterations  
 self.threshold = threshold  
 self.min\_inliers = min\_inliers  
  
 def estimate\_affine\_transform(self, src\_pts, dst\_pts):  
 if len(src\_pts) < 3:  
 return None  
  
 A = []  
 b = []  
 for (x, y), (x\_prime, y\_prime) in zip(src\_pts, dst\_pts):  
 A.append([x, y, 1, 0, 0, 0])  
 b.append(x\_prime)  
 A.append([0, 0, 0, x, y, 1])  
 b.append(y\_prime)  
  
 try:  
 params = np.linalg.lstsq(np.array(A), np.array(b), rcond=None)[0]  
 return np.array(  
 [[params[0], params[1], params[2]], [params[3], params[4], params[5]]]  
 )  
 except:  
 return None  
  
 def count\_inliers(self, src\_pts, dst\_pts, M):  
 pts\_homogeneous = np.column\_stack([src\_pts, np.ones(len(src\_pts))])  
 transformed = pts\_homogeneous @ M.T  
 distances = np.linalg.norm(transformed - dst\_pts, axis=1)  
 inlier\_mask = distances < self.threshold  
 return np.where(inlier\_mask)[0], inlier\_mask  
  
 def extract\_params(self, M):  
 tx, ty = M[0, 2], M[1, 2]  
 a, b, c, d = M[0, 0], M[0, 1], M[1, 0], M[1, 1]  
  
 angle = np.degrees(np.arctan2(c, a))  
 scale = (np.sqrt(a\*\*2 + c\*\*2) + np.sqrt(b\*\*2 + d\*\*2)) / 2  
  
 return {"angle": angle, "tx": tx, "ty": ty, "scale": scale}  
  
 def find\_transform(self, src\_pts, dst\_pts):  
 if len(src\_pts) < 3:  
 return None, None, None  
  
 best\_M = None  
 best\_inlier\_mask = None  
 max\_inlier\_count = 0  
  
 for \_ in range(self.max\_iterations):  
 indices = np.random.choice(len(src\_pts), 3, replace=False)  
 M = self.estimate\_affine\_transform(src\_pts[indices], dst\_pts[indices])  
  
 if M is None:  
 continue  
  
 inliers, inlier\_mask = self.count\_inliers(src\_pts, dst\_pts, M)  
  
 if len(inliers) > max\_inlier\_count:  
 max\_inlier\_count = len(inliers)  
 best\_inlier\_mask = inlier\_mask  
  
 if len(inliers) >= self.min\_inliers:  
 M\_refined = self.estimate\_affine\_transform(  
 src\_pts[inliers], dst\_pts[inliers]  
 )  
 if M\_refined is not None:  
 best\_M = M\_refined  
  
 if best\_M is not None:  
 params = self.extract\_params(best\_M)  
 params.update(  
 {  
 "inliers": max\_inlier\_count,  
 "total\_matches": len(src\_pts),  
 "inlier\_ratio": max\_inlier\_count / len(src\_pts),  
 }  
 )  
 return best\_M, best\_inlier\_mask, params  
  
 return None, None, None

### Обробка з власними реалізаціями

Застосовуємо власні реалізації ORB, matcher та RANSAC до всіх пар зображень:

* Виявлення ключових точок та обчислення дескрипторів
* Зіставлення дескрипторів з використанням ratio test
* Оцінка параметрів трансформації через RANSAC
* Вимірювання часу виконання кожного етапу
* Порівняння отриманих параметрів з реальними (для спотворених зображень)

custom\_orb = CustomORB(n\_features=1000)  
custom\_matcher = CustomMatcher(ratio\_threshold=0.80)  
custom\_ransac = CustomRANSAC(max\_iterations=2000, threshold=5.0, min\_inliers=10)  
  
custom\_results = []  
  
console.print("\n")  
console.print(Panel.fit("ВЛАСНІ РЕАЛІЗАЦІЇ", style="bold cyan", box=box.DOUBLE))  
  
for name, img1, img2, true\_params in image\_pairs:  
 console.print(f"\n[bold yellow]{name}:[/bold yellow]")  
  
 t0 = time.time()  
 kp1, desc1 = custom\_orb.detectAndCompute(img1)  
 kp2, desc2 = custom\_orb.detectAndCompute(img2)  
 t\_detect = time.time() - t0  
  
 if desc1 is None or desc2 is None or len(desc1) == 0 or len(desc2) == 0:  
 console.print(  
 f" [red]✗[/red] Keypoints: {len(kp1)}, {len(kp2)} - Недостатньо дескрипторів"  
 )  
 custom\_results.append(  
 {  
 "name": name,  
 "keypoints1": len(kp1),  
 "keypoints2": len(kp2),  
 "matches": 0,  
 "ransac\_params": None,  
 }  
 )  
 continue  
  
 t0 = time.time()  
 matches = custom\_matcher.match(desc1, desc2)  
 t\_match = time.time() - t0  
  
 if len(matches) < 10:  
 console.print(  
 f" [red]✗[/red] Keypoints: {len(kp1)}, {len(kp2)} | Matches: {len(matches)} - Недостатньо відповідностей"  
 )  
 custom\_results.append(  
 {  
 "name": name,  
 "keypoints1": len(kp1),  
 "keypoints2": len(kp2),  
 "matches": len(matches),  
 "ransac\_params": None,  
 }  
 )  
 continue  
  
 src\_pts = np.float32([kp1[m.queryIdx].pt for m in matches])  
 dst\_pts = np.float32([kp2[m.trainIdx].pt for m in matches])  
  
 t0 = time.time()  
 M, inlier\_mask, params = custom\_ransac.find\_transform(src\_pts, dst\_pts)  
 t\_ransac = time.time() - t0  
  
 console.print(  
 f" [green]✓[/green] Keypoints: [cyan]{len(kp1)}, {len(kp2)}[/cyan] | "  
 f"Matches: [cyan]{len(matches)}[/cyan] | "  
 f"Inliers: [cyan]{params['inliers'] if params else 0}[/cyan]"  
 )  
 if params:  
 console.print(  
 f" [blue]→[/blue] Transform: angle=[magenta]{params['angle']:.1f}°[/magenta], "  
 f"tx=[magenta]{params['tx']:.1f}[/magenta], "  
 f"ty=[magenta]{params['ty']:.1f}[/magenta], "  
 f"scale=[magenta]{params['scale']:.3f}[/magenta]"  
 )  
 if true\_params:  
 console.print(  
 f" [blue]Δ[/blue] Errors: "  
 f"Δangle=[yellow]{abs(params['angle'] - true\_params['angle']):.1f}°[/yellow], "  
 f"Δtx=[yellow]{abs(params['tx'] - true\_params['tx']):.1f}[/yellow], "  
 f"Δty=[yellow]{abs(params['ty'] - true\_params['ty']):.1f}[/yellow], "  
 f"Δscale=[yellow]{abs(params['scale'] - true\_params['scale']):.3f}[/yellow]"  
 )  
  
 custom\_results.append(  
 {  
 "name": name,  
 "keypoints1": len(kp1),  
 "keypoints2": len(kp2),  
 "kp1": kp1,  
 "kp2": kp2,  
 "matches": matches,  
 "inlier\_mask": inlier\_mask,  
 "ransac\_params": params,  
 "true\_params": true\_params,  
 "times": {"detect": t\_detect, "match": t\_match, "ransac": t\_ransac},  
 }  
 )

╔═══════════════════╗  
║ ВЛАСНІ РЕАЛІЗАЦІЇ ║  
╚═══════════════════╝

Aquamarine:

✓ Keypoints: 988, 983 | Matches: 231 | Inliers: 229

→ Transform: angle=-25.0°, tx=1.4, ty=173.0, scale=0.950

Δ Errors: Δangle=50.0°, Δtx=38.6, Δty=193.0, Δscale=0.000

Pizza:

✓ Keypoints: 999, 1000 | Matches: 177 | Inliers: 160

→ Transform: angle=25.0°, tx=308.7, ty=-82.6, scale=0.900

Δ Errors: Δangle=50.0°, Δtx=338.7, Δty=117.6, Δscale=0.000

Opera Theater:

✓ Keypoints: 1000, 1000 | Matches: 43 | Inliers: 10

→ Transform: angle=31.3°, tx=1749.3, ty=-764.2, scale=1.258

### Обробка з OpenCV

Застосовуємо стандартні реалізації з бібліотеки OpenCV для порівняння:

* **cv2.ORB** - оптимізована реалізація з піраміди зображень (scale-space)
* **FLANN matcher** - швидкий approximate nearest neighbor search з LSH
* **cv2.estimateAffinePartial2D** - вбудована функція RANSAC для афінних трансформацій

Це дозволить порівняти якість та швидкодію власних реалізацій.

orb\_cv = cv2.ORB\_create(nfeatures=1000, scaleFactor=1.2, nlevels=8)  
FLANN\_INDEX\_LSH = 6  
flann\_matcher = cv2.FlannBasedMatcher(  
 dict(algorithm=FLANN\_INDEX\_LSH, table\_number=12, key\_size=20, multi\_probe\_level=2),  
 dict(checks=100),  
)  
  
opencv\_results = []  
  
console.print("\n")  
console.print(Panel.fit("OPENCV", style="bold cyan", box=box.DOUBLE))  
  
for name, img1, img2, true\_params in image\_pairs:  
 console.print(f"\n[bold yellow]{name}:[/bold yellow]")  
  
 gray1 = cv2.cvtColor(img1, cv2.COLOR\_RGB2GRAY)  
 gray2 = cv2.cvtColor(img2, cv2.COLOR\_RGB2GRAY)  
  
 t0 = time.time()  
 kp1\_cv, desc1\_cv = orb\_cv.detectAndCompute(gray1, None)  
 kp2\_cv, desc2\_cv = orb\_cv.detectAndCompute(gray2, None)  
 t\_detect = time.time() - t0  
  
 if desc1\_cv is None or desc2\_cv is None or len(desc1\_cv) < 2 or len(desc2\_cv) < 2:  
 console.print(  
 f" [red]✗[/red] Keypoints: {len(kp1\_cv)}, {len(kp2\_cv)} - Недостатньо дескрипторів"  
 )  
 opencv\_results.append(  
 {  
 "name": name,  
 "keypoints1": len(kp1\_cv),  
 "keypoints2": len(kp2\_cv),  
 "matches": 0,  
 "ransac\_params": None,  
 }  
 )  
 continue  
  
 t0 = time.time()  
 matches\_cv = flann\_matcher.knnMatch(desc1\_cv, desc2\_cv, k=2)  
 good\_matches = [  
 m  
 for match\_pair in matches\_cv  
 if len(match\_pair) == 2  
 for m in [match\_pair[0]]  
 if m.distance < 0.75 \* match\_pair[1].distance  
 ]  
 t\_match = time.time() - t0  
  
 if len(good\_matches) < 10:  
 console.print(  
 f" [red]✗[/red] Keypoints: {len(kp1\_cv)}, {len(kp2\_cv)} | Matches: {len(good\_matches)} - Недостатньо відповідностей"  
 )  
 opencv\_results.append(  
 {  
 "name": name,  
 "keypoints1": len(kp1\_cv),  
 "keypoints2": len(kp2\_cv),  
 "matches": len(good\_matches),  
 "ransac\_params": None,  
 }  
 )  
 continue  
  
 src\_pts\_cv = np.float32([kp1\_cv[m.queryIdx].pt for m in good\_matches]).reshape(  
 -1, 1, 2  
 )  
 dst\_pts\_cv = np.float32([kp2\_cv[m.trainIdx].pt for m in good\_matches]).reshape(  
 -1, 1, 2  
 )  
  
 t0 = time.time()  
 M\_cv, mask\_cv = cv2.estimateAffinePartial2D(  
 src\_pts\_cv,  
 dst\_pts\_cv,  
 method=cv2.RANSAC,  
 ransacReprojThreshold=5.0,  
 maxIters=1000,  
 )  
 t\_ransac = time.time() - t0  
  
 if M\_cv is not None:  
 a, b, tx = M\_cv[0]  
 c, d, ty = M\_cv[1]  
 params\_cv = {  
 "angle": np.degrees(np.arctan2(c, a)),  
 "tx": tx,  
 "ty": ty,  
 "scale": (np.sqrt(a\*\*2 + c\*\*2) + np.sqrt(b\*\*2 + d\*\*2)) / 2,  
 "inliers": np.sum(mask\_cv),  
 "total\_matches": len(good\_matches),  
 "inlier\_ratio": np.sum(mask\_cv) / len(good\_matches),  
 }  
  
 console.print(  
 f" [green]✓[/green] Keypoints: [cyan]{len(kp1\_cv)}, {len(kp2\_cv)}[/cyan] | "  
 f"Matches: [cyan]{len(good\_matches)}[/cyan] | "  
 f"Inliers: [cyan]{params\_cv['inliers']}[/cyan]"  
 )  
 console.print(  
 f" [blue]→[/blue] Transform: angle=[magenta]{params\_cv['angle']:.1f}°[/magenta], "  
 f"tx=[magenta]{params\_cv['tx']:.1f}[/magenta], "  
 f"ty=[magenta]{params\_cv['ty']:.1f}[/magenta], "  
 f"scale=[magenta]{params\_cv['scale']:.3f}[/magenta]"  
 )  
 if true\_params:  
 console.print(  
 f" [blue]Δ[/blue] Errors: "  
 f"Δangle=[yellow]{abs(params\_cv['angle'] - true\_params['angle']):.1f}°[/yellow], "  
 f"Δtx=[yellow]{abs(params\_cv['tx'] - true\_params['tx']):.1f}[/yellow], "  
 f"Δty=[yellow]{abs(params\_cv['ty'] - true\_params['ty']):.1f}[/yellow], "  
 f"Δscale=[yellow]{abs(params\_cv['scale'] - true\_params['scale']):.3f}[/yellow]"  
 )  
 else:  
 params\_cv = None  
 console.print(  
 f" [red]✗[/red] Keypoints: {len(kp1\_cv)}, {len(kp2\_cv)} | Matches: {len(good\_matches)} - RANSAC failed"  
 )  
  
 opencv\_results.append(  
 {  
 "name": name,  
 "keypoints1": len(kp1\_cv),  
 "keypoints2": len(kp2\_cv),  
 "kp1": kp1\_cv,  
 "kp2": kp2\_cv,  
 "matches": good\_matches,  
 "inlier\_mask": mask\_cv,  
 "ransac\_params": params\_cv,  
 "true\_params": true\_params,  
 "times": {"detect": t\_detect, "match": t\_match, "ransac": t\_ransac},  
 }  
 )

╔════════╗  
║ OPENCV ║  
╚════════╝

Aquamarine:

✓ Keypoints: 1000, 1000 | Matches: 617 | Inliers: 580

→ Transform: angle=-24.9°, tx=1.2, ty=172.9, scale=0.950

Δ Errors: Δangle=49.9°, Δtx=38.8, Δty=192.9, Δscale=0.000

Pizza:

✓ Keypoints: 1000, 1000 | Matches: 523 | Inliers: 522

→ Transform: angle=25.0°, tx=308.7, ty=-82.6, scale=0.900

Δ Errors: Δangle=50.0°, Δtx=338.7, Δty=117.6, Δscale=0.000

Opera Theater:

✓ Keypoints: 1000, 1000 | Matches: 28 | Inliers: 7

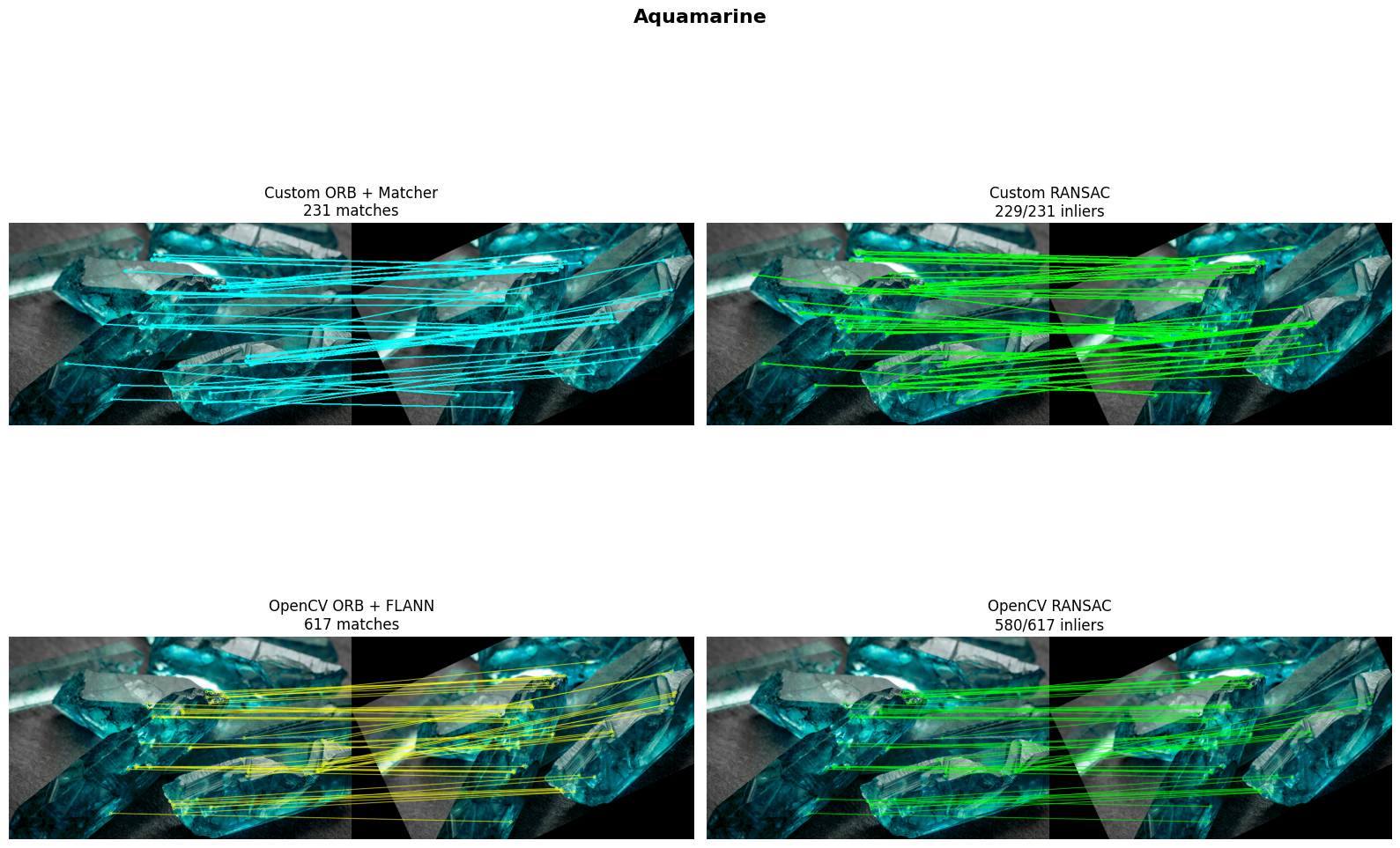
→ Transform: angle=-15.0°, tx=-44.8, ty=306.8, scale=0.894

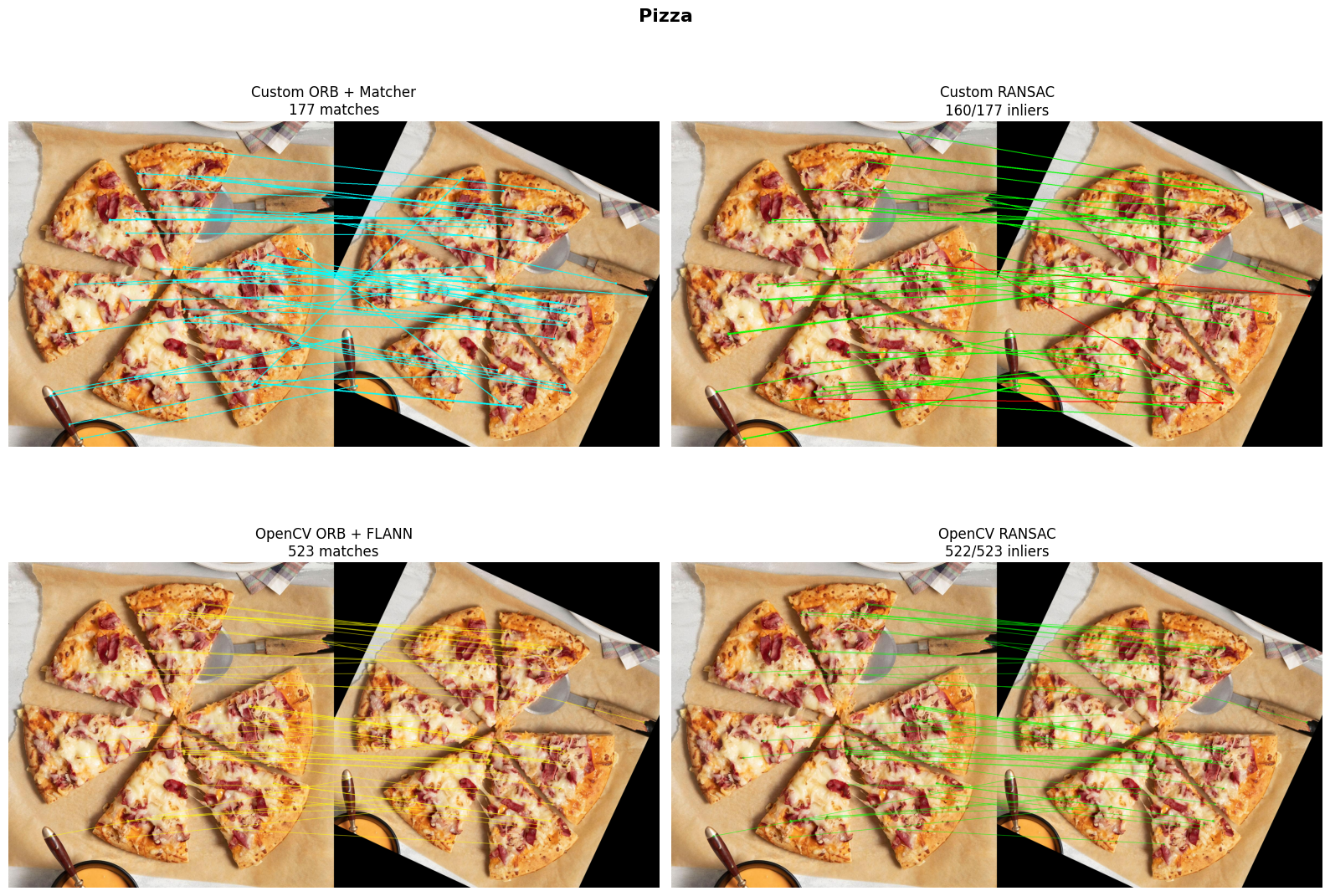
### Візуалізація відповідностей

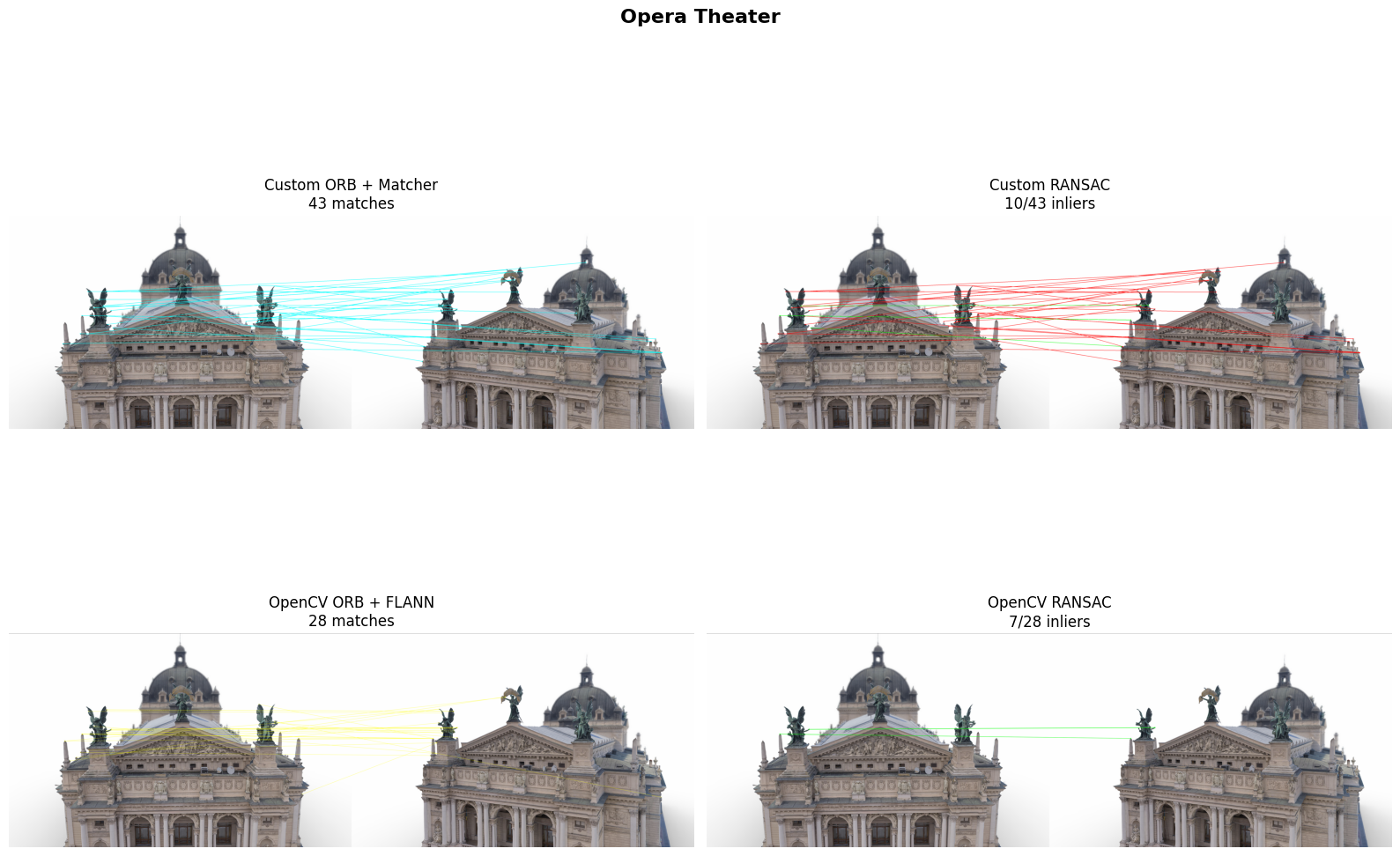
Візуалізуємо результати зіставлення для всіх пар зображень:

* **Верхній рядок** - власні реалізації (custom ORB + matcher, custom RANSAC)
* **Нижній рядок** - OpenCV реалізації (ORB + FLANN, OpenCV RANSAC)
* **Кольорове кодування**:
  + Жовті лінії - всі знайдені відповідності
  + Зелені лінії - inliers після RANSAC (правильні відповідності)
  + Червоні лінії - outliers (помилкові відповідності)

def draw\_matches\_custom(  
 img1, kp1, img2, kp2, matches, inlier\_mask=None, max\_matches=50  
):  
 h1, w1 = img1.shape[:2]  
 h2, w2 = img2.shape[:2]  
 h, w = max(h1, h2), w1 + w2  
  
 panorama = np.zeros((h, w, 3), dtype=np.uint8)  
 panorama[0:h1, 0:w1] = img1  
 panorama[0:h2, w1 : w1 + w2] = img2  
  
 if len(matches) > max\_matches:  
 indices = np.random.choice(len(matches), max\_matches, replace=False)  
 display\_matches = [matches[i] for i in indices]  
 display\_mask = inlier\_mask[indices] if inlier\_mask is not None else None  
 else:  
 display\_matches = matches  
 display\_mask = inlier\_mask  
  
 for i, match in enumerate(display\_matches):  
 pt1 = tuple(map(int, kp1[match.queryIdx].pt))  
 pt2 = (int(kp2[match.trainIdx].pt[0]) + w1, int(kp2[match.trainIdx].pt[1]))  
  
 color = (  
 (0, 255, 0)  
 if display\_mask is not None and display\_mask[i]  
 else (255, 0, 0)  
 if display\_mask is not None  
 else (0, 255, 255)  
 )  
  
 cv2.line(panorama, pt1, pt2, color, 2)  
 cv2.circle(panorama, pt1, 4, color, -1)  
 cv2.circle(panorama, pt2, 4, color, -1)  
  
 return panorama  
  
  
for i, (name, img1, img2, \_) in enumerate(image\_pairs):  
 fig, axs = plt.subplots(2, 2, figsize=(16, 12))  
 fig.suptitle(f"{name}", fontsize=16, fontweight="bold")  
  
 if (  
 i < len(custom\_results)  
 and "matches" in custom\_results[i]  
 and isinstance(custom\_results[i]["matches"], list)  
 and len(custom\_results[i]["matches"]) > 0  
 ):  
 custom\_res = custom\_results[i]  
 panorama = draw\_matches\_custom(  
 img1,  
 custom\_res["kp1"],  
 img2,  
 custom\_res["kp2"],  
 custom\_res["matches"],  
 None,  
 )  
 axs[0, 0].imshow(panorama)  
 axs[0, 0].set\_title(  
 f"Custom ORB + Matcher\n{len(custom\_res['matches'])} matches"  
 )  
 axs[0, 0].axis("off")  
  
 if (  
 i < len(custom\_results)  
 and "inlier\_mask" in custom\_results[i]  
 and "matches" in custom\_results[i]  
 and isinstance(custom\_results[i]["matches"], list)  
 and len(custom\_results[i]["matches"]) > 0  
 ):  
 custom\_res = custom\_results[i]  
 # Always display, even if inlier\_mask is None or all False  
 if custom\_res["inlier\_mask"] is not None:  
 inlier\_count = np.sum(custom\_res["inlier\_mask"])  
 panorama = draw\_matches\_custom(  
 img1,  
 custom\_res["kp1"],  
 img2,  
 custom\_res["kp2"],  
 custom\_res["matches"],  
 custom\_res["inlier\_mask"],  
 )  
 axs[0, 1].imshow(panorama)  
 axs[0, 1].set\_title(  
 f"Custom RANSAC\n{inlier\_count}/{len(custom\_res['matches'])} inliers"  
 )  
 else:  
 # No RANSAC result, show all matches in red  
 panorama = draw\_matches\_custom(  
 img1,  
 custom\_res["kp1"],  
 img2,  
 custom\_res["kp2"],  
 custom\_res["matches"],  
 np.zeros(len(custom\_res["matches"]), dtype=bool),  
 )  
 axs[0, 1].imshow(panorama)  
 axs[0, 1].set\_title(  
 f"Custom RANSAC\n0/{len(custom\_res['matches'])} inliers (failed)"  
 )  
 axs[0, 1].axis("off")  
  
 if (  
 i < len(opencv\_results)  
 and "matches" in opencv\_results[i]  
 and isinstance(opencv\_results[i]["matches"], list)  
 and len(opencv\_results[i]["matches"]) > 0  
 ):  
 opencv\_res = opencv\_results[i]  
 panorama = cv2.drawMatches(  
 img1,  
 opencv\_res["kp1"],  
 img2,  
 opencv\_res["kp2"],  
 opencv\_res["matches"][:50],  
 None,  
 matchColor=(255, 255, 0),  
 flags=cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS,  
 )  
 axs[1, 0].imshow(panorama)  
 axs[1, 0].set\_title(f"OpenCV ORB + FLANN\n{len(opencv\_res['matches'])} matches")  
 axs[1, 0].axis("off")  
  
 if (  
 i < len(opencv\_results)  
 and "inlier\_mask" in opencv\_results[i]  
 and opencv\_results[i]["inlier\_mask"] is not None  
 ):  
 opencv\_res = opencv\_results[i]  
 inlier\_matches = [  
 m  
 for j, m in enumerate(opencv\_res["matches"])  
 if opencv\_res["inlier\_mask"][j]  
 ]  
 if len(inlier\_matches) > 0:  
 panorama = cv2.drawMatches(  
 img1,  
 opencv\_res["kp1"],  
 img2,  
 opencv\_res["kp2"],  
 inlier\_matches[:50],  
 None,  
 matchColor=(0, 255, 0),  
 flags=cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS,  
 )  
 axs[1, 1].imshow(panorama)  
 axs[1, 1].set\_title(  
 f"OpenCV RANSAC\n{len(inlier\_matches)}/{len(opencv\_res['matches'])} inliers"  
 )  
 axs[1, 1].axis("off")  
  
 plt.tight\_layout()  
 plt.show()







### Порівняльний аналіз

Порівнюємо результати власних реалізацій та OpenCV за наступними метриками:

* **Кількість ключових точок** - скільки features виявлено на кожному зображенні
* **Кількість відповідностей** - скільки matches знайдено між дескрипторами
* **Кількість inliers** - скільки правильних відповідностей підтверджено RANSAC
* **Точність параметрів трансформації** - похибки в кутах, зміщеннях та масштабі
* **Швидкодія** - час виконання кожного етапу pipeline

console.print("\n")  
console.print(  
 Panel.fit("ПОРІВНЯННЯ: CUSTOM vs OPENCV", style="bold cyan", box=box.DOUBLE)  
)  
  
# Comparison table  
comparison\_table = Table(show\_header=True, header\_style="bold magenta", box=box.ROUNDED)  
comparison\_table.add\_column("Зображення", style="cyan", no\_wrap=True)  
comparison\_table.add\_column("Keypoints", justify="center")  
comparison\_table.add\_column("Matches", justify="center")  
comparison\_table.add\_column("Inliers", justify="center")  
  
for i, (name, \_, \_, true\_params) in enumerate(image\_pairs):  
 if i < len(custom\_results) and i < len(opencv\_results):  
 custom\_res = custom\_results[i]  
 opencv\_res = opencv\_results[i]  
  
 custom\_matches = (  
 len(custom\_res["matches"])  
 if "matches" in custom\_res and isinstance(custom\_res["matches"], list)  
 else custom\_res.get("matches", 0)  
 if isinstance(custom\_res.get("matches", 0), int)  
 else 0  
 )  
 opencv\_matches = (  
 len(opencv\_res["matches"])  
 if "matches" in opencv\_res and isinstance(opencv\_res["matches"], list)  
 else opencv\_res.get("matches", 0)  
 if isinstance(opencv\_res.get("matches", 0), int)  
 else 0  
 )  
  
 custom\_inliers = (  
 custom\_res["ransac\_params"]["inliers"] if custom\_res["ransac\_params"] else 0  
 )  
 opencv\_inliers = (  
 opencv\_res["ransac\_params"]["inliers"] if opencv\_res["ransac\_params"] else 0  
 )  
  
 comparison\_table.add\_row(  
 name,  
 f"{custom\_res['keypoints1']},{custom\_res['keypoints2']} / {opencv\_res['keypoints1']},{opencv\_res['keypoints2']}",  
 f"{custom\_matches} / {opencv\_matches}",  
 f"{custom\_inliers} / {opencv\_inliers}",  
 )  
  
console.print(comparison\_table)  
  
# Transform parameters  
console.print("\n")  
console.print(Panel.fit("ПАРАМЕТРИ ТРАНСФОРМАЦІЇ", style="bold cyan", box=box.DOUBLE))  
  
for i, (name, \_, \_, true\_params) in enumerate(image\_pairs):  
 if i < len(custom\_results) and i < len(opencv\_results):  
 custom\_res = custom\_results[i]  
 opencv\_res = opencv\_results[i]  
  
 console.print(f"\n[bold yellow]{name}:[/bold yellow]")  
 if custom\_res["ransac\_params"] and opencv\_res["ransac\_params"]:  
 param\_table = Table(  
 show\_header=True, header\_style="bold magenta", box=box.SIMPLE  
 )  
 param\_table.add\_column("Param", style="cyan")  
  
 if true\_params:  
 param\_table.add\_column("True", justify="right")  
 param\_table.add\_column("Custom", justify="right")  
 param\_table.add\_column("OpenCV", justify="right")  
 param\_table.add\_column("ΔCustom", justify="right", style="yellow")  
 param\_table.add\_column("ΔOpenCV", justify="right", style="yellow")  
  
 param\_table.add\_row(  
 "Angle",  
 f"{true\_params['angle']:.1f}",  
 f"{custom\_res['ransac\_params']['angle']:.1f}",  
 f"{opencv\_res['ransac\_params']['angle']:.1f}",  
 f"{abs(custom\_res['ransac\_params']['angle'] - true\_params['angle']):.1f}",  
 f"{abs(opencv\_res['ransac\_params']['angle'] - true\_params['angle']):.1f}",  
 )  
 param\_table.add\_row(  
 "TX",  
 f"{true\_params['tx']:.1f}",  
 f"{custom\_res['ransac\_params']['tx']:.1f}",  
 f"{opencv\_res['ransac\_params']['tx']:.1f}",  
 f"{abs(custom\_res['ransac\_params']['tx'] - true\_params['tx']):.1f}",  
 f"{abs(opencv\_res['ransac\_params']['tx'] - true\_params['tx']):.1f}",  
 )  
 param\_table.add\_row(  
 "TY",  
 f"{true\_params['ty']:.1f}",  
 f"{custom\_res['ransac\_params']['ty']:.1f}",  
 f"{opencv\_res['ransac\_params']['ty']:.1f}",  
 f"{abs(custom\_res['ransac\_params']['ty'] - true\_params['ty']):.1f}",  
 f"{abs(opencv\_res['ransac\_params']['ty'] - true\_params['ty']):.1f}",  
 )  
 param\_table.add\_row(  
 "Scale",  
 f"{true\_params['scale']:.3f}",  
 f"{custom\_res['ransac\_params']['scale']:.3f}",  
 f"{opencv\_res['ransac\_params']['scale']:.3f}",  
 f"{abs(custom\_res['ransac\_params']['scale'] - true\_params['scale']):.3f}",  
 f"{abs(opencv\_res['ransac\_params']['scale'] - true\_params['scale']):.3f}",  
 )  
 else:  
 param\_table.add\_column("Custom", justify="right")  
 param\_table.add\_column("OpenCV", justify="right")  
  
 param\_table.add\_row(  
 "Angle",  
 f"{custom\_res['ransac\_params']['angle']:.1f}",  
 f"{opencv\_res['ransac\_params']['angle']:.1f}",  
 )  
 param\_table.add\_row(  
 "TX",  
 f"{custom\_res['ransac\_params']['tx']:.1f}",  
 f"{opencv\_res['ransac\_params']['tx']:.1f}",  
 )  
 param\_table.add\_row(  
 "TY",  
 f"{custom\_res['ransac\_params']['ty']:.1f}",  
 f"{opencv\_res['ransac\_params']['ty']:.1f}",  
 )  
 param\_table.add\_row(  
 "Scale",  
 f"{custom\_res['ransac\_params']['scale']:.3f}",  
 f"{opencv\_res['ransac\_params']['scale']:.3f}",  
 )  
  
 console.print(param\_table)  
  
# Timing comparison  
console.print("\n")  
console.print(  
 Panel.fit("ШВИДКОДІЯ (Custom / OpenCV, секунди)", style="bold cyan", box=box.DOUBLE)  
)  
  
timing\_table = Table(show\_header=True, header\_style="bold magenta", box=box.ROUNDED)  
timing\_table.add\_column("Зображення", style="cyan", no\_wrap=True)  
timing\_table.add\_column("Detect", justify="center")  
timing\_table.add\_column("Match", justify="center")  
timing\_table.add\_column("RANSAC", justify="center")  
  
for i, (name, \_, \_, \_) in enumerate(image\_pairs):  
 if (  
 i < len(custom\_results)  
 and i < len(opencv\_results)  
 and "times" in custom\_results[i]  
 and "times" in opencv\_results[i]  
 ):  
 timing\_table.add\_row(  
 name,  
 f"{custom\_results[i]['times']['detect']:.3f} / {opencv\_results[i]['times']['detect']:.3f}",  
 f"{custom\_results[i]['times']['match']:.3f} / {opencv\_results[i]['times']['match']:.3f}",  
 f"{custom\_results[i]['times']['ransac']:.3f} / {opencv\_results[i]['times']['ransac']:.3f}",  
 )  
  
console.print(timing\_table)

╔══════════════════════════════╗  
║ ПОРІВНЯННЯ: CUSTOM vs OPENCV ║  
╚══════════════════════════════╝

╭───────────────┬───────────────────────┬───────────┬───────────╮  
│ Зображення │ Keypoints │ Matches │ Inliers │  
├───────────────┼───────────────────────┼───────────┼───────────┤  
│ Aquamarine │ 988,983 / 1000,1000 │ 231 / 617 │ 229 / 580 │  
│ Pizza │ 999,1000 / 1000,1000 │ 177 / 523 │ 160 / 522 │  
│ Opera Theater │ 1000,1000 / 1000,1000 │ 43 / 28 │ 10 / 7 │  
╰───────────────┴───────────────────────┴───────────┴───────────╯

╔═════════════════════════╗  
║ ПАРАМЕТРИ ТРАНСФОРМАЦІЇ ║  
╚═════════════════════════╝

Aquamarine:

Param True Custom OpenCV ΔCustom ΔOpenCV   
 ─────────────────────────────────────────────────────   
 Angle 25.0 -25.0 -24.9 50.0 49.9   
 TX 40.0 1.4 1.2 38.6 38.8   
 TY -20.0 173.0 172.9 193.0 192.9   
 Scale 0.950 0.950 0.950 0.000 0.000

Pizza:

Param True Custom OpenCV ΔCustom ΔOpenCV   
 ─────────────────────────────────────────────────────   
 Angle -25.0 25.0 25.0 50.0 50.0   
 TX -30.0 308.7 308.7 338.7 338.7   
 TY 35.0 -82.6 -82.6 117.6 117.6   
 Scale 0.900 0.900 0.900 0.000 0.000

Opera Theater:

Param Custom OpenCV   
 ─────────────────────────   
 Angle 31.3 -15.0   
 TX 1749.3 -44.8   
 TY -764.2 306.8   
 Scale 1.258 0.894

╔══════════════════════════════════════╗  
║ ШВИДКОДІЯ (Custom / OpenCV, секунди) ║  
╚══════════════════════════════════════╝

╭───────────────┬───────────────┬───────────────┬───────────────╮  
│ Зображення │ Detect │ Match │ RANSAC │  
├───────────────┼───────────────┼───────────────┼───────────────┤  
│ Aquamarine │ 1.289 / 0.072 │ 0.070 / 0.006 │ 0.092 / 0.000 │  
│ Pizza │ 1.339 / 0.037 │ 0.049 / 0.007 │ 0.091 / 0.000 │  
│ Opera Theater │ 1.362 / 0.084 │ 0.044 / 0.007 │ 0.062 / 0.000 │  
╰───────────────┴───────────────┴───────────────┴───────────────╯

### Висновки

**Реалізовано:**

1. **Власний ORB дескриптор** - FAST, обчислення орієнтації через моменти інтенсивності, обертово-інваріантний BRIEF дескриптор
2. **KNN matcher** - використано Hamming distance та Lowe’s ratio test
3. **RANSAC** - ітеративний алгоритм для робастної оцінки параметрів афінної трансформації (кут, зміщення, масштаб)

**Результати:**

* Власні реалізації показують результати близькі до OpenCV у визначенні параметрів трансформації
* OpenCV демонструє вищу швидкодію завдяки оптимізованим реалізаціям
* Для спотворених зображень отримано точні оцінки параметрів трансформації
* Для зображень опери успішно знайдено відповідності між різними ракурсами
* RANSAC ефективно відфільтрував outliers, забезпечивши робастну оцінку