Uçaklardaki Motorların Yerlerini Tespit Etme (Türkçe)

Determination of Engine Locations in Aircraft (In English)

Soray CENGİZ
Elektronik Mühendisliği Bölümü, Gebze Teknik Üniversitesi
soraycengiz.2016@gtu.edu.tr

Özetçe—

Bir uçakta kullanım amaçlarına bağlı olarak 1-8 arasında motor bulunmaktadır motorlar çoğu zaman uçağın kanatları altında konumlandırılmaktadır bunun dışında uçağın arka kuyruğunda da zaman zaman motor görebilmekteyiz.

Uçaklarda bulunabilecek motor tipleri genelleştirilmiş olarak 3'e ayrılmaktadır bunlar; turbojet, turbofan ve turboprob motor tipleridir. Proje sadece turbofan motora sahip uçaklar için tasarlanmıştır. Turbofan motor tipinin Figür 1'de bir örneği gözükmektedir çevresinde herhangi bir pervane barındırmaz bunun yerine ön tarafında bir fan barındırır. Projede çeşitli açılardan çekilmiş uçak görüntülerine bakarak, çıkışında motorların nerede olduğunu belirleyen motorları yuvarlak içerisine alan aynı zamanda motorların sayısını hesaplayan bir algoritma tasarlanmıştır. Uçaktaki motorları tespit ederken motorun iç kısmındaki karanlık alandan yararlanılmıştır dolayısıyla herhangi bir pervaneli motorda algoritma çalışmayacaktır. Algoritmayı tasarlarken görüntülerdeki motorların boyutları, konumları, sayılarını sınırlayacak bir ön koşul bulunmamaktadır, aynı zamanda giriş görüntülerinde normalinden daha fazla aydınlık veya karanlık olması, bunun haricinde etraftaki ekstra nesnelerinde görüntüde bulunmasında sakınca yoktur genel uçak görüntülerinde doğru cevabı verecek şekilde tasarlanmıştır. Algoritma tasarlanırken Python yazılım dili kullanılmıştır ve birçok görüntü işleme tekniğinden faydalanılmıştır.

Abstract—

There are between 1-8 engines in an airplane, depending on the purpose of use, the engines are often positioned under the wings of the airplane, but different type airplanes we can see engines in the rear tail of the airplane.

Engine types that can be found in airplanes are generally divided into 3. These are turbojet, turbofan and turboprob engine types. The project designed only for aircraft with turbofan engines. An example of the turbofan engine type can be seen in Figure 1, it does not have any propellers around it, but instead has a fan on the front. In the project, an algorithm that determines where the engines are located at the exit by looking at the aircraft images taken from various angles, takes the engines into a circle and calculates the number of engines. While detecting the engines in the aircraft, the dark area inside the engine has been used, so the algorithm will not work in any turboprob engine which has propellers. While designing the algorithm, there are no prerequisites to limit the dimensions, locations, number of the engines in the images, at the same time, it is designed to give the correct answer in the general aircraft images. While designing the algorithm, Python software language was used and many image processing techniques were used.

Figure 1.1

I. GİRİŞ

Projede ele alınan problem, çeşitli açılardan çekilmiş bir veya birden fazla motora sahip uçakların sahip oldukları motorları tespit edip, tespit edilen motor sayısını hesaplamaktır.

Araştırmalarım sonucu daha önce bu konu hakkında herhangi bir çalışma yapılmadığını gördüm fakat proje konusuna yakın konuları araştırdığımda tepeden bakış ile havaalanındaki uçakların tespiti, uçakların kanat genişlikleri üzerinde çeşitli çalışmalar yapıldığı gözlemlenmiştir fakat herhangi bir motor tespiti gözlemlenememiştir. Uçaklardaki motor tespitine yakın konular olarak bir görüntüde yuvarlak olan objelerin tespiti, bunların görüntüdeki sayısını hesaplama veya örneğin bir masa üzerinde bulunan madeni paraları tespit etme onların çaplarına göre miktarlarını belirleme işlemleri projeyi yaparken yararlanılan yardımcı kaynaklar olmuşlardır.

Bununla beraber bir ağaçtaki veya sepetteki meyveleri tespit etme konusu üzerine yapılan çalışmalar da bu proje yapımında elde edilmeye çalışılan objelerin geometrik benzerliği bakımından yakın olduğundan bu tarz konularda yapılan çalışmalar da incelenmiştir.

Raporun devamında sırası ile yöntemler başlığı altında denediğim yöntemlerden bahsedeceğim bu yöntemler içerisinde algoritmada kullanılan ve kullanılmayan yöntemler beraber yer alacaktır. Algoritma başlığı altında sadece kullanılan yöntemler detaylı açıklanacaktır bununla birlikte tüm algoritmanın sözde-kod ve akış diyagramı verilecektir. Deneyler ve Tartışma altında girdi görüntüleri üzerinden yorumlar yapılacak hangi girdi görüntüsü neden çalışmadı veya daha iyi nasıl çalışabilirdi yöntemlerin alternatiflerinin algoritmaya nasıl yarar veya zarar sağlanacağı değerlendirilecek. Sonuç kısmında oluşturulan algoritmanın performansı ve verimi üzerine bir anlatım gerçekleşecektir. Son olarak Kaynakça altında yararlanılan referanslar paylaşılacaktır.

II. YÖNTEMLER

A. Görüntüyü Yeniden Boyutlandırma (Resize)

Kurulacak algoritmaya çeşitli boyutlarda görüntüler girdi olarak verilecektir, farklı boyutlar beraberinde farklı giriş görüntülerinde bulunan motorların çaplarındaki devasa farkları ortaya çıkarabiliyor bunu bir miktar ortadan kaldırabilmek için giriş görüntüleri yeniden boyutlandırılmalıdır. Örneğin a görüntüsündeki motorun çapı 15 pixel değerinde ve görüntünün boyutu (600,400) olsun b görüntüsünün motor çapları 100 pixel ve görüntü (2500, 1600) boyutunda olsun aradaki fark çok büyüktür bunu azaltabilmek için ve kodda bazı fonksiyonların çalışma periyotları örneğin Hough Transform for Circle fonksiyonunu belirli çap aralıklarında çalışmasını genelleştirebilirsek algoritma daha verimli çalışacaktır. Bu gibi sorunları bir miktar da olsa tamamen kaldıramayız örneğin (100,600)' lük görüntüyü üst pixellere yeniden boyutlandırdığımızda genişlemiş bir görüntü elde edeceğiz. Bir miktar olsun bu sorunu ortadan kaldırabilmek için Python'da kurulan algoritmada görüntüyü gri seviyeli bir şekilde koda çağırdıktan sonra ilk olarak çağırılan görüntüyü yeniden boyutlandırmak tercih edildi.

Yeniden boyutlandırma yapılırken cv2.resize(src, dsize, interpolation) fonksiyonu kullanıldı bu fonksiyon parametre olarak sırası ile giriş görüntüsünü, yeni boyutu tuple içerisinde (fx, fy) olarak veya harici fx , fy olarak girebilmemiz mümkündür ve son olarak interpolation olarak hangi tip interpolasyon kullanarak yeniden boyutlandırma işleminin yapılacağını istemektedir. Interpolasyon teknikleri NEAREST, LINEAR, CUBIC, AREA' dır.

Projede nearest, cubic ve linear interpolasyon yöntemlerinin hepsi denenmiştir cubic ve linear, nearest'e göre çok daha iyi sonuç verdiğinden ve linear ve cubic arasında çok fazla bir fark olmadığından fonksiyonun default değeri olan lineer interpolasyon tekniği tercih edildi.

Nearest interpolasyon tekniği basitçe "en yakın komşu" pikselini belirler ve o pikselin değerini yeni görüntüde ilgili yere atar. Lineer interpolasyon tekniğinde ise "en yakın 4 komşu" hücreyi kullanıp ağırlıklı ortalamayı alarak ilgili yere yeni piksel yoğunluğunu atar ve son olarak kübik interpolasyon tekniği ise "16 komşu" hücreyi kullanıp bunların ortalamasına bakarak bir hesaplama yapmaktadır. Figür 2.1' de algoritmada kullanılan iki interpolasyon tekniğinin nasıl farklı sonuçlar verdiği gösterilmektedir. Algoritmada tüm giriş görüntüleri 1280x800 boyutuna yeniden yapılandırılmıştır. Figür 2.1'deki giriş görüntüsü orijinal hali 750x500 boyutundadır ve üst kısımda nearest interpolasyon tekniğinin uygulanarak 1280x800' e çıkartma işlemi alt kısımda ise aynı işlemi lineer interpolasyon tekniği kullanarak yapıldığını iyice yaklaştırarak görebilmekteyiz. Lineer interpolasyonun motor bölümlerinde daha doğru sonuçlar verdiğini görüyoruz bundan dolayı bu teknik kullanılmıştır.

B. Görüntüdeki aydınlatma problemlerini ve kontrastını iyileştirme (Histogram Equ.)

Görüntü yeniden boyutlandırıldıktan sonra görüntülerdeki aydınlatma problemini çözmek için Normalized Histogram Equalization, Adaptive Histogram Equalization ve Gamma Correction methodları yeniden boyutlandırılmış görüntüye uygulanarak en uygun method seçildi.

Gamma Correction metodu görüntülerdeki aydınlık ve karanlık sorunlarını çözmek için kullanılmaktadır. Gamma değerine bağlı olarak görüntüdeki aydınlık giriş pixel değerleri çıkışta daha dar bir pixel aralığına sıkıştırılır örneğin gamma 1'den küçük bir değere sahip ise görüntü aydınlatılır gamma 1'den büyük değere sahip ise görüntü karartılır. Gamma Correction yöntemini algoritmaya uygulamak için görüntünün pixel değerlerinin ortalamasına bakarak bir gamma değeri bulma işlemi yapıldı örneğin ortalama değer 100-120 arasında olan görüntlere gamma = 0.6 değeri kullanılarak bir gamma Correction uygulandı.

$$coefficient = 255 \div 255^{\gamma} \quad (1)$$

$$S = coefficient * c * R^{\gamma} \quad (2)$$

Şeklinde bir algoritma kullanılmıştır. Burada S çıktı görüntüsü R ise gamma Correction uygulanan giriş görüntüsüdür. Fakat bu yöntemi uygularken giriş görüntüsünün ortalama değerine bakarak bir gamma değeri elde etmek iyi bir sonuç vermedi bir genelleme yapılamadı. Giriş görüntüsünün ortalamasına bakarak bir gamma değeri belirlemek sorunu çözemedi.

2.yöntem olarak Normalleştirilmiş Histogram Eşiklemesi metodu kullanıldı bu yöntemi kullanırken python'da cv2.equalizeHist(src)fonksiyonu kullanıldı src giriş görüntüsü olmak üzere fonksiyon kendi içersinde görüntüdeki pixel değerlerinin CDF'lerine bakarak bir görüntü iyileştirme yapmaktadır fakat bu yöntem de iyi sonuçlar vermemiştir.

Bunu Figure 2.2' de net bir şekilde görebiliyoruz.

Son olarak Adaptive Histogram Equalization (CLAHE) tekniği kullanıldı. CLAHE tekniği görüntünün tamamında değiş belli MxN bölgelerinde çalışmaktadır. Görüntüdeki belirli karoda yani sınırlı alanda çalışır.

Bu yöntemi kullanmak için öncelikle obj = cv.createCLAHE(clipLimit, tileGridSize) sınıfının bir objesi üretilmiştir. Burada bu sınıf constructor olarak clipLimit değeri yani kontrast sınırını belirler, tileGridSize görüntüdeki bölünecek karo pixel alanı. Obje oluşturulduktan sonra hist_out = obj.apply(src) ile objenin member fonksiyonu çağırılıp girdi olarak yeniden boyutlandırdığımız görüntü verilir. Bizim

algoritmamızda clipLimit = 5.0 ve tileGridSize = (12,12) seçilmiştir. clipLimit'i çok arttırmak beraberinde aşırı kontrast artışı getirmektedir.

Figür 2.2 incelendiğinde en iyi iyileştirme yönteminin CLAHE yöntemi olduğunu görebiliyoruz. Figür 2.2 de sırası ile ilk olarak input görüntüsünü görüyoruz ardından input görüntüsüne normalleştirilmiş histogram uygulandığı sonucu görüyoruz ve son olarak CLAHE uygulanan son görüntüyü görebiliyoruz. CLAHE tekniği diğer tüm giriş görüntülerinde de benzer bir iyileştirme sağlamıştır bundan dolayı bu teknik kullanılmıştır. Bu 3 görüntünün histogramları üzerinden yorum yapacak olursak Figür 2.3'de yine aynı sıra ile olmak üzere histogramları görebiliriz, 2.Histogram olan Normalize Histogram sonucunun giriş görüntüsünün histogramını geniş bir alana yaydığı ve bazı piksel değerlerini yok ettiğini görüyoruz fakat 3.Histogram olan CLAHE'de böyle bir sorun yok aksine bir iyileştirme gözükmektedir.

C. Görüntüyü yumuşatma (Smoothing)

Görüntüye bir Adaptive Histogram Eşiktleme yöntemi uygulandıktan sonra görüntüdeki geçişler yani kenarlar çok keskin olmuştur buradaki geçişlerin keskinliği beraberinde istenmeyen detayları da arttıracaktır, bununla birlikte görüntüde oluşabilecek gürültüyü filtreleme işlemi de görüntüyü yumuşatarak gerçekleşecektir. Bununla birlikte kenarları bulmak isteyeceğimizden çok fazla kenar bulacak ve istenmeyen kenarlar da yorumlanmak zorunda kalınacaktır.

Algoritmada "Gaussian Blurring Technique" kullanılmıştır. Python'da ise cv2.GaussianBlur(src, (kx, ky), 0) fonksiyonu bu işlemi yapmıştır. Algoritmada src olarak CLAHE yapılmış görüntüyü veriyoruz. Kernel olarak (kx, ky) tuple'ını giriyoruz ve son olarak sabit 0 sınır tipini girilmektedir.

Gaussian Blur tekniğinin yaptığı işlem aslında bir görüntüyü bir maske ile filtrelemektedir maskenin şekli ve içerisinde barındırdığı piksel değerlerine bağlı olarak görüntüdeki yumuşama oranı artabilir veya azalabilir. Bu filtre için yapılan şey giriş görüntüsündeki tüm piksel değerlerine aynı filtreyi uygulamak ve filtrenin boyutuna göre pikseldeki komşulukları da kullanarak ortalama bir değer almaktır.

D. Kenarları Tespit Etme (Canny)

CLAHE + BLURRING işleminin ardından görüntüdeki kenarları ortaya çıkartıp görüntüdeki motorların çeperlerinden bir yorum yapmak planlanmıştır. Kenarları ortaya çıkarabilmek için tek yöntem denenmiştir ve bu yöntem ile istenen performans elde edilebildiğinden farklı yöntemlere başvurulmamıştır uygulanan yöntem "Canny Edge Detection" yöntemidir.

Algoritmada bu yöntemi uygulamak için pythonda cv2.canny(src, Tao1, Tao2) fonksiyonu kullanılmıştır. Burada kısaca Canny'nin kendi içerisinde yaptığı şeyi özetlemek istersek;

İlk olarak fonksiyona giren src görüntüsü bir Gaussian Filter kullanarak yumuşatılır. 5x5 bir filtre kullanılır Gaussian Filtresini Figure 2.4' de üst kısımda maske halini ikinci kısımda ise Gauss şekline sahip 3boyutlu bir şeklini görmekteyiz. Gaussian filtresinde tüm piksellerin toplamı 1 olacaktır. 5x5 tüm pikselleri toplayıp 1/273 ile çarparsak 1 sonucu elde edeceğiz. 3boyutlu olarak 2.görüntüyü incelediğimizde aslında sivri bir yapı görmekteyiz bu sivri yapı görüntüde yüksek frekansların filtreleneceğini bize söyler, yüksek frekanslar ise hızlı geçişlere sahip olan kenarlardır. Gaussian filtresinin ortası sivri olmasının nedeni görüntünün (0,0) konumu yani DC konumu orta noktaya taşındığında orta noktadan kenarlara doğru gidildiğinde yüksek frekanslara doğru gidiliyor gibi düşünülebilir bu da orta noktadan uzak olan yerlerin yüksek frekanslar olduğunu ortaya çıkarır 3boyutlu Gauss görüntüsünde kenarların sıfıra yakın olduğunu görüyoruz bu da bize görüntüdeki yüksek frekansların bastırılacağını söylemektedir aslında bir nevi Alçak Geçiren Filtre mantığı gibi düşünülebilir.

Yumuşatma işleminin ardından "Gradient Calculation" işlemi uygulanmaktadır. Bu aşamada yumuşatma yapılmış görüntünün x ve y yönündeki türevleri alınmaktadır. Kenarlar türev operatörüne karşı çabuk tepki verebilmektedirler, dolayısıyla x ve y yönündeki türevleri hesaplayarak görüntünün x ve y yönündeki kenarlarını tespit edebiliriz. Bu türevleri hesaplamak için x ve y yönünde türev alan maskeler vardır bu maskeler görüntüye ayrı ayrı uygulanıp elde edilen sonuçların mutlak değerlerinin toplamı bize Gradient yapılmış görüntüyü yani kenarları bulunmuş görüntüyü verecektir. Figure 2.5' e bakarak x ve y yönünde türev alabilen Sobel filtrelerini görebilmekteyiz 2.görüntüde ise bu maskelerin uygulandıktan sonra mutlak değerli gradient ifadesini ve altında gradient yönünün görebilmekteyiz gradient yönü bize edge yönü hakkında bilgi verecektir gradient yönü ile edge yönü arasındaki açı 90 derecedir.

3. adım olarak "Non-Maximum Suppression" işlemi gerçekleşecektir. Gradient sonucu elde edilen görüntüde çok fazla kenar tespit edilecektir her piksel için pozitif ve negatif gradyan yönlerinde iki komşu bulun ve yan yana gelmiş 3 kenarlardan piksel büyüklüklerine bakarak en büyük olanı bırakıp geriye kalan 2 pikseli bastırmaya dayalı bir yapı işlemektedir.

4. adımda "Double Threshold and Hysteresis" mantığı söz konusudur kısaca açıklamak gerekirse çift eşik adımı 3 çeşit pikseli tanımlamayı amaçlar bunlar; güçlü, zayıf ve alakasız olarak sınıflandırılır. Güçlü pikseller yüksek yoğunluğa sahip piksellerdir, zayıf pikseller güçlü pikseller kadar yoğunluğu olmayan ancak alakasız olarak da kabul edilmeyecek piksel yoğunluğuna sahiplerdir. Diğer pikseller kenar için alakasız kabul edilir. Fonksiyon Tao1 ve Tao2 olarak iki eşik kabul etmektedir Tao2 parametresi burada yüksek eşiği tanımlar, Tao1 ise alakalı olmayan pikselleri tanımlamak için düşük eşik kullanır. Her iki eşik arasında yoğunluğa sahip olan tüm pikseller zayıf olarak işaretlenir ve biri sonraki adımda yorumlanmak için hazırlanmaktadırlar. Figür 2.6'ya bakarak bir yorum yapabiliriz, A kenarı Tao2 yani maxVal'in üzerindedir, bu nedenle "kesin kenar" olarak kabul edilir. C kenarı maxVal'in altında olmasına rağmen, A kenarına bağlıdır, böylece geçerli kenar olarak kabul edilir ve biz yukarıdaki tam eğriyi elde edebiliriz. Ancak B kenarı Tao1 yani minVal'in üzerinde olmasına ve C kenarı ile aynı bölgede olmasına rağmen, herhangi bir "kesin kenara" bağlı olmadığı için algoritma alt kısımdaki eğriyi bir kenar olarak yorumlayamayacaktır. Tao1 ve Tao2 ye bağlı olarak kenar yorumları değişebilmektedir, algoritmada kullanılan parametreler bir sonraki bölümde açıklanacaktır.

E. Görüntüdeki kenarlardan yuvarlak tespit etme(Hough Transform for Circles)

İnput_3.pgm giriş görüntüsü için; Canny işleminden sonra elde edilen görüntüyü Resim_1.jpg' de görebilirsiniz bu resimde motorlar ile beraber etrafta bir çok detay küçük ve büyük olmak üzere yuvarlak olarak algılanabilecek bir sürü detay bulunmaktadır.

Bu görüntüden çapı bilinmeyen yuvarlakları bulma işlemi çok zordur, ancak bir yaklaşım yapılabilir bu yaklaşım şu şekilde olacaktır; Yuvarlağa yakın olan görüntüdeki tüm yerleri al ve başka bir görüntüye ata. Bu yaklaşıma bağlı kalarak kenarları bulunmuş görüntüye Hough Circle Transform uygulanacak ve istenilen çap aralığındaki ve istenilen accumulator parametresine bağlı olarak görüntüde olası iyi veya kötü circle'a benzeyen tüm içerik başka bir görüntüye aktarılacaktır. Görüntüdeki yuvarlağa benzeyen kısımların çıktısını Resim_2.jpg'de görebilirsiniz, ve son olarak Resim_3.jpg'ye bakara bu yuvarlakların merkezlerinin konumlarını ve sahip oldukları çap değerlerini görebileceğiniz bir çıktı yer almaktadır.

Burada bulunan ve hesaplanan yuvarlaklar daha sonra Algoritma başlığı altında anlatılarak algoritmada ne amaçla kullanıldığı gösterilecektir. Hough Transformu sadece motorların olası konumlarını barındıran yuvarlakları bulmuştur. Burada bulunan her bir yuvarlak konumunda aslında olası bir motor bulunma ihtimalini beraberinde getirecektir. Hesaplanan ve Resim_3'de de görülen konum ve yarıçap değerleri bir sonraki aşamada kullanılacaktır.

"Hough Transform for Circle" yönteminin çalışma prensibi bir accumulator'e dayanmaktadır. Basit bir çember denkleminden yola çıkarak Figure 2.7' ye bakarak görüntü düzleminde Circle detection yaparken;

$$(x-a)^2 + (y-b)^2 = r^2$$

$$x = a + R * cos\theta$$
$$y = b + R * sin\theta$$

Şeklinde düşünecek olursak ve görüntüdeki yuvarlak nesnelerin yarıçağını bilmiyorsak her x,y ikilisine bu formül uygulandığında ortaya çıkabilecek tüm r çapına sahip a,b merkezli yuvarlaklar bulunacaktır bu çok fazla işlem gerektirir, bulunacak bu circle'lar Accumulator'de çizdirilir, ve yeteri kadar üst üste kesişim var ise accumulator orada ilgili piksel konumunda bir circle olduğunu bize söyler. Figure 2.8'e bakarak bir yorum yapabiliriz burada daha önce de bahsedildiği gibi masa üzerindeki paraların tespitinde kullanılan bir accumulator yapısını görmekteyiz masa üzerinde her hangi bir para için accumulator'de yüzlerce yuvarlak yapı yer almaktadır bu yuvarlakların kesişimlerine bakacak olursak bir parıltı gözükmektedir algoritmada bu kesişim sayısına bağlı olarak bir yuvarlak belirleme işlemine gidildi, kesişim sayısı minumum yarıçap maksimum yarıçap gibi diğer özellikler ile beraber kullanıldı.

Python'da Hough Transform bir muhtemel motor konumlarını bulmak için kullanılmıştır amacı sadece görüntüdeki belirli spesifikasyonları sağlayan değerler için yuvarlaklar bulmak ve bu yuvarlakların konumlarını ve yarıçaplarını bir listede depolamak için kullanılmıştır bulunan bu olası motor yuvarlakları sayesinde görüntüde motor arama işlemi görüntünün tamamında değiş sadece bu olası konumlarda aranacaktır. Algoritma başlığı altında daha detaylı örnekler ile bir açıklama yapılmıştır.

Python'da cv2.HoughCircles(src, hough_type, dp, minDist, param1, param2, minRadius, maxRadius) fonksiyonu kullanılmıştır.

Src olarak kenarları bulunmuş görüntü verilmektedir, hough type olarak sadece tek metot olan cv2.HOUGH-GRADIENT kullanılmıştır, dp accumulator ile görüntü çözünürlüğü arasındaki ters orandır bu oran 1 olarak alınmıştır , minDist tespit edilecek yuvarlakların merkezleri arasındaki minumum mesafedir, param1 değişkeni kenar algılamayı işlemek için kullanılan gradyan değeridir, param2 ise accumulator'de elde edilecek yuvarlakların kesişmesine bağlı olarak belirlenen bir threshold parametresidir örneğin bu değer 20 veya 40 tutulursa ne elde edilir bunu Resim2_jpg ve Resim_4.jpg görüntülerine bakarak görebilirsiniz. Accumulator threshold parametresi küçük tutulmuştur bunun sebebi olası hiçbir motor konumunu kaçırmamaktır. minRadius ve maxRadius tespit edilecek olan yuvarlakların max ve min yarıçap bilgileridir.

Hough Transform uyguladıktan sonra elde edilecek parametreler kenarları bulunmuş görüntü kullanarak elde ediliyordu fakat daha sonra bu elde edilen yuvarlak konumları ve yarıçap bilgileri kullanılarak CLAHE + BLUR uygulanmış ve grayscale'de bulunan görüntünün bu piksel konumlarındaki küçük görüntü parçaları ayrı ayrı değerlendirilecek ve circle parçaları üzerinden bir threshold işlemi yaparak binary formata geçiş sağlanacaktır.

Yani burada Hough Transform Canny' yapılmış görüntüye uygulandı ve sonucunda muhtemel motor konumları bulundu ve bir array'e kaydedildi daha sonra bu array'deki piksel konumları gray-scale formatta olan görüntüde parçalar halinde alındı ve parça parça thresholding işlemi yapıldı bu işlem daha detaylı örnekler ile Algoritma başlığında anlatılacaktır.

F. Eşikleme Yöntemi (Thresholding)

Bir önceki yöntem ile muhtemel motor konumları bulunduktan sonra elde edilen konumlarda gray-scale görüntüde ilgili konumlarda bir thresholding yapılacaktır bu işlem daha detaylı algoritma başlığı altında adım adım anlatılacaktır. Eşikleme tekniği olarak Simple Thresholding tekniği kullanıldı ilgili görüntüde maksimum pikselin 20/100'üne bakarak bir threshold belirlendi.

G. Görüntünün Tersini Alma İşlemi (Invers)

Threshold yapıldıktan sonra elde edilen görüntüyü Resim_5.jpg' de görebilirsiniz. Burada görüleceği üzere motorlara odaklanacak olursak motorların iç kısmının neredeyse tamamen siyah olduğunu görebiliyoruz fakat etrafında bazı birleşimler görüyoruz bunlardan ayırıp sadece motorun siyahlıklarını ve arka planı ayırmak istiyorum. Öncelikle bu görüntünün tersini alıyoruz bunu Resim_6.jpg' de görebilirsiniz. Tersini kullanarak bazı morfolojik işlemler yapılacaktır bir sonraki başlıkta anlatılacaktır.

H. Görüntüye Morfolojik İşlemler Uygulanması (Morphological Operators)

Daha anlaşılır olması için Figure 2.9' a bakarak bir yorum yapabiliriz. İlk görüntü inverse yapılarak motorun iç kısmını

beyazlatan görüntüdür, bu görüntüde motor kısmının tamamen herhangi bir başka component ile bağlantısını istemiyoruz çünkü daha sonra kullanacağımız connected componnets tekniğinin iyi çalışabilmesi için her bir motoru arka plan ile ayrı tutmak ve ekstra diğer nesneler ile ayrı tutmak istiyoruz figürdeki ilk görüntüde motorun yuvarlak beyaz kısmı ile sağ taraftaki obje bitişik veya bitişmeye çok yakın tam anlaşılamıyor bu yüzden görüntüye bu aşamada bir erosion işlemi yapılması gerekmektedir buradaki motorun tamamen başka bir obje ile bağıntısını kesmeyi garantilemek için erosion yapılıyor hemen erosion ardından erosion yapılıp elede edilen görüntüye bir dilation yapılıyor ki aynı motor çapına geri dönebilelim.

Erosion olarak pythonda cv2.erode(src, kernel, iterations) ve dilation için cv2.dilate(src, kernel, iterations) fonksiyonları kullanılmıştır burada src yukarıda söylendiği gibi input görüntüleridir, kernel ise 3x3 boyuta sahip bir disktir, iterations ise kaç kere erosion veya dilation yapılmasını istiyorsanız oraya o değer yazılmaktadır biz her ikisi için de 1 seçtik.

I. Bağlı Bileşenler (Conencted Components)

Bir önceki yöntemde mümkün olduğunca motorları diğer komponentlerden ve aynı zamanda arka plandan ayırmaya çalıştık bu aşamada birbiri ile bağlantısı olmayan tüm componentler yani binary olarak 1 değerine sahip tüm componentlere etiketleme işlemi yapılarak componentlere bir etiket numarası verilecektir. Bu sayede motorların sahip olduğu etiketleri bulmaya çalışacağız.

Connected components mantığında algoritmada binary formatta dilation sonrası elde edilen birbirinden olabildiğince ayrı konumlanmış objeleri birbiri ile bağlantılı olanlara aynı etiket numarası vererek her bir farklı objeyi farklı etiket numarası vermeye dayalı bir yöntemdir. Figure 2.10'a bakarak bir yorum yapabiliriz, a görüntüsünde birbirinden ayrı olan binary 1 değerine sahip objeleri görebiliyoruz bu görüntülerin her birinde piksellerin birbiri ile olan komşuluk bağlantılarına bakarak bir etiketleme işlemi yapılıyor, komşuluk bağlantıları 4-connectivity ve 8-connectivity olarak ikiye ayrılır ilgili pikselin sadece sağ, sol, üst ve alt kısmındaki komşularını değerlendirmeye alıyorsak bu 4-connectivity bunların dışında sağ üst çapraz, sol üst çapraz, sağ alt çapraz , sol alt çapraz komşuluklarını da değerlendiriyorsak bu 8-conenctivity komşuluktur. Figüre 2.10'da b görüntüsünde 3 farklı objeye 3 farklı etiket numarası verilmiştir, yani birbirinden ayrı şekilde 3 farklı obje görüntüde yer almaktadır.

Bizim algoritmamızda etrafta çok fazla detay olduğundan bazı durumlarda motorların etraftaki objeler ile olan bağlantısı kesilemedi ancak mümkün olduğunca maksimum şekilde bağlantıları ayırmak için bir tasarım yapıldı.

Kendi giriş görüntülerinden örnek verecek olursak Resim_8'e bakarak bir yorum yapabiliriz, resimde motorların farklı bir renk ile gösterildiğini ve arkaplanın siyah olduğunu aynı zamanda etrafta bir çok birbiri ile bağlantılı component tespit edilmiştir fakat bizim amacımız olan motorları yuvarlak bir componnet şeklinde elde edebilme işlemi gerçekleşebilmiştir. Figure 2.9'a tekrar bakacak olursak burada erosion uygulanmasaydı sağdaki tek motoru etiketleme işlemi tehlikeye girebilecekti yuvarlak bir şekilde etiketleme mümkün olmayabilirdi

Algoritmada connected_component_label başlığı altında bir fonksiyon yazılmıştır ve bu fonksiyon sonucunda görüntüdeki tüm etiket gruplarını ve görüntünün representetive halini return edecektir representetive'den kastım renkli bir görüntü return ediliyor fakat renkli olmasının oradaki piksel değerleri ile bir alakası olmamasıdır. Daha sonra return edilen bu etiket gruplarını yorumlayacak bir algoritma geliştirilecektir, etiketlerin bazı özelliklerine bakarak motorların tespiti yapılmaya çalışılacaktır.

Python'da cv2.connectedComponents(src) fonksiyonu kullanarak binary giriş görüntüsünden etiketlenmiş grupları elde edebiliriz, daha detaylı açıklama Algoritma başlığı altındadır.

J. Bağlı Bileşenlerden Yuvarlak Tespiti

Elde edilen etiket gruplarının bazı özellikleri kullanılarak yuvarlak şekle sahip etiketlerin tespit edilmesi sağlanacaktır. Öncelikle bulunan birbiri ile bağlı olan componentlerin yani aynı etikete sahip olan etiket gruplarının görünümünü görebilmek için Resim-9 ve Resim-10' a bakınız. Resim 10'a bakarsak burada her bir etiket grubunun üzerine Mouse ile geldiğimizde bize farklı özelliklerini gösterecektir her bir etiket grubu için bu özellikler mevcuttur ve bu özelliklerden bazılarını kullanarak yuvarlak yani motor tespiti yapmak amaçlanmıştır.

Algoritmada kullanılan özellikler sırası ile şunlardır; eccentricity, area, major ve minör axis length'dir. Bu özellikler bir ön koşul olmuştur if else yapıları kullanarak bu özellikler altında belirli değeri sağlayan etiket gruplarının geçmesine diğerlerinin elenmesi sağlanmıştır. Buradaki figürü çizdirebilmek için plotly modülü içinde bazı fonksiyonlar kullanılmıştır, bir kaynak kodda görülüp bir benzerini buradaki algoritmaya göre değiştirip uygulanmıştır.

Bu başlık altında kullanılan Figürler:

Figure 2.1

Figure 2.2

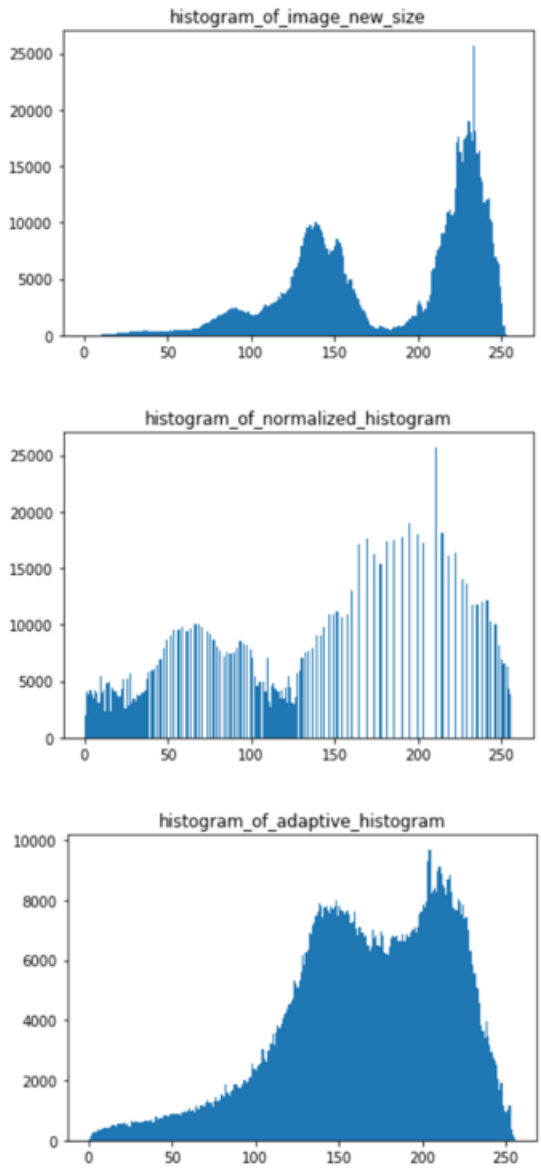

Figure 2.3

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

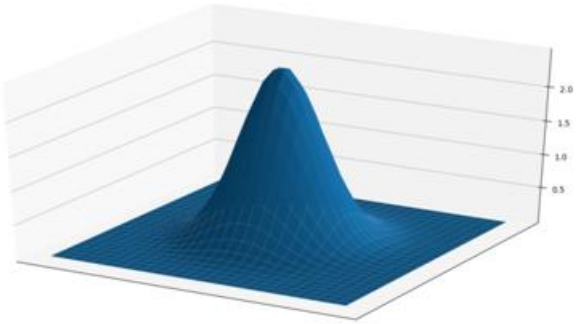

Figure 2.4

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

Sobel filters for both direction (horizontal and vertical)

$$|G| = \sqrt{I_x^2 + I_y^2},$$
$$\theta(x,y) = arctan\left(\frac{I_y}{I_x}\right)$$

Gradient intensity and Edge direction

Figure 2.5

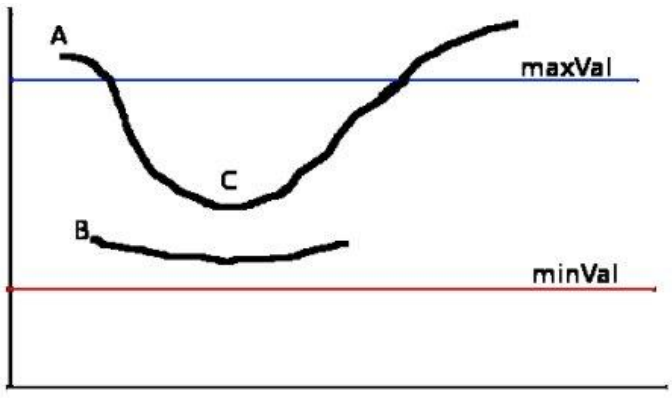

Figure 2.6

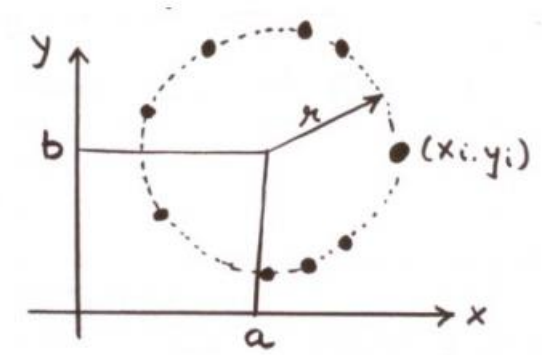

Figure 2.7

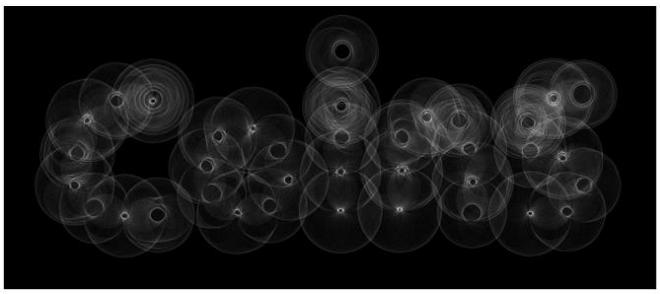

Figure 2.8

Figure 2.9

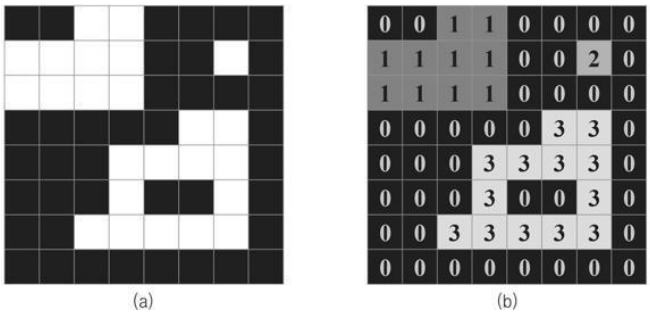

Figure 2.10

III. ALGORİTMA

Algoritma sözde kodu:

1.adım:

input_image = cv2.imread(‘name_of_image’, 0)

imread() fonksiyonu aracılığı ile kod ile aynı klasör içerisinde yer alan herhangi bir .pgm, .jpg, vs uzantılı dosyalar koda gri seviyeli formata çağırılabilir. ‘name_of_image’ çağırılmak istenen görüntünün ismidir ve 0 parametresi ise görüntünün gri seviyeli bir şekilde çağırılacağını bize söylemektedir.

2.adım:

image_new_size = cv2.resize

(input_image, (1280,800), interpolation = cv2.INTER_LINEAR)

resize() fonksiyonu ile giriş görüntüsünü yeniden boyutlandırabiliyoruz interpolasyon tipi olarak Linear Interpolation tercih edilmiştir detaylı bilgi Yöntemler-A başlığı altında verilmiştir.

3.adım:

obj = cv2.createCLAHE(clipLimit = 5.0, tileGridSize=(12,12))

histogram_out = obj.apply(image_new_size)

cv2.createCLAHE() aracılığı ile bir obje oluşturuluyor ve bu objeye apply member fonksiyonu uygulanıyor, burada clipLimit ve tileGridSize parametrelerine daha önce Yöntemler-B başlığı altında değinilmiştir. Bizim algoritmamızda clipLimit = 5.0 seçilmiştir daha yüksek seçilmesi durumunda kontrast artışı çok fazla olacaktır ve bu da beraberinde birçok sorunu getirecektir, tileGridSize = (12,12) seçilmiştir bunun sebebi 12’lik karolar ile bu işlem yapıldığında herhangi bir problem çıkmadı (8,8) veya (16,16) kullanınca pek bir fark görülmedi fakat, clipLimit parametresi arttırıldıkça kontrast artışı gözlemlendi dolayısı ile 5.0 gibi bir değer tercih edildi. Daha önce de bahsedildiği gibi burada gamma Correction ve Normalized Histogram Equalization teknikleri de denendi fakat iyi sonuçlar alınamadı.

4.adım:

image_blur = cv2.GaussianBlur(histogram_out, (5,5), 0)

cv2.GaussianBlur() fonksiyonu kullanılarak bir önceki adımda CLAHE sonucunda elde edilen iyileştirilmiş görüntüye bir yumuşatma işlemi uygulanmıştır. (5,5) değeri bize burada uygulanacak gaussian filtresinin boyutunu vermektedir (5,5) seçilmesi algoritmada için uygun bir seçim olmuştur (3,3) denenmiştir fakat istenilen kadar kenarlardaki geçişler yumuşatılamamış istenildiği kadar gürültü filtrelenememiştir.

5.adım: (8.adımda kullanılmaktadır)

image_to_loop = image_blur ataması yapılarak daha sonra image_to_loop değişkeni .. adımda kullanılacaktır. image_to_loop giriş görüntüsüne RESIZE + CLAHE + BLUR yapılmış formudur. İsminin böyle seçilmesinin sebebi kullanıldığı fonksiyonlarda çok fazla döngü olmasıdır.

6.adım:

İmage_edges = cv2.Canny(image_blur, 100, 140)

cv2.Canny() fonksiyonu aracılığı ile yumuşatılmış görüntüde kenarların tespiti hedeflenmiştir. Canny mümkün olabildiğince birbirinden kopuk olmayan edgeler bulmayı hedefleyen bir tekniktir algoritmada kenarları bulmak içi sadece bu teknik uygulanmıştır ve verimli de olmuştur diğer teknikler denenmemiştir. Burada Tao1 ve Tao2 değişkenlerinin ne olduğundan Yöntemler başlığı altında bahsedilmişti Tao1 ve Tao2’yi seçerken yaklaşık olarak en iyi bu parametreler kullanıldığında iyi bir sonuç verildiği gözlemlenmiştir bunun dışında 100,200 veya 60,200 gibi değerler girildiğinde de aynı sonucu verdiğini gözlemledim farklı parametreler verildiğinde kenarların daha detaylı veya daha az sayıda kenar bulunmasını bekleniyordu fakat CLAHE işlemi ile görüntünün kontrastı bazı görüntülerde aşırı iyileştirmeye sebep olduğundan bazı görüntülerde Tao1 ve Tao2 değişiminin çıkışı etkilemediği gözlemlendi.

7.adım:

marked_circles =

cv.HoughCircles(image_edges, cv.HOUGH_GRADIENT, 1, 80, param1 = 80, param2 = 20, minRadius = 15, maxRadius = 80)

cv2.HoughCircles() fonksiyonu aracılığı ile giriş görüntüsünün kenarları bulunmuş görüntüsünde olası motor konumları hesaplanmıştır. Tekrar Resim_2.jpg ve Resim_3.jpg’ye bakarak burada yapılan daha net anlaşılabilir. Elde edilen bu marked_circles datası bir tuple return eder. Tuple bünyesinde kaç adet circle bulduğu ve her bir circle’ın x,y merkez konumları ki bunlara kodda ccols ve crows denmiştir ve son olarak circle’ların çaplarını tutmaktadır. Elde edilen bu data daha sonra kullanılacaktır.

HoughCircles için parametreler param1 = 80 seçilmiştir bu parametrenin değişimi neredeyse hiçbir şey değiştirmeyecektir çünkü elde edilen kenarlar çok güçlüdür param1 elde edilen kenarların yoğunluğuna bakan bir parametredir, param2 = 20 seçilmesi aslında hough transform for circle için kötü bir seçimdir fakat burada accumulator mantığında üst üste gelen circle’ların çoğunu almak istiyorum ki bu circle’lar potansiyel motorun çeperi de olabilir. Amaç öncelikle motorun çeperini en kötü görüntüde dahi olsa bulabilmek ve onu değerlendirmeye alabilmektir dolayısıyla bu parametre düşük tutulmuştur, minRadius ve maxRadius’da aynı şekilde min için küçük bir değer verilmesi tüm görüntülerde motorların çaplarını bilemeyeceğimizden min’i küçük max’ı büyük tuttum.

Algoritmada Hough for circle için 2 farklı extra fonksiyon yazılmıştır bunlar circle_coordinates() ve Show_marked_circles()’ dir sırası ile bu fonksiyonlar marked_circles parametresi yollanarak çağırıldığında Resim_3 ve Resim_4.jpg çıktıları elde edilecektir.

8.adım:

Not: Fonksiyon parametre olarak 5.adımdaki image_to_loop görüntüsünü ve marked_circles değerlerini almaktadır.

def output_of_thresholded_marked_image(i_image, marked_circles_parameters):

rows, cols = shapes of input image

create tmp_image[rows, cols]

```
size = size of marked circles

for item in range(size):
    ccols = marked circle col
    rrows = marked circle row
    circle_diameter = diameter of circle

    part_row = 2 * diameter + 40
    part_col = 2* diameter + 40
    create empty part_image[part_row, part_col]

    dia = diameter
    for i in range(crows – dia – 20, crows + dia +20):
       for j in range(ccols – dia – 20,  ccols + dia + 20):
          if j > 1280 – dia or i > 800 – dia:
             pass
          else:
             tmp_image[i, j] = image_to_loop[i,j]

    for i_part, i zip(range(0, part_row), range(same with up))
       for j_part, j zip(range(0, part_col), range(,,,))
          if : same with up
             pass
          else:
             part_image[i_part, j_part] = tmp_image[i,j]

    T = np.amax(part_image)
    T = T * %20
for i_part, i …..
    for j_part, j …
          if ….
             Pass
          else:
             tmp_image[i,j] = part_image[i_part, j_part]
return tmp_image
```

Bu fonksiyonun yaptıklarını Örnek resimler ile anlatmak daha anlamlıdır. Bizim bir önceki adımda bulduğumuz tüm marked_circles'ları Resim_2.jpg'de görmüştük buradaki her bir circle'ı sırası ile çağırıp bu circle boyutlarına bazı eklemeler yapıp kodda tek tek işlemek amaçlanmıştır.

Öncelikle Fonksiyon algoritmada şu isim ile şu şekilde çağırılmıştır;

```
output_of_threshold_image = output_of_thresholded_marked_image(imaeg_to_loop, marked_circles_round)
```

Burada önemli noktalar fonksiyona girdi görüntüsü olarak 5.adımda belirlenen image_to_loop yollanacaktır 2.adımda belirlenen image yollanır ise kod düzgün çalışmayacaktır. Sırası ile Resim_2.jpg ardından Resim_3.jpg'yi inceledikten sonra, Resimlerden First_iteration_1, 2 ve 3 görüntülerini sırası ile açınız burada her bir iç içe for lopları sırası ile 1.görüntüyü 2. Ve 3. Görüntüyü verecektir daha anlaşılır olması için sırası ile görüntüleri kontrol ediniz burada yapılan ilk iterasyonun sonucu budur. Toplam size kadar yani Hough Circle Transform'un bulduğu muhtemel circle sayısı kadar bir iterasyon gerçekleşecektir. Resimlerden Sixth_iteration_1, 2 ve 3 görüntülerini sırası ile açınız ve buradan net bir şekilde görülecektir ki motorlar yavaş yavaş bulunup belirlenen threshold'a göre her bir görüntü parçacığı için aynı threshold tekniği uygulanacaktır ardından Resimlerden Tenth_iteration_1, 2 ve 3 görüntülerini sırası ile açınız burada da 3.motor belirecektir.

Bu şekilde fonksiyon yazma ihtiyacı görüntülerde muhtemelen circle olmayan yerler ile ilgilenmeden sadece muhtemel motorların bulunabileceği yerler ile ilgilenmektir. Threshold değeri her bir girdi parçasının buna part_image[] denmiştir maximum pixel değerinin yüzde 20'si kadar olacaktır bu kötü bir threshold belirleme tekniğidir fakat istenilen şey zaten motorların içindeki koyu alanın belirlenmesi onun diğer objelerden ayrılmasıdır.

Tüm iterasyonlar bittikten sonra Resim_5.jpg elde edikmiştir bize motorların siyah iç kısımlarının belirgin olduğu bir görüntü oluşturmuştur, fakat hem morfolojik operatörleri kullanmak hem de connected components tekniğinin çalışabilmesi için bu siyah kısımların beyaz olması gerekmektedir.

Her bir iç içe for yapısına bir if yapısı eklenmiştir bunun sebebi görüntünün dışına çıkabilecek pixel değerlerini engellemektir görüntüde olmayan bir pixel konumu for içerisinde kalmamalıdır.

9.adım:

Morfolojik operatörlerin kullanımı burada morfolojik operatör olarak ilk önce erosion kullanılmıştır Resim-6.jpg incelendiğinde motor kısımlarının kenarlarında başka objeler ile bağlantı gözükmektedir bu istemeyeceğimiz bir şey olduğundan bunu ortadan kaldırmak için erosion ardından aynı yapılandırma elemanı ile dilation yapılmaktadır.

Burada yapılan işlemi Figure 2.9'a bakarak tekrar hatırlayabiliriz.

10.adım:

```
connected_component_label(i_image):
    dilation yapıldıktan sonra oluian görüntüden connected_componnets'e geçir.
    number_of_labels, all_labels = cv2.connectedComponents(i_image)
    labeled_img için bir yöntem belirle ve etiketlenmiş görünyüuü representetive olarak göster.
```

return all_labels, labeled_img

label_groups = fonksiyonun return ettiği ilk parametre görüntüdeki tüm ediketlerin bulunduğu bir ndarray' yapısıdır.

Output_of_labeled_image = fonksiyonun return ettiği ikinci parametre görüntünün representetive formudur. Resim_8.jpg' ye bakara görebilirsiniz.

Burada fonksiyon yazılırken cv2.connectedComponents() fonksiyonu kullanılmıştır yollanan binary görüntüde birbiri ile bağlantıları olmayan tüm objeleri tespit eder ve onlara etiket değerleri atar.

11.adım:

Figür işlemleri:

İstediğimiz binary görüntütünün etiketlerini içeren array'i kullanarak etiketlerin bazı parametrelerini kullanacağız ve onları kıyaslayacağız.

Props = measure.regionprops() fonksiyonu aracılığı ile etiketlenen her bir etiket grubunda etiket gruplarının bazı parametrelerini bize yansıtılmasını sağlayan fonksiyondur. Motor belirleme işlemi için bize yuvarlak şekle sahip etiket grupları gerekmektedir. Yuvarlak şeklini 'eccentricity' parametresine bakarak bu parametreyi kullanarak etiketlenmiş grupların bazılarını eleyebiliriz eccentricity değeri düşük olan etiket grubu yuvarlak şekle daha yakın olmaktadır fakat sadece bu parametreye bakarak bir belirleme işlemi yapılamaz. Figure 3.1'de eccentiricty'nin yuvarlak objeler üzerindeki değerlerini görebilmekteyiz. Bunun dışında etiketin kapladığı alana ve major ve minor axis length leri arasındaki farka bakarak da bir belirleme işlemi söz konusu olacaktır.

Motor = 0

For index in range(1, label_groups.max()):

Label = props[index]

Look up eccentricity for label[index]

İf eccentricity < 0.70:

Look up area for label[index]

İf area > 600:

Look up differ major and minor for label[index]

İf major – minor < 20 :

Motor += 1

Tüm if yapılarından geçtikten sonra

contour = measure.find_contours()

fonksiyonu aracılığı ile bu şartları sağlayan etiket grubunun tüm y,x değerleri bu contour içerisinde tutulur ve tutulan bu y,x değerleri fig.add_trace(go.Scatter()) yapısında iteratif olarak çizdirilir. Örneğin ilk motoru bulduktan sonra for içerisinde bu çizim yapılır ardından 2.motor bulunduğunda tekrar çizim yapılır en son for bittikten sonra yapılan tüm çizimler çıktı olarak gösterilir. Resim_10.jpg'ye bakara bunu net görebiliriz.

Burada tüm adımlar açıklanmıştır algoritma en iyi cepheden çekilmiş motor görüntülerinde çalışacaktır yandan çekimlerde tam bir motorun iç yuvarlağı elde edilmesi söz konusu olamamaktadır. Dolayısı ile karşıdan bakıldığında +45, -45 derece açılara kadar bazı görüntülerde motor tespit işlemleri başarılı olmulştur anca +90 -90 gibi yandan bakışlarda motor bulamayacağız çünkü temel aldığımız şey olan motorların içerisindeki karanlık özelliğinden yararlanamıyor olacağız. Bunun dışında uçak görüntüsünün içinde uçak ile beraber insanların yer alması bu algoritmayı çalışamaz hale getirecektir örneğin Resim_12. Ve Resim 13.jpg ye bakarak bu durumu anlayabiliriz.

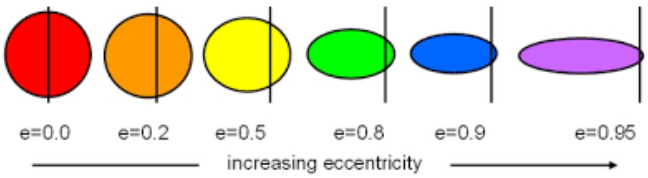

Figure 3.1

IV. DENEYLER VE TARTIŞMA

A. Veri Seti/Deneyler/Senaryolar

Kullanılan veri seti olması gerektiği gibi kolay, orta, Zor olarak 3 farklı gruba ayrılabilir. 17 adet deney setinin haricinde çeşitli harici görüntüler denenmiştir. 17 görüntü üzerinden bir verim tanımı yapacak olursak verilen 17 görüntüden 6'sında tam olarak doğru çalışmaktadır, 6'sında hiç çalışmamaktadır yani ya hiç motor bulamamaktadır ya da farklı objeleri motor olarak bulmaktadır, kalan 5 görüntüde ise motorlar bulunmuştur fakat motorlar ile beraber bazı başka görüntüler de bulunup onlar da saydırılmıştır. Bunun sebebi en son kısımda belirlenen eccentricity, area ve major – minor parametreleri olduğu düşünülebilmektedir veya bu aşamaya gelene kadar daha düzgün bir yuvarlak elde etmek de bu sorunu ortadan kaldırabilir. Daha düzgün bir yuvarlak elde etmek her görüntü için olası olmamaktadır cepheden bakılan görüntüler için gayet yüksek eccentricity'e sahip motorlar bulunmaktadır fakat farklı ekstrem açılardan bakılınca tam yuvarlak elde etme imkanı azalmaktadır bu da beraberinde motorun tespit edilememesini getirmektedir.

B. Sonuçlar

Algoritmaya uygulanan tüm sonuçların çıktısı gözlemlendi ve Outputs klasörü içerisinde zip dosyasının içerisinde paylaşıldı.

C. Tartışma

A başlığı altında da belirtildiği gibi elimizde olan 17 giriş görüntüsünden 6 tanesi sorunsuz, 6 tanesi çok yanlış cevaplar içeren ve kalan 5 tanesi de nerede ise doğru şeklinde çalışmaktadır. Sadece verilen input görüntülerine bakarak bir verim tanımı yapmak anlamsızdır ekstra görüntüler yükleyerek testler yapılmıştır yine görüntülerin zorluğuna ve uygunluğuna bakarak kusursuz çalışma yüzdesi %40'ı geçememektedir. (Not bu yüzde sadece cepheden çekilmiş görüntüler içermiyor veri setinde paylaşılan zorluklarda görüntüler içermektedir.)

Algoritmada bariz bir parametre değişince değişen sonuçlar bütünü vardır bu parametrelerden en başta gelen parametre eccentricity parametresidir neredeyse hiçbir zaman 0.30'un üstünde eccentricity'e sahip olan bir motor tespit edilememiştir. Eccentiricty'si 0.90 üstünde olan motorlar dahi zaman zaman karşımıza çıkmıştır fakat bu motorlar if'den geçemeyeceği için tespit edilememiştir. Örneğim output_7 görüntüsü için eccentricity 0.85 ve area < 500 yapıldığı an algoritma 2 motoru da bulabilmektedir. Her şeyden önce etiketleme yapmadan önce çok çok iyi bir yuvarlak elde edilememektedir hatta açının değişimine göre berbat yuvarlaklar elde edilebilmektedir. Örneğin output_9 görüntüsünde motoru diğer tüm objelerden aynı bir component gibi etiketleyebiliyoruz fakat elde edilen etiket yuvarlaktan çok çok uzak olduğu için o görüntüde bir motor bulamıyoruz. Son olarak output_14'e bakacak olursak bu görüntü input_14'de resize edilmemiş hali çok çok küçük bir görüntüdür resize yapmak görüntüde genişlemeye yol açmıştır bunun neticesinde istenilen motorlar tespit edilememiştir halbuki görüntünün ilk hali resize edilmemiş halinden görebiliriz ki görüntü istenilen tüm şartları sağlıyor cepheden çekilmiş ve motor kısımlarının iç tarafı koyu renkli.

Farklı yöntemler kullanarak görüntülerdeki sorunları bulma görüntülerde istenilen spesifikasyona göre obje tespiti yapılmıştır. Bunlar yapılırken belirli bir görüntüye bağlı kalınmamıştır genelleme yapılarak sorunlar çözülmeye çalışılmıştır.

V. SONUÇ

Sonuç olarak algoritmamız daha önce de bahsedildiği üzere en iyi karşıdan çekilmiş veya +45, -45 derece sağ veya soldan çekimlerde aynı zamanda motorun iç kısmının koyu olduğu görüntülerde çok iyi çalışabilmektedir. Önerdiğim algoritmada connected components yapana kadar ayrı bir yorum yapılacak olursa bu aşamaya kadar daha iyi bir motorların yuvarlaklıklarının anlaşılır daha düzgün olduğu bir binary görüntü elde etmek algoritmayı iyileştirebilir.

Algoritmayı iyileştirebilmek için motor içerisindeki koyu bölgenin mümkün olabildiğince yuvarlağa yakın bir şekilde elde etmeye çalışarak daha iyi bir verim elde edebiliriz. Burada önerim cv2.watershed() fonksiyonunu algoritmaya monte etmeye çalışabiliriz bu fonksiyon beraberinde üst üste gelmiş yuvarlakları tespit edebilecek bir fonksiyondur. Algoritmada bu denendi fakat beklenen sonuç elde edilemedi fakat daha önceden masa üzerinde para saydırma gibi konularda kullanılan bu fonksiyon o tarz işleri kusursuz yapabilmektedir.

VI. BİLGİLENDİRME

Bu raporu yaparken giriş kısmında da bahsedildiği gibi daha önce bu konu üzerinde çalışma yapılmamış olmasından dolayı buna yakın konular olan para saydırma ve meyve tespiti konuları üzerine yapılan araştırmalar incelendi ancak sadece 1 yerde bir kod parçası referans alındı kendi algoritmama uyarlandı. Uyarlanan kısım son bölümde plotly içerisinde yer alan fonksiyonlar bütünüdür iteratif bir figür çizdirme işlemlerinin bazıları başka bir koddan uyarlanmıştır. Bunun sebebi etiketlerin özelliklerini Mouse ile üzerine gelerek görebilmemizdir daha önce plotly ile çalışılmadığından bu kısım referans alındı. Direkt kopyala yapıştır yapılmadı sadece birkaç kısım bu algoritmaya göre düzenlendi.

KAYNAKÇA

[1] "OpenCV", generated on Sun Jan 3 2021
https://docs.opencv.org/master/da/d6e/tutorial_py_geometric_transformations.html

[2] "OpenGenus IQ"
https://iq.opengenus.org/connected-component-labeling/

[3] "sckit-image"
https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_regionprops.html

[4] "OpenCv", generated on Sun Jan 3 2021
https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html

[5] "Rafael C. Gonzalez University of Tennessee"
Digital Image Processing Third Edition

[6] "Joseph Howse and Joe Minichino"
Learning OpenCV 4 Computer Vision with Python 3 Third Edition

[7] "OpenCv", generated on Sun Jan 3 2021
https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html

[8] "OpenCv", generated on Sun 3 2021
https://docs.opencv.org/3.4/d3/de5/tutorial_js_houghcircles.html