Belge Tarayıcı
Document Detection

Barış Onur Uzgur
Elektronik Mühendisliği Bölümü, Gebze Teknik Üniversitesi
barisonur.uzgur2016gtu.edu.tr

***Özetçe*—Bu projede belirli algoritmalar kullanılarak bir belge tarayıcı oluşturulmuştur. Çevrimiçi eğitimin vazgeçilmez bir parçası haline gelen çevrimiçi sınav, ödev ve proje teslimleri belge tarayıcılar kullanılarak dijital ortama aktarılmaktadır. Bu programlar sayesinde ödev fotoğraflarından belge otomatik olarak kırpılıp, iyileştirilerek okunması kolaylaştırılır. Oluşturulan bu program Python programlama dili ve çok bilinen OpenCV kütüphanesi kullanılarak gerçekleştirilmiştir. Kullanılan görüntüler karanlık-aydınlık ve arka plan rengi gibi farklı durumlarda çekilen fotoğraflardır.**

Anahtar Kelimeler — Belge Tarayıcı, Python, OpenCV, Nesne Tespiti

***Abstract*— In this project, a document scanner has been created using certain algorithms. Online exams, homework and project submissions, which have become an indispensable part of online education, are digitalized using document scanners. Thanks to these programs, the document is automatically cropped from the homework photos and improved to make it easier to read. This program was created using the Python programming language and the well-known OpenCV library. The images used are photographs taken in different situations such as dark-light and background color.**

Keywords —Documentation Scanner, Python, OpenCV, Object Detection.

I. GİRİŞ

Günümüzde, pandemi koşullarının gerektirdiği şartlarda eğitim çevrimiçi olarak devam etmektedir. Bu koşullarda hayatımıza yeni giren yöntemler ve uygulamalar bulunmaktadır. Bunlardan biri de belge taramak. Ödevler, sınavlar ve projeler "Camscanner" gibi belge tarayıcı programlarla dijital ortama aktarılarak teslim edilmektedir. Belge tarayıcı elinizde bulunan bir belge fotoğrafını kırparak ve iyileştirerek dijital ortama aktarmanızı sağlayan bir programdır. Bu projede asıl hedef belge görüntüsündeki kâğıdı tespit edip orijinal görüntüden tespit edilen kâğıt görüntüsünün kırpılmasıdır. Ayrıca görüntünün kırpılma işleminden sonra iyileştirilmesi hedeflenmektedir.

Günümüzde belge taramak için birçok mobil uygulama bulunmaktadır. Bunlardan en çok bilineni "Camscanner" programıdır. Ayrıca başka yazılımcılar tarafından birçok proje örneği de bulunmaktadır.

Yöntemler bölümünde proje yapımında kullanılan yöntemlerden bahsedilecektir. Algoritma başlığında oluşturulan algoritma ayrıntılı bir şekilde anlatılacaktır. Deneyler ve Tartışma başlığında veri setleri ve sonuçlardan bahsedilerek daha iyi bir uygulama için yöntemlerden bahsedilecektir. Sonuç kısmında algoritma yorumlanacak ve özeleştiri yapılacaktır.

II. YÖNTEMLER

Proje gerçekleme aşamasında iki ana yöntem kullanılmıştır.

A. Nesne Tespiti

Görüntüyü daha rahat işlemek ve işimize yaramayacak detaylardan kaçınmak için görüntü bulanıklaştırırız.

B. İyileştirme

Görüntü üzerindeki gürültüleri azaltmak veya üzerinde nesneyi daha kolay belirlemek için eşikleme yaparız.

C. Denoising(Gürültü Giderme)

Görüntü üzerinde gürültüler olacaktır. Bu gürültüler bizim Çalışmamızı engelleyebilir veya bozabilir. Bu sebep gürültü arındırma işlemi yaparız.

D. Morfolojik İşlemler

Görüntü üzerindeki nesnelerin sınır bölgelerinin aşındırma ve görüntü üzerindeki yazıyı kapatıp beyaz alanı artırıp gürültüyü azaltmak için morfolojik işlemlerden "erosion "ve "closing" kullanırız.

E. Çizgi tespiti

Tespit edeceğimiz nesne bir dikdörtgen olduğu için çizgi tespit yöntemi kullanırız. Bu yöntem "Canny Edge Detection" olarak adlandırılır.

F. Kontur Bulma

Görüntü üzerindeki konturleri bulmak nesne tespiti için oldukça gereklidir. Bunun sebebi konturler aracılığı nesne tespitinin oldukça kolay olmasıdır.

G. En büyük konturu bulmak

Çevresi en büyük olan kontur bize istediğimiz bölgeyi verecektir. Bunun için en büyük konturu buluruz.

H. Kontur Koordinatlarını bulmak

En büyük kontoru bulduktan sonra original görüntüde bu çevreyi çıkartmak için konturun koordinarlarını bulup bir dikdörtgen haline çeviririz.

İ. Dört Nokta Dönüşüm

Konturun koordinatlarını kullanarak konturun genişliğini ve yüksekliğini hesaplarız. Daha sonra warpRespective fonksiyonu ile konturu kırparız.

J. Filtreleme

Kırpma işleminden sonra görüntünün okunabilirliğini arttırmak için görüntüyü keskinleştiririz. Keskinleştirme için belirlediğimiz maske ile filtreleme yaparız.

III. ALGORİTMA

Algoritmayı sözde kod üzerinden açıklamak daha anlaşılır olacağından bu bölümde sözde kod paylaşılacaktır.

Sözde Kod:

1. Başla
2. Görüntüyü oku
3. Gri seviyeye çevir
4. Bulanıklaştırma
   Bulanıklaştırma için "Averaging" yöntemi kullanılmıştır. Bunun sebebi kullanışlı ve basit olmasıdır. Ayrıca diğer bulanıklaştırma Yöntemleri ile bir farklılık gözlemlenmemiştir.
5. Eşikle
   "Adaptive Gaussian Thresholding" yöntemi kullanılmıştır. Eşikleme yöntemleri kıyaslandığında kağıt üzerindeki gürültüyü azaltmak için daha uygun olduğu tespit edilmiştir. [1]
6. Gürültü Gider
   Gürültü gidermek için "fastNlMeansDenoising" fonksiyonu kullanılmıştır. Araştırmalar sonucunda gürültü gidermek için basit ve hızlı bir fonksiyon olduğundan kodda kullanılmıştır.
7. Erosion
   Görüntü üzerindeki nesnelerin sınırlarının daha rahat belirlenmesi için aşındırma işlemi uygulanmıştır.
8. Closing
   Sadece belgenin yerini tespit edip kırpmakla uğraştığımız için yazıyı kapatıp beyaz alanı arttırmak bize kolaylık sağlayacağından kapanım işlemi yapılmıştır.
9. Kenar Bulma

   Canny Edge Detection yöntemi ile kenarlar belirginleştirildi.
10. Konturleri bul
    İlgilendiğimiz şey bir dikdörtgen çevresi ve alanı olduğundan kontur bulma işlemi yaparız.
11. En büyük Konturu bul
    Görüntüdeki en büyük alanlı dikdörtgen bizim nesnemiz oluğundan en büyük konturu buluruz.
12. En büyük konturu kırpmak
    Orijinal resimden en büyük konturu çıkartmamız lazım. Bunun için iki aşamalı bir yöntem izlenmiştir.
    1. Koordinat bulma
       En büyük konturun koordinatları bulunarak bir dikdörtgen oluşturulur.

    2. Dört Nokta dönüşümü

    Koordinatlar bulunarak konturun yükseklik ve genişliğini hesaplarız. Ardından konturu hazır kod ile kırparız.
13. Keskinleştir
    Kırpılan görüntünün okunaklı olabilmesi için görüntüyü keskinleştiririz. Keskinleştirme yöntemi olarak "Sharpening Spatial Filter" kullanılmıştır. Bunun sebebi kolay ve kullanışlı olmasıdır.
14. Görüntüyü Bastır

Bir sonraki bölümde aşamaların sonuçları görüntü şeklinde verilmiştir.

Algoritmanın sözde kodu ve açıklaması böyledir. Algoritma benim kullandığım görüntülerde çalışmaktadır. Fakat uzaktan çekilmiş veya karanlığın fazla olduğu bir görüntüde çalışması zorlaşacaktır. Algoritma normal arka plan renginin farklı, ışığın doğru oranda yansıması, çekilen açının uygun olması durumlarında düzgün çalışacaktır. Algoritmada eşikleme, keskinleştirme, bulanıklaştırma ve en büyük konturu bulma adımlarında alternatifler bulunabilir. Fakat sözde kodda anlatıldığı gibi bazı durumlarda kolaylık, bazı durumlarda ise uygunluk göz önüne alınarak bu yöntemler arasında seçim yapılmıştır. Bulanıklaştırma, eşikleme ve morfolojik işlemler adımlarında farklı yöntemler denenmiş fakat verim alınamamıştır. Algoritma görüntüde farklı nesneler de eklenerek denendiğinde bir hata vermemiştir. Fakat gölgenin fazla olduğu yani kenarların belirlenmesinin zor olduğu durumlarda algoritmanın performansı düşecektir.

IV. DENEYLER VE TARTIŞMA

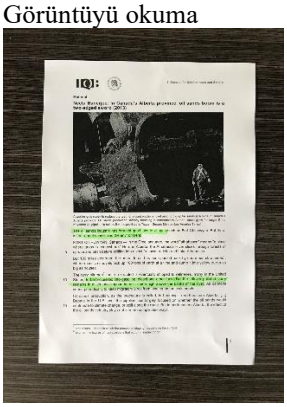
A. Veri Seti/Deneyler/Senaryolar

Projede kullanılan iki veri seti vardır (Eğitim ve Test). Bu setler kolay orta zor görüntülerle hazırlanmıştır. İki sette de dörder görüntü bulunmaktadır. Görüntünün zorluk derecesi arka plan, ışıklandırma, nesne sayısı ve çekilen açı etkenleri göz önüne alınarak hazırlanmıştır. Eğitim seti üzerinde algoritma geliştirilirken, test seti ile algoritma test edilmiştir.

B. Sonuçlar

Sözde kod aşamalarının çıktıları sırasıyla bu bölümde gösterilmiştir. Örnek giriş görüntüsü olarak "egitim1.jpeg" görüntüsü alınmıştır.

1. Görüntüyü okuma

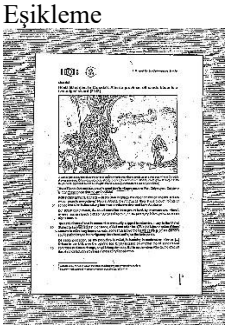


Şekil 1.

2. Bulanıklaştırma

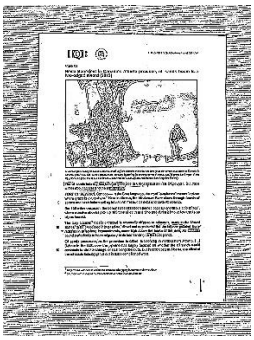
Şekil 2.

3. Eşikleme



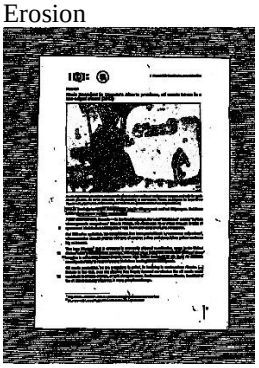
Şekil 3.

4. Gürültü Giderme



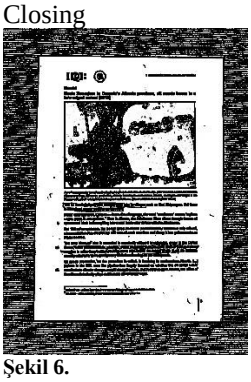
Şekil 4.

5. Erosion



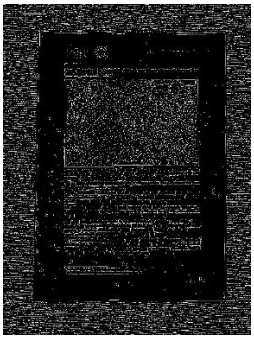
Şekil 5.

6. Closing



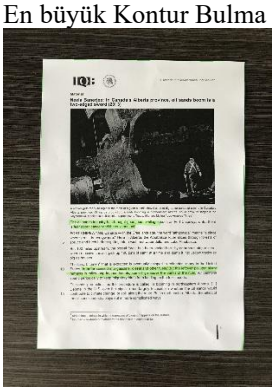
Şekil 6.

7. Kenar Bulma



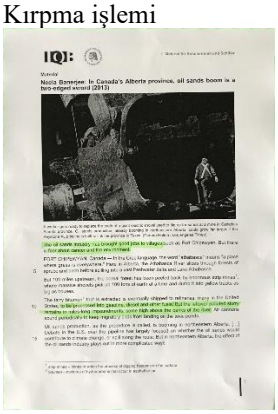
Şekil 7.

8. En büyük Kontur Bulma



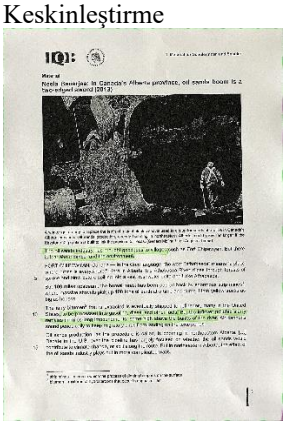
Şekil 8.

9. Kırpma işlemi



Şekil 9

10. Keskinleştirme



Şekil 10.

C. Tartışma

İlk denemelerimde eşikleme yöntemi olarak otsu eşikleme yöntemini kullanmıştım fakat kontur tam olarak belirlenemiyorum belgenin tamamını kırpamıyordum. Bunun için eşikleme yöntemlerini araştırıp değiştirdim. Ayrıca projenin başlangıcında dilation işlemini kullanmıştım. Fakat giriş görüntülerini değiştirdikten sonra algoritmanın hata verdiğini gördüm. Daha sonra morfolojik işlemleri deneyip en uygun olanına karar verdim. Ayrıca bulanıklaştırmak için denediğim filtrelerin sonucu etkilemediğini gördüm ve kullanışı kolaylığı bakımından ortalama yöntemini seçtim.

V. SONUÇ

Sonuçları incelediğimizde seçilen veri setleriyle algoritmanın düzgün çalıştığı gözlemlenmiştir. Algoritma fazla karmaşık olmamakla birlikte derleme süresi açısından oldukça hızlıdır. Ayrıca anlaşılması da güç değildir. Algoritmanın kısıtları ise üst üste iki belgeyi ayırt edememesi ve kullanıcının istediği alanı kırpamamasıdır.

Algoritmanın hatalı bir şekilde çalıştığını varsayalım. Belge tam olarak kırpılamayacak ve burada el ile müdahale gerekecek. Fakat algoritmamız kullanıcaya bir seçenek sunmuyor otomatik olarak kırpma işlemi yapıyoruz. Algoritma kameradan direkt eş zamanlı veri ile çalışması daha kullanışlı olacaktır.

VI. BİLGİLENDİRME

Rapor düzenlemesi yaparken [7]. Kaynaktaki rapordan yararlandım.
Proje yapımındaysa en çok yararlandığım kod [4]. Kaynaktaki projeydi. Kendi çalışmalarım sonuncu kontur bulduktan sonra bir çıkmaza girmiştim ve gerekli olan konturu ayırt edemiyordum birçok kod örneği gördükten sonra en verimli algoritmanın bu kaynaktaki en büyük konturu bulma algoritması olduğunu düşündüm.
Kodun geri kalanındaysa (Kırpma işlemi) herkes tarafından kullanılan hazır fonksiyonlardır bir çok kaynakta gözlemleyerek kullandım.

KAYNAKÇA

[1] https://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html

[2] https://ozengineer.com/?p=1253

[3] https://levelup.gitconnected.com/create-your-own-camscanner-using-python-opencv-66251212270

[4] https://www.pyimagesearch.com/2014/09/01/build-kick-ass-mobile-document-scanner-just-5-minutes/

[5] https://www.murtazahassan.com/courses/opencv-projects/lesson/code-4/

[6] https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html

[7] https://github.com/ertugruldeniz/plakatanimasistemi

[8] http://datahacker.rs/004-how-to-smooth-and-sharpen-an-image-in-opencv/

[9] https://medium.com/@hamzaerguder/s%C3%BCr%C3%BCc%C3%BCs%C3%BCz-ara%C3%A7lar-i%C3%A7in-hough-transform-ile-%C5%9Ferit-bulma-%C3%A7al%C4%B1%C5%9Fmas%C4%B1-9c5ab8d1bcfb

[10] https://www.pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/

[11] https://www.yalova.edu.tr/Files/UserFiles/70/aliiskurt/IMAGE_PROCESS/morfolojik_30Ekim.pdf

[12] https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html