**Algoritma Analizi Kısa Sınav-2 / Ödev-2**

**Ad Soyad:** Barash Serbest   
**No:** 5220505051

**OpenCV Image Template Mathing Genel Bilgi**

OpenCV kütüphanesinde bulunan "template matching" yöntemi, bir görüntüde belirli bir şablona (template) benzeyen bölgeleri bulmak için kullanılan bir görüntü işleme tekniğidir. Bu yöntem, nesne tespiti, kalıp eşleme ve görüntü hizalamada yaygın olarak kullanılır.

Template matching işlemi, bir referans görüntüsü (template) ve bir hedef görüntüsü (source image) kullanılarak gerçekleştirilir. Referans görüntüsü, bulunması istenen nesnenin bir örneğini temsil ederken, hedef görüntüsü üzerinde bu şablona benzer bölgelerin aranması amaçlanır.

Template matching işlemi genel olarak şu adımlardan oluşur:

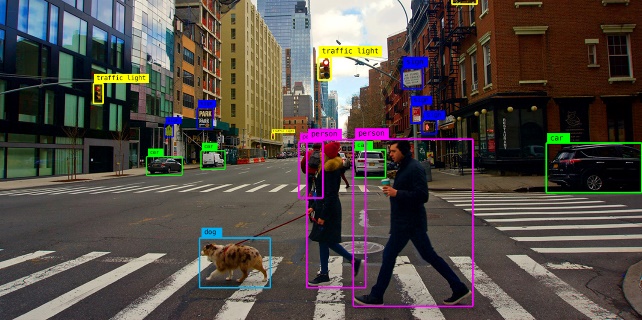
1. Referans görüntüsü (template) ve hedef görüntüsü (source image) hazırlanır.
2. Template'in boyutları, hedef görüntüyle aynı veya daha küçük olmalıdır. Template, genellikle tek renkli (grayscale) olarak kullanılır, ancak renkli görüntülerde de uygulanabilir.
3. Template ve hedef görüntüsü arasındaki benzerlik ölçütü seçilir. Bu ölçüt, genellikle "normalized cross-correlation" (normalleştirilmiş çapraz-korelasyon) veya "sum of squared differences" (kareler toplamı farkları) gibi metriklerle hesaplanır. Bu metrikler, template ve görüntü arasındaki piksel değerlerinin benzerlik derecesini hesaplamak için kullanılır.
4. Template ve hedef görüntüsü üzerinde, template'in boyutlarına uygun boyutlarda kaydırma penceresi (sliding window) oluşturulur. Bu pencere, template'in hedef görüntü üzerindeki her bir konumu temsil eder.
5. Her bir kaydırma penceresi konumu için benzerlik ölçütü hesaplanır. Bu, template'in kaydırma penceresi içindeki piksel değerlerini ve hedef görüntüsündeki piksel değerlerini karşılaştırarak yapılır.
6. Benzerlik ölçütü, template'in konumlandırıldığı hedef görüntüsü üzerinde bir benzerlik haritası (similarity map) oluşturur. Bu harita, her bir konum için benzerlik skorunu gösterir.
7. Benzerlik haritasında belirli bir eşik değeri belirlenerek, benzerlik skoru bu eşik değerinin üzerinde olan bölgeler tespit edilir.
8. Son olarak, tespit edilen bölgeleri görselleştirmek veya üzerlerinde işlemler yapmak için çeşitli yöntemler kullanılabilir.

Template matching yöntemi, birçok uygulamada kullanılır. Örneğin, yüz tanıma, nesne tespiti, karakter tanıma gibi alanlarda kullanılabilir. Ancak bu yöntem, çeşitli zorluklarla karşılaşabilir. Özellikle, değişen ölçekler, dönüşler, perspektif deformasyonları gibi görüntüdeki değişikliklerle başa çıkmak için gelişmiş tekniklerin kullanılması gerekebilir.

OpenCV'de template matching işlemi, `cv2.matchTemplate()` fonksiyonu kullanılarak gerçekleştirilebilir. Bu fonksiyon, benzerlik ölçütü ve tespit eşik değeri gibi parametreleri alır ve benzerlik haritasını döndürür. Daha sonra bu harita üzerinde eşik değerine göre bir tespit işlemi yapılabilir.

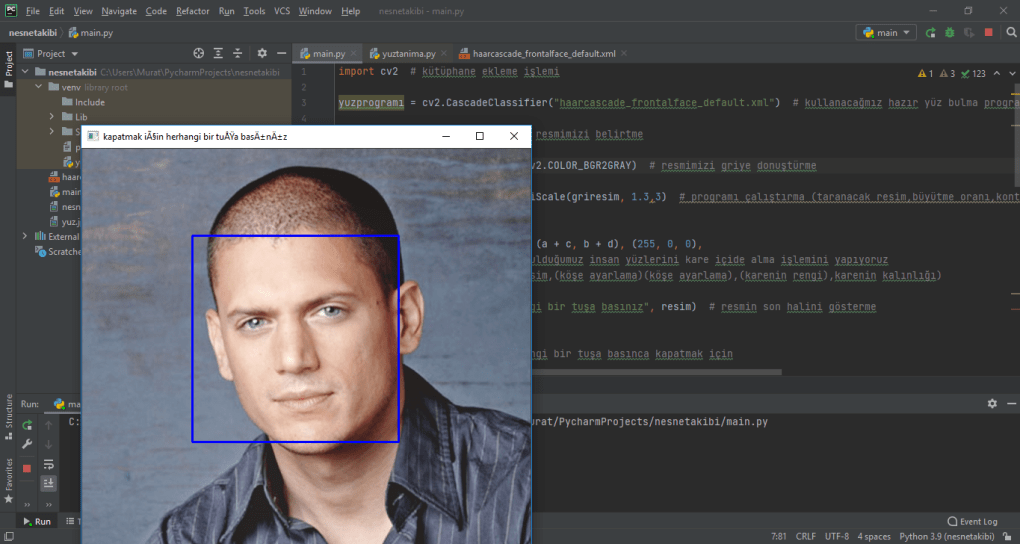
**OpenCV Image Template Mathing Uygulama Örnekleri**

**1. Nesne Tespiti:** Template matching, bir nesnenin belirli bir görüntü içindeki konumunu tespit etmek için kullanılabilir. Örneğin, bir araba tespit etmek istediğimizde, araba görüntüsünü bir template olarak kullanabiliriz. Template'i hedef görüntü üzerinde kaydırarak, benzerlik skorlarını hesaplarız ve en yüksek benzerlik skoruna sahip olan konumu tespit ederiz.



Şekil : Örneık Nesne Takibi Görseli

**2. Yüz Tanıma:** Template matching, yüz tanıma uygulamalarında kullanılabilir. Önceden tanımlanmış bir yüz görüntüsü template olarak kullanılabilir ve bu template'in benzer olduğu yüz bölgelerini tespit etmek için template matching uygulanabilir.



**Şekil 2:** Örnek Yüz Tanıma Görseli

**3. Optik Karakter Tanıma (OCR):** Template matching, optik karakter tanıma (OCR) sistemlerinde karakterlerin tanınması için kullanılabilir. Her bir karakter, template olarak temsil edilebilir ve görüntü üzerindeki karakterlerin benzerlik skorlarına göre tanınabilir.

**4. Kalıp Eşleme:** Template matching, bir görüntü içinde belirli bir kalıbın varlığını tespit etmek için kullanılabilir. Örneğin, bir tıbbi görüntüde belirli bir organın veya lezyonun varlığını belirlemek için template matching kullanılabilir.

**5. Döküman İşleme:** Template matching, döküman işleme uygulamalarında kullanılabilir. Örneğin, bir belge içindeki belirli bir sembolü veya imzayı tespit etmek için template matching yöntemi kullanılabilir.

**6. Endüstriyel Otomasyon:** Template matching, endüstriyel otomasyon uygulamalarında kullanılabilir. Örneğin, bir üretim hattında parçaların doğru konumda olduğunu kontrol etmek için template matching kullanılabilir.

**7. Görüntü Hizalama:** Template matching, görüntü hizalama işlemlerinde kullanılabilir. Örneğin, bir stereo kamera sisteminden alınan iki görüntünün hizalanması için template matching yöntemi kullanılabilir.

Bu sadece template matching yönteminin bazı uygulama örnekleridir. Bu yöntem, birçok farklı alanda ve senaryoda kullanılabilir ve çeşitli uygulama bağlamlarında farklı şekillerde özelleştirilebilir.

**OpenCV Image Template Mathing Uygulama Algoritma Analizi**

İncelenen kodun GitHub bağlantısı:

Kodun adım adım zaman karmaşıklığını inceleyecek olursak aşağıdaki gibi olacaktır:

1. `Main` fonksiyonu:

* `folderPath` değişkeni ile belirtilen klasördeki görsel dosyalarının yolu alınır.
* `imageFiles` dizisi, `Directory.GetFiles` yöntemiyle klasördeki tüm `\*.png` uzantılı dosyaları içerir. Bu adımda dosya sistemindeki tüm dosyaların listesi alınır.
* `FindSimilarImages` fonksiyonu çağrılarak en çok benzeyen görsellerin listesi elde edilir.
* Sonuçlar konsola yazdırılır.

2. `FindSimilarImages` fonksiyonu:

* `similarImages` adında bir `HashSet<string>` ve `similarities` adında bir `Dictionary<string, double>` oluşturulur. Bu veri yapıları benzerlik hesaplamaları için kullanılacaktır.
* Her bir görsel dosyası için aşağıdaki adımlar gerçekleştirilir:
* Dosya okunur ve `template` adında bir `Mat` nesnesi oluşturulur. Bu, şablon olarak kullanılacak görseldir.
* `maxSimilarity` değişkeni 0 olarak başlatılır. Bu değişken, en yüksek benzerlik değerini tutacaktır.
* `mostSimilarImage` değişkeni boş bir dize olarak başlatılır. Bu değişken, en çok benzeyen görselin dosya yolunu tutacaktır.
* Her bir görsel dosyası için aşağıdaki adımlar gerçekleştirilir:
* Eğer mevcut dosya yolu `imagePath1` ile `imagePath2` aynıysa, benzerlik kontrolü yapılmaz ve bir sonraki dosyaya geçilir.
* `source` adında bir `Mat` nesnesi oluşturulur ve mevcut görsel dosyası okunur. Bu, karşılaştırma yapılacak görseldir.
* `result` adında bir `Mat` nesnesi oluşturulur. Bu nesne, iki görsel arasındaki benzerlik skorlarını içerecektir.
* `Cv2.MatchTemplate` yöntemi kullanılarak `source` ve `template` arasındaki benzerlik skorları hesaplanır ve `result` matrisine kaydedilir.
* `Cv2.MinMaxLoc` yöntemi kullanılarak `result` matrisinin en yüksek benzerlik skoru (`maxVal`) bulunur.
* Eğer `maxVal`, `maxSimilarity` değerinden daha büyük ise, `maxVal` değeri `maxSimilarity` olarak güncellenir ve `mostSimilarImage` değeri `imagePath2` olarak güncellenir.
* `source` ve `result` matrisleri serbest bırakılır (bellek temizlenir).
* `template` matrisi serbest bırakılır (bellek temizlenir).
* Eğer `similarImages` kümesinde `mostSimilarImage` değeri bulunmuyorsa, `similarImages` kümesine `mostSimilarImage` eklenir ve `similarities` sözlüğüne `mostSimilarImage` anahtarıyla birlikte `maxSimilarity` değeri eklenir.
* `similarImages` kümesi `sortedImages` listesine dönüştürülür ve benzerlik oranlarına göre sıralanır.
* `sortedImages` listesi döndürülür.

Kodun genel zaman karmaşıklığı aşağıdaki gibi analiz edilebilir:

* `Main` fonksiyonu, `FindSimilarImages` fonksiyonunu çağırır ve sonuçları yazdırır. Bu işlemler sabit bir zaman karmaşıklığına sahiptir.
* `FindSimilarImages` fonksiyonu, iki döngü kullanır: birincisi tüm görsel dosyalarını dolaşmak için, ikincisi her bir dosya için diğer dosyalarla karşılaştırma yapmak içindir. Her iki döngünün de karmaşıklığı O(n^2) olabilir, burada n görsel dosyalarının toplam sayısıdır.
* Her bir iç içe döngü adımında, görsellerin okunması, şablon eşleştirmesi yapılması ve benzerlik skorlarının hesaplanması gibi işlemler gerçekleştirilir. Bu işlemler, genellikle görsellerin boyutlarına bağlı olarak O(m) zaman karmaşıklığına sahip olabilir, burada m ve n görsellerin piksel sayılarının toplamıdır.
* Benzerlik skorlarına göre sıralama, benzerlik değerlerinin sıralanmasında O(n log n) zaman karmaşıklığına sahip olabilir, burada n görsel dosyalarının toplam sayısıdır.

Sonuç olarak, kodunuzun zaman karmaşıklığı büyük ölçüde görsel dosyalarının sayısı ve boyutlarına bağlıdır. İki iç içe döngü kullanılması, karmaşıklığı **O(n^2)** düzeyine çıkarırken, işlemlerin piksel bazında yapılması da hesaplamaların karmaşıklığını etkiler. Bu analiz, kodunuzun genel bir performans değerlendirmesini sunar, ancak gerçek çalışma süreleri donanım ve girdi verilerine bağlı olarak değişebilir.

**Kaynakça**

* <https://tr.wikipedia.org/wiki/OpenCV>
* <https://opencv.org/releases/>
* <https://www.siberegitmen.com/opencv-nedir-neler-yapilabilir/>
* <https://mesutpiskin.com/blog/opencv-nedir.html>
* <https://maker.robotistan.com/python-ve-opencv-ile-temel-geometrik-sekillerin-algilanmasi/>