T.C.

Ege Üniversitesi

Mühendislik Fakültesi

Elektrik Elektronik Mühendisliği Bölümü

Lisans Bitirme Projesi Sonuç Raporu

Televizyon Monitörlerindeki Marka Amblemlerinin Görüntü İşleme Yöntemleri ile Üretim Kontrolü

Production Control of Brand Emblems on Television Monitors With Image Processing Methods

Muhammet Yasin ALPEREN

05170000504, m_yasinalperen@hotmail.com, +90 505 646 21 69

Proje Danışmanı: Prof. Dr. Aydoğan SAVRAN

Bornova, İZMİR, Haziran 2022

E.Ü. ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ
LİSANS BİTİRME PROJESİ BİLGİ FORMU

1- Projenin Yapıldığı Dönem: 2021-2022	2- Rapor Tarihi:22 Haziran 2022

3- Projenin Başlangıç ve Bitiş Tarihleri: Ekim 2021-Haziran 2022

4- Projenin Adı: Televizyon Monitörlerindeki Marka Amblemlerinin Görüntü İşleme Yöntemleri ile Üretim Kontrolü

Project Name: Production Control of Brand Emblems on Television Monitors With Image Processing Methods

Projenin Toplam Maliyeti: 220 TL

5- Proje Yürütücüsü Öğrenciler ve iletişim bilgileri (adres, e-posta, tel.)

Muhammet Yasin ALPEREN
05170000504, m_yasinalperen@hotmail.com, +90 505 646 21 69

6- Projenin Yürütüldüğü Kuruluş: VESTEL

7- Destekleyen Kuruluş(ların) Adı, Adresi ve Destek Miktarı: VESTEL Mustafa Kemal Blv. No:10, 45030 Keçiliköy Osb/Manisa Merkez/Manisa

Destek Miktarı : 0 TL (Talep edilmemiştir)

8- Öz:

Vestel firması farklı marklar adında birçok TV monitörü üretmektedir. Bu monitörler üretildikten sonra test birimine gitmektedir. Test biriminde çeşitli kontroller yapılmaktadır, bu kontrollerden biri de marka logosunun monitörün üzerine doğru şekilde basılıp basılmamasıdır. Bu kontrolü normal şartlarda tekniker gözüyle kontrol etmektedir ve bazen gözden kaçırdığı hatalar olabilmektedir. Projemin konusu bilgisayarlı görü teknolojisi ile üretilen ürünlerdeki logo ve markadaki varsa yazım hataları bularak kullanıcıya (teknikere) bunu bildirmektir

Anahtar Kelimeler: Kalite Kontrol,Görüntü İşleme ,SIFT

9- Danışmanının Öğretim Üyesi Adı/Soyadı ve Görüşü:
Prof. Dr. Aydoğan Savran

10- Bölüm Kurulu Görüşü:

11- Projenin Başarı Durumu: Projenin Aldığı Not:

Önsöz

Bu projede Vestel firmasının TV monitörü üretirken karşılaştığı bir problem olan marka amblemlerindeki hataların kontrol edilmesi için bir görüntü işleme programı yapılacaktır. Projenin amacı televizyon monitörleri üzerindeki marka amblemlerinin doğru monitöre hatasız bir şekilde (marka ambleminin doğru açı ile, silik olmadan vb.) basılıp basılmadığını kontrol edebilen görüntü işleme yöntemleri kullanarak kalite kontrol testi yapan bir program yapmak. Bu projede Vestel firmasının televizyon kalite kontrol birimiyle birlikte çalışılacaktır.

Özet Makale

Öz

Bu projede Vestel firmasının TV monitörü üretirken karşılaştığı bir problem olan marka amblemlerindeki hataların kontrol edilmesi için bir görüntü işleme programı yapılacaktır. Projenin amacı televizyon monitörleri üzerindeki marka amblemlerinin doğru monitöre hatasız bir şekilde (marka ambleminin doğru açı ile, silik olmadan vb.) basılıp basılmadığını kontrol edebilen görüntü işleme yöntemleri kullanarak kalite kontrol testi yapan bir program yapmak. Projemizde Python dili ve OpenCV kütüphanesi uyguladık. Marka amblemlerinin orijinal resim ile karşılaştırmasını SIFT algoritmasını yaptık. Sonuçları otomatik excel dosyasına yazdırma işlemi yaparak kullanıcıya belge niteliğinde de bir dosya ulaştırmış olduk. Kameranın dış etkenlerden etkilenmemesi ve stabil bir şekilde aynı uzaklıkta fotoğrafların çekilmesi içinde ayrı bir kamera aparatı geliştirdik.

Anahtar Kelimeler: Kalite Kontrol,Görüntü İşleme ,SIFT

Abstract

In this project, an image processing program will be developed to check the errors in the brand logos, which is a problem Vestel encounters while producing TV monitors. The aim of the project is to make a program that performs quality control tests using image processing methods that can check whether the brand emblems on the television monitors are printed on the right monitor without errors (with the right angle, without fading, etc.). We applied Python language and OpenCV library in our project. We compared the brand emblems with the original image using the SIFT algorithm. By printing the results to an excel file automatically, we delivered a document as a document to the user. We have developed a separate camera apparatus to ensure that the camera is not affected by external factors and that photos are taken at the same distance in a stable manner.

Index Terms: Quality Control ,Computer Vision ,SIFT

Giriş ve Literatür Özeti

Bu tarz çalışmalar genelde fabrika içi çözüm olarak kalıp herhangi bir çalışma olarak yayınlanmıyor. Bu nedenle bu amaçla yapılan bir çalışmaya herhangi bir kaynakta rastlanmamıştır. Fakat resimler arasındaki farklılıkları bulma ve test etme ile ilgili çalışmalar mevcuttur. Birkaç bireysel çalışma olarak farklı logoların arasından istenen logoyu bulabilen bir ufak bir çalışma gördüm. Bu çalışmayı Literatür kısmında paylaştım. Bu çalışmada ORB algoritması ile anahtar noktalar bulunmuş fakat ben daha kesin sonuçlar doğurduğu için SIFT algoritması kullandım. ORB algoritmasının SIFT algoritmasına göre avantajlı yönü ticari kullanımlarda ücretsiz olmasıdır. Bizim çalışmamız diğerlerinden farklı olarak metin kontrolü yapmaktadır ve herhangi bir eksiklik veya orijinal logoya göre kullanıcı tarafından sorun edilecek bir hata varsa onu tespit etmektedir.

Robust logo detection with OpenCV

Daha önceden tanımlanmış herhangi bir markanın logosunu farklı markaların logolarından ayıran bir projedir. ORB yöntemini kullanarak orijinal logo ile test görüntüsü arasında benzerlikler arar ve benzer bulduğu noktaların yoğunlaştırdığı yere sınır kutusu çizer.

Gerçek Zamanlı Görsel Özellik Çıkarımı için Hızlı SIFT Tasarımı

SIFT yöntemini kullanarak yapılmış bir çalışma anahtar noktaların nasıl çıktığını SIFT algoritmasının nasıl işlediğini, Gauss yöntemini nasıl kullanıldığını matematiksel formülleri açıklayan bir çalışmadır. Görüntülerin matrislerinin nasıl işlediğini ve algoritmanın akış şemalarını anlatmıştır. Bu çalışmanın en önemli tarafı bu algoritmanın nasıl gerçek zamanlı bir şekilde kullanılabileceği ile ilgili tasarımıdır.

Nesne Tanıma için SIFT Tabanlı Tanımlayıcıların Performans Değerlendirmesi

Eğitim görüntüsünden çıkarılan öznitelikler kümesi, iyi nesne tanıma performansı için kritik öneme sahiptir. Ölçek Değişmez Özellik Dönüşümü (SIFT), 1999'da David Lowe tarafından önerildi; SIFT özellikleri yereldir ve nesne tanıma için etkilidir. Bu yazıda, SIFT tanımlayıcısı ile ilgili son zamanlarda yapılan bir araştırmayı yürüttüler, nesne tanıma için değerlendirme kriterlerini analiz ettik ve ortak özelliklere ve değerlendirme kriterlerine dayalı olarak SIFT tanımlayıcısının ve genişletilmiş SIFT tanımlayıcılarının performansını analiz ettik. Bu çalışma, SIFT tanımlayıcısının ve önerilen uzantıların iyileştirme stratejilerini ve eğilimlerini belgelemektedir.

Araç Plaka Tanıma Teknolojisi

Plaka tanıma, araçları plakalarına göre tanımlamak için kullanılan bir görüntü işleme teknolojisidir. Bu teknoloji, çeşitli güvenlik ve trafik uygulamalarında kullanılmaktadır. Plakaların üzerindeki yazılar farklı öğrenme algoritmaları ile veya metin tanıyan bazı görüntü işleme algoritmalarıyla tanımlanabilir [1].

Araç Logosu Tanıma ve Sınıflandırma: Özellik Tanımlayıcıları vs. Şekil Tanımlayıcıları

Bu çalışmada üreticinin logosuna odaklanarak bir aracın markasını otomatik olarak tanımlamak için çeşitli görüntü işleme yöntemlerini araştırıyorlar. Bulguları, parlaklıktaki büyük değişikliklerin, ön plandaki araç özelliklerinin ve aynasal yansımaların, ölçekte değişmeyen özellik dönüşümü (SIFT) yaklaşımını pratik olarak işe yaramaz hale getirdiğini ortaya koymaktadır [2].

SSIM ile Görüntüler Arasındaki Farkı Tanımlama

Kredi kartındaki orijinal amblemlerin sahte amblemlerden farklılıkları anlamak ve kredi kartı dolandırıcılığının önüne geçilmesi için görüntü işleme yöntemi kullanılır, bu çalışmada kredi kartının bazı özelliklerinin bir görüntü işleme yöntemi olan SSIM (Structural Similarity Index) ile taranıp farklılıklar bulunmaya çalışılmıştır [3].

Silgi Fabrikası için Kalite Kontrol Sistemi

Bir silginin bazı kalite standartlarına olup olmadığını tespit etmek için üzerine çalışılmış bir sistem örneği. Sistemde silginin iyi bir alana sahipliği, üründeki bariz kusurları, üretici logosunun silgi üzerindeki konum doğruluğu, farklı bir ürünün barkodu olup olmaması, barkodun konumu gibi özellikleri görüntü işlemi ile kontrol etmek amaçlanmıştır [4].

SURF Algoritması ile Daha Önce Öğretilmiş Nesneyi Tanıma

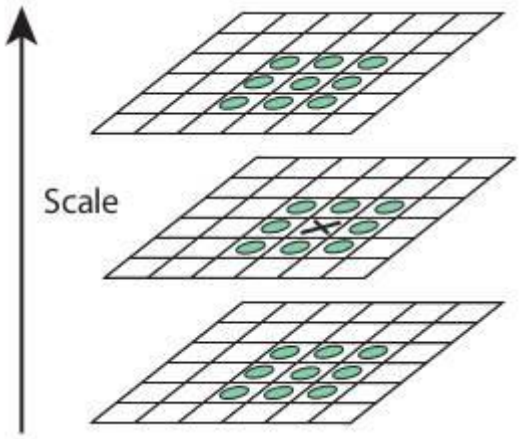
Bilgisayarın önce belirli özelliklere sahip belirli bir piksel grubunun (nesne, yazı vb.) "x nesnesi" olarak adlandırıldığını öğrenmesi, bu bilgiyi hatırlaması ve gelecekte test görüntüsünün bir x nesnesi veya x nesnesi değil gibi veriye sahip olup olmadığını çıkarım için kullanması gerekir. SURF algoritması da bu makine öğrenme yöntemlerinden biridir. Bu çalışmada bu algoritma kullanarak test görüntüsü gerçek zamanlı çalışan bir sistemde bulunmaya çalışılmıştır [5].

Sistemin Tanıtımı ve Çalışma Prensipleri

Projede hatalı logoyu anlamak için SIFT yöntemi kullanılmıştır. Bu yöntemle hatalı logonun doğru logo ile benzerlik oranını hesaplayıp belli bir oranda benzer olan logoyu doğru üretilmiş olduğunu gösterip geçer değeri verilmiştir. Bu benzerlik oranını algoritma ilk başta verilen orijinal resmi inceler daha sonra resimden bazı anahtar noktalar bularak bunları diğer test edilecek fotoğrafın anahtar noktaları ile karşılaştırıp arasında benzerlikler arar. Bu anahtar noktaların eşleşmeleri her zaman mükemmel olmuyor projede yüksek oranda benzer olan eşleşmeler değerlendirmeye alınmıştır. SIFT yöntemi projenin gerçekleştirilmesinde büyük kolaylık sağlamıştır. SIFT yöntemine daha yakından bakalım.

Scale Invariant Feature Transform (SIFT)

Scale Invariant Feature Transform (SIFT) , Türkçe karşılığı ölçek değişmez unsur dönüşümü olan, görüntü tabanlı eşleştirme ve tanıma için bir tanımlayıcıdır. SIFT algoritması, 3 boyutlu bir sahnenin farklı görünümleri arasında nokta eşleştirme ve görünüm tabanlı nesne tanıma ile ilgili bilgisayar vizyonunda çok sayıda hedef için kullanılır. SIFT tanımlayıcısı, görüntü alanındaki çevirilere, döndürmelere ve ölçekleme dönüşümleri ile değişmez ve perspektif dönüşümlerini ve aydınlatma değişikliklerini yumuşatmaya dayanıklıdır.



Şekil 1.0

SIFT algoritması çok detaylıdır fakat burada açıklamak gerekirse ana olarak 4 kısım özellikten oluşmaktadır. Bunlar;

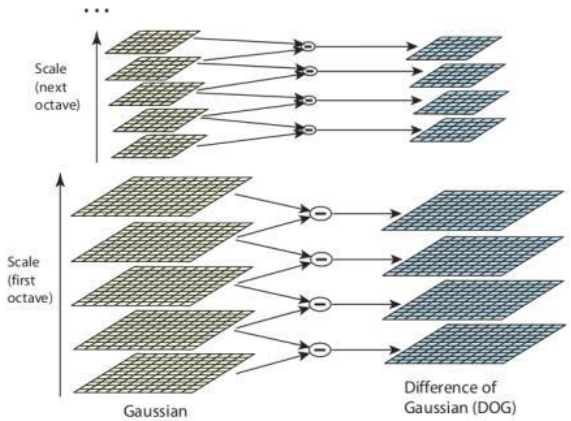
Ölçek alanı tepe seçimi: Özellikleri bulmak için potansiyel konumdur.

Anahtar Nokta Yerelleştirme: Özellik kilit noktalarını doğru bir şekilde bulmadır.

Yönlendirme Ataması: Anahtar noktalara yön atamadır.

Anahtar nokta tanımlayıcı: Anahtar noktaları yüksek boyutlu bir vektör olarak tanımlamadır.

Anahtar Nokta Eşlemesi da diğer bir yöntemidir.



Şekil 1.1

Algoritma projenin ana konusu olmadığı için fazla detaya inilmedi ,fakat yararlanılan kaynakları kaynakça kısmında bulabilirsiniz.[6]

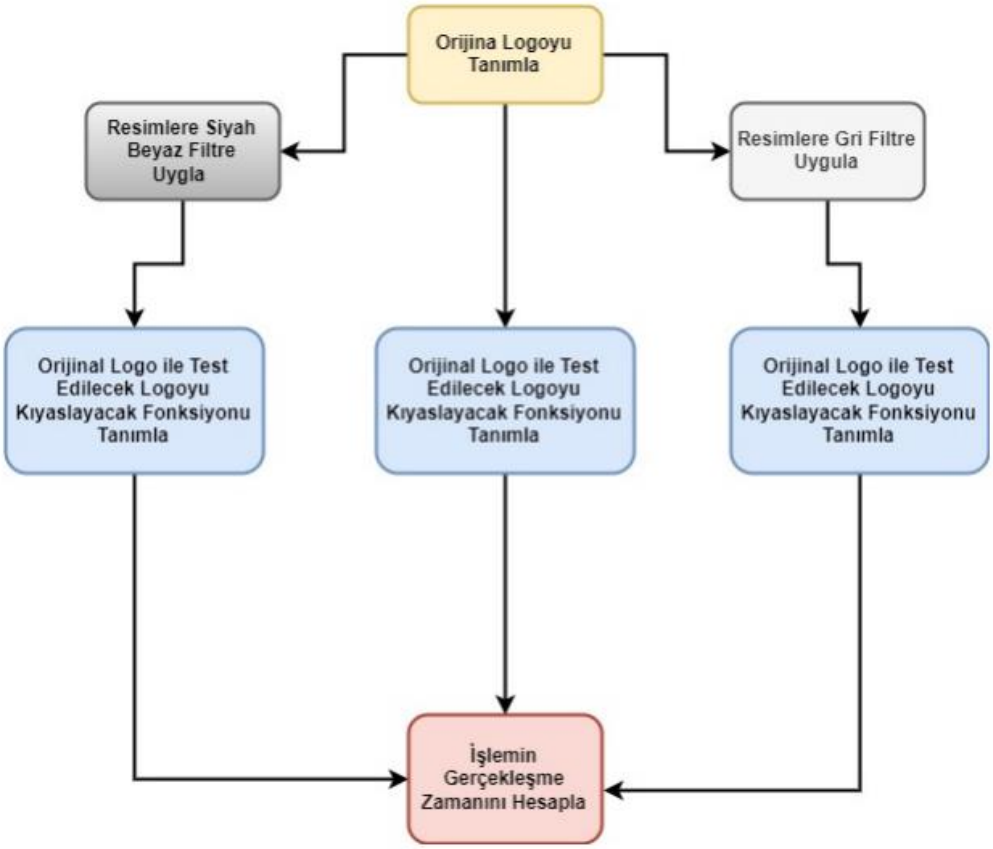
Yazılım Bazında Simülasyonlarla Sistemin İncelenmesi

Projenin Programlanması

Projenin kodları *Python* dilinde yazılmıştır ve *OpenCV* görüntü işleme kütüphanesi kullanılmıştır. Bunun dışında görüntüler matrislerden oluştuğu için *numpy* kütüphanesi ve işlem zamanını hesaplamak için *time* kütüphanesi kullanılmıştır.

Yazılım Akış Şemaları

Algoritmanın genel akış şeması Şekil 1.0'daki gibidir.

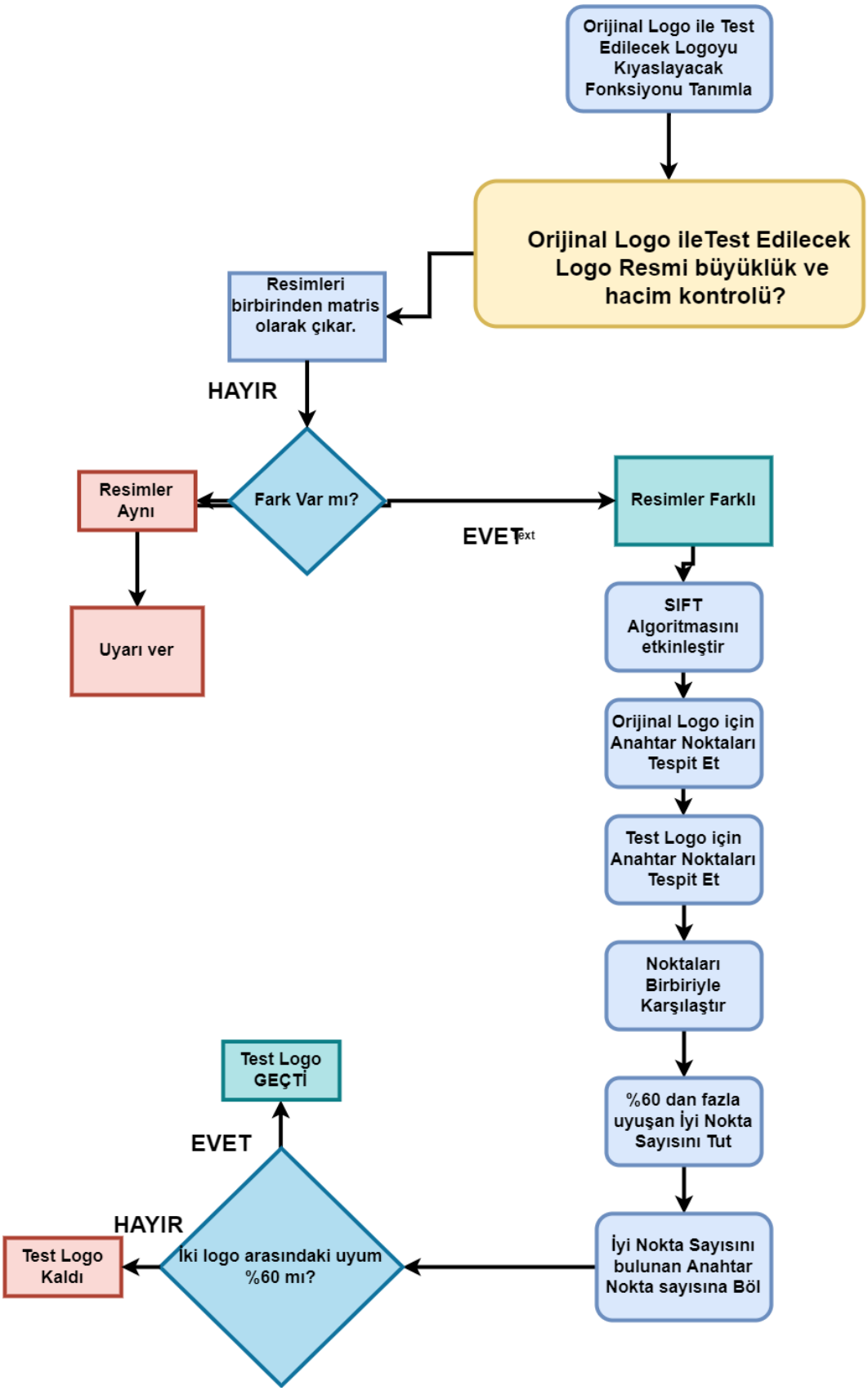


Şekil1.0: Genel Algoritma Şeması

Şekil1.0'daki şemadan kısaca bahsetmek gerekirse tanımladığımız orijinal logoyu farklı filtrelere sokup orijinal logo ile test logosu arasındaki farkları ve benzerlikleri bulacak

fonksiyonun içine atıyoruz. Ve bunu yaparken işlem süresini öğrenmek için her bir filtrede bu işlem kaç saniye sürüyor onu hesaplıyoruz.

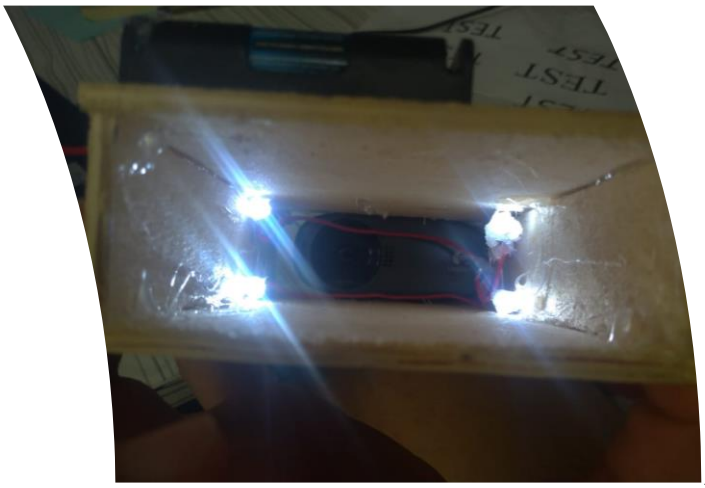
Orijinal logo ile test edilecek logoyu kıyaslayacak fonksiyonun algoritma şeması Şekil 1.1'deki gibidir.

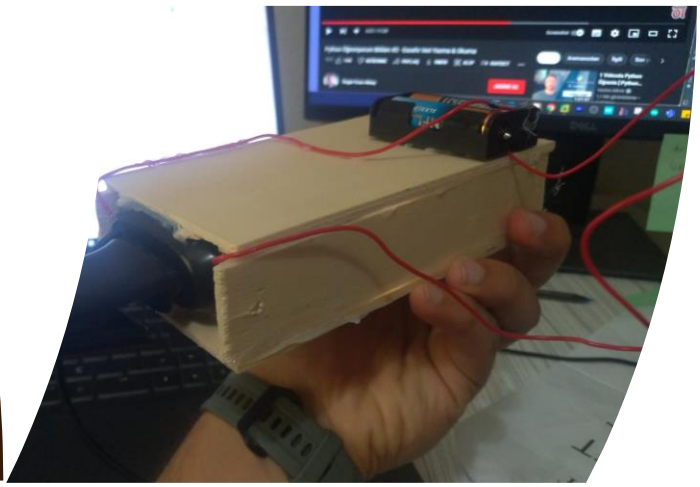

Orijinal Logo ile Test Edilecek Logoyu Kıyaslayacak Fonksiyonu Tanımla
Orijinal Logo ileTest Edilecek Logo Resmi büyüklük ve hacim kontrolü?
Resimleri birbirinden matris olarak çıkar.
HAYIR
Fark Var mı?
Resimler Aynı
Uyarı ver
EVET
Resimler Farklı
SIFT Algoritmasını etkinleştir
Orijinal Logo için Anahtar Noktaları Tespit Et
Test Logo için Anahtar Noktaları Tespit Et
Noktaları Birbiriyle Karşılaştır
%60 dan fazla uyuşan İyi Nokta Sayısını Tut
İyi Nokta Sayısını bulunan Anahtar Nokta sayısına Böl
İki logo arasındaki uyum %60 mı?
HAYIR
Test Logo Kaldı
EVET
Test Logo GEÇTİ

Donanım Bazında Deneysel Olarak Sistemin İncelenmesi

Projemde donanım bazında deneysel olarak incelenecek bir devre kurulumum yoktur. Çalışmam genel olarak yazılım ağırlıklı bir çalışmadır. Fakat test fotoğrafı çekerken fotoğrafların ışık ,uzaklık ,odak gibi konularda aynı şartlar altında olması için bir kamera aparatı tasarladık.

Geliştirilen Aparat

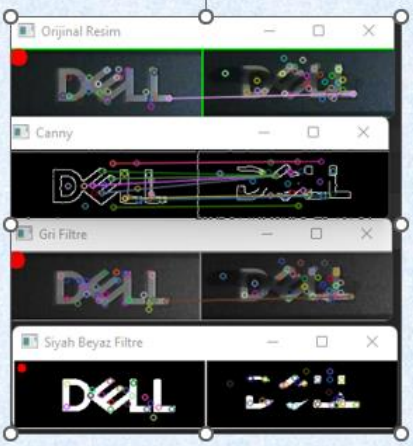




Sonuçlar ve Tartışma

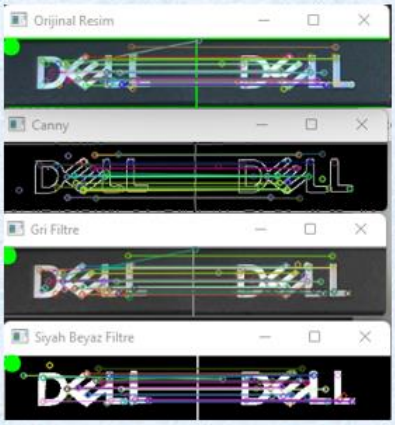
Daha önceden de bahsettiğimiz gibi yaptığımız test sonuçlarının daha anlamlı ve dışardan etkilenmemesi için bir aparat geliştirdik ve testlerimizi standart koşullarda, her bir test için aynı ortamda yapmaya başladık. Ek olarak yazılım algoritmamıza Canny filtresini de ekleyerek logonun kenarlarını kullanarak bir karşılaştırma da yaptık. Bir monitörde yaptığımız testler aşağıda belirtildiği gibidir. Fotoğrafların üstündeki kırmızı noktalar test logonun yanlış olduğunu yeşil noktalar ise doğru olduğunu simgesel olarak ayrıca belirtir.

Üstü Çizik Logo Testi



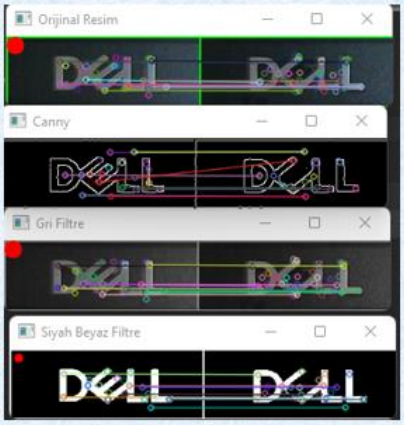
Test Filtresi	Doğruluk Oranı	İşlem Gerçekleşme Zamanı
Orijinal (Filtresiz) Fotoğraf	%4,70	0.022 saniye
Gri Filtreli Fotoğraf	%4,70	0.012 saniye
Siyah Beyaz Filtreli Fotoğraf	%0	0.013 saniye
Canny Filtreli	%24	0.010 saniye

Doğru Logo Testi



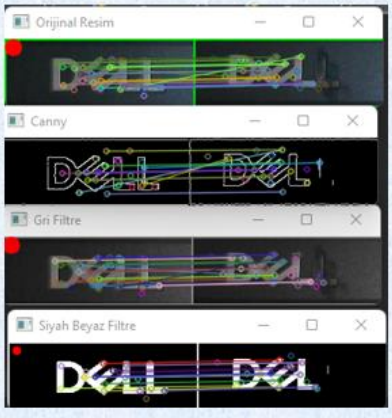
Test Filtresi	Doğruluk Oranı	İşlem Gerçekleşme Zamanı
Orijinal (Filtresiz) Fotoğraf	%61,70	0.046 saniye
Gri Filtreli Fotoğraf	%61,70	0.030 saniye
Siyah Beyaz Filtreli Fotoğraf	%66,66	0.031 saniye
Canny Filtreli	%60	0.033 saniye

E'nin Çizgisi Eksik Logo Testi



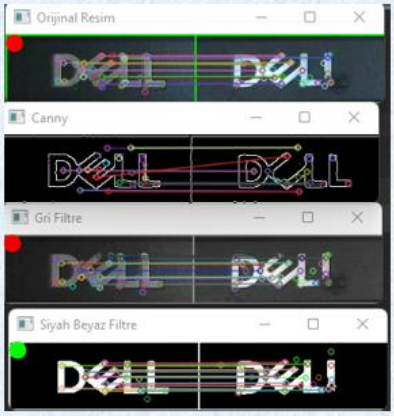
Test Filtresi	Doğruluk Oranı	İşlem Gerçekleşme Zamanı
Orijinal (Filtresiz) Fotoğraf	%26,19	0.020 saniye
Gri Filtreli Fotoğraf	%26,19	0.013 saniye
Siyah Beyaz Filtreli Fotoğraf	%30,30	0.011 saniye
Canny Filtreli	%32	0.011 saniye

Eksik Harf Logo Testi



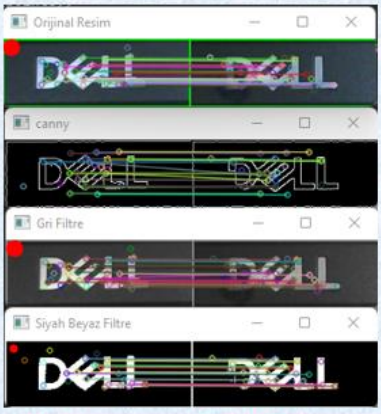
Test Filtresi	Doğruluk Oranı	İşlem Gerçekleşme Zamanı
Orijinal (Filtresiz) Fotoğraf	%53,84	0.026 saniye
Gri Filtreli Fotoğraf	%53,84	0.032 saniye
Siyah Beyaz Filtreli Fotoğraf	%57,66	0.035 saniye
Canny Filtreli	%45,16	0.053 saniye

L'nin Kuyruğu Silik Testi



Test Filtresi	Doğruluk Oranı	İşlem Gerçekleşme Zamanı
Orijinal (Filtresiz) Fotoğraf	%30,70	0.046 saniye
Gri Filtreli Fotoğraf	%30,70	0.030 saniye
Siyah Beyaz Filtreli Fotoğraf	%63,66	0.031 saniye
Canny Filtreli	%52,75	0.033 saniye

Ucu Silik Logo Testi



Test Filtresi	Doğruluk Oranı	İşlem Gerçekleşme Zamanı
Orijinal (Filtresiz) Fotoğraf	%36,7	0.046 saniye
Gri Filtreli Fotoğraf	%36,7	0.030 saniye
Siyah Beyaz Filtreli Fotoğraf	%53,8	0.031 saniye
Canny Filtreli	%45,5	0.033 saniye

Kaynakça

[1] https://medium.com/programming-fever/license-plate-recognition-using-opencv-python-7611f85cdd6c

[2] https://www.semanticscholar.org/paper/Vehicle-Logo-Recognition-and-Classification-%3A-vs-.-Burkhard/14d950df7b92648658c3bf3df129dbac901c512f

[3] https://www.pyimagesearch.com/2017/06/19/image-difference-with-opencv-and-python

[4] https://answers.opencv.org/question/116508/quality-control-system-for-eraser-factory/

[5] http://robocv.blogspot.com/2012/02/real-time-object-detection-in-opencv.html

[6] https://aylablgn.medium.com/%C3%B6l%C3%A7ek-de%C4%9Fi%C5%9Fmez-unsur-d%C3%B6n%C3%BC%C5%9F%C3%BCm%C3%BC-scale-invariant-feature-transform-sift-makine-%C3%B6%C4%9Frenmesi-6-439341081ecf

İçindekiler

Öz

Bu projede Vestel firmasının TV monitörü üretirken karşılaştığı bir problem olan marka amblemlerindeki hataların kontrol edilmesi için bir görüntü işleme programı yapılacaktır. Projenin amacı televizyon monitörleri üzerindeki marka amblemlerinin doğru monitöre hatasız bir şekilde (marka ambleminin doğru açı ile, silik olmadan vb.) basılıp basılmadığını kontrol edebilen görüntü işleme yöntemleri kullanarak kalite kontrol testi yapan bir program yapmak. Projemizde Python dili ve OpenCV kütüphanesi uyguladık. Marka amblemlerinin orijinal resim ile karşılaştırmasını SIFT algoritmasını yaptık. Sonuçları otomatik excel dosyasına yazdırma işlemi yaparak kullanıcıya belge niteliğinde de bir dosya ulaştırmış olduk. Kameranın dış etkenlerden etkilenmemesi ve stabil bir şekilde aynı uzaklıkta fotoğrafların çekilmesi içinde ayrı bir kamera aparatı geliştirdik.

Anahtar Kelimeler: Kalite Kontrol,Görüntü İşleme ,SIFT

Abstract

In this project, an image processing program will be developed to check the errors in the brand logos, which is a problem Vestel encounters while producing TV monitors. The aim of the project is to make a program that performs quality control tests using image processing methods that can check whether the brand emblems on the television monitors are printed on the right monitor without errors (with the right angle, without fading, etc.). We applied Python language and OpenCV library in our project. We compared the brand emblems with the original image using the SIFT algorithm. By printing the results to an excel file automatically, we delivered a document as a document to the user. We have developed a separate camera apparatus to ensure that the camera is not affected by external factors and that photos are taken at the same distance in a stable manner.

Index Terms: Quality Control ,Computer Vision ,SIFT

Giriş Bölümü

Problem, Vestel firması farklı marklar adında birçok TV monitörü üretmektedir. Bu monitörler üretildikten sonra test birimine gitmektedir. Test biriminde çeşitli kontroller yapılmaktadır, bu kontrollerden biri de marka logosunun monitörün üzerine doğru şekilde basılıp basılmamasıdır. Bu kontrolü normal şartlarda tekniker gözüyle kontrol etmektedir ve bazen gözden kaçırdığı hatalar olabilmektedir. Projemin konusu bilgisayarlı görü teknolojisi ile üretilen ürünlerdeki logo ve markadaki varsa yazım hataları bularak kullanıcıya (teknikere) bunu bildirmektir

Bu tarz bir proje ürün üretimi yapan fabrikalarda logo kontrolü için kullanılabilir. Bu şekilde kalite departmanında insan gözü ile yapılan gözlem bilgisayar ile test edilebilir.

Literatür Özeti

Bu tarz çalışmalar genelde fabrika içi çözüm olarak kalıp herhangi bir çalışma olarak yayınlanmıyor. Bu nedenle bu amaçla yapılan bir çalışmaya herhangi bir kaynakta rastlanmamıştır. Fakat resimler arasındaki farklılıkları bulma ve test etme ile ilgili çalışmalar mevcuttur. Birkaç bireysel çalışma olarak farklı logoların arasından istenen logoyu bulabilen bir ufak bir çalışma gördüm. Bu çalışmayı literatür kısmında paylaştım. Diğer literatir taramalarına kaynakça kısmındaki [1], [2], [3], [4], [5] maddelerşnden ulaşabilirsiniz. Bu çalışmada ORB algoritması ile anahtar noktalar bulunmuş fakat ben daha kesin sonuçlar doğurduğu için SIFT algoritması kullandım. ORB algoritmasının SIFT algoritmasına göre avantajlı yönü ticari kullanımlarda ücretsiz olmasıdır. Bizim çalışmamız diğerlerinden farklı olarak metin kontrolü yapmaktadır ve herhangi bir eksiklik veya orijinal logoya göre kullanıcı tarafından sorun edilecek bir hata varsa onu tespit etmektedir

Gelişme Bölümü

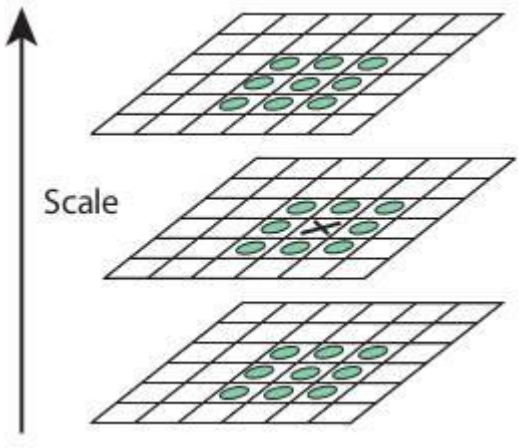
Teori

Çalışmamız iki resim arasında görüntü işleme kütüphaneleri ile fark arama üzerinedir. SIFT algoritması kullanılmıştır. SIFT algoritması gerekli başlıklar altında detaylıca açıklanmıştır.

Scale Invariant Feature Transform (SIFT)

Projede hatalı logoyu anlamak için SIFT yöntemi kullanılmıştır. Bu yöntemle hatalı logonun doğru logo ile benzerlik oranını hesaplayıp belli bir oranda benzer olan logoyu doğru üretilmiş olduğunu gösterip geçer değeri verilmiştir. Bu benzerlik oranını algoritma ilk başta verilen orijinal resmi inceler daha sonra resimden bazı anahtar noktalar bularak bunları diğer test edilecek fotoğrafın anahtar noktaları ile karşılaştırıp arasında benzerlikler arar. Bu anahtar noktaların eşleşmeleri her zaman mükemmel olmuyor projede yüksek oranda benzer olan eşleşmeler değerlendirmeye alınmıştır. SIFT yöntemi projenin gerçekleştirilmesinde büyük kolaylık sağlamıştır. SIFT yöntemine daha yakından bakalım.

Scale Invariant Feature Transform (SIFT) , Türkçe karşılığı ölçek değişmez unsur dönüşümü olan, görüntü tabanlı eşleştirme ve tanıma için bir tanımlayıcıdır. SIFT algoritması, 3 boyutlu bir sahnenin farklı görünümleri arasında nokta eşleştirme ve görünüm tabanlı nesne tanıma ile ilgili bilgisayar vizyonunda çok sayıda hedef için kullanılır. SIFT tanımlayıcısı, görüntü alanındaki çevirilere, döndürmelere ve ölçekleme dönüşümleri ile değişmez ve perspektif dönüşümlerini ve aydınlatma değişikliklerini yumuşatmaya dayanıklıdır.



Şekil 1.0

SIFT algoritması çok detaylıdır fakat burada açıklamak gerekirse ana olarak 4 kısım özellikten oluşmaktadır. Bunlar;

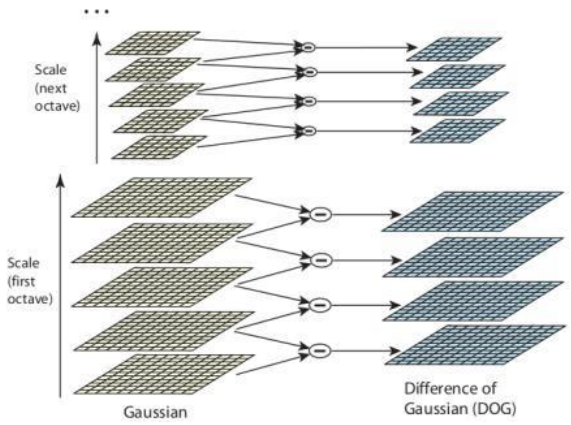
Ölçek alanı tepe seçimi: Özellikleri bulmak için potansiyel konumdur.

Anahtar Nokta Yerelleştirme: Özellik kilit noktalarını doğru bir şekilde bulmadır.

Yönlendirme Ataması: Anahtar noktalara yön atamadır.

Anahtar nokta tanımlayıcı: Anahtar noktaları yüksek boyutlu bir vektör olarak tanımlamadır.

Anahtar Nokta Eşlemesi da diğer bir yöntemidir.



Şekil 1.1

Algoritma projenin ana konusu olmadığı için fazla detaya inilmedi, fakat yararlanılan kaynakları kaynakça kısmında bulabilirsiniz.[6]

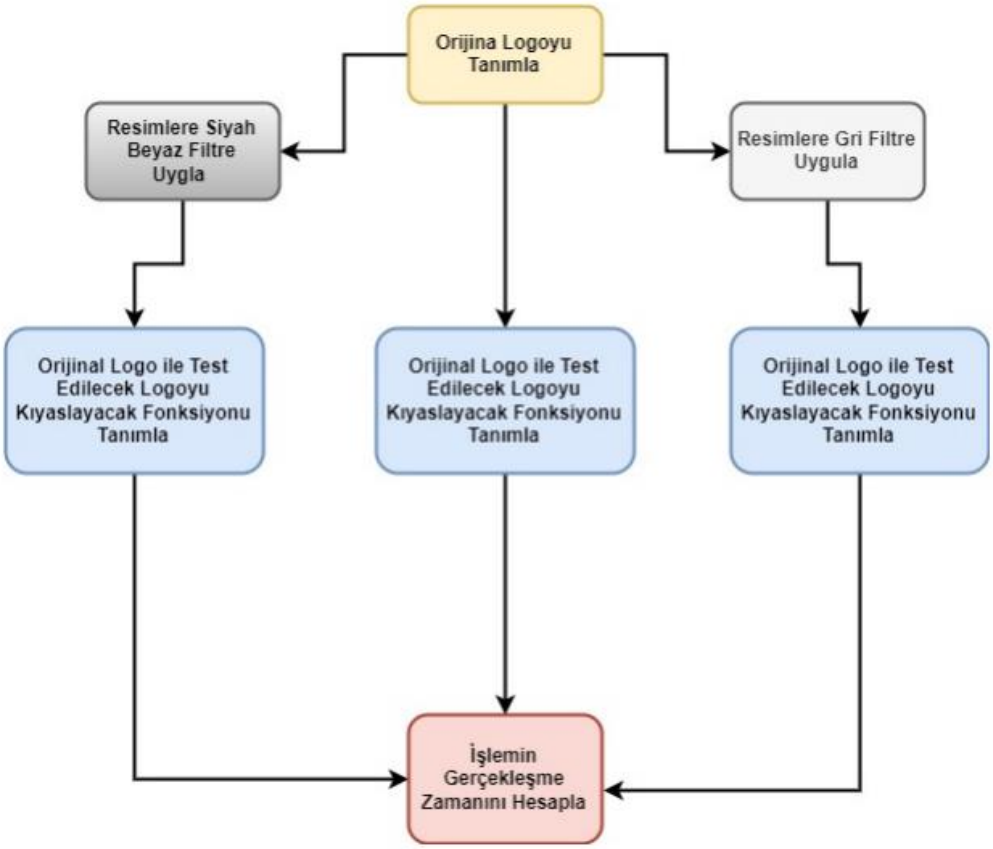
Deney

Projenin Programlanması

Projenin kodları *Python* dilinde yazılmıştır ve *OpenCV* görüntü işleme kütüphanesi kullanılmıştır. Bunun dışında görüntüler matrislerden oluştuğu için *numpy* kütüphanesi ve işlem zamanını hesaplamak için *time* kütüphanesi kullanılmıştır. Projede kullanılan kodlar *EK-A* kısmında verilmiştir.

Yazılım Akış Şemaları

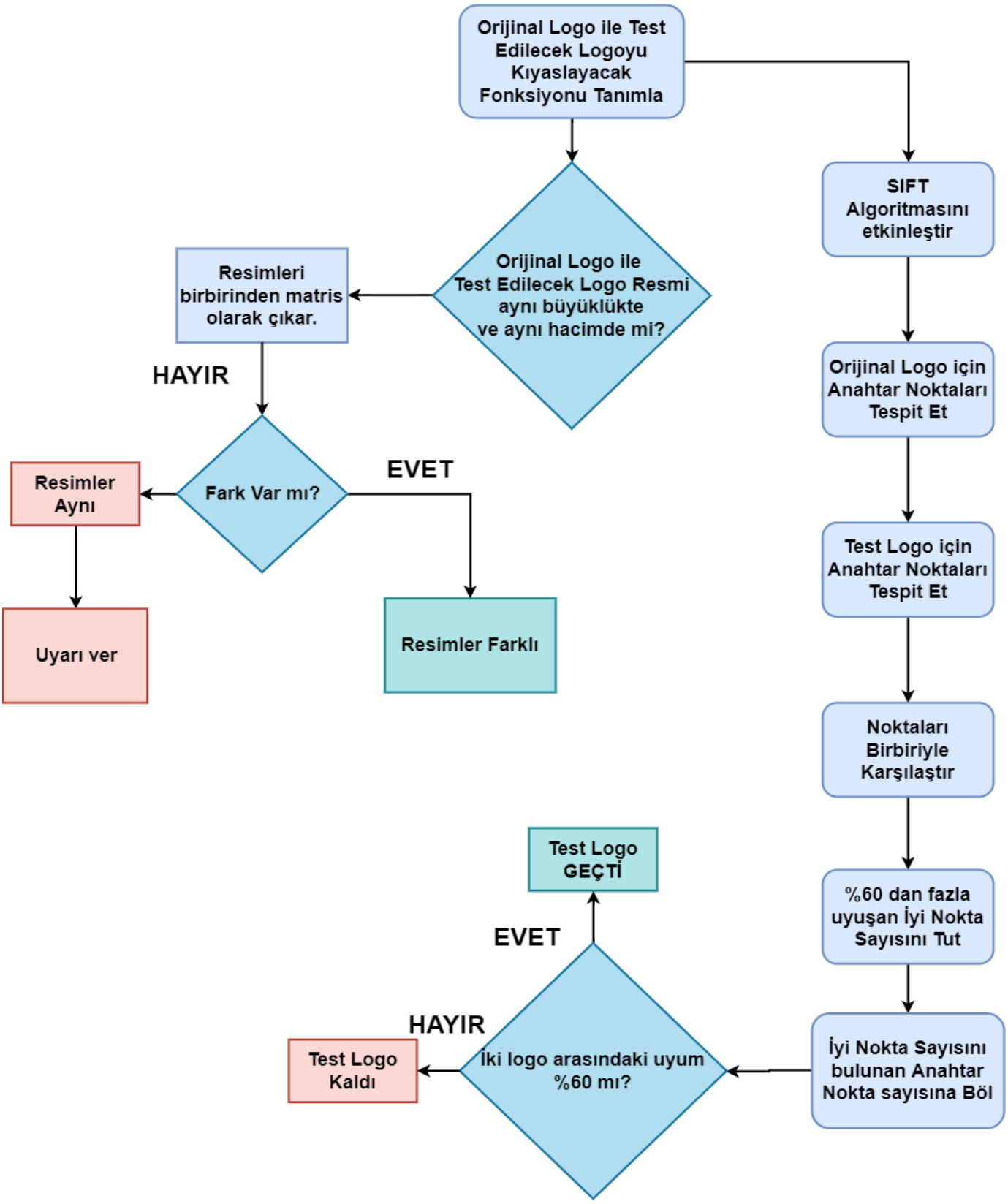
Algoritmanın genel akış şeması Şekil 1.0'daki gibidir.



1. *Şekil1.0: Genel Algoritma Şeması*

Şekil1.0'daki şemadan kısaca bahsetmek gerekirse tanımladığımız orijinal logoyu farklı filtrelere sokup orijinal logo ile test logosu arasındaki farkları ve benzerlikleri bulacak fonksiyonun içine atıyoruz. Ve bunu yaparken işlem süresini öğrenmek için her bir filtrede bu işlem kaç saniye sürüyor onu hesaplıyoruz.

Orijinal logo ile test edilecek logoyu kıyaslayacak fonksiyonun algoritma şeması Şekil 1.1'deki gibidir.



2. Şekil1.1: Genel Algoritma Şeması

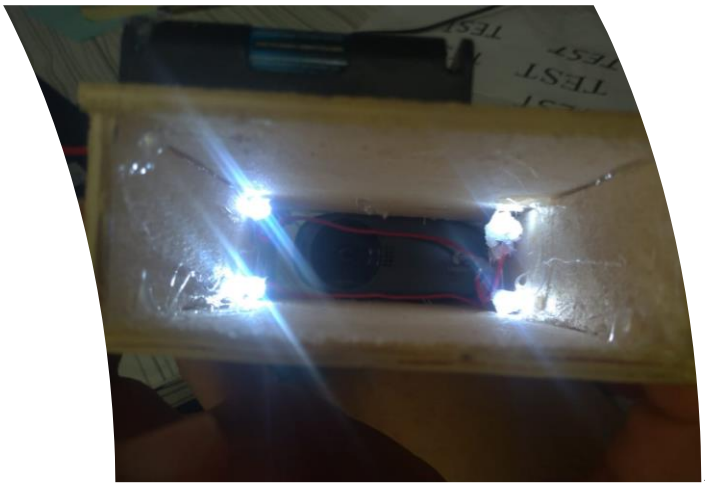
Ölçümler

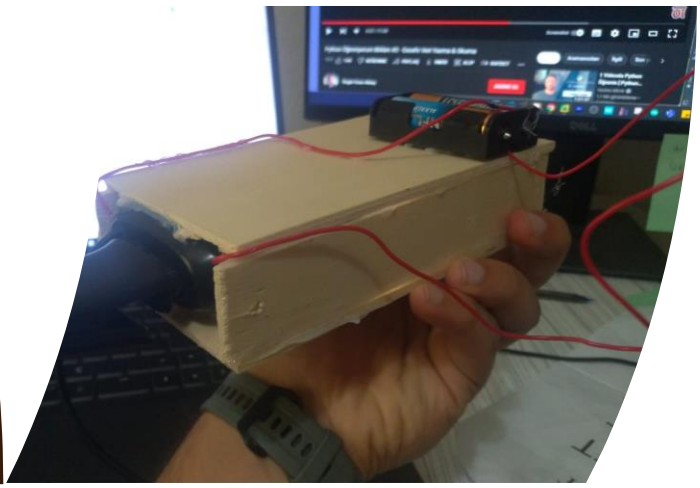
Projemde herhangi bir devre olmadığından ötürü herhangi bir ölçüm sonucum yoktur fakat yazılım programını test etmek için yaptığım çalışmaların logo test sonuçlarımı performans başlığı altında detaylı bir şekilde verdim.

Performans

Gelişme raporunda da bahsedildiği gibi yaptığımız test sonuçlarının daha anlamlı ve çevre koşullarından etkilenmemesi için bir aparat geliştirdik ve testlerimizi standart koşullarda, her bir test için aynı ortamda yapmaya başladık.

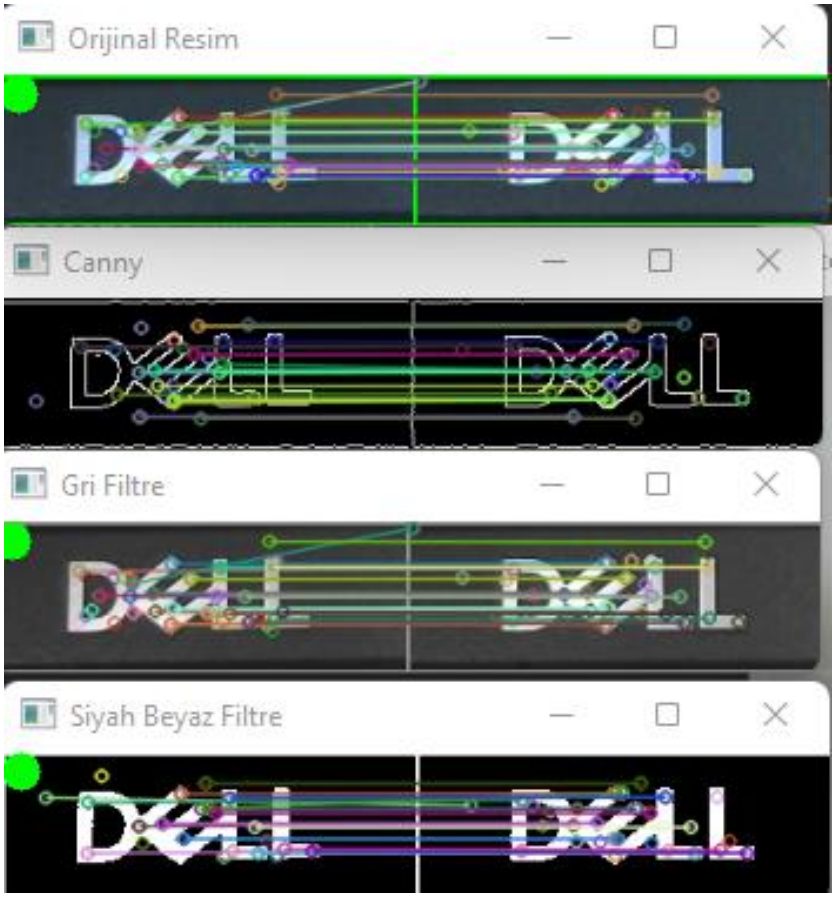
Geliştirilen Aparat





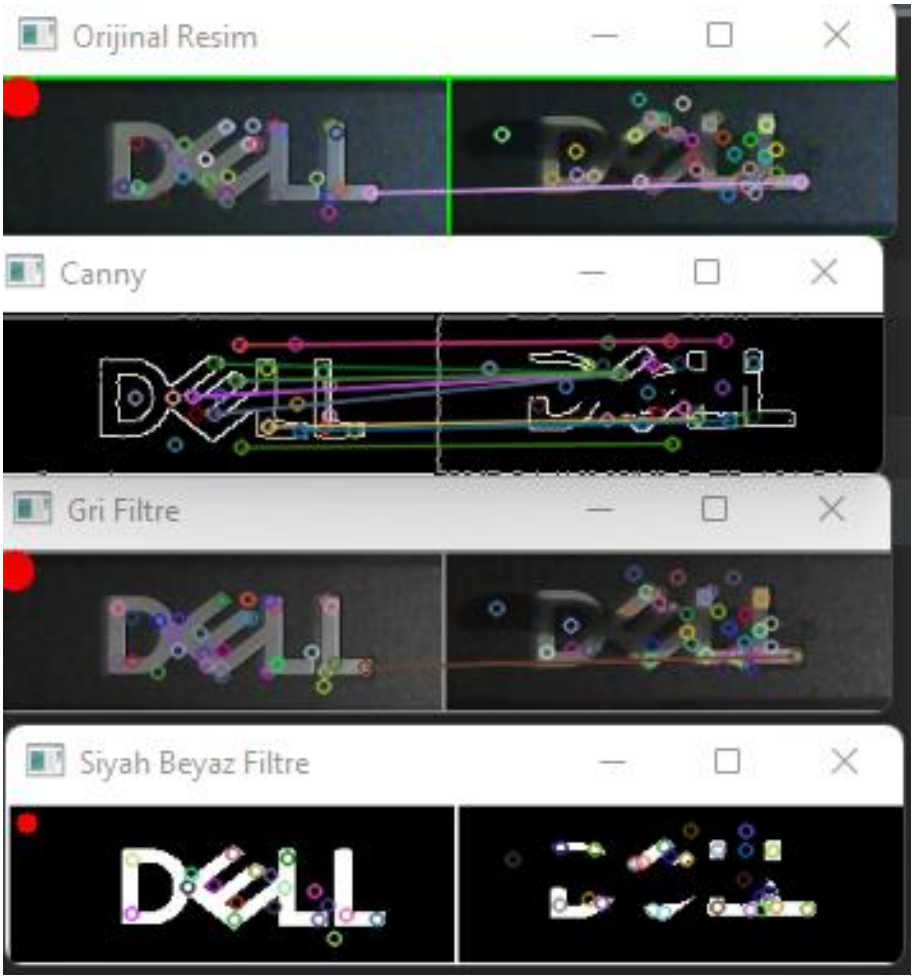
Ek olarak yazılım algoritmamıza Canny filtresini de ekleyerek logonun kenarlarını kullanarak bir karşılaştırma da yaptık. Farklı filtreler eklenerek testler yapılıp logo tahmini doğruluk oranı ayarlanabilir. Böyle bir programı üretim bandına koymadan önce hangi logonun bu programda ne kadarlık bir benzerlik çıkardığı tespit edilip eşik değeri farklı denemler sonucunda ayarlanıp o şekilde kalite kontrol testinde kullanım haline gelebilir. Bir monitörde yaptığımız testler aşağıda belirtildiği gibidir. Bu testler sonucunda geliştirilen programın %98 oranda başarılı olduğu görülmüştür fakat elimizde birebir üretim bandında görüntüler olmadığı için şu an gerçek hayatta yüzde kaçlık bir başarıya ulaşacağı bilinmiyor.

Doğru Logo Testi



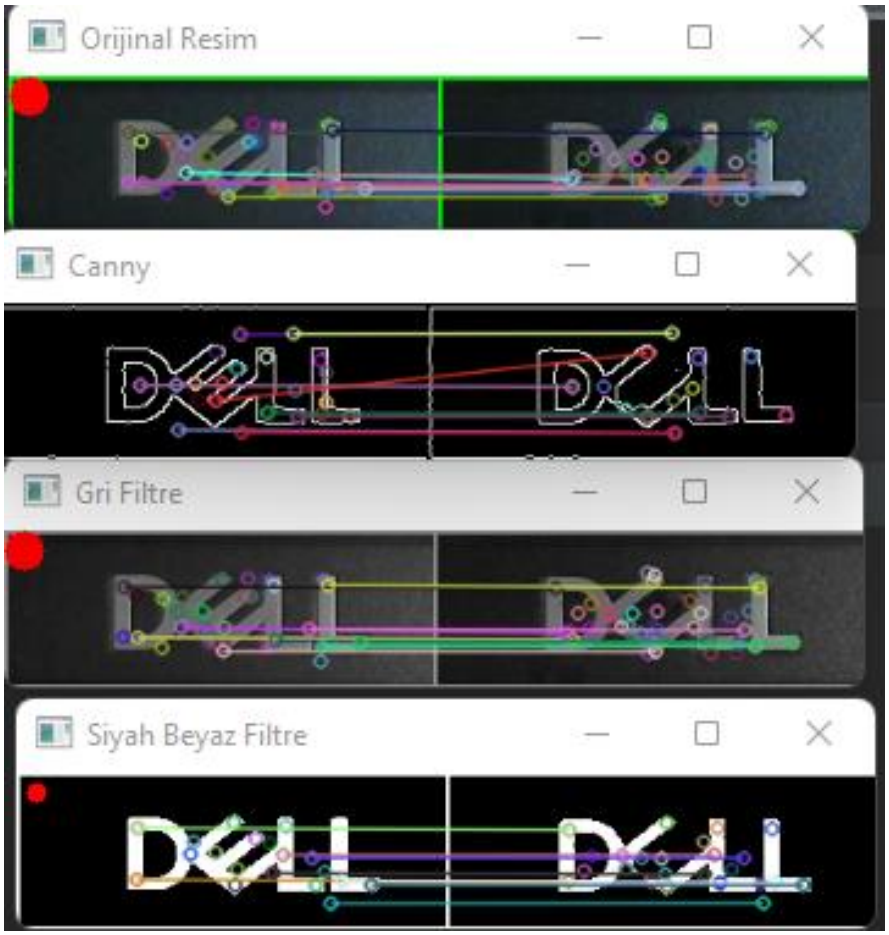
Test Filtresi	Doğruluk Oranı	İşlem Gerçekleşme Zamanı
Orijinal (Filtresiz) Fotoğraf	%61,70	0.046 saniye
Gri Filtreli Fotoğraf	%61,70	0.030 saniye
Siyah Beyaz Filtreli Fotoğraf	%66,66	0.031 saniye
Canny Filtreli	%60	0.033 saniye

Üstü Çizili Logo Testi



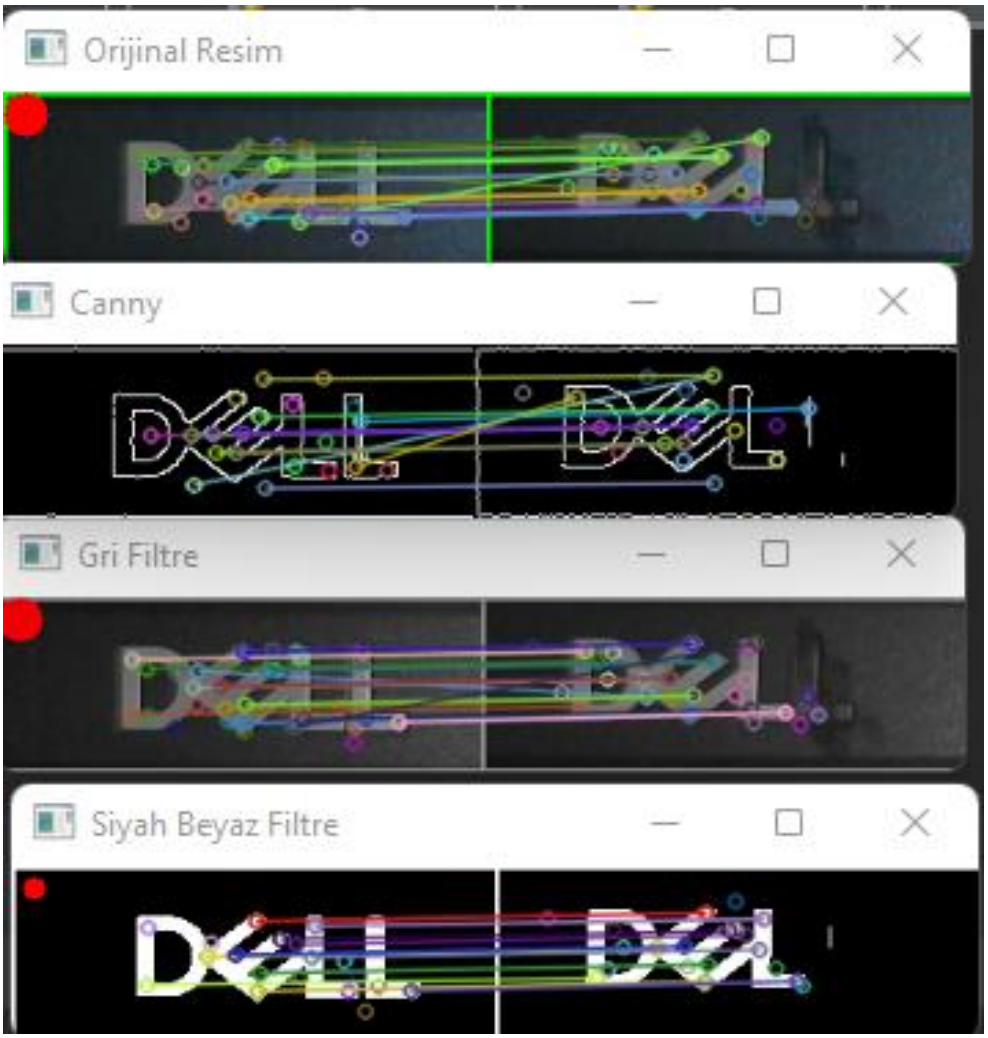
Test Filtresi	Doğruluk Oranı	İşlem Gerçekleşme Zamanı
Orijinal (Filtresiz) Fotoğraf	%4,70	0.022 saniye
Gri Filtreli Fotoğraf	%4,70	0.012 saniye
Siyah Beyaz Filtreli Fotoğraf	%0	0.013 saniye
Canny Filtreli	%24	0.010 saniye

E’nin Çizgisi Eksik Logo Testi



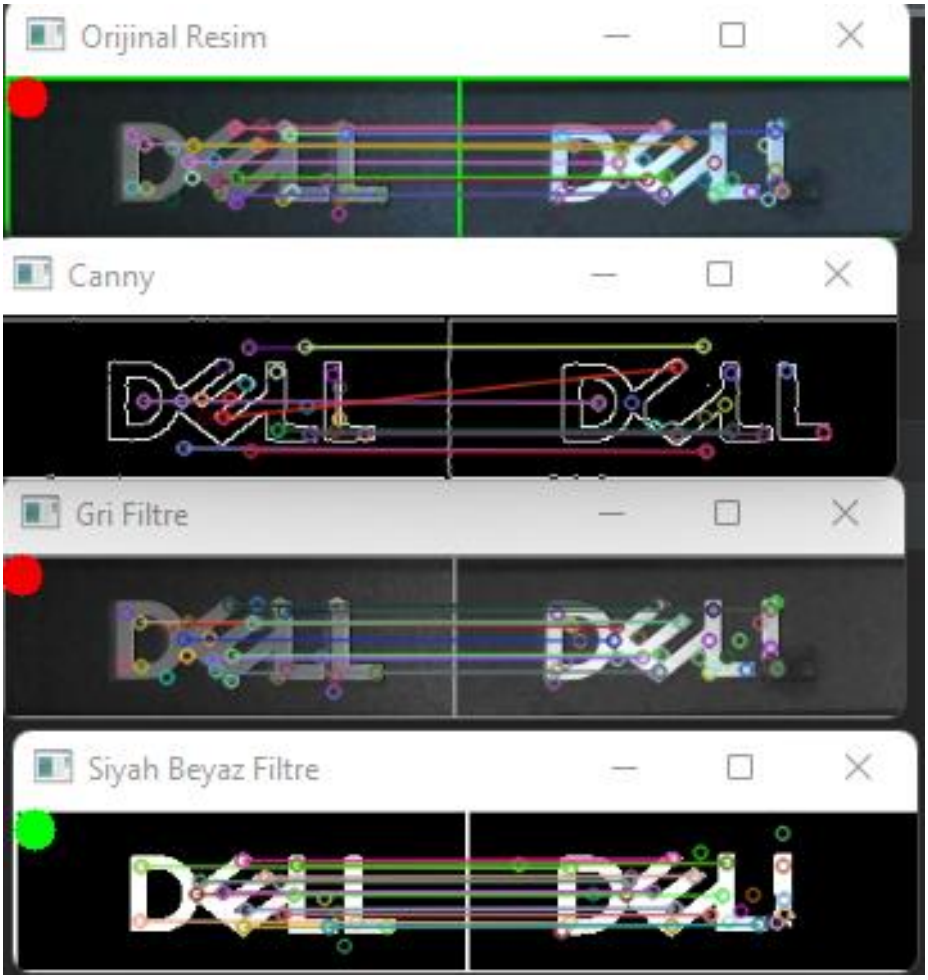
Test Filtresi	Doğruluk Oranı	İşlem Gerçekleşme Zamanı
Orijinal (Filtresiz) Fotoğraf	%26,19	0.020 saniye
Gri Filtreli Fotoğraf	%26,19	0.013 saniye
Siyah Beyaz Filtreli Fotoğraf	%30,30	0.011 saniye
Canny Filtreli	%32	0.011 saniye

Eksik Harf Logo Testi



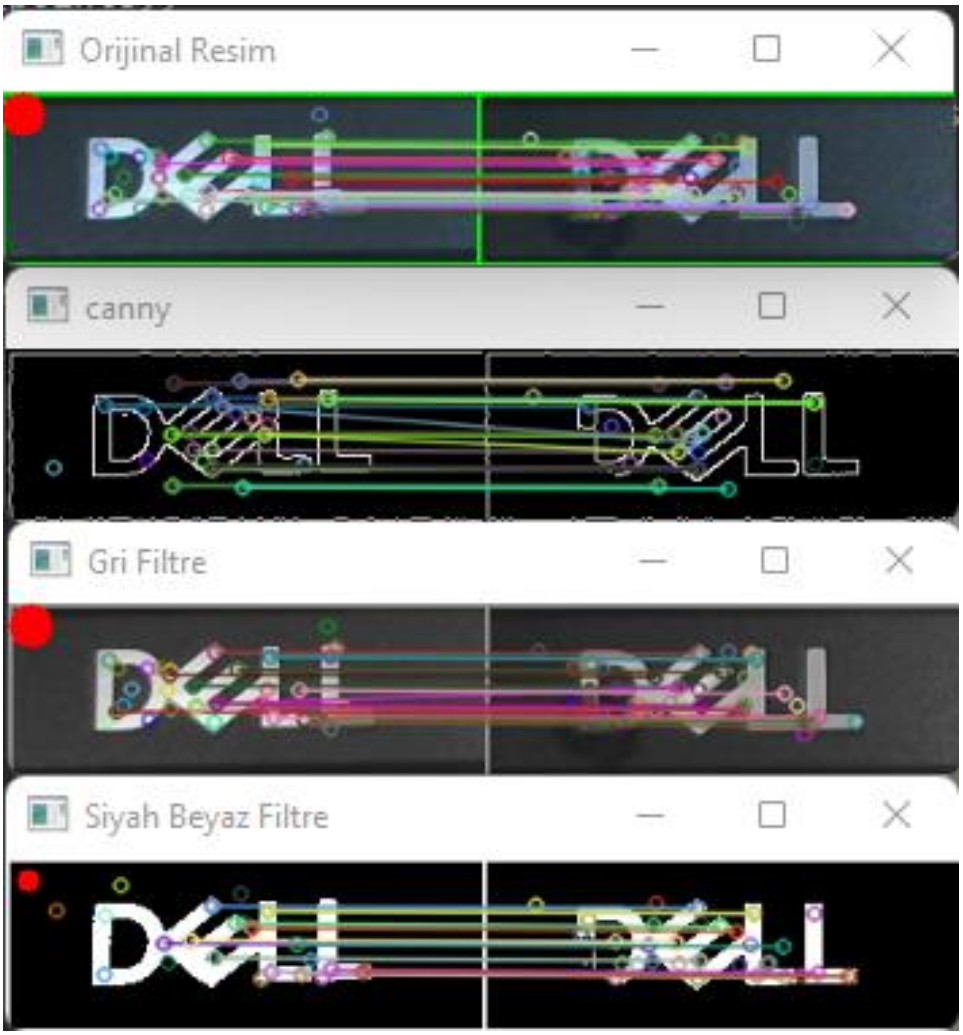
Test Filtresi	Doğruluk Oranı	İşlem Gerçekleşme Zamanı
Orijinal (Filtresiz) Fotoğraf	%53,84	0.026 saniye
Gri Filtreli Fotoğraf	%53,84	0.032 saniye
Siyah Beyaz Filtreli Fotoğraf	%57,66	0.035 saniye
Canny Filtreli	%45,16	0.053 saniye

L’nin Kuyruğu Silik Testi



Test Filtresi	Doğruluk Oranı	İşlem Gerçekleşme Zamanı
Orijinal (Filtresiz) Fotoğraf	%30,70	0.046 saniye
Gri Filtreli Fotoğraf	%30,70	0.030 saniye
Siyah Beyaz Filtreli Fotoğraf	%63,66	0.031 saniye
Canny Filtreli	%52,75	0.033 saniye

Ucu Silik Logo Testi



Test Filtresi	Doğruluk Oranı	İşlem Gerçekleşme Zamanı
Orijinal (Filtresiz) Fotoğraf	%36,7	0.046 saniye
Gri Filtreli Fotoğraf	%36,7	0.030 saniye
Siyah Beyaz Filtreli Fotoğraf	%53,8	0.031 saniye
Canny Filtreli	%45,5	0.033 saniye

Sonuç Bölümü

Sonuç olarak projenin amacına uygun bir şekilde başarılı olarak bittiğini düşünüyorum. Sadece Vestel firmasında değil aynı zamanda başka firmalarda da bu şekilde bir logo kalite kontrolü yapılabilir. Yazılan yazılım programı sadece tekli bir logo için değil çoklu logo için tasarlanmıştır. Bu nedenle kullanışlıdır.

Program farklı kullanım alanlarında incelenerek doğruluk oranına bakılıp test amaçlı diğer logo kontrollerinde kullanılabilir.

Tasarım ve Davranış Özellikleri:

Herhangi bir devre olmadığı için davranış ve tasarım analizi yapılmamıştır.

İleriye dönük çalışmalar

Program kullanışlı bir ara yüz tasarımıyla kullanıcı dostu hale gelebilir ve seri üretim bandında kullanılır hale gelebilir. Ayrıca kaliteli bir aparat tasarlayıp üzerine fotoğraf çekebilmesi ve ara yüzü kontrol edebilmesi için birkaç tane tuş atanabilir.

Projede Vestel firması ile çalışılmıştır fakat bu çalışma sorunun belirtilmesi ve fabrikada ön izlenim yapılmasından ibarettir. Herhangi bir maddi veya projenin hazırlanması için teknik destek yardımı talep edilmemiştir.

Bu projeyi logo üreten firmalar veya logolarını test etmek isteyen firmalar için kullanılabilir. Bu şekilde bir network sağlanıp kurumlarla iletişime geçilip program test edilebilir.

Kaynakça

[1] https://medium.com/programming-fever/license-plate-recognition-using-opencv-python-7611f85cdd6c

[2] https://www.semanticscholar.org/paper/Vehicle-Logo-Recognition-and-Classification-%3A-vs-.-Burkhard/14d950df7b92648658c3bf3df129dbac901c512f

[3] https://www.pyimagesearch.com/2017/06/19/image-difference-with-opencv-and-python

[4] https://answers.opencv.org/question/116508/quality-control-system-for-eraser-factory/

[5] http://robocv.blogspot.com/2012/02/real-time-object-detection-in-opencv.html

[6] https://aylablgn.medium.com/%C3%B6l%C3%A7ek-de%C4%9Fi%C5%9Fmez-unsur-d%C3%B6n%C3%BC%C5%9F%C3%BCm%C3%BC-scale-invariant-feature-transform-sift-makine-%C3%B6%C4%9Frenmesi-6-439341081ecf

EK-5

1. Projeniz tasarım boyutu nedir (prototip ğerçekleştirme, benzetme veya analiz) ?

Projem yazılım üzerinedir.Python dili OpenCV kütüphanesi kullanılarak bir görüntü işleme çalışmasıdır.Donanım olarak sadece bir kamera kullanımı ve kameranın dış ortam şartlarından etkilenmemesi için bir aparat vardır.

2. Kullandığınız tasarim yöntem (yöntemleri) acıklayınız:

Herhangi bir donanım tasarımı olmamıştır fakat yazılım tasarmda nesne yönelimli bir kod yazılımına dikkat edilmiştir.

3. Kullandığınız veya dikkate aldığınız mühendislik standartları nelerdir?

Projemde mühendislik standartlarından herhangi birinin olmasını gerektirecek bir donanım tasarımı bulunmamaktadır. Projem genel olarak yazılım üzerine olduğundan dolayı açık bir kod yazmaya çalıştım. Gerektiği yerlere yorum ekleyerek daha sonra okuyacak insanlara açıklık getirmeye çalıştım.

4. Kullandığınız veya dikkate aldığınız gerçekçi kısıtlar nelerdir? (Ekonomi, Çevre sorunları, Sürdürülebilirlik, Üretilebilirlik, Etik, Sağlık, Güvenlik, Sosyal ve politik sorunlar).

Ürünümüz bir yazılım hizmeti olduğundan dolayı herhangi bir güvenlik sıkıntısı bulunmamaktadır. Fakat kullandığımız algoritma ticari kullanımlarda lisans ücreti talep edebilmektedir. Bu nedenle projenin ticarileşmesi durumunda SIFT algoritması geliştiricilerine ücret ödenmelidir. Ekonomik olarak bunun çok büyük bir eksi olacağını zannetmiyorum. Projenin üretilmesinde herhangi bir engel yoktur. Spesifik bir sorunu çözdüğünden ötürü kullanım alanları kısıtlı fakat projenin sürdürülebilir bir çözüm ürettiğini düşünüyorum.

5. Proje yönetimi nasıl gerçeklestirdiğinizi acıklayınız - her öğrencinin sorumlu olduğu görevleri iş yükü (görev tanımı ve yüzde olarak) ve zaman süreleri ile beraber belirtiniz.

Bireysel çalışma olduğundan dolayı bütün projeyi kendi başıma yaptım.

6. Öneri raporunda yapılan takvime ve/veya konuya gore değişiklik/sapma oldu mı? Evet ise nedeni acıklayınız.

Öneri raporunda anlatılan konuda herhangi bir değişiklik olmadı sadece takvimdeki program iki haftalık sapmaya uğradı bu nedenle proje tam anlamıyla kullanılabilir bir hale Mart ayında gelemedi.Bu sapmalar yıl içinde yaşadığım sağlık sorunları nedeni ile olmuştur.

7. Proje çalışması sırasında ne tür problemler ve sorunlarla karşılaştınız? Bu problemlere ve sorunlara ne tür çözümler getirdiniz.

Projede algoritmayı geliştirirken ilk olarak yaşadığım sıkıntı test ürün bulmaktı. Gerçek hayatta yaşanmış örnekler , üretilmiş ürün ve bu ürünün hatalı hallerini bulmaya çalıştım fakat başarılı olamadım. Bu probleme çözümüm kendi test ürünümü yapmak ve bunu çeşitli şekilde hatalı hallerini üretmek oldu. Aynı yöntemi daha sonra farklı ürünlere uyguladım

Algoritmayı yazarken bir ürünün geçmesi için belli bir yüzde benzerlik oranı vermemiz gerekiyor. Logolar hatalıda olsa çoğunluk hatalı logonun büyük bir kısmı orijinal logoya benzediği için bu benzerlik yüzdesini iyi belirlememiz gerekiyor. Bu problemin çözümünü farklı renk filtreleri deneyerek logoyu daha belirgin hale getirmeye çalışarak ve yaptığımız test sayısını arttırarak bir nebze çözdük.

Ürün gerçek zamanlı olarak kullanılması gereken bir yazılım hizmeti olduğundan dolayı gerçekleşme süresi çok uzun olmaması lazım. Bu soruna çözümümüz algoritmamız kameranın her gördüğü alanı taramayacak sadece logonun içinde bulunduğu çerçeveli alanı tarayacak şekilde tasarlandı.

Kontrol edilecek logonun fotoğrafını çekerken logonun dış ortamdan etkileneceği bir çok unsur vardı (Işık, gölgeleme, odak ayarı, uzaklık vb.) bu sorunu kamera için bir aparat yaparak hallettik

EK-A

```
import cv2
import numpy as np
import time


cam = cv2.VideoCapture(1)
cv2.namedWindow("test")
img_counter = 0

GOLD="GOLD.png"
DELL="DELL.png"
Brand_Name=DELL

#gold imageların tanımlanması
goldImg=cv2.imread(Brand_Name)
goldImg_gray =cv2.cvtColor(goldImg,cv2.COLOR_BGR2GRAY)
thresh, goldImg_wb = cv2.threshold(goldImg_gray, 100, 255,
cv2.THRESH_BINARY)
imgCanny= cv2.Canny(goldImg,200,200)


def test(original,image_to_compare):
    image_to_compare=cv2.imread(image_to_compare)
    #image_to_compare= cv2.cvtColor(image_to_compare1, cv2.COLOR_BGR2GRAY)



    # 1) Check if 2 images are equals
    if original.shape == image_to_compare.shape:
        print("The images have same size and channels")
        difference = cv2.subtract(original, image_to_compare)
        b, g, r = cv2.split(difference)

        if cv2.countNonZero(b) == 0 and cv2.countNonZero(g) == 0 and
cv2.countNonZero(r) == 0:
            print("The images are completely Equal")
        else:
            print("The images are NOT equal")

    # 2) Check for similarities between the 2 images

    sift = cv2.xfeatures2d.SIFT_create()
```

```
    kp_1, desc_1 = sift.detectAndCompute(original, None)
    kp_2, desc_2 = sift.detectAndCompute(image_to_compare, None)

    index_params = dict(algorithm=0, trees=5)
    search_params = dict()
    flann = cv2.FlannBasedMatcher(index_params, search_params)

    matches = flann.knnMatch(desc_1, desc_2, k=2)

    good_points = []
    for m, n in matches:
        if m.distance < 0.6 * n.distance:
            good_points.append(m)

    # Define how similar they are

    number_keypoints = 0
    if len(kp_1) <= len(kp_2):
        number_keypoints = len(kp_1)
    else:
        number_keypoints = len(kp_2)

    print("Keypoints 1ST Image: " + str(len(kp_1)))
    print("Keypoints 2ND Image: " + str(len(kp_2)))
    print("GOOD Matches:", len(good_points))
    print("Renkli DOGRULUK Oranı: ", len(good_points) / number_keypoints *
100)
    result = cv2.drawMatches(original, kp_1, image_to_compare, kp_2,
good_points, None)

   # cv2.putText(result, "Logo Testi", (10, 40), cv2.FONT_HERSHEY_SIMPLEX,
2.0, (255, 0, 0), 4)
    if (len(good_points) / number_keypoints * 100 )> 60:

        #result = cv2.drawMatches(original, kp_1, image_to_compare, kp_2,
good_points, None)
        cv2.circle(result,(10,10), 10, (0,255,0), -1)



        cv2.imshow("Orijinal Resim", cv2.resize(result, None, fx=0.8,
fy=0.8))
        cv2.imwrite("feature_matching.jpg", result)
```

```
    else:

        # result = cv2.drawMatches(original, kp_1, image_to_compare, kp_2,
good_points, None)
        cv2.circle(result,(10,10), 10, (0,0,255), -1)
        cv2.imshow("Orijinal Resim", cv2.resize(result, None, fx=0.8,
fy=0.8))
        cv2.imwrite("feature_matching.jpg", result)



def test_gray(original,image_to_compare):
    image_to_compare1=cv2.imread(image_to_compare)
    image_to_compare= cv2.cvtColor(image_to_compare1, cv2.COLOR_BGR2GRAY)




    sift = cv2.xfeatures2d.SIFT_create()
    kp_1, desc_1 = sift.detectAndCompute(original, None)
    kp_2, desc_2 = sift.detectAndCompute(image_to_compare, None)

    index_params = dict(algorithm=0, trees=5)
    search_params = dict()
    flann = cv2.FlannBasedMatcher(index_params, search_params)

    matches = flann.knnMatch(desc_1, desc_2, k=2)

    good_points = []
    for m, n in matches:
        if m.distance < 0.6 * n.distance:
            good_points.append(m)

    # Define how similar they are

    number_keypoints = 0
    if len(kp_1) <= len(kp_2):
        number_keypoints = len(kp_1)
    else:
        number_keypoints = len(kp_2)

    print("Keypoints 1ST Image: " + str(len(kp_1)))
```

```
    print("Keypoints 2ND Image: " + str(len(kp_2)))
    print("GOOD Matches:", len(good_points))
    print("Gri DOGRULUK Oranı: ", len(good_points) / number_keypoints *
100)
    result = cv2.drawMatches(original, kp_1, image_to_compare, kp_2,
good_points, None)

    #cv2.putText(result, "Logo Testi", (10, 40), cv2.FONT_HERSHEY_SIMPLEX,
2.0, (255, 0, 0), 4)
    if (len(good_points) / number_keypoints * 100 )> 60:

        #result = cv2.drawMatches(original, kp_1, image_to_compare, kp_2,
good_points, None)
        cv2.circle(result,(10,10), 10, (0,255,0), -1)


        cv2.imshow("Gri Filtre", cv2.resize(result, None, fx=0.8, fy=0.8))
        cv2.imwrite("feature_matching.jpg", result)

    else:

       # result = cv2.drawMatches(original, kp_1, image_to_compare, kp_2,
good_points, None)
        cv2.circle(result,(10,10), 10, (0,0,255), -1)
        cv2.imshow("Gri Filtre", cv2.resize(result, None, fx=0.8, fy=0.8))
        cv2.imwrite("feature_matching.jpg", result)


def test_black_white(original,image_to_compare):
    image_to_compare2=cv2.imread(image_to_compare)
    image_to_compare_gray= cv2.cvtColor(image_to_compare2,
cv2.COLOR_BGR2GRAY)
    thresh, image_to_compare =
cv2.threshold(image_to_compare_gray,100,255,cv2.THRESH_BINARY)



    sift = cv2.xfeatures2d.SIFT_create()
    kp_1, desc_1 = sift.detectAndCompute(original, None)
    kp_2, desc_2 = sift.detectAndCompute(image_to_compare, None)

    index_params = dict(algorithm=0, trees=5)
    search_params = dict()
```

```
    flann = cv2.FlannBasedMatcher(index_params, search_params)

    matches = flann.knnMatch(desc_1, desc_2, k=2)

    good_points = []
    for m, n in matches:
        if m.distance < 0.6 * n.distance:
            good_points.append(m)

    # Define how similar they are

    number_keypoints = 0
    if len(kp_1) <= len(kp_2):
        number_keypoints = len(kp_1)
    else:
        number_keypoints = len(kp_2)

    print("Keypoints 1ST Image: " + str(len(kp_1)))
    print("Keypoints 2ND Image: " + str(len(kp_2)))
    print("GOOD Matches:", len(good_points))
    print("Siyah Beyaz DOGRULUK Oranı: ", len(good_points) /
number_keypoints * 100)
    result = cv2.drawMatches(original, kp_1, image_to_compare, kp_2,
good_points, None)

    #cv2.putText(result, "Logo Testi", (10, 40), cv2.FONT_HERSHEY_SIMPLEX,
2.0, (255, 0, 0), 4)
    if (len(good_points) / number_keypoints * 100 )> 60:

        #result = cv2.drawMatches(original, kp_1, image_to_compare, kp_2,
good_points, None)
        cv2.circle(result,(10,10), 10, (0,255,0), -1)


        cv2.imshow("Siyah Beyaz Filtre", cv2.resize(result, None, fx=0.8,
fy=0.8))
        cv2.imwrite("feature_matching.jpg", result)

    else:

       # result = cv2.drawMatches(original, kp_1, image_to_compare, kp_2,
good_points, None)
        cv2.circle(result, (10, 10), 5, (0, 0, 255), -1)
```

```
        cv2.imshow("Siyah Beyaz Filtre", cv2.resize(result, None, fx=0.8,
fy=0.8))
        cv2.imwrite("feature_matching.jpg", result)


def test_canny(original,image_to_compare):
    image_to_compare2=cv2.imread(image_to_compare)
    image_to_compare_gray= cv2.cvtColor(image_to_compare2,
cv2.COLOR_BGR2GRAY)
    thresh, image_to_compare_wb =
cv2.threshold(image_to_compare_gray,100,255,cv2.THRESH_BINARY)
    image_to_compare=cv2.Canny(image_to_compare_gray,200,200)


    # 2) Check for similarities between the 2 images
    print("test2")
    sift = cv2.xfeatures2d.SIFT_create()
    kp_1, desc_1 = sift.detectAndCompute(original, None)
    kp_2, desc_2 = sift.detectAndCompute(image_to_compare, None)

    index_params = dict(algorithm=0, trees=5)
    search_params = dict()
    flann = cv2.FlannBasedMatcher(index_params, search_params)

    matches = flann.knnMatch(desc_1, desc_2, k=2)

    good_points = []
    for m, n in matches:
        if m.distance < 0.6 * n.distance:
            good_points.append(m)

    # Define how similar they are

    number_keypoints = 0
    if len(kp_1) <= len(kp_2):
        number_keypoints = len(kp_1)
    else:
        number_keypoints = len(kp_2)

    print("Keypoints 1ST Image: " + str(len(kp_1)))
    print("Keypoints 2ND Image: " + str(len(kp_2)))
    print("GOOD Matches:", len(good_points))
```

```
    print("Canny Doğruluk Oranı: ", len(good_points) / number_keypoints *
100)
    result = cv2.drawMatches(original, kp_1, image_to_compare, kp_2,
good_points, None)

    #cv2.putText(result, "Logo Testi", (10, 40), cv2.FONT_HERSHEY_SIMPLEX,
2.0, (255, 0, 0), 4)
    if (len(good_points) / number_keypoints * 100 )> 60:
        print("test success")
        #result = cv2.drawMatches(original, kp_1, image_to_compare, kp_2,
good_points, None)
        cv2.circle(result, (10, 10), 5, (0, 255 , 0), -1)


        cv2.imshow("canny", cv2.resize(result, None, fx=0.8, fy=0.8))
        cv2.imwrite("feature_matching.jpg", result)

    else:
        print("test failed")
       # result = cv2.drawMatches(original, kp_1, image_to_compare, kp_2,
good_points, None)
        cv2.circle(result, (10, 10), 5, (0, 0, 255), -1)
        cv2.imshow("Canny", cv2.resize(result, None, fx=0.8, fy=0.8))
        cv2.imwrite("feature_matching.jpg", result)

    a = len(good_points) / number_keypoints * 100
    b = round(a, 2)
    return b


def all_test(original,image_to_compare):
    start_time_gray = time.time()
    test_gray(goldImg_gray, img_name)
    end_time_gray = time.time()
    execution_time_gray = end_time_gray - start_time_gray

    start_time_wb = time.time()
    test_black_white(goldImg_wb, img_name)
    end_time_wb = time.time()
    execution_time_wb = end_time_wb - start_time_wb
```

```
while True:
    ret, frame = cam.read()
    width=int(cam.get(3))
    height=int(cam.get(3))
    if not ret:
        print("failed to grab frame")
        break

    camera = cv2.rectangle(frame, (int ((width/2)-220),int((height/2)-100)
), (int((width/2)+10),int((height/2)-20)), (0, 255, 0), 2)

    cropped_image = frame[220:300, 100:320]
    cv2.imshow("cropped", cropped_image)

    cv2.imshow("test", camera)
    k = cv2.waitKey(1)
    if k%256 == 27:
        # ESC pressed
        print("ECS tuşu,kapatiliyor...")
        break
    elif k%256 == 32:
        # SPACE pressed
        start_time=time.time()
        img_name = "opencv_frame_{}.png".format(img_counter)
        #cv2.imwrite(img_name, frame)
        cv2.imwrite(img_name, cropped_image)
        print("{} written!".format(img_name))
        img_counter += 1
        test(goldImg,img_name)
        end_time=time.time()
        execution_time= end_time-start_time


        start_time_gray = time.time()
        test_gray(goldImg_gray, img_name)
        end_time_gray= time.time()
        execution_time_gray = end_time_gray- start_time_gray

        start_time_wb = time.time()
        test_black_white(goldImg_wb, img_name)
        end_time_wb = time.time()
        execution_time_wb = end_time_wb - start_time_wb
```

```
        start_time_canny = time.time()
        test_canny(imgCanny, img_name)
        end_time_canny = time.time()
        execution_time_canny = end_time_canny - start_time_canny

        print("Ex.Time Renkli          :{}".format(execution_time))
        print("Ex.Time Gri          :{}".format(execution_time_gray))
        print("Ex.Time Siyah ve Beyaz :{}".format(execution_time_wb))
        print("Ex.Time Canny :{}".format(execution_time_canny))
        print("Zaman Kazancı 1:{}".format(execution_time -
execution_time_gray))
        print("Zaman Kazancı 2 :{}".format(execution_time -
execution_time_wb))
        print("Zaman Kazancı 3 :{}".format(execution_time_gray -
execution_time_wb))




cam.release()

cv2.destroyAllWindows()
```