

ANKARA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

EKSEN DIŞI PARABOLİK AYNALI KOLİMATÖR VE EĞİK KENAR
YÖNTEMİYLE MODÜLASYON AKTARIM FONKSİYONU ÖLÇÜM
SİSTEMİNİN TASARIMI VE GERÇEKLEŞTİRİLMESİ

Yiğit Can ACAR

FİZİK ANABİLİM DALI

ANKARA
2025

Her hakkı saklıdır

ÖZET

Yüksek Lisans Tezi

EKSEN DIŐI PARABOLİK AYNALI KOLİMATÖR VE EĐİK KENAR YÖNTEMİYLE MODÜLASYON AKTARIM FONKSİYONU ÖLÇÜM SİSTEMİNİN TASARIMI VE GERÇEKLEŐTİRİLMESİ

YiĐit Can ACAR

Ankara Üniversitesi
Fen Bilimleri Enstitüsü
Fizik Anabilim Dalı

Danışman: Prof. Dr. Şengöl KURU

Eő Danışman: Dr. Öğr. Üyesi. Özgür SELİMOĐLU

Bu tez çalışmasında optik ve elektro-optik sistemlerin görüntüleme kapasitelerinin temel bir ölçüm kıstası olan modülasyon aktarım fonksiyonu (MTF) ölçüm sisteminin tasarımı ve deneysel olarak testleri gerçekleştirilmiştir. Yapay (sentetik) bir eğik kenar görüntüsünün yazılan Python kodlarıyla MTF grafiĐinin çıkarılmasıyla başlayan süreç, kurulan deneysel düzeneklerden alınan sonuçların birbirinden baĐımsız yöntemler ile kontrol edilmesiyle devam etmiştir. Tasarlanan sistem ve yazılan kodlar, yapısal olarak iki farklı yöntem ve Zemax ortamında gerçekleştirilen bir simülasyondan gelen sonuçlar ile bilimsel metriklere uygun bir şekilde sınanmış ve birbirleriyle uyumluluk gösteren daĐılımlar elde edilmiştir.

Ocak 2025, 89 sayfa

Anahtar Kelimeler: Optik tasarım, Kolimatör, Parabolik Ayna, Modülasyon Aktarım Fonksiyonu (MTF)

ABSTRACT

Master Thesis

DESIGN AND REALIZATION OF MODULATION TRANSFER FUNCTION MEASUREMENT SYSTEM WITH AN OFF-AXIS PARABOLIC MIRROR COLLIMATOR AND SLANTED EDGE METHOD

Yiğit Can ACAR

Ankara University
Graduate School of Natural and Applied Sciences
Department of Physics

Supervisor: Prof. Dr. Şengül KURU

Co-supervisor: Asst. Prof. Dr. Özgür SELİMOĞLU

In this thesis, design and experimental testing of modulation transfer function (MTF) measurement system, which is a basic measurement criterion for the imaging capabilities of optical and electro-optical systems, have been carried out. The process, which started with obtaining MTF graphs of an artificial (synthetic) slanted edge image using Python codes, continued with the validating of the results obtained from experimental setups with independent methods. Designed system and written codes have been tested in accordance with scientific metrics with the results from two structurally different methods and a simulation performed in Zemax environment. Finally, distributions that are compatible with each other have been obtained.

January 2025, 89 pages

Keywords: Optical Design, Collimator, Parabolic Mirror, Modulation Transfer Function (MTF)

TEŞEKKÜR

Eğitim hayatım boyunca benden desteğini esirgemeyen aileme, öğretilerime ve Ankara Üniversitesi'nde bulunduğum zaman süresince ilminden faydalandığım bana her konuda ışık tutan değerli hocalarım Prof. Dr. Şengül KURU (Ankara Üniversitesi Fizik Anabilim Dalı) ve Dr. Öğr. Üyesi Özgür Selimoğlu (Ankara Üniversitesi Enerji Sistemleri Mühendisliği Anabilim Dalı)'na sonsuz teşekkürler.

Yüksek lisans öğrenimim boyunca BİDEB 2210-A Genel Yurt İçi Yüksek Lisans Burs Programı aracılığıyla kurumsal desteğini almış olduğum Türkiye Bilimsel ve Teknolojik Araştırma Kurumu TÜBİTAK'a içtenlikle teşekkürlerimi sunarım. Tez çalışmalarım sırasında laboratuvarlarını kullanma imkânı sunan POLOPTECH Optik ve Uzay Sanayi firmasına ve çalışanlarına teşekkür ederim.

Yiğit Can ACAR
Ankara, Ocak 2025

İÇİNDEKİLER

TEZ ONAYI	
ETİK.....	i
ÖZET.....	ii
ABSTRACT.....	iii
TEŞEKKÜR	iv
KISALTMALAR DİZİNİ	vi
ŞEKİLLER DİZİNİ	vii
1. GİRİŞ.....	1
2. OPTİKSEL TRANSFER FONKSİYONU	3
2.1 Dürtü Tepkisi.....	3
2.2 Optiksel Dürtü Tepkisi	4
2.3 Transfer Fonksiyonu.....	5
2.3.1 Optik transfer fonksiyonun doğrusal uzamsal değişmezliği	5
2.4 Modülasyon Aktarım Fonksiyonu	7
2.5 Eğik Kenar Yöntemi	8
2.5.1 Veri yumuşatma yöntemi	10
2.5.1.1 Gradyan açısı bulma yöntemi.....	14
2.5.1.2 Kenar yayılım fonksiyonu çıkarımı.....	17
2.5.2 Python ile nümerik türev alma	22
2.5.3 Hamming penceresi.....	24
2.5.4 Tek frekansta Fourier dönüşümü.....	29
2.5.5 Eğri uydurma yöntemi.....	34
2.6 Zemax Görüntü Simülasyonu	40
3. DENEYSEL ÇALIŞMALAR	42
3.1 Karşılaştırmalı MTF Analizi.....	52
3.2 Python ile Arayüz Uygulaması.....	59
3.2.1 Otomatik kenar tespit algoritması.....	60
4. SONUÇ VE TARTIŞMA	66
KAYNAKLAR	67
EKLER.....	68
EK1 Veri Yumuşatması ile MTF Kodları.....	68
EK2 Eğri Uydurma ile MTF Kodları.....	75
EK3 Otomatik Kenar Tespit Algoritması.....	83
ÖZGEÇMİŞ.....	89

KISALTMALAR DİZİNİ

ESF	Kenar Yayılım Fonksiyonu (<i>Edge Spread Function</i>)
PSF	Nokta Yayılım Fonksiyonu (<i>Point Spread Function</i>)
LSF	Çizgi Yayılım Fonksiyonu (<i>Line Spread Function</i>)
MTF	Modülasyon Aktarım Fonksiyonu (<i>Modulation Transfer Function</i>)
OTF	Optik Aktarım İşlevi (<i>Optical Transfer Function</i>)
LTI	Doğrusal Zamanla Değişmez (<i>Linear Time Invariant</i>)
LSI	Doğrusal Uzamsal Değişmez (<i>Linear Shift Invariant</i>)
ROI	İlgili Çalışma Bölgesi (<i>Region of Interest</i>)
SWIR	Kısa Dalga Kızılötesi (<i>Short Wave Infrared</i>)
GUI	Kullanıcı Arayüzü (<i>Graphical User Interface</i>)

ŞEKİLLER DİZİNİ

Şekil 2.1 Optik sistem birim dürtü girdi fonksiyonu	4
Şekil 2.2 Obje düzlemindeki delta fonksiyonu görüntü düzleminde bulanık bir fonksiyonla eşleşir (Boreman, 2001).	4
Şekil 2.3 Örnek bir optik sistem yapısı	5
Şekil 2.4 İdeal ve gerçek durumu gösteren MTF grafikleri	7
Şekil 2.5 Eğik kenar analizinin matematiksel gösterimi	9
Şekil 2.6 Piksel izdüşümünün grafiksel gösterimi	9
Şekil 2.7 Kenar açısının örnek bir gösterimi	10
Şekil 2.8 Faydalı Python kütüphaneleri	10
Şekil 2.9 İlgili görüntünün Python ortamında açılması	11
Şekil 2.10 Eğik kenar görüntüsü	11
Şekil 2.11 Matris üzerinden ROI seçimi	12
Şekil 2.12 Eğik Kenar noktalarının tespiti	12
Şekil 2.13 Lineer regresyon yöntemi ile açı hesabı	13
Şekil 2.14 Eğik kenar açı değeri	13
Şekil 2.15 Imatest görüntü analiz çıktısı	13
Şekil 2.16 Eğik kenar açısının hesabı	14
Şekil 2.17 “x” yönünde türev alan kod parçası	15
Şekil 2.18 “y” yönünde türev alan kod parçası	15
Şekil 2.19 Eğik kenarın açı hesabı	16
Şekil 2.20 Hesaplanan açı değeri	16
Şekil 2.21 Imatest görüntü programı çıktısı	16
Şekil 2.22 Kenar yayılım fonksiyonu şematiği (Zhang, H vd. 2018)	17
Şekil 2.23 Python’da kenar yayılım fonksiyonu algoritması	18
Şekil 2.24 Kenar yayılım fonksiyonu (ESF)	19
Şekil 2.25 Kenar yayılım fonksiyonunun belirli bir aralıkta çizdirilmesi	19
Şekil 2.26 Veri yumuşatma yöntemi için kullanılan kenar yayılım fonksiyonu algoritması	20
Şekil 2.27 Kenar yayılım fonksiyonunun yumuşatılması için yazılan kod parçası	20
Şekil 2.28 Kenar yayılım fonksiyonunun yumuşatılmış hali	21
Şekil 2.29 Yumuşatılmış ESF grafiğinin türevini alan kod parçası	23
Şekil 2.30 Çizgi yayılım fonksiyonu (LSF)	23
Şekil 2.31 Hamming penceresi grafiği	24
Şekil 2.32 Hamming penceresinin Python kodları	25
Şekil 2.33 Hamming penceresi sonrası LSF	25
Şekil 2.34 Fourier dönüşümü için yazılan Python kod parçası	26
Şekil 2.35 Modülasyon aktarım fonksiyonu	26
Şekil 2.36 MTF değerinin frekans tablosu	27
Şekil 2.37 Imatest MTF grafiği çıktısı	27
Şekil 2.38 Imatest görüntü analiz çıktısı	28
Şekil 2.39 Sinüsoidal fonksiyon MATLAB kod parçası	30

Şekil 2.40 Zaman alanında sinüs sinyali	30
Şekil 2.41 Sinüsoidal sinyalin Fourier dönüşümü	31
Şekil 2.42 Tek bir frekansta Fourier dönüşümü için yazılan MATLAB kod parçası	31
Şekil 2.43 Tek bir frekansta Fourier dönüşümü fiziksel çıktısı	32
Şekil 2.44 Tek bir frekansta Fourier dönüşümü	33
Şekil 2.45 MTF sonuçları ve tek frekansta Fourier dönüşümü	33
Şekil 2.46 Tek bir frekansta hesaplanan Fourier dönüşümü	34
Şekil 2.47 Spyder ortamında Fermi fonksiyonunun Python kodlanmış hali	35
Şekil 2.48 Optimizasyon için yazılan kod parçası	35
Şekil 2.49 Veriye uydurulan eğri	36
Şekil 2.50 Kenar yayılım fonksiyonu	36
Şekil 2.51 Python kodlarıyla türev alma işlemi	37
Şekil 2.52 Çizgi yayılım fonksiyonu	37
Şekil 2.53 Fourier dönüşümü alan kod parçası	37
Şekil 2.54 Modülasyon aktarım fonksiyonu	38
Şekil 2.55 Eğri uydurma yöntemi ile MTF değerinin frekans tablosu	38
Şekil 2.56 Imatest MTF grafiği çıktısı	39
Şekil 2.57 İdeal mercekli kolimatör teleskop yapısı	40
Şekil 2.58 Zemax Lens Data Editor	41
Şekil 2.59 Zemax ortamında görüntü simülasyonu	41
Şekil 3.1 Işık homojenleştirici küre (integrating sphere)	42
Şekil 3.2 Sadece küre ile aydınlatılan görüntü	43
Şekil 3.3 Küre ve teflon malzeme ile aydınlatılan düzenek	43
Şekil 3.4 Teflon malzemenin görüntü üzerindeki etkisi	44
Şekil 3.5 Kenar yayılım fonksiyonu (ESF)	44
Şekil 3.6 Çizgi yayılım fonksiyonu (LSF)	45
Şekil 3.7 Modülasyon transfer fonksiyonu (MTF)	45
Şekil 3.8 50mm f/1.4 objektif	45
Şekil 3.9 Basler kamera	46
Şekil 3.10 Kamera tutucu	46
Şekil 3.11 Basler kameranın monte edilmiş hali	47
Şekil 3.12 Karanlık ortam laboratuvar düzeneği	47
Şekil 3.13 Yeni küre ile aydınlatılan görüntü	48
Şekil 3.14 Yeni küre ile kenar yayılım fonksiyonu	48
Şekil 3.15 Yeni küre ile çizgi yayılım fonksiyonu	48
Şekil 3.16 Yeni küre ile modülasyon aktarım fonksiyonu	49
Şekil 3.17 SWIR kamera ile kurulan deneysel düzenek	49
Şekil 3.18 Kameradan elde edilen görüntü	50
Şekil 3.19 Kenar yayılım fonksiyonu	51
Şekil 3.20 Çizgi yayılım fonksiyonu	51
Şekil 3.21 Modülasyon aktarım fonksiyonu	51
Şekil 3.22 SWIR kamera ölçüm düzeneği	53
Şekil 3.23 Deneysel görüntü	53
Şekil 3.24 SWIR kamera kenar yayılım fonksiyonu (ESF)	54
Şekil 3.25 SWIR kamera çizgi yayılım fonksiyonu (LSF)	54

Şekil 3.26 SWIR kamera modülasyon aktarım fonksiyonu (MTF)	54
Şekil 3.27 Duvara asılı eğik kenar hedefi ve SWIR kamera.....	55
Şekil 3.28 Duvar aydınlatması ve MTF ölçümü	56
Şekil 3.29 Duvara asılı deneysel görüntü.....	56
Şekil 3.30 Duvar aydınlatması kenar yayılım fonksiyonu (ESF)	57
Şekil 3.31 Duvar aydınlatması çizgi yayılım fonksiyonu (LSF)	57
Şekil 3.32 Duvar aydınlatması modülasyon aktarım fonksiyonu (MTF)	57
Şekil 3.33 Birbirleriyle ilintili olmayan 3 farklı yöntemin çıktıları	58
Şekil 3.34 MTF ölçüm sistemi arayüzü (GUI)	59
Şekil 3.35 Duvar aydınlatmasından gelen görüntü	61
Şekil 3.36 Duvar aydınlatması	61
Şekil 3.37 Sobel filtre uygulaması	62
Şekil 3.38 Sobel filtre uygulaması sonraki görüntü	62
Şekil 3.39 Görüntü normalizasyonu.....	63
Şekil 3.40 Normalize edilmiş Sobel görüntüsü.....	63
Şekil 3.41 Eşik algoritması uygulanması sonucu elde edilen görüntü.....	64
Şekil 3.42 Bağlı bileşen analizi.....	64
Şekil 3.43 ROI merkez koordinatların belirlenmesi	65
Şekil 3.44 Maksimum kenar görüntüsü	65

1. GİRİŞ

Günümüzde optik, optronik ve elektro-optik cihazların önemi ve yaygın kullanımı giderek artmaktadır. Kameralar, sensörler, mercekler, teleskoplar vb. aygıtlar sadece askeri alanda değil sivil hayatta da karşımıza çıkmaktadırlar. Dolayısıyla bu tür aletlerden görüntüler alıp işleyebilmek ve kaliteli görüntüler elde edebiliyor olmak, mühendis ve fizikçilerin çözmesi gereken bir gerçek dünya problemi haline gelmiştir. Yükselen teknolojiyle birlikte kaliteli görüntüleme tekniklerine olan ihtiyaç, uzay/uydu görüntüleme sistemlerinden tıbbi uygulamalara kadar kendisini geniş bir spektrumda göstermektedir (Fiete, 2010).

Elektro-optik kameraların görüntü kalitesini belirleyen en önemli parametrelerden birisi modülasyon aktarım fonksiyonu (MTF)'dur. Bir görüntüleme sisteminin MTF ölçümü yapılarak, sistemin tasarımı, üretimi, hizalaması ve odaklamasının performans değeri elde edilebilir (Smith, 1990). Bu tez kapsamında görünür ile kısa kızıl-ötesi bantları kapsayan bir kolimatör tasarımını içeren örnek bir test sisteminin ve kameralardan alınan görüntüyü otomatik olarak analiz eden açık kaynak kodlu (Python) bir MTF ölçüm yazılımının geliştirilmesi amaçlanmıştır.

Görüntü kalitesi ölçümü sayesinde sahada kullanılan sorunlu optik veya elektro-optik cihazların tespiti, yeni geliştirilen cihazlardaki iyileştirmelerin ortaya konulabilmesi ve ürün kabullerinin daha güvenilir kıstaslara göre yapılması mümkün olmaktadır. Bu çalışmayla birlikte birçok alanda kullanılan kameraların görüntü kalitelerini ölçebilen bir sistem geliştirilecek ve optik sistemlerdeki hatalar, niceliği ölçülerek ortaya konulabilecektir. Kameraların görüntü kalitelerinin ölçülmesi sayesinde optik endüstrisindeki kaliteli ürün geliştirilmesi konusunda rekabetçilik artacak ve daha kaliteli ürünlere ulaşım imkânı doğacaktır. Her geçen gün daha geniş alanda çalışan optik sistemlere hitap etmek amacıyla geliştirilecek sistem, görünür-yakın kızılötesi (VNIR) ve kısa kızıl-ötesi (SWIR) bantta çalışabilir olacaktır.

Hazırlanan bu tez çalışmasının ikinci bölümünde optik sistemlerin teorik ve matematiksel modellemeleri hakkında bilgiler sunulmuştur. Evrişim ve Fourier dönüşümünün doğası hakkında bilgiler verilmiştir. Tezin ikinci bölümü birden fazla alt başlığa bölünmüş olup tez kapsamında yapılmak istenen çalışmaların özü burada açıklanmıştır. Burada, tez çalışmasının en temel ölçülmek istenen parametresi olan modülasyon aktarım fonksiyonunun (MTF) tanımlanması yapılmış olup ilk olarak yapay (sentetik) bir görüntü üzerinden MTF çıkarımı yapan Python kodları anlatılmıştır. Yazılan Python kodlarının ekran görüntüleri alınarak tek tek her parça ve bu parçaların çıktıları kullanım amaçlarıyla birlikte detaylıca anlatılmıştır. MTF ölçümünde kullanılan en temel yöntemlerden biri olan eğik kenar yöntemi (Slanted Edge Method) anlatılmış ve kodlamalar bu algoritmaya göre yapılmıştır. Temel algoritmaya bağlı kalacak şekilde iki farklı yöntem ile kodlama yapıp elde edilen sonuçlar kıyaslanmıştır. Daha sonrasında sistemin optik tasarımı Zemax OpticStudio programı kullanılarak yapılmış ve görüntü simülasyonu gerçekleştirilmiştir.

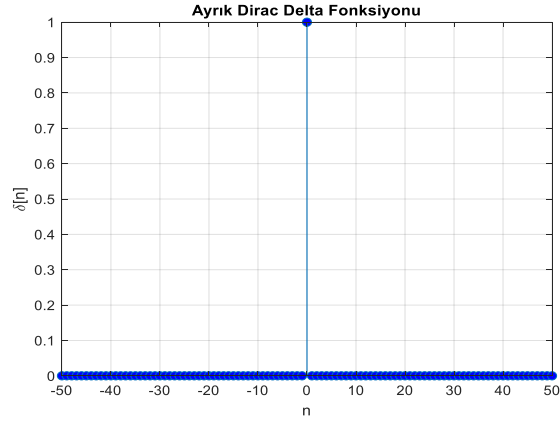
Tezin üçüncü bölümünde kurulan deneysel düzenekler anlatılmıştır. Deneysel düzeneklerden alınan örnek görüntülere yer verilmiş ve bu görüntülerden yazılan kodlar aracılığıyla MTF grafikleri çıkarılmıştır. Birbirinden optik açıdan farklı şekillerde kurulan deneysel düzeneklerin MTF çıktıları Zemax simülasyonundan gelen sonuçlar da dâhil edilerek karşılaştırmalı olarak analiz edilmiştir. Kameradan alınan görüntüyü doğrudan bir arayüz ekranına yansıtan ve üzerine tıklanılan butonlar sayesinde otomatik MTF ölçümü yapabilen bir arayüz uygulaması geliştirilmiş ve tezin yine üçüncü bölümü altında sunulmuştur. Tezin dördüncü bölümünde yapılan çalışmaların sonuçları tartışılmış ve son kısımda EK'ler bölümünde hesaplamalarda kullanılan kodlar sunulmuştur.

2. OPTİKSEL TRANSFER FONKSİYONU

Optik bir sistemin birim nokta girişine verdiği dürtü tepkisinin frekans alanında ifade edilmiş haline optiksel/optik transfer fonksiyonu denir. Optik transfer fonksiyonu aynı zamanda bir görüntüleme sisteminin frekans tepkisini gösteren kompleks değerli bir matematiksel ifade olarak da tanımlanabilmektedir. Giriş-çıkış ilişkilerinin zaman veya uzamsal alanda betimlendiği optik sistemlerin performanslarını frekans alanında değerlendirmede oldukça işlevsel olan bir fonksiyondur (Saleh ve Teich, 1991). Doğrusal sistemler teorisi, optik ve elektro-optik sistemleri analiz etmek için çeşitli imkânlar sunmaktadır. Sistemin uzamsal dürtü yanıtı, uzamsal frekanslı optik transfer fonksiyonunu elde etmek için Fourier dönüşümüne uğratılır. Görüntü kalitesi kavramını frekans alanında basitçe ifade etmek, kendi başına herhangi bir yeni bilgi üretmemektedir. Bununla birlikte, bakış açısındaki kavramsal değişiklik özellikle birkaç alt sistemin birleştiği ortak durumda bir görüntüleme sisteminin davranışına ilişkin bilgi edinilmesini sağlar. Bu durumun temel sebebi artık bir nokta boyutu yerine bir frekans tepkisinin dikkate alınmasıdır. Sisteme ait genel bir transfer fonksiyonu elde etmek için alt bileşenlerin transfer fonksiyonları çarpılmaktadır. Bu prosedür, uzamsal alan analizi için gerekli olacak tekrarlanan evrişimlerden (harmanlama) daha kolaydır ve alt sistemlerin her birinin performansı açısından toplam sistemin performans sınırlamalarının anında görselleştirilmesine izin verir. Temelde optik aktarım işlevi (OTF: Optical Transfer Function), bir görüntüleme sisteminin yanıtını uzamsal frekansın bir işlevi olarak tanımlayan karmaşık değerli bir fonksiyondur (Boreman, 2001).

2.1 Dürtü Tepkisi

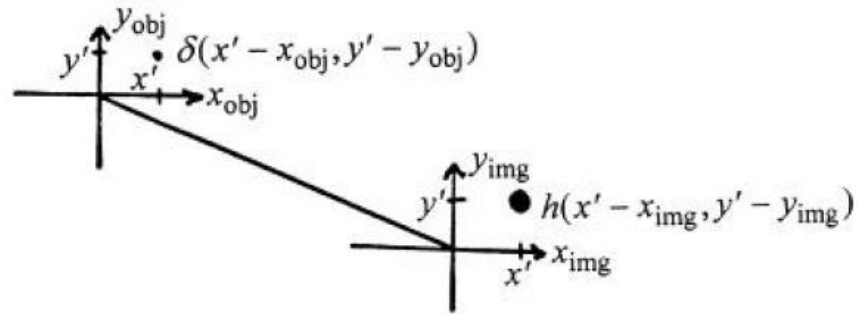
Dürtü tepkisi sinyal ve görüntü işlemede oldukça önemli ve temel bir kavramdır. Bir sistemin dürtü tepkisi, sistemi karakterize eden bir fonksiyondur. Doğrusal zamanda değişmez (LTI: Linear Time Invariant) ve doğrusal ve konum (kayma) değişmez (LSI: Linear Shift Invariant) sistemler için dürtü tepkisi tanımlamak mümkündür. Sistemin girişine zamanda neredeyse sıfır genişlikte, sonsuz büyüklükte ve birim alanlı (Dirac Delta: $\delta(t)$) bir giriş uygulandığında sistemin verdiği çıkış, tam anlamıyla birim dürtü tepkisi olarak tanımlanır (Şekil 2.1).



Şekil 2.1 Optik sistem birim dürtü girdi fonksiyonu

2.2 Optiksel Dürtü Tepkisi

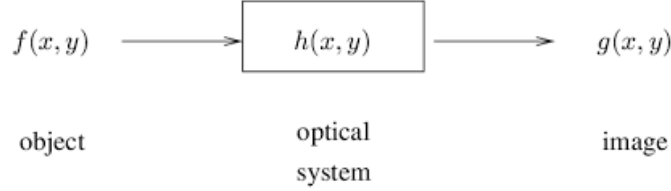
Optiksel dürtü tepkisi, optik bir sistemin ideal durumdaki en temel yapısını ifade eder. Obje düzlemine nokta boyutunda bir cisim konulduğunda sistemin bu noktaya verdiği yanıtı yani görüntü düzlemindeki halini betimler. Bir diğer deyişle darbe yanıtı veya dürtü tepkisi, bir optik sistemin oluşturabileceği en küçük görüntü detayıdır. Obje düzlemindeki nokta, görüntü düzleminde bulanık bir noktaya karşılık gelecektir. Bu durum yani darbe tepkisinin sonlu genişliği, ışığın kırınımının ve optik sistemin sebep olduğu bozunumların (aberrasyonların) sonucudur ve fizik kuralları gereği sistemin doğasında mevcuttur (Şekil 2.2).



Şekil 2.2 Obje düzlemindeki delta fonksiyonu görüntü düzleminde bulanık bir fonksiyonla eşleşir (Boreman, 2001).

2.3 Transfer Fonksiyonu

Optik bir sistemde giriş-çıkış ilişkileri Şekil 2.3'teki bir görsel ile ifade edilebilmektedir:



Şekil 2.3 Örnek bir optik sistem yapısı

Şekil 2.3'te bir şematiği verilen düzenekte uzamsal alanda (domain) çalışma yapılmaktadır. Buna göre, $f(x, y)$ objeyi, $h(x, y)$ optik sistemin birim dürtü (nokta) fonksiyonunu (PSF: Point Spread Function) ve $g(x, y)$ fonksiyonu ise görüntüyü temsil etmektedir. Bu üç fonksiyon arasındaki ilişki aşağıdaki gibidir:

$$g(x, y) = f(x, y) \otimes h(x, y)$$

Burada $\otimes$ işareti, evrişim anlamına gelmektedir.

2.3.1 Optik transfer fonksiyonunun doğrusal uzamsal değişmezliği

Zamanda evrişim frekans alanında basit bir çarpıma denktir. Göstermek istediğimiz eşitlik aşağıdaki gibi olsun:

$$y(t) = h(t) \otimes x(t) \Leftrightarrow Y(jw) = H(jw)X(jw)$$

$F\{y(t)\} = Y(jw)$ olacak şekilde

$$Y(jw) = \int_{-\infty}^{+\infty} y(t) * e^{-jw t} dt$$
$$Y(jw) = \int_{-\infty}^{+\infty} [h(t) * x(t)] * e^{-jw t} dt$$

Yukarıdaki integral içerisinde evrişim integralinin tanımından faydalanılarak bir integral daha yazıldığında,

$$Y(j\omega) = \int_{-\infty}^{+\infty} \left[\int_{-\infty}^{+\infty} x(\tau) * h(t - \tau) d\tau \right] * e^{-j\omega t} dt$$

$$Y(j\omega) = \int_{-\infty}^{+\infty} x(\tau) \left[\int_{-\infty}^{+\infty} h(t - \tau) * e^{-j\omega t} dt \right] d\tau$$

$$Y(j\omega) = \int_{-\infty}^{+\infty} x(\tau) [H(j\omega) * e^{-j\omega \tau}] d\tau$$

$$Y(j\omega) = H(j\omega) \int_{-\infty}^{+\infty} x(\tau) e^{-j\omega \tau} d\tau$$

$$Y(j\omega) = H(j\omega)X(j\omega)$$

Böylece zamanda veya uzayda evrişimin (harmanlamanın) frekans alanında basit bir çarpıma dönüştüğü gösterilmiştir. Bu özellik, hesaplamalarda oldukça önemli bir avantaj sağlamaktadır.

Bu durumda yukarıda verilen görüntü eşitliğinin her iki tarafının da Fourier dönüşümü alındığında üzerinde kolaylıkla işlemlerin yapılabileceği fonksiyonlar elde edilmiş olunur. Frekans alanındaki yeni eşitlik aşağıdaki gibidir:

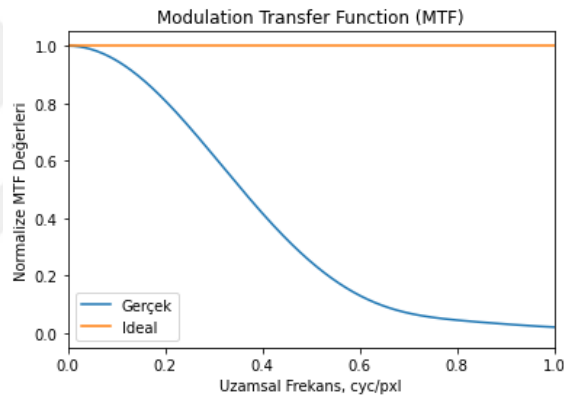
$$G(f_x, f_y) = F(f_x, f_y) * H(f_x, f_y)$$

Yukarıdaki denklemde yer alan $H(f_x, f_y)$ fonksiyonu optik sistemin transfer fonksiyonu veya sistemin frekans tepkisi olarak adlandırılır. f_x, f_y 'ler ise x ve y yönlerindeki uzamsal frekansları göstermektedir.

2.4 Modülasyon Aktarım Fonksiyonu

Modülasyon Aktarım Fonksiyonu (MTF) optik sistemin performansını belirlemede kullanılan önemli bir fiziksel parametredir. Temelde bir optiğin görüntüyü karşı tarafa hangi netlik ve keskinlikte aktardığının bir ölçüsüdür. İdeal durumda MTF'nin değeri 1'e eşittir (Robinet vd. 2018).

MTF için *Ayrıntı Aktarım Fonksiyonu* tanımlaması da yapılabilmektedir. Görüntüye ait ince detayların optik sistemden geçtikten sonra karşı tarafa ne ölçüde aktarıldığını frekans alanında betimler (Şekil 2.4).



Şekil 2.4 İdeal ve gerçek durumu gösteren MTF grafikleri

Optiksel transfer fonksiyonu bir genliğe ve kompleks üstel bir fonksiyon olacak şekilde bir de faz terimine sahiptir. MTF fonksiyonu burada genlik ifadesine karşılık gelmektedir.

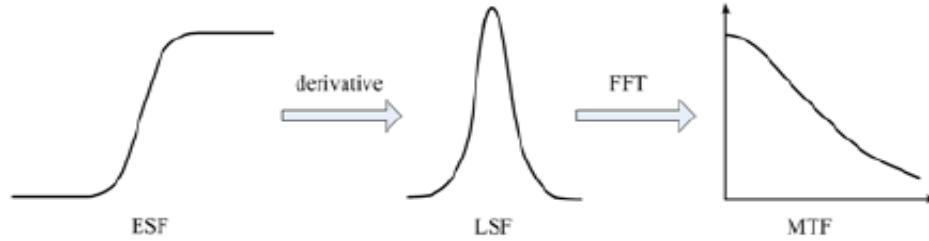
$$OTF \equiv H(\xi, \eta) = |H(\xi, \eta)|e^{-j\theta(\xi, \eta)}$$

$$MTF \equiv |H(\xi, \eta)|$$

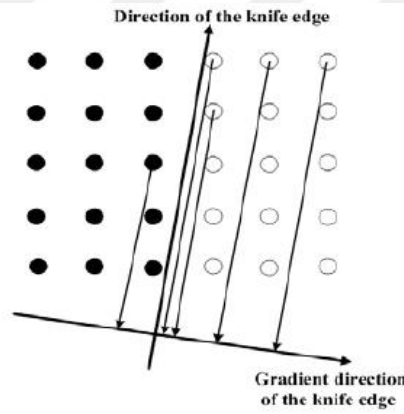
2.5 Eğik Kenar Yöntemi

Bu yöntem, optik görüntüleme sistemlerinin MTF ölçümünde kullanılan temel analiz metotlarından bir tanesidir. Temelde belli bir açıda eğime sahip bir görüntünün kenar analizinin yapılmasına dayanır. Bir optik sistemin eğik ve keskin bir kenara sahip objeye verdiği tepki görüntüsü üzerinden örnekleme işlemi yapılır. Bu örnekleme işleminden önce görüntü üzerinde çalışılacak bölge seçilir (ROI: Region of Interest). Bu bölge içerisinde görüntü üzerindeki keskin kenara karşılık gelen doğrunun eğim açısına ihtiyaç duyulmaktadır. Bu açı, ilgili matrisin her bir satırı için sırasıyla türevler alınıp piksel altı hassasiyetine inilerek keskin kenar eğrisinin görüntüyle kesiştiği merkez noktaların tespit edilmesiyle hesaplanır. Bu noktalar, artık tam sayı koordinat değerleri değildir. Bu noktalar kullanılarak nümerik bir eğri uydurma yöntemi olan lineer regresyon fonksiyonu yardımıyla bir doğru denklemi elde edilir. Doğru denklemi yardımıyla doğrunun eğim açısı hesaplandıktan sonra doğruya dik yönde olacak şekilde yeni bir yatay eksene pikseller iz düşürülür. Buradaki yatay eksen MTF üzerindeki sinyal ötüşmelerinin (signal aliasing) etkisini azaltmak amacıyla normal örnekleme aralığının $\frac{1}{4}$ 'ü oranında genişliğe sahip aralıklara bölünmüştür. Her bir aralıkta toparlanan piksel değerlerinin ortalaması alınır ve böylece bir boyutlu bir *kenar yayılım fonksiyonu* (ESF: Edge Spread Function) elde edilir. Alternatif bir yöntem olarak, elde edilen ayırık kenar verisine (ESF) bir fonksiyon uydurulup kenar yayılım fonksiyonu elde edilebilir. Kenar/Köşe yayılım fonksiyonunun birinci mertebeden türevi *çizgi yayılım fonksiyonunu* (LSF: Line Spread Function) vermektedir. LSF'nin Fourier dönüşümü ise *modülasyon aktarım fonksiyonunu* (MTF: Modulation Transfer Function) vermektedir. Bu işlemin bir standardı mevcuttur ve bu standart ISO 12233 Standart'ı olarak adlandırılmaktadır. Burada üzerinde çalışılan sistemler doğrusal uzamsal konum (kayma) değişmez (LSI) optiksel sistemlerdir. Doğrusal konum değişmez sistemlerin başlıca avantajları, üzerlerinde Fourier dönüşümü, evrişim vb. matematiksel operasyonların uygulanabilir olması ve bu sistemlerin deterministik (belirlenimci) olmalarıdır (Şekil 2.5). Optik sistemlerin doğrusal konum değişmez sistemler olarak modellenabiliyor olması matematiksel işlemler bakımından mühendis ve fizikçilere önemli kolaylıklar getirmektedir (Masaoka, K vd. 2014).

Eğik kenar yöntemi, günümüzde dijital kamera ve merceklerin testleri, tıbbi görüntüleme, uydu ve havacılık görüntüleme sistemleri ve endüstriyel optik cihazların çözünürlük ve optik kalite testlerinde oldukça yaygın ve profesyonel bir biçimde kullanılmaktadır.

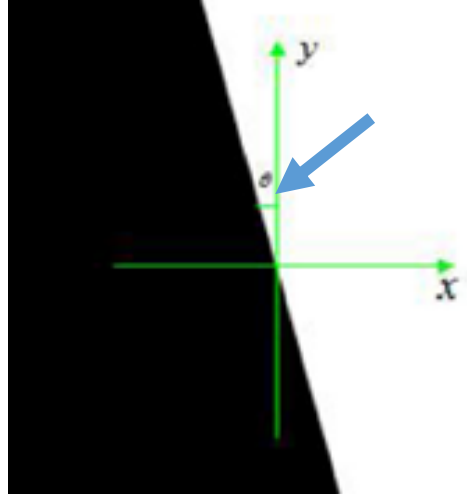


Şekil 2.5 Eğik kenar analizinin matematiksel gösterimi



Şekil 2.6 Piksel izdüşümünün grafiksel gösterimi

Şekil 2.6’da gösterildiği üzere görüntü üzerinde siyah ve beyaz pikselleri ayıran bir bıçak kenarı doğrusu bulunmaktadır. Bu doğrunun eğim açısı hesaplandıktan sonra doğruya paralel bir şekilde pikseller yeni bir eksene iz düşürülür. Buradaki yeni eksen, keskin kenar doğrusuna dik bir eksenidir ve standart olarak piksel boyutunun $\frac{1}{4}$ ’ü oranında pencerelere bölünmüştür. Piksel değerleri bu pencerelerde gruplandırılıp her bir pencerede toparlanan piksellerin ortalaması alınıp çizdirildiğinde kenar yayılım fonksiyonu/verisi elde edilebilmektedir. Eğik kenar analiz basamaklarında önemli adımlar arasında çalışılacak bölgenin seçilmesi (ROI) ve kenar doğrusunun belirlenmesi yer almaktadır (Şekil 2.7).



Şekil 2.7 Kenar açısının örnek bir gösterimi

Bu aşamada MTF hesabını yapan Python ortamında yazılmış bir kodlama çalışması yer almaktadır. Bu kodlama çalışmasında iki farklı yöntem kullanılmıştır. Birincisi piksel örnekleme sonucu elde edilen veriyi yumuşatan yöntem, diğeri ise elde edilen veriye hiç dokunmadan fonksiyon uyduran yöntemdir.

2.5.1 Veri yumuşatma yöntemi

Öncelikli olarak ilgili Python kütüphaneleri çağrılır ve görüntü Python ortamında açılır (Şekil 2.8 ve Şekil 2.9).

```
import numpy as np
import pandas as pd
import scipy.constants as sci
import cv2 as cv
import matplotlib.pyplot as plt
from scipy import stats
from skimage.util import random_noise
import scipy.constants as sci
import math
from scipy import ndimage
from scipy.misc import derivative
```

Şekil 2.8 Faydalı Python kütüphaneleri

Açılan görüntü üzerinden belirli bir bölge kırılır ve çalışma bölgesi (ROI) seçilir. Önemli olan yeterince büyük bir ROI'nin seçilmiş olmasıdır (Şekil 2.11).

```
ROI_Image = deepcopy(image_trial[150:351, 350:700])
#ROI_Image = image_trial
cv.imshow("ROI_OS", ROI_Image)
cv.waitKey(80000)
cv.destroyAllWindows()
```

Şekil 2.11 Matris üzerinden ROI seçimi

Şekil 2.11 ile ROI seçildikten sonra yapılması gereken işlem, eğik kenar açısının bulunmasıdır. Bu amaçla bir fonksiyon ve devamında bir kod parçası yazılmıştır. Bu fonksiyon görüntü üzerinde parlaklık geçişlerinin olduğu merkez (centroid) noktalarını piksel altı hassasiyette tespit edebilmektedir. Devamındaki kod parçasında ise ROI satır satır taranarak her satırdaki eğik kenarın görüntü ile kesiştiği merkez (centroid) noktaları tespit edilip bir numpy dizisinde tutulmaktadır.

```
# Define a function to find the edge centroid for a given row
def find_edge_centroid(row):
    # Take the 1D derivative along the row
    gradient = np.gradient(row)
    # Get absolute gradient values
    abs_gradient = np.abs(gradient)
    # Compute the centroid (weighted average of indices)
    indices = np.arange(len(row))
    centroid = np.sum(indices * abs_gradient) / np.sum(abs_gradient)
    return centroid

# Extract edge locations (centroids) for each row
edge_locations = []
for row in ROI_Image:
    edge_centroid = find_edge_centroid(row)
    edge_locations.append(edge_centroid)

# Perform linear regression to find the slope of the edge
y_coords = np.arange(len(edge_locations)) # Row indices
x_coords = np.array(edge_locations)        # Edge centroids
```

Şekil 2.12 Eğik Kenar noktalarının tespiti

Şekil 2.12'deki kod parçasının sonunda her bir satır indisi ve bu indislere karşılık gelen kenar (edge) merkez koordinatları elde edilmiştir. Yani elimizde x ve y değerlerine ait ondalıklı sayıları da içeren noktalar kümesi bulunmaktadır. Bu noktalardan geçen

doğruya ihtiyaç duyulmaktadır. Bu durumda nümerik analiz eğri uydurma tekniklerinden lineer regresyon metodu kullanılmıştır ve böylece doğru bulunup eğimi elde edilmiştir. Doğrunun eğiminin tanjant değeri de saat yönünde görüntü üzerindeki eğik kenar açısını vermektedir (Şekil 2.13).

```
from scipy.stats import linregress
# Use scipy's linear regression
slope, intercept, _, _, _ = linregress(y_coords, x_coords)

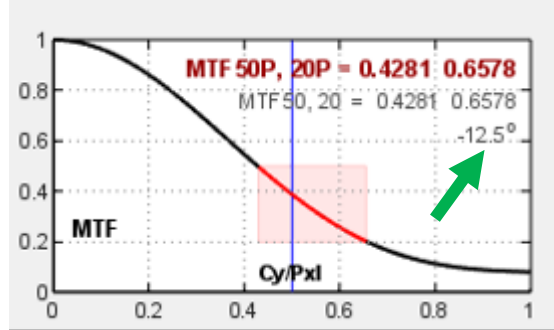
# Compute the angle in degrees
#theta = math.pi/2 + math.atan(slope) # Angle in radians
theta = math.atan(slope)*-1
theta_in_degrees = math.degrees(theta)

print(f"Estimated Edge Angle: {theta_in_degrees} degrees")
```

Şekil 2.13 Lineer regresyon yöntemi ile açı hesabı

```
first_image: (728, 1051) uint8
Estimated Edge Angle: 12.492845026165398 degrees
```

Şekil 2.14 Eğik kenar açı değeri

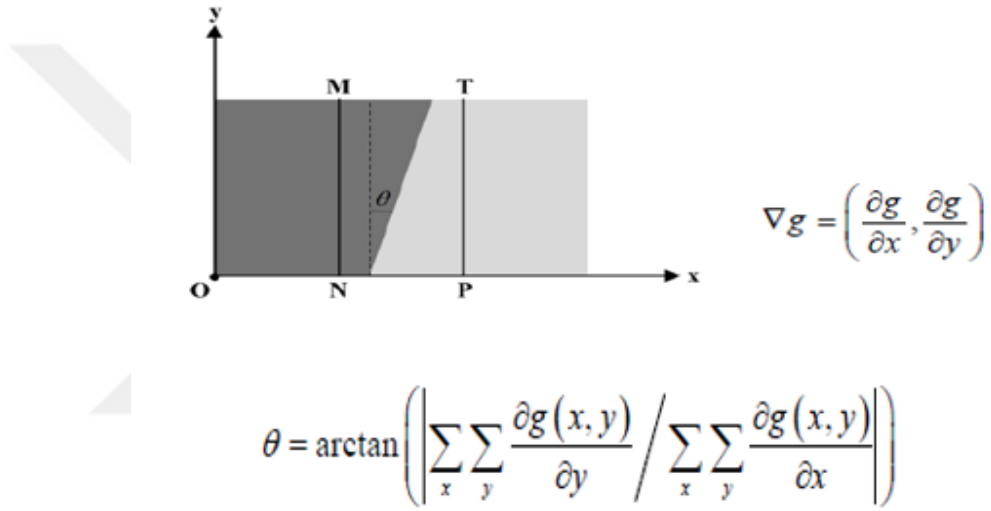


Şekil 2.15 Imatest görüntü analiz çıktısı

Burada hesaplanan açı değerini kontrol etmek amacıyla ticari bir yazılım olan IMATEST görüntü analiz programı kullanılmıştır ve bu programdaki açı değerinin 12.5 derece olduğu görülmüştür. Bu değer, yazılan Python kodlarının sonucuna oldukça yakın bir değerdir. Şekil 2.14 ve Şekil 2.15 incelendiğinde hesaplanan açı değerlerinin arasında 0.01 derecelik bir farkın olduğu görülmektedir.

2.5.1.1 Gradyan açı bulma yöntemi

Şekil 2.10'daki görüntü üzerinde nümerik ve matematiksel işlemler yapılacağı için “ROI_Image” değişkeni “float64” biçimine dönüştürülür. Bir sonraki aşama eğik kenar açısını hesaplamaktır (Şekil 2.16). Bu hesap için görüntü üzerinde x ve y yönleri boyunca türevler alınarak sırasıyla x ve y yönlerindeki değişimleri tutan matrisler elde edilir. Matrislerdeki değerler toplanıp birbirlerine oranlandığında ise doğrunun eğim açısı olan theta (θ) açısına geçilmiş olunur (Li, H vd. 2016).



Şekil 2.16 Eğik kenar açısının hesabı

Bir matris üzerinde belli bir yönde türev almak için onu ayırık türev alan bir kernel (çekirdek) matrisiyle evriştirmek gerekir. Bunun için Python OpenCV kütüphanesinin “cv.filter2D” komutu mevcuttur. Öncelikle türev alıcı kernel matrisi tanımlanır ve yukarıda daha önceden sunulmuş olan “ROI_Image” matrisiyle birlikte cv.filter2D metodunun içine atılır. Çıkan sonuç sırasıyla x ve y yönlerindeki değişim miktarlarını içeren matrisleri vermektedir (Şekil 2.17 ve Şekil 2.18).

```

### Taking Derivative in x Direction:

# Kernel Definition of Derivating Matrix in x - direction:
Kx = np.array([[ -1,1],[ -1,1]])
Kx = (1/2)*Kx
# Convolution Operation:
Fx = cv.filter2D(ROI_Image, -1, Kx)
Fx = abs(Fx)

```

Şekil 2.17 “x” yönünde türev alan kod parçası

```

### Taking Derivative in y Direction:

# Kernel Definition of Derivating Matrix in y - direction:
Ky = np.array([[1,1],[ -1,-1]])
Ky = (1/2)*Ky
# Convolution Operation:
Fy = cv.filter2D(ROI_Image, -1, Ky)
Fy = abs(Fy)

```

Şekil 2.18 “y” yönünde türev alan kod parçası

Burada hesaplanan değerler sırasıyla F_x ve F_y matrislerinde tutulmaktadır. Açığı hesaplamak için bu matrislerdeki değerlerin birbirine oranları gerekmektedir. İlk başta radyal olarak hesaplanan açı değeri daha sonra derece cinsine çevrilerek Python kod derleyicisi olan Spyder ekranında gösterilmektedir (Şekil 2.19 ve Şekil 2.20).

```
# Slanted Edge Angle (theta) Calculation:
```

```
# Buradaki hesap için " https://opg.optica.org/josk/abstract.cfm?URI=josk-20-3-381 " adresinde yer alan  
# makaledeki matematiksel işlemler referans alınmıştır.
```

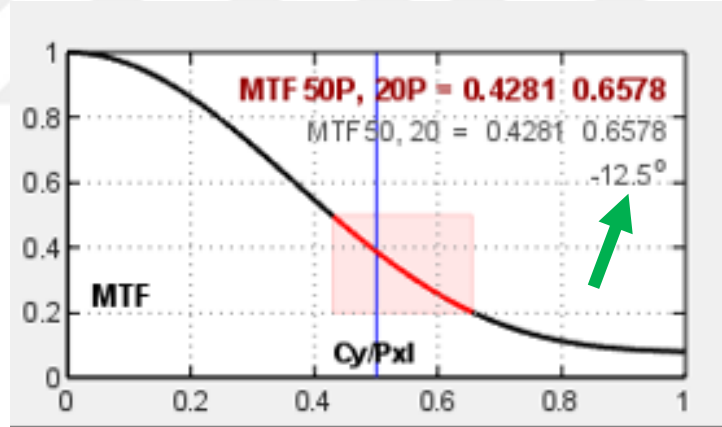
```
theta = math.atan(np.sum(Fy)/np.sum(Fx)) # in radians  
#theta = 0.218166156499292
```

```
#theta = math.atan(abs(np.sum(Fy)/np.sum(Fx))) # in radians  
theta_in_degrees = 180*(theta/np.pi); # in degrees
```

Şekil 2.19 Eğik kenarın açısı hesabı

theta_in_degrees	float	1	12.493116821214102
------------------	-------	---	--------------------

Şekil 2.20 Hesaplanan açı değeri

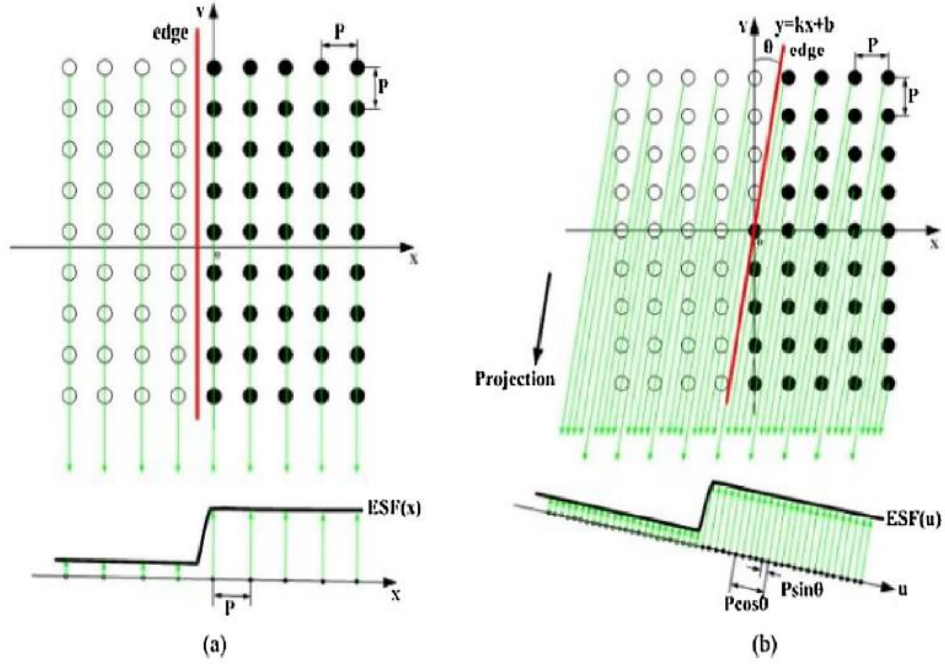


Şekil 2.21 Imatest görüntü programı çıktısı

Şekil 2.20 ve Şekil 2.21 incelendiğinde hesaplanan açı değerleri arasındaki farkın yalnızca 0.01° derece olduğu görülür. Yapılan hesaplamalar ve analizlerden görüldüğü üzere iki farklı açı hesaplama yöntemiyle de aynı sonuçlar elde edilmektedir. Bir sonraki aşama görüntü üzerinden kenar yayılım fonksiyonunun çıkarılması olacaktır.

2.5.1.2 Kenar yayılım fonksiyonu çıkarımı

Açı hesabından sonraki aşama, kenar yayılım fonksiyonunun hesaplanmasıdır. Bu fonksiyonun nasıl hesaplandığı ile ilgili bir görsel aşağıda sunulmuştur:



Şekil 2.22 Kenar yayılım fonksiyonu şematığı (Zhang, H vd. 2018)

Şekil 2.22 dikkatlice incelendiğine ilk başta sol tarafta verilen görüntü üzerindeki pikseller, olduğu gibi bir yatay eksene iz düşürülmüştür ve keskin bıçak kenarı (görsel üzerindeki kırmızı doğrusal çizgi) 0° derece eğim açısına sahiptir. Pikseller arası uzaklık “p” kadardır ve bu değer izdüşüm işlemi sonucunda korunmuştur. Burada standart bir izdüşüm işlemi yapılmıştır. Sağ tarafta verilen görselde ise kırmızı renk ile çizilen doğru saat yönünde belirli bir theta (θ) açısı kadar döndürülmüş ve bu doğru boyunca pikseller sinüs (sin) ve kosinüs (cos) fonksiyonlarıyla çarpılarak doğruya dik yöndeki yeni bir eksen üzerine iz düşürülmüştür. Burada yapılan işlemin temel mantığı elde edilmek istenen örneklem sayısını artırmaktır yani bu kısımda bir aşırı-örnekleme (oversampling) olduğu açıktır. Bu örnekleme işlemi optik sistemin kenar objesine karşı vermiş olduğu tepkiyi karakterize etmek bakımından oldukça önemlidir zira kenar yayılım fonksiyonu oluşturmak için yeterli sayıda büyüklüğe sahip dizilere veya bir diğer deyişle vektörlere

ihtiyaç duyulmaktadır. Optik sistemin bir eğik kenar girdisine verdiği tepkiyi karakterize edebilmek amacıyla böyle bir metoda ihtiyaç duyulmuştur. Böylece kenar yayılım fonksiyonu bu algoritma kullanılarak çıkarılmaktadır.

```
# ESF Hesaplama Algoritması:

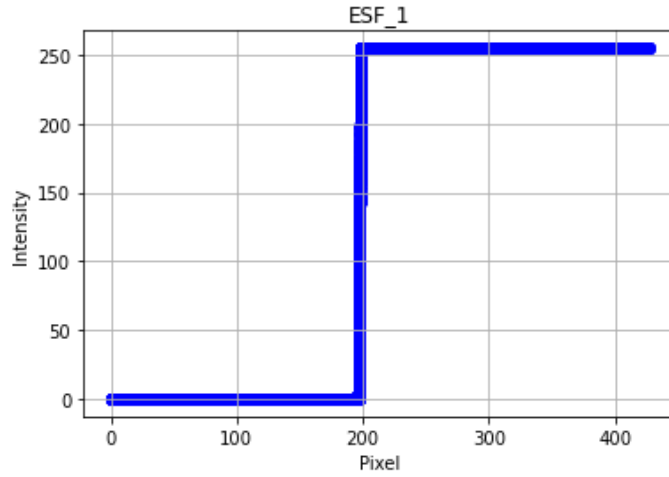
i = 0;
for k in range(0,rows):
    for m in range(0, cols):
        Value_Matrix[m + i,0] = (m*1.0 + math.tan(theta)*k)*math.cos(theta)
        Value_Matrix[m + i,1] = ROI_Image[k,m] # Intensity Alignment
    i = i + cols

# Aşağıdaki Kod Parçasında üretilen iki boyutlu matrix'ten (Value Matrix) ...
# ... küçükten büyüğe bir sorting (sıralama) işlemi yapılarak ESF'nin son haline geçiliyor:

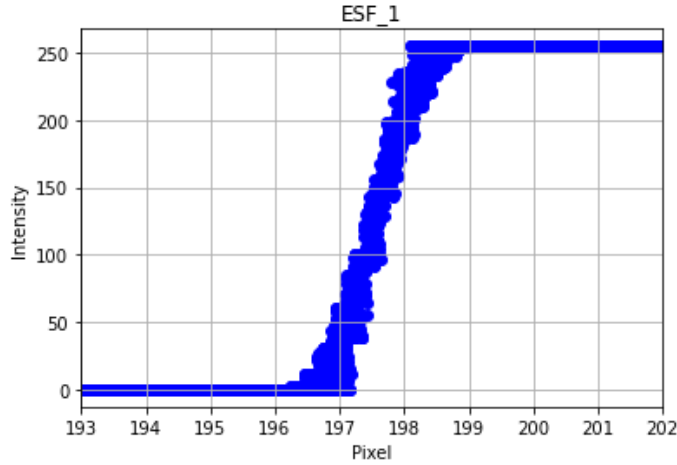
Value_Matrix_Trial_2 = np.copy(Value_Matrix)
ind = np.lexsort((Value_Matrix_Trial_2[:,1], Value_Matrix_Trial_2[:,0]))
ESF_Matrix = Value_Matrix_Trial_2[ind] # ESF'nin son hali
```

Şekil 2.23 Python'da kenar yayılım fonksiyonu algoritması

Şekil 2.23'te verilen algoritmada öncelikle 2 boyutlu (2-D: grayscale) bir görüntü üzerinde işlemlerin yapıldığı bilinmelidir. Daha sonrasında bilgisayar ortamında bir Python numpy dizisi olarak 2 boyutlu bir matris şeklinde tutulan görüntü üzerinden yeni bir iki boyutlu vektör türetilecektir. Bu vektörün birinci sütunu yeni örneklenmiş (trigonometrik fonksiyonlarla çarpılmış) piksel koordinatlarını ikinci sütunu ise piksel değerlerinin taşıdığı parlaklık (intensity) değerlerini gösterecektir. Bu işlemi gerçekleştirmek için görüntünün sol en üst noktasından başlayarak tüm görüntü matrisini satır ve sütunlar boyunca tarayarak koordinat ekseninin x değerlerini trigonometrik fonksiyonlar ile çarpıp y değerlerini yani parlaklık değerlerini ise değiştirmeden karşılık gelen ilgili yeni x değerinin karşısına yerleştirerek bir ön vektör çıkarımı yapılır. Daha sonra Şekil 2.23'te belirtilen kod parçasının son kısmında “np.lexsort()” komutuyla oluşturulan ön vektör üzerinde küçükten büyüğe bir sıralama yapılarak kenar yayılım fonksiyonuna ait ayırık ham veri elde edilir.

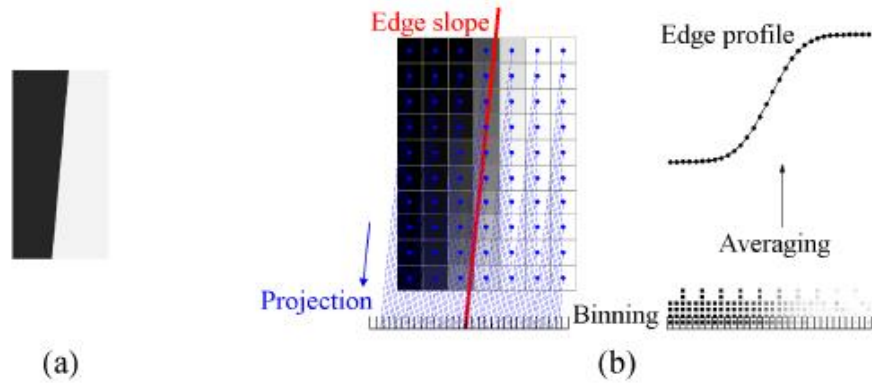


Şekil 2.24 Kenar yayılım fonksiyonu (ESF)



Şekil 2.25 Kenar yayılım fonksiyonunun belirli bir aralıkta çizdirilmesi

Şekil 2.24 ve Şekil 2.25'te sunulan grafikler üzerinden görüldüğü gibi kenar yayılım fonksiyonu ayrık dağılıma sahip değerler içeren bir ham data (veri) fonksiyonudur. Bir sonraki adım, türev ve Fourier dönüşümü elde etmek olacağından sonraki adıma ilerleyebilmek için bu fonksiyonun mutlaka analitik, türevlenebilir ve bir boyutlu bir fonksiyon biçiminde ifade edilmesi gerekmektedir. Tam bu noktada veri yumuşatması (data smoothing) devreye girmektedir. ESF verisi piksel büyüklüğünün standart olarak 0.25'inden daha yüksek bir örnekleme sıklığı tercih edildiğinden piksel büyüklüğünün 0.1'ine karşılık gelecek şekilde genişliklere sahip pencerelere (bin'ler) gruplandırılıp her bir gruptaki (bin'deki) piksel değerleri yani parlaklık (intensity) değerlerinin ortalaması alındığında Şekil 2.26'da görüldüğü gibi bir boyutlu bir ESF grafiği elde edilir.



Şekil 2.26 Veri yumuşatma yöntemi için kullanılan kenar yayılım fonksiyonu algoritması

```

range_size = 0.10;

def frange(start, stop, step=1.0):
    i = start
    while i < stop:
        yield i
        i += step

x_max = x_values.max();
x_min = x_values.min()
print(x_max)

empty_array = np.array([[0, 0]]) # Array Initialization

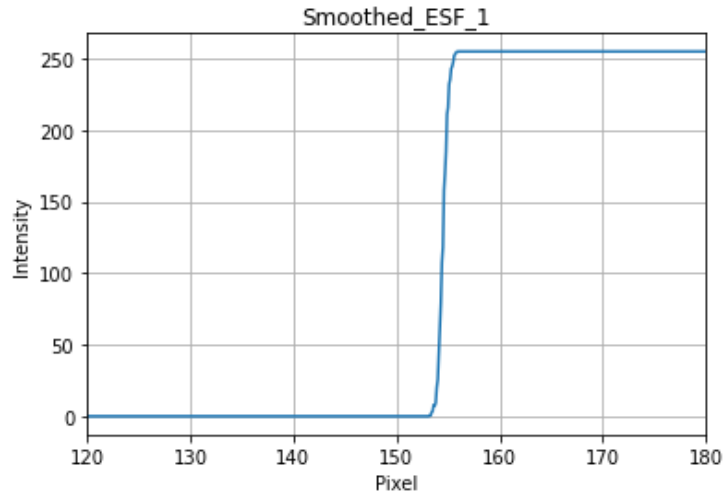
for i in frange(x_min, x_max, range_size):
    solutions = np.argwhere(np.logical_and(x_values >= i, x_values < (i + range_size))) # Index'le
    sum = 0.0; # Intensity'leri toplamadan önceki initialization
    for index in solutions:
        sum += y_values[index] # İlgili indeks'lerdeki Intensity değerlerini topluyoruz !!
    if solutions.size != 0:
        bin_middle_point = (2*i + range_size) / 2; # Bin'in ortalaması
        bin_averaged_intensity = sum[0] / solutions.size; # Intensity Değerlerinin Ortalaması
        empty_array = np.append(empty_array, [[bin_middle_point, bin_averaged_intensity]], axis=0)

```

Şekil 2.27 Kenar yayılım fonksiyonunun yumuşatılması için yazılan kod parçası

Şekil 2.27’te verilen kod parçası detaylıca analiz edildiğinde öncelikle bir fonksiyonun tanımlandığı görülür. Buradaki “f_range” fonksiyonu örneklenmiş x koordinat değerlerini tutan yatay eksendeki pencereler boyunca adım adım ilerlemeyi sağlayan fonksiyondur. Örneklenmiş ve iz düşürülmüş x koordinat değerlerinin minimum ve maksimum değerlerinden başlayarak fonksiyona girilen adım büyüklüğünü gösteren “range_size” isimli değişkeni içine alarak bir döngü oluşturulmuş olup yeni örneklenmiş x koordinat değerlerini tutan dizi içerisindeki elemanların tek tek döngü değişkeni ve döngü değişkeninin “range_size” fazlası kadar bir aralıkta olup olmadığı tespit edilmiştir. Daha

sonrasında ilgili x koordinatlarının kendi değerlerinden ziyade indekslerine ihtiyaç duyulmaktadır; çünkü bu indeks değerlerine karşılık gelen y değerleri yani piksel parlaklık değerlerinin ortalaması alınıp ilgili pencereye karşılık gelen aralığın tam orta değerine eşleştirilecek şekilde yeni bir vektör türetilmektedir. Türetilen bu 2 boyutlu vektörün birinci ve ikinci sütunundaki değerler birbirlerine göre çizdirildiğinde kenar yayılım fonksiyonu elde edilir (Şekil 2.28).



Şekil 2.28 Kenar yayılım fonksiyonunun yumuşatılmış hali

Şekil 2.28 incelendiğinde ilgili fonksiyon üzerinde sapmalar olduğu gözlemlenmektedir. Bu durum, temelde yazılan algoritmada kullanılan örnekleme sıklığına (range_size) bağlıdır. Bu grafiğin nümerik türevi alındığında grafik üzerindeki sapmaların etkisi ortaya çıkacaktır. Bu aşamadan sonraki adımlar oldukça sade bir şekilde ilerlemektedir. Önce Şekil 2.28'te verilen fonksiyonun türevi alınacaktır. Daha sonra Hamming window (Hamming pencere) adı verilen bir tür filtreleme işleminden geçirilerek elde edilen son fonksiyonun Fourier dönüşümü alınıp MTF grafiği çıkarılacaktır. Kenar yayılım fonksiyonunun türevi, çizgi yayılım fonksiyonu olarak geçmektedir ve MTF hesabında sondan bir önceki basamağı temsil etmektedir.

2.5.2 Python ile nümerik türev alma

Belirli bir fonksiyonun sonsuz küçük aralıkta nasıl değiştiği türev alınarak incelenebilmektedir. Günlük hayatta analitik fonksiyonların türevleri alınırken bu hesap oldukça kullanışlıdır. Ancak bilgisayar ortamında türev alınırken belirli bir aralıkta çalışıldığından bu aralığın nümerik olarak tanımlanması gerekmektedir. Bu duruma ek olarak bilgisayarlar ayırık zamanlı çalışan sistemlerdir ve bu zamana kadar alışlagelinmiş olan türev alma yöntemi artık analitik yerine nümerik yöntemlerle yapılmalıdır. Üzerinde çalışılan probleme bağlı olarak karmaşık denklemlerde türev almak oldukça zor olabilir veya davranışı bilinen bir fonksiyon yerine ayırık noktalardan oluşan bir veri grubuyla çalışılması durumunda nümerik türeve başvurulmalıdır. C++, Java ve Python gibi programlama dilleriyle yüksek mertebelerden türev alabilen algoritmalar yazmak mümkündür. Python programlama dilinin numpy kütüphanesinde yer alan bir fonksiyon olan `np.gradient()` fonksiyonu ile ayırık türev alınabilmektedir ve bu fonksiyonun kullanımı oldukça yaygındır (Şekil 2.29 ve Şekil 2.30).

Belirli bir dizi için, `np.gradient()` fonksiyonu hesaplanmak istenen türevi iç noktalarda merkezi farklar yöntemini ve sınırlarda birincil farklar yöntemini kullanarak hesaplar. Örneğin, `f` dizisindeki bir iç nokta olan `f[i]` için türev, merkezi farklar yöntemi kullanılarak bulunur. Δx noktalar arasındaki uzaklığa karşılık gelecek şekilde

$$Gradient[i] = \frac{f[i + 1] - f[i - 1]}{2 \cdot \Delta x}$$

olarak ifade edilir.

Sınır noktalarında merkezi farklar yöntemini uygulamak mümkün değildir. Bu durumda ileri farklar ve geri farklar yöntemi kullanılmaktadır. Başlangıç noktasında (`f[0]`),

$$Gradient[0] = \frac{f[1] - f[0]}{\Delta x}$$

formülü kullanılırken bitiş noktasında ise aşağıdaki bir matematiksel yapıdan faydalanılabilir:

$$\text{Gradient}[N - 1] = \frac{f[N - 1] - f[N - 2]}{\Delta x}$$

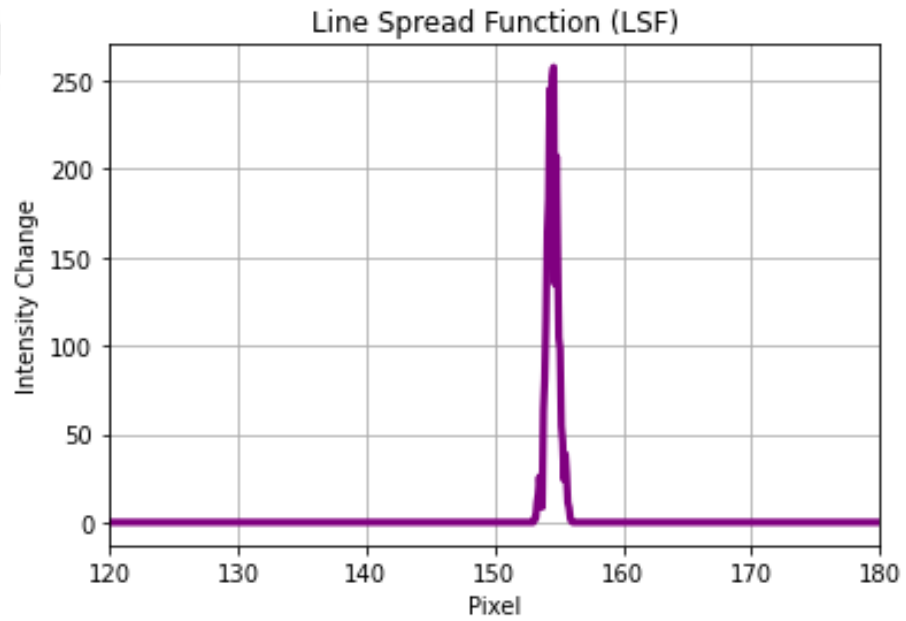
```

#####
# LSF Derivation:

LSF = abs(np.gradient(y2,x2))
plt.figure()
plt.plot(x2,LSF,linewidth = 3.2, color = 'purple' )
plt.xlim([193, 202])
plt.title('Line Spread Function (LSF)')
plt.grid()
plt.xlabel('Pixel')
plt.ylabel('Intensity Change')
plt.show()

```

Şekil 2.29 Yumuşatılmış ESF grafiğinin türevini alan kod parçası

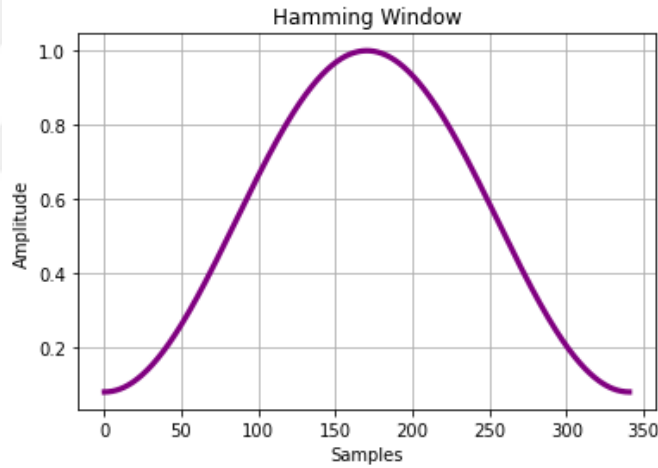


Şekil 2.30 Çizgi yayılım fonksiyonu (LSF)

2.5.3 Hamming penceresi

Sinyal işleme ve istatistikte kullanılan pencere fonksiyonu belirli seçilmiş bölgenin dışında 0 değerine sahip; genelde bu bölge etrafında simetrik, orta kısımda maksimuma ulaşan ve diğer kısımlarda gittikçe azalan bir yapıya sahiptir. Bu fonksiyon, bir sinyali analiz ederken ya da filtreleme sırasında pencereleme işlemini daha etkili hale getirir. Pencereleme, sonsuz bir sinyali veya veriyi, analiz edilebilecek sınırlı bir boyuta indirgeyerek spektrumda yan lobları (sidelobes) azaltmak için uygulanır. Hamming penceresi için matematiksel bir ifade de mevcuttur.

$$w[n] = a_0 - (1 - a_0) \cdot \cos\left(\frac{2\pi n}{N}\right), 0 \leq n \leq N$$



Şekil 2.31 Hamming penceresi grafiği

Burada Şekil 2.31’de verilen fonksiyon, çizgi yayılım fonksiyonunun (LSF) aldığı değerlerle doğrudan çarpılmaktadır ve 0-400 aralığı dışındaki değerlerin sıfırlandığı görülmektedir. Diğer bölgelerde ise ilgili data [0-1] aralığındaki değerler ile çarpılmaktadır. Bu bağlamda Hamming penceresinin bir tür filtreleme işlevi gördüğü sonucuna varılabilmektedir (Şekil 2.32 ve Şekil 2.33)

```

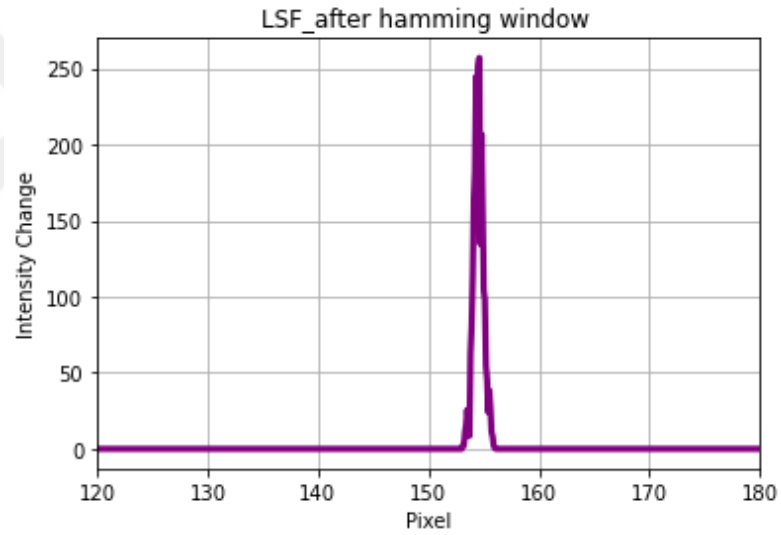
#####

hamming = np.hamming(len(LSF))
plt.figure()
plt.plot(x2,hamming,linewidth = 3.2, color ='purple' )
plt.title('Hamming window Utilized')
plt.grid()
plt.show()

after_hamming = np.multiply(hamming,LSF)
plt.figure()
plt.plot(x2, after_hamming, linewidth = 3.2, color ='purple' )
plt.title('LSF_after hamming window ')
plt.xlabel('Pixel')
plt.ylabel('Intensity Change')
plt.xlim([193,202])
plt.grid()
plt.show()
#####

```

Şekil 2.32 Hamming penceresinin Python kodları



Şekil 2.33 Hamming penceresi sonrası LSF

Son aşama olarak geriye Fourier dönüşümü kalmaktadır ve Python kütüphaneleri aracılığıyla bu işlemi gerçekleştirmek mümkündür. Bu aşamada Fourier dönüşümü alınmış veri normalize edilerek çizdirilmiştir.

```

# Part 6:

# Bu aşamda ise LSF(Line Spread Function)'ın DFT(Discrete Fourier Transform)' u hesaplanmaktadır.
# Fourier Transformu alınmış data normalize edilerek MTF fonksiyonu çizdirilmiştir.

from scipy.fft import fftfreq,rfft
N_1 = len(x2)

f3 = fftfreq(len(x2),np.mean(np.diff(x2))) ## -- >> Cycles per pixel

x2_FFT = abs(rfft(after_hamming)) ## Taking Fourier Transform !!

norm_x2_FFT = x2_FFT/x2_FFT[0] # By dividing zero frequency component, normalizing is done.

plt.figure()
plt.plot(f3[:N_1//2], norm_x2_FFT[:N_1//2])
plt.xlabel('spatial freq, cyc/pxl')
plt.title('MTF_7.5')
plt.grid()
plt.xlim([0,1])
plt.show()

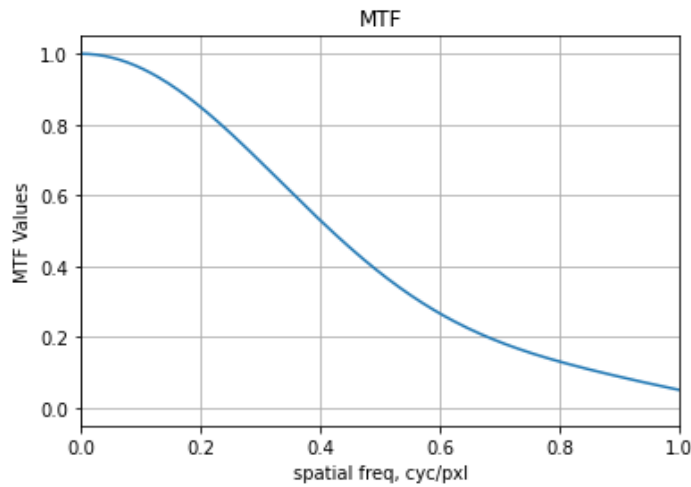
a3 = norm_x2_FFT[:N_1//2] ## Normalize MTF değerleri
a4 = f3[:N_1//2] ## Frekans eksenini.

MTF_Matrix = np.zeros((a4.shape[0],2))
MTF_Matrix[:,0] = a3
MTF_Matrix[:,1] = a4

```

Şekil 2.34 Fourier dönüşümü için yazılan Python kod parçası

Şekil 2.34'te verilen kod parçasında önce LSF sinyalinin uzunluğuna bakılmaktadır; çünkü bir örnekleme frekansı seçilmelidir. Bu uzunluk elde edildikten sonra Fourier alanında bir frekans eksenini Python kütüphanesinin “fftfreq” fonksiyonu kullanılarak elde edilmektedir. Fourier dönüşümü alınmış değerler ise hem kompleks hem de reel değerlerdir. Burada ilgilenilen kısım, reel sonuçlardır. Bu sonuçlar, dönüşüm sonrası elde edilen değerlerden sıfır frekansına karşı gelen değere bölünerek normalize edilir. Böylece [0,1] arasında MTF grafiğine ulaşılır (Şekil 2.35).



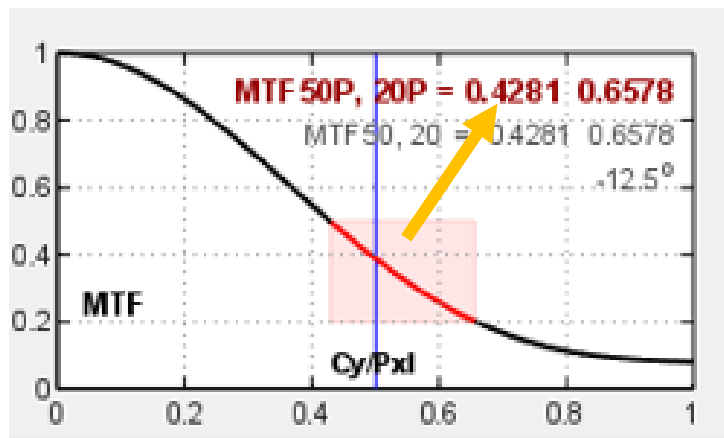
Şekil 2.35 Modülasyon aktarım fonksiyonu

Elde edilen MTF verisinin doğruluğunu kıyaslamak amacıyla IMATEST v.24.3 ticari görüntü yazılımından faydalanılmıştır. Bunu yaparken temel ölçüt, MTF değerinin %50'sine düştüğü değere bakmaktır ve bu değerleri kıyaslamaktır.

	0	1
156	0.519297	0.406461
157	0.515148	0.409067
158	0.511013	0.411672
159	0.506893	0.414278
160	0.502788	0.416883
161	0.498697	0.419489
162	0.494622	0.422095
163	0.490562	0.4247
164	0.486519	0.427306

Şekil 2.36 MTF değerinin frekans tablosu

Şekil 2.36'daki tabloda MTF değerinin yaklaşık % 50'ye düştüğü frekans değeri 0.4168 cyc/pxl (piksel başına döngü) olduğu gözlemlenmektedir. Imatest programında ise bu değer 0.4281 cyc/pxl olduğu saptanmıştır. Yani arada yaklaşık 0.0113'lük bir fark mevcuttur ve bu fark Imatest yazılımı çıktısı üzerinden görülebilmektedir (Şekil 2.37).



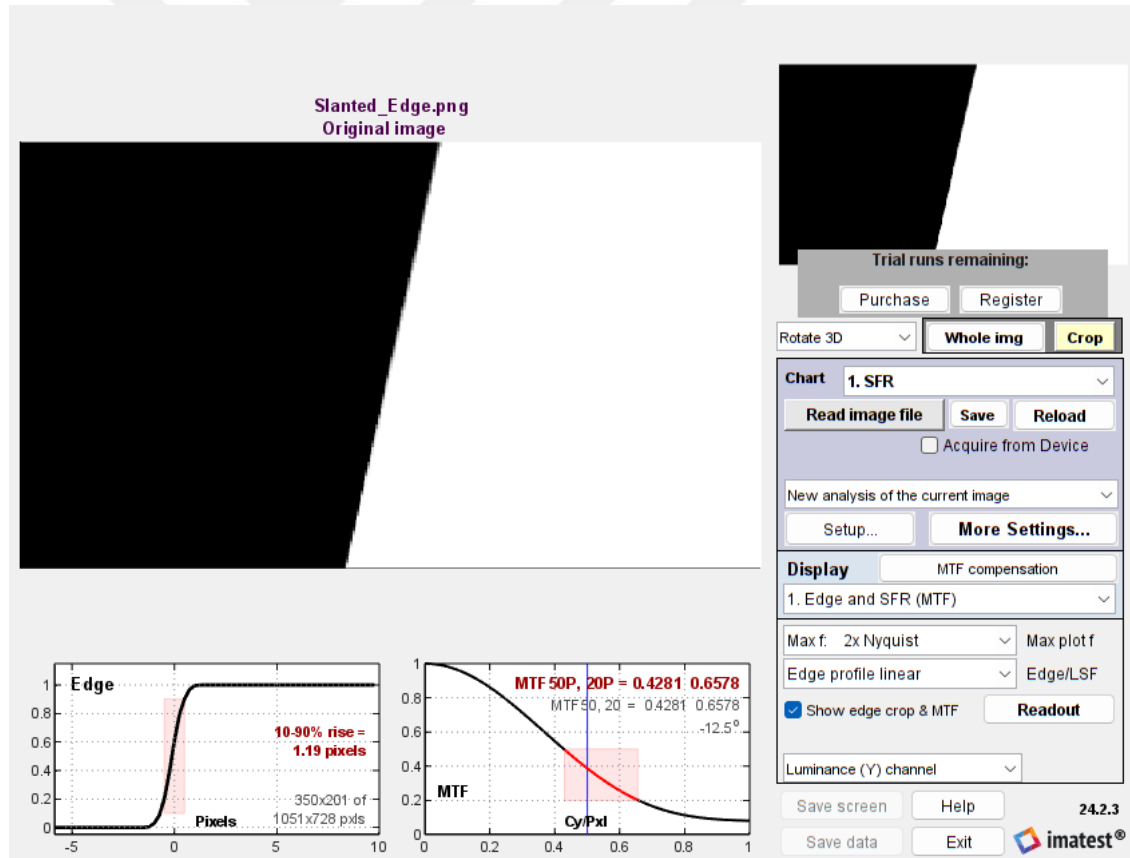
Şekil 2.37 Imatest MTF grafiği çıktısı

Imatest yazılımının sonuçlarıyla yazılan Python kodlarının verdiği değer arasındaki farkı yüzdesel olarak ifade edebilmek mümkündür:

$$Fark \% = \frac{Imatest - Python}{Imatest} * 100$$

$$Fark \% = \frac{0.4281 - 0.4168}{0.4281} * 100 = \%|2.63|$$

Yukarıdaki hesaptan görüldüğü üzere % |2.63|'lük bir fark mevcuttur. Daha detaylı analiz için Imatest yazılımının çıktısı sunulmuştur (Şekil 2.38).



Şekil 2.38 Imatest görüntü analiz çıktısı

2.5.4 Tek frekansta Fourier dönüşümü

Sürekli zamanlı sinyallerde bir sinyalin sadece tek bir frekans bileşeni içerdiği durumlarda yapılan Fourier analizi işlemi, tek frekansta Fourier dönüşümü olarak adlandırılmaktadır. Normal Fourier dönüşümü formülü aşağıdaki gibidir:

$$X(f) = \int_{-\infty}^{+\infty} x(t) * e^{-j*2*\pi*f*t} dt$$

Yukarıdaki denklemde sırasıyla,

$X(f)$: Frekans alanındaki genlik ve faz bilgisini içeren kompleks fonksiyon

$x(t)$: Zaman alanındaki sinyal

f : Frekans (Hz)

$e^{-j*2*\pi*f*t}$: $x(t)$ sinyalinin üzerine izdüşüm yapıldığı ortonormal kümenin baz fonksiyonu (kompleks üstel fonksiyon). Tek bir frekansta salınımı yapan sinusoidal ise aşağıdaki gibi ifade edilebilmektedir:

$$x(t) = A * \sin(2 * \pi * f_0 * t)$$

A : Sinyalin genliği

f_0 : Sinyalin frekansı

ϕ : Fazı (radyan cinsinden)

t : Zaman

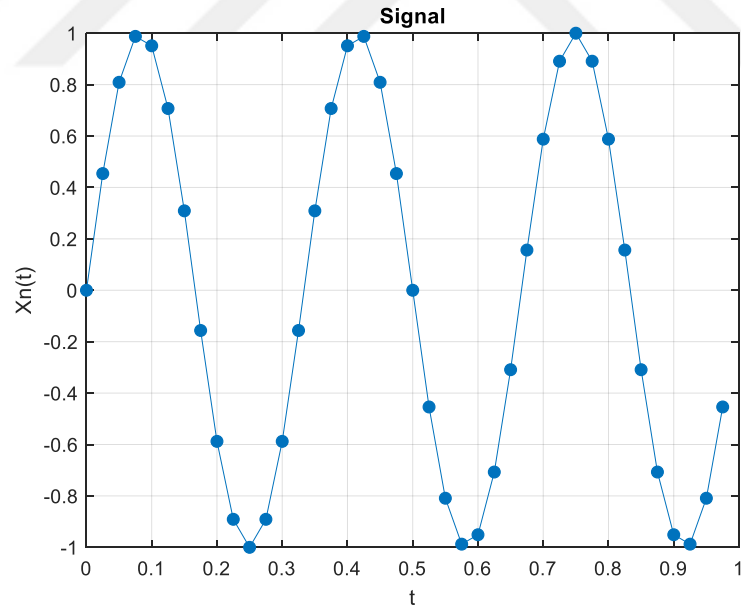
Bu sinyalin Fourier dönüşümü aldığı frekans uzayında $+f_0$ ve $-f_0$ frekanslarında olmak üzere iki adet dirac delta fonksiyonu biçimine benzeyen pik dürtü fonksiyonları elde edilir (Şekil 2.41). Öncelikle zaman alanında sinüsodali çizdiren MATLAB

kodlarına yer verilmiş, daha sonrasında bu kodun çıktıları sunulmuştur (Şekil 2.39 ve Şekil 2.40).

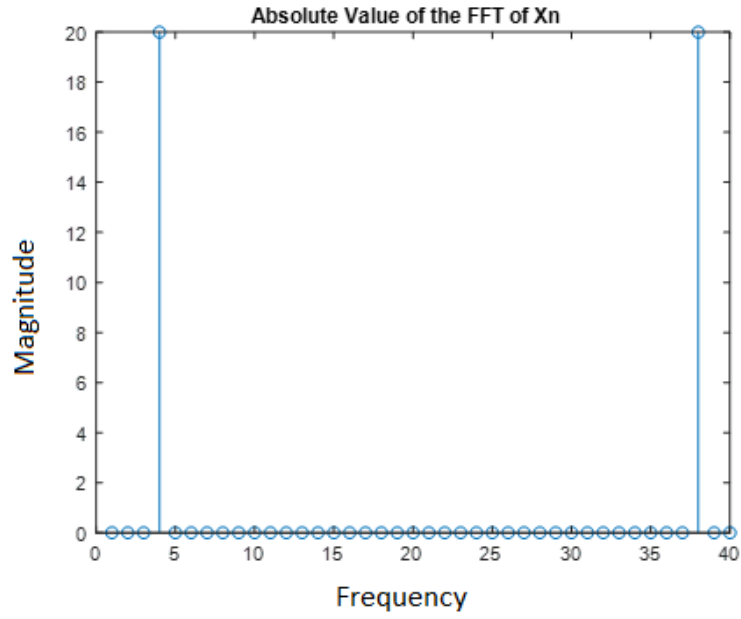
```
Fs = 40;      % Sampling Frequency
T = 1/Fs;     % Sampling Period
N = 40;       % Length of Signal
t = (0:N-1)*T'; % Time Vector
f = 3;        % 3 Hz
Xn = sin(2*pi*f*t); % Normalized 3 Hz Signal

plot(t,Xn, '-.', 'MarkerSize', 20);
title('Signal')
xlabel('t')
ylabel('Xn(t)')
```

Şekil 2.39 Sinüsoidal fonksiyon MATLAB kod parçası



Şekil 2.40 Zaman alanında sinüs sinyali



Şekil 2.41 Sinüsoidal sinyalin Fourier dönüşümü

Şekil 2.41’de frekans alanında 3Hz’deki pik’i görebilmek için MATLAB kodları yeniden düzenlenmiş ve çıktılarıyla birlikte sunulmuştur (Şekil 2.42 ve Şekil 2.43).

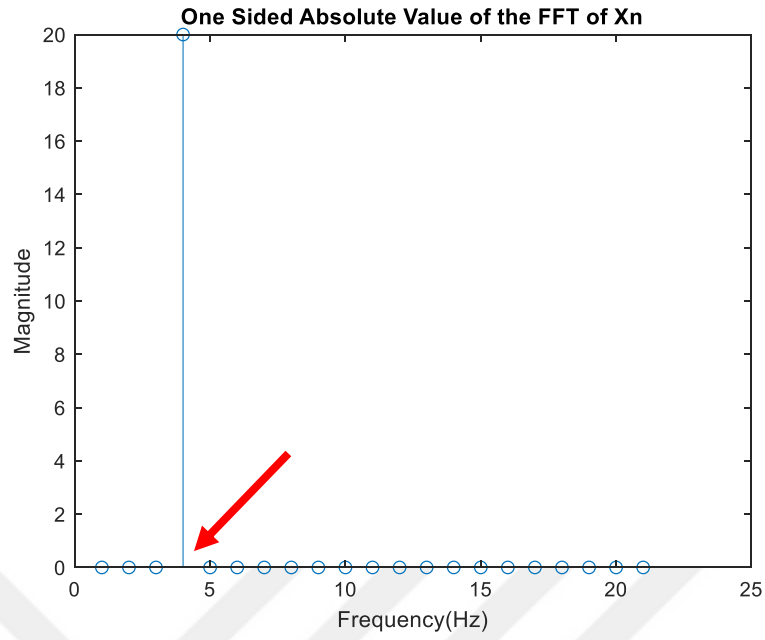
```
T = 1/Fs; % Sampling Period
N = 40; % Length of Signal
t = (0:N-1)*T'; % Time Vector
f = 3; % 3 Hz
Xn = sin(2*pi*f*t); % Normalized 3 Hz Signal

plot(t,Xn, '-.', 'MarkerSize', 20);
title('Signal')
xlabel('t')
ylabel('Xn(t)')
grid on

k = 0:N/2; % k starts at 0 so the length of k here is N/2+1
freq = k*Fs/N;

Y = fft(Xn);
figure;
stem(abs(Y(1:N/2+1)));
title('One Sided Absolute Value of the FFT of Xn')
```

Şekil 2.42 Tek bir frekansta Fourier dönüşümü için yazılan MATLAB kod parçası



Şekil 2.43 Tek bir frekansta Fourier dönüşümü fiziksel çıktısı

Bir sinyalin Fourier dönüşümü alındığında frekans alanında bir spektrum elde edilir ve bu spektrum bütün frekans bileşenlerini içermektedir. Tek frekansta Fourier dönüşümünde ise spektral dağılımdaki sadece bir frekans bileşenine ait bilgi elde edilmektedir. Bu işlem, Fourier dönüşümünün tam olarak hesaplanması yerine sadece bir frekanstaki davranışı incelemek için yapılır. Bu durum genellikle performans kaygıları olan uygulamalarda veya belirli frekans bileşenlerini izole etmek istenildiğinde kullanılır. Bu yöntem hem sürekli zamanlı sinyallere hem de bilgisayar ortamındaki ayrık zamanlı sinyallere uygulanabilmektedir.

$$X[f_0] = \sum_{n=0}^{N-1} x[n] * e^{-j*2*\pi*f_0*n/f_s}$$

f_0 : İncelenen frekans

f_s : Sinyal örnekleme frekansı

N: Sinyalin toplam örnek sayısı

$e^{-j*2*\pi*f_0*n/f_s}$: Kompleks üstel (sinüs ve kosinüs bileşenleri)

```

delta_x = np.mean(np.diff(x2))
sampling = delta_x

def my_fourier(signal, target_frequency, sampling_rate):
    N = len(signal)
    F_target = np.sqrt(2.8)*np.sum(signal * np.exp(-2j * np.pi * target_frequency * np.arange(N) * sampling_rate))/N
    return F_target

target_frequency = 0.15 # in cycles per pixel
deneme = np.abs(my_fourier(LSF, target_frequency, sampling))

```

Şekil 2.44 Tek bir frekansta Fourier dönüşümü

Şekil 2.44'te verilen kod parçasında Python'un kütüphanelerinden alınan hazır "fft" komutu yerine direkt matematiksel ifade kodlanmıştır. Öncelikle bir örnekleme frekansı belirlenmiştir. Yazılan fonksiyon Fourier dönüşümü alınacak sinyal, dönüşümün yapılmak istendiği hedef frekans ve örnekleme frekansı olacak şekilde temelde 3 adet parametre almaktadır. Fonksiyon çalıştırıldıktan sonra kompleks değerli bir ifade dönmektedir. Bu ifadenin "np.abs()" komutu ile mutlak değeri alınması gerekmektedir.

```

MTF_Matrix = np.zeros((a4.shape[0],2))
MTF_Matrix[:,0] = a3
MTF_Matrix[:,1] = a4

# MTF_Matrix'in ikinci sütununda k değerine en yakın olan frekans bulma
#k = target_frequency # Örnek olarak belirli bir frekans değeri verilmiş
frekanslar = MTF_Matrix[:, 1] # MTF_Matrix'in ikinci sütunu (frekans eksenini)
mtf_degerleri = MTF_Matrix[:, 0] # MTF_Matrix'in birinci sütunu (MTF değerleri)

# k'ye en yakın frekansın indeksini bulma
en_yakin_indeks = np.argmin(np.abs(frekanslar - target_frequency))

# Bu indekste yer alan MTF değerini alma
en_yakin_mtf_degeri = mtf_degerleri[en_yakin_indeks]
en_yakin_frekans = frekanslar[en_yakin_indeks]

print(f"hedef frekans değerine en yakın frekans: {en_yakin_frekans}")
print(f"En yakın MTF değeri: {en_yakin_mtf_degeri}")

```

Şekil 2.45 MTF sonuçları ve tek frekansta Fourier dönüşümü

Şekil 2.45'te verilen kod parçasında ise yazılan kodlar sonucunda elde edilen MTF grafiğinin x ve y eksenlerindeki değerleri sırasıyla ayrı birer numpy dizisinde tutulmaktadır. Bu diziler birleştirilerek bir MTF matrisi oluşturulmuştur. MTF grafiğinin frekans eksenindeki değerleri arasından Fourier dönüşümünün yapıldığı hedef frekansa en yakın frekans değeri hesaplandıktan sonra bu frekansın indeks değerine ulaşılır zira MTF matrisi üzerinden ilgili MTF değerine ihtiyaç duyulmaktadır. Elde edilen bu değerler, daha sonrasında yazılan kod parçası yardımıyla karşılaştırılacaktır.

```
Manuel olarak tek bir frekansta hesaplanan Fourier Dönüşümü  
at f=0.15: 0.8988264828685197  
hedef frekans değerine en yakın frekans: 0.14984772845814237  
En yakın MTF değeri: 0.8977553526460914
```

Şekil 2.46 Tek bir frekansta hesaplanan Fourier dönüşümü

Şekil 2.46'daki görselde bütün spektrumdan alınan değer ile spektrum üretmeden sadece tek bir frekansta hesaplanan Fourier dönüşümü işlemlerinin sonuçları gösterilmektedir. Buna göre, spektrumdan bulunan 0.1498 değeri hedef frekans olan 0.15'e en yakın değer iken tek bir frekansta yaklaşık olarak hesaplanan 0.899 değeri ile MTF spektrumundan alınan 0.898 değeri arasındaki fark sadece 0.001 olmaktadır.

2.5.5 Eğri uydurma yöntemi

Bu yöntemde görüntü üzerinden kodlar yazılarak çıkarılmış ayırık ve ham kenar yayılım fonksiyonuna hiç dokunulmadan bu veriye bir eğri uydurularak devam edilmiştir (Şekil 2.47). Eldeki dağılım istatistiksel fizikteki Fermi fonksiyonuna benzediği için analitik olarak benzetilecek fonksiyon Fermi fonksiyonu olarak seçilmiştir (Tzannes vd. 1995).

$$F = \frac{a}{\exp\left(-1 * \left(\frac{x-b}{c}\right)\right) + 1}$$

```
# Bir Fermi fonksiyonu aşağıdaki gibi tanımlanır ve ESF data'sına fit edilmesi sağlanırsa:

# Fermi Fonksiyonun Tanımlanması:

def fermi_f(x,a,b,c):
    result = a/(np.exp(-1*((x-b)/c)) + 1)
    return result
```

Şekil 2.47 Spyder ortamında Fermi fonksiyonunun Python kodlanmış hali

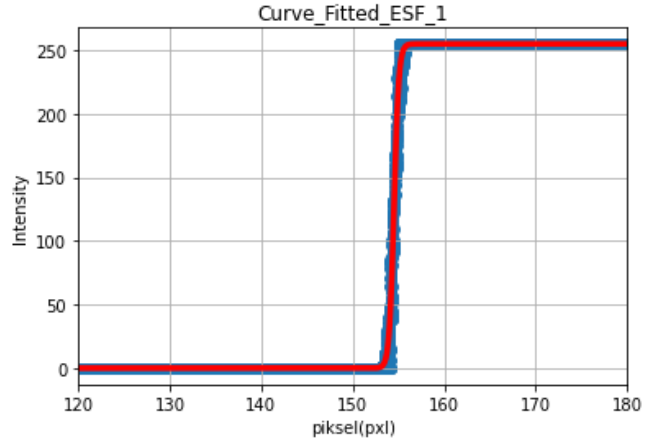
Bu aşamada eğri uydurma yapılacaktır. Bu işlem, Python kütüphanelerinden alınmış bir fonksiyon olan “curve_fit” fonksiyonu ile gerçekleştirilecektir. Bu fonksiyona benzetmek istenilen analitik fonksiyon ve elimizdeki veri grubunun x ve y eksenindeki değerleri girilmektedir. Daha sonra “curve_fit” fonksiyonunun döndüğü bu yeni parametrelerle birlikte Fermi fonksiyonu tekrar çizdirilip veriye uydurulmuş nihai fonksiyon elde edilecektir (Şekil 2.48 ve Şekil 2.49).

```
from scipy.optimize import curve_fit
#popt, pcov = curve_fit(fermi_f, x, y, p0 = [255, 197.5, 0.2])
popt, pcov = curve_fit(fermi_f, x, y)
print(popt)

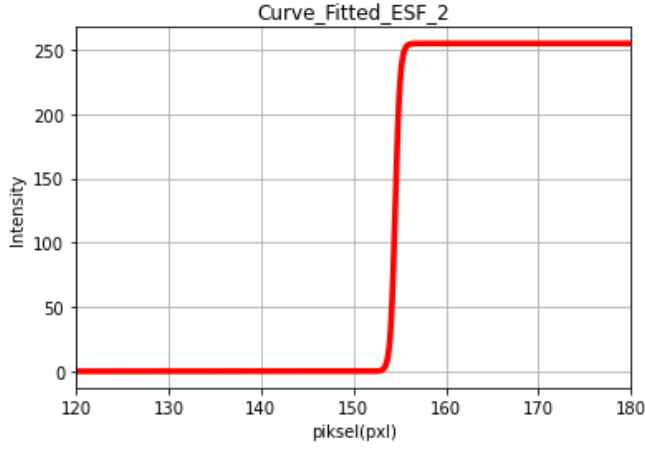
a_opt, b_opt, c_opt = popt # curve_fit fonksiyonunun döndüğü optimal parametreler
x_model = np.linspace(min(x), max(x),100000)
#y_model = fermi_f(x_model, a_opt, b_opt, c_opt)
y_model = fermi_f(x_model, *popt)

plt.figure()
plt.scatter(x,y) # ESF DATA
plt.plot(x_model,y_model, linewidth = 3.6, color = 'r') # Fitted Function
#plt.xlim([100, 120])
plt.title('Curve_Fitted_ESF_1')
plt.xlabel('piksel(px1)')
plt.ylabel('Intensity')
plt.grid()
plt.show()
```

Şekil 2.48 Optimizasyon için yazılan kod parçası



Şekil 2.49 Veriye uydurulan eğri



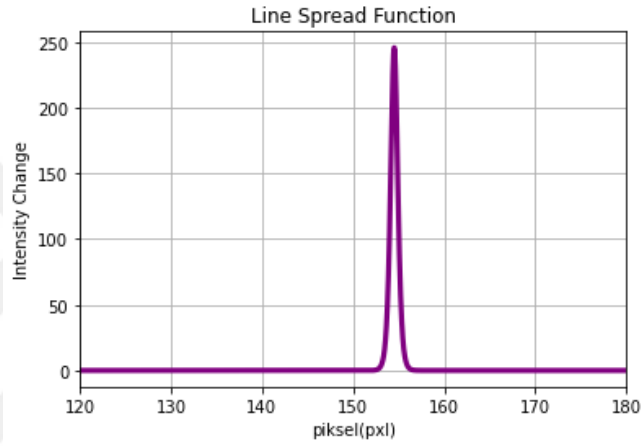
Şekil 2.50 Kenar yayılım fonksiyonu

Kenar yayılım fonksiyonu Şekil 2.50’deki gibi elde edildikten sonra birinci mertebeden türevi alınarak çizgi yayılım fonksiyonu elde edilir. Türev alınırken Python numpy kütüphanesine ait bir fonksiyon olan “`np.gradient()`” metodu kullanılmıştır. Bu metod ile türev alınıp elde edilen grafik çizdirilmiştir (Şekil 2.51 ve Şekil 2.52). Daha sonraki aşamada ise yine Python kod parçası yazılarak bir Fourier dönüşümü alınıp MTF fonksiyonuna ulaşılmıştır (Şekil 2.53 ve Şekil 2.54).

```
# LSF Derivation:
# Bu aşamada fitting işlemi sonucunda yeni fonksiyonun x değişkenine göre türevi hesaplanmaktadır.

LSF = abs(np.gradient(y_model,x_model))
plt.figure()
plt.plot(x_model,LSF,linewidth = 3.2, color ='purple' )
#plt.xlim([100, 120])
plt.title('LSF_after_Fitted_ESF_1')
plt.xlabel('piksel(px1)')
plt.ylabel('Response')
plt.grid()
plt.show()
```

Şekil 2.51 Python kodlarıyla türev alma işlemi



Şekil 2.52 Çizgi yayılım fonksiyonu

```
from scipy.fft import fftfreq,fft
x = x_model
N = len(x)

f2 = fftfreq(len(x),np.mean(np.diff(x))) ## -- >> Cycles per pixel

x1_FFT = abs(fft(LSF)) ## Taking Fourier Transform !!

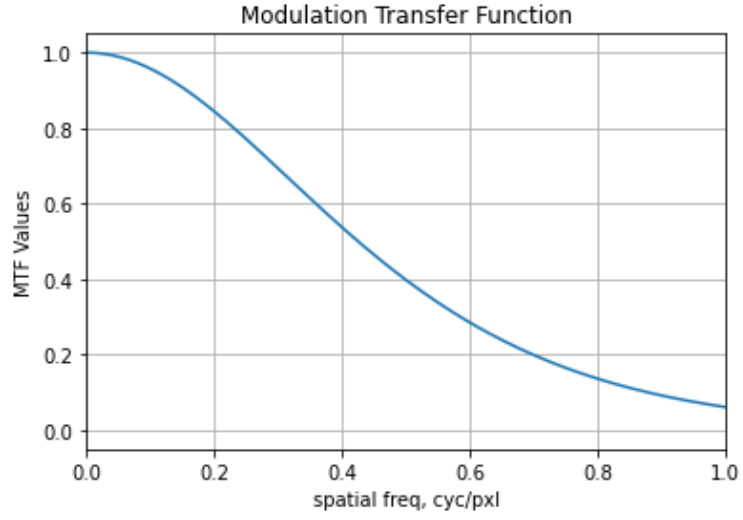
norm_x1_FFT = x1_FFT/x1_FFT[0] # By dividing zero frequency component, normalizing is done.

plt.figure()
plt.plot(f2[:N//2], norm_x1_FFT[:N//2])
plt.xlabel('spatial freq, cyc/px1')
plt.title('MTF_7.5')
plt.grid()
plt.xlim([0,1])
plt.show()

a3 = norm_x1_FFT[:N//2] ## Normalize MTF değerleri
a4 = f2[:N//2] ## Frekans eksenı.

MTF_Matrix = np.zeros((a4.shape[0],2))
MTF_Matrix[:,0] = a3
MTF_Matrix[:,1] = a4
```

Şekil 2.53 Fourier dönüşümü alan kod parçası



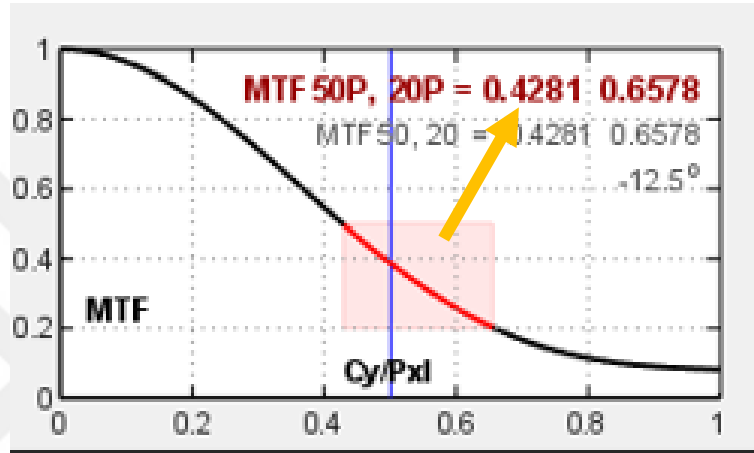
Şekil 2.54 Modülasyon aktarım fonksiyonu

MTF grafiği 2. yöntem ile elde edildikten sonra yapılması gereken işlem, IMATEST görüntü analiz yazılımının sonuçları ile bir karşılaştırmadır. Bu karşılaştırma sonucunda IMATEST yazılımından yüzde olarak farklılık hesaplanacaktır.

0	1
0.515997	0.414291
0.512164	0.416897
0.508345	0.419503
0.50454	0.422108
0.500749	0.424714
0.496973	0.427319
0.493211	0.429925
0.489464	0.432531
0.485733	0.435136
0.482016	0.437742
0.478315	0.440348

Şekil 2.55 Eğri uydurma yöntemi ile MTF değerinin frekans tablosu

Şekil 2.55’te ilk sütun MTF değerini, ikinci sütun ise cyc/pixel (piksel başına döngü) cinsinden frekans değerini göstermektedir. Mavi renk ile işaretlenmiş satır incelenecek olursa MTF değerinin %50’sine düştüğü frekans değerinin 0.4247 cyc/pxl olduğu görülmektedir. Bu değer, birinci yöntem ile hesaplanan 0.4168 cyc/pxl değerine oldukça yakın bir değerdir. Eğri uydurma yöntemi ile hesaplanan 0.4247 cyc/pxl değeri Imatest değerinden yaklaşık 0.0034 değerinde bir farklılık göstermektedir (Şekil 2.56).



Şekil 2.56 Imatest MTF grafiği çıktısı

Şekil 2.56’da sunulan Imatest yazılımının sonuçlarıyla eğri uydurma yönteminin verdiği değer arasındaki farkı yüzdesel olarak ifade edebilmek mümkündür:

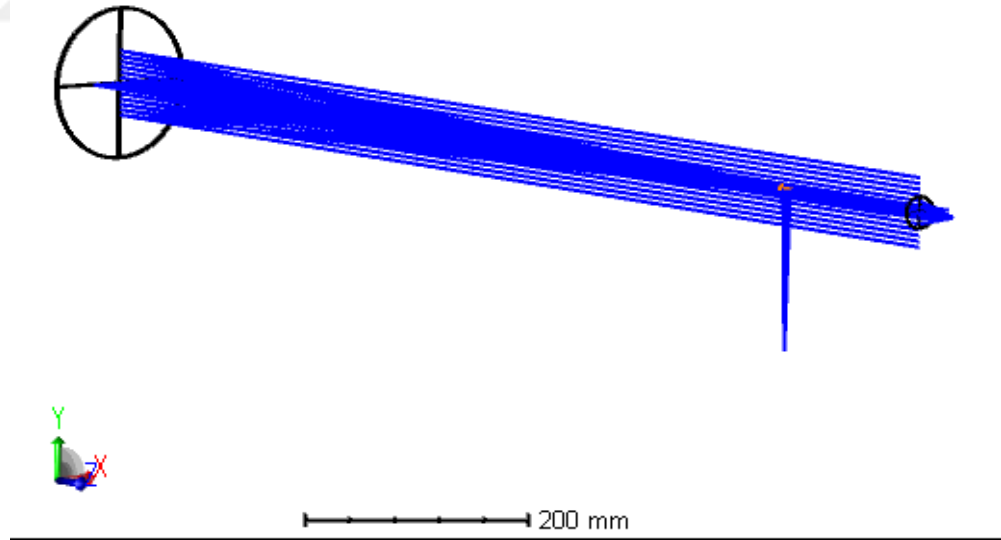
$$Fark \% = \frac{Imatest - Python}{Imatest} * 100$$

$$Fark \% = \frac{0.4281 - 0.4247}{0.4281} * 100 = \% |0.79|$$

Buradaki hesaptan görülmektedir ki kodlara test amaçlı verilen sentetik bir görüntüden iki farklı kodlama yöntemiyle elde edilen sonuçların Imatest yazılımından farkı % |3|’ün altındadır.

2.6 Zemax Görüntü Simülasyonu

Bu tez kapsamında tasarlanan ve bir örneği gerçekleştirilen MTF ölçüm sistemi mekanik, optik ve yazılım tasarımlarını içermektedir. Bu tez çalışmasında ağırlıklı olarak yazılım tasarım kısmına yoğunlaşmış olunmakla birlikte Zemax OptikStudio programı üzerinden bir kolimatör teleskop tasarımı yapılmıştır (Şekil 2.57). Sonrasında ise deneysel çalışmalar kısmında anlatılacağı üzere SWIR kameraya takılan 50mm f/1.4 özelliklerine sahip bir objektifi modelleyebilmek için Zemax ortamında ideal paraxial lens (mercek) seçilmiş ve odak uzaklığı mekanik ölçüleriyle birlikte Zemax'ın LENS Data Editor kısmına girilmiştir (Şekil 2.58). Sıralı modda ışın izleme analizi yoluyla odak düzlemine ışıklar yoğunlaştırılarak bir görüntü simülasyonu (benzetimi) yapılmıştır.



Şekil 2.57 İdeal mercekli kolimatör teleskop yapısı

	Surface Type	Comment	Radius	Thickness	Material	Coating	Clear Semi-Dia	Chip Zone	Mech Semi-Dia	Conic
0	Standard ▼		Infinity	200,000			0,000	0,000	0,000	0,000
1	Coordinate Break ▼			0,000	-		0,000	-	-	
2	Standard ▼		Infinity	0,000	MIRR...		7,000 U	0,000	7,000	0,000
3	Coordinate Break ▼			-1000,0...	-		0,000	-	-	
4	Standard ▼		2500,000	1200,000	MIRR...		20,000 U	0,000	20,000	-1,0...
5	Paraxial ▼			50,080			15,000 U	-	-	
6	Standard ▼		Infinity	-			3,527E-07	0,000	3,527E-07	0,000

Şekil 2.58 Zemax Lens Data Editor

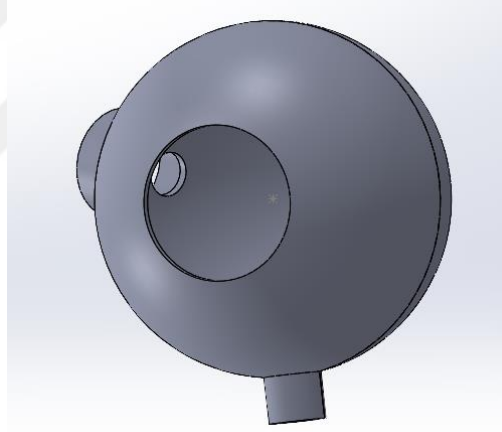
Zemax'ta tasarlanan MTF ölçüm sistemi, ideal bir ortamı temsil etmekle birlikte kırınım etkileri Zemax programı tarafından mümkün kılınmaktadır. Buna göre Şekil 2.10'da sunulan görüntü Şekil 2.58'deki Zemax tasarımına girdi olarak verilip görüntü simülasyonu yaptırıldıktan sonra optik kırınımın limitleri ve etkilerinin görüldüğü bir şekil, ideal mercek odağında belirmektedir (Şekil 2.59). Bir sonraki aşama ise benzetimi yapılmış Zemax çıktısının MTF grafiğinin yazılan Python kodları tarafından çıkarılmasıdır. Bu kısma ait sonuçlar sonraki bölümde sunulacaktır.



Şekil 2.59 Zemax ortamında görüntü simülasyonu

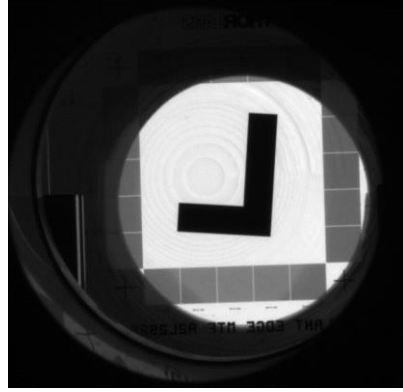
3. DENEYSEL ÇALIŞMALAR

Laboratuvar ortamında farklı ışık kaynakları ve detektörler kullanılarak oluşturulan deneysel düzeneklerin çıktıları üzerinden yazılan kodlar test edilmiştir. Daha önceden sentetik (yapay) görüntüler üzerinde çalışan MTF kodlarının deneysel düzenekler ile de başarılı bir şekilde çalıştığı gösterilmiştir. Laboratuvar ortamında optik masa üzerinde deneysel bir optik düzenek kurulmuştur. Öncelikle kullanılacak ışık kaynakları ve detektörler belirlenmiştir. Bu aşamada ışığı homojen hale getirmek ve tekdüze bir ışıma ile eğik kenar hedefini aydınlatmak önemli bir deneysel parametredir. Bu amaçla ışığı homojenize eden küreler Solidworks yazılımıyla çizilip 3-D (3 boyutlu) yazıcı tarafından üretilmiş ve deneysel sistemlerde kullanılmıştır.



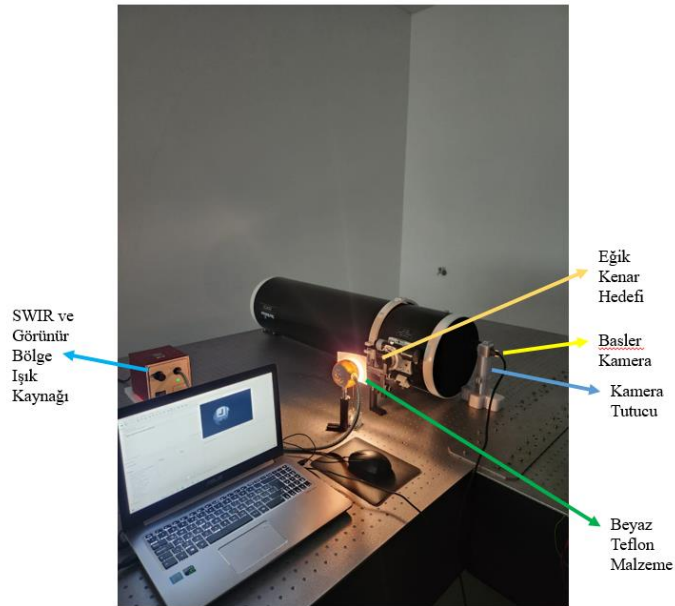
Şekil 3.1 Işık homojenleştirici küre (integrating sphere)

Şekil 3.1'deki küre PLA beyaz renkli malzemeden üretilmiştir. Kürenin içerisinde ışık yansıtıcılığının yüksek olması istenildiğinden beyaz renkli bir malzeme kullanılmıştır. Kürenin sol üst bölgesinden SWIR ve görünür bölgede ışıma yapabilen ışık kaynağı veya 1550nm ışık veren LED yerleşecek şekilde bir delik bölmesi ayrılmıştır. Ancak bu durumda bile ışığın yeterince tekdüze ve homojen olmadığı görülmüştür. Kürenin arka kısmından yansıyan taraklı halkasal bir yapı detektörde elde edilen görüntü üzerinde belirmiştir (Şekil 3.2).

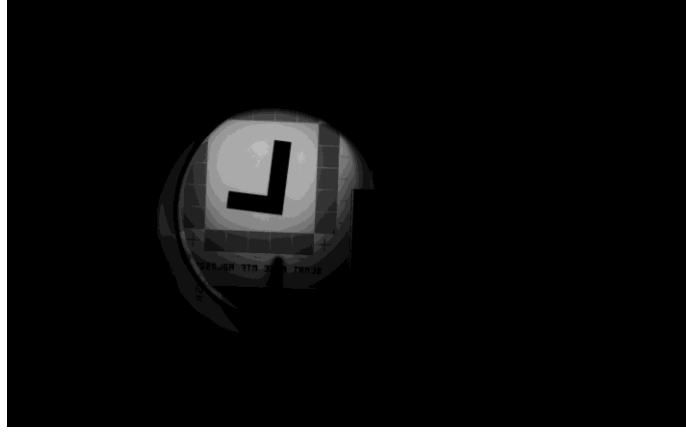


Şekil 3.2 Sadece küre ile aydınlatılan görüntü

Şekil 3.2’den görüldüğü üzere siyah ve eğimli “L” harfinin arka kısmında küreye ait bir halka deseni mevcuttur. Bu durum, deneysel olarak kabul edilebilir bir gözlem değildir zira doğrudan görüntü kalitesini etkilemektedir. Bu halkalı yapıyı ortadan kaldırmak amacıyla daha iyi ışık homojenizasyon teknikleri denenmiştir. Örneğin yansıtıcılığı oldukça yüksek bir malzeme olan teflon Şekil 3.3’te verilen kürenin çıkış açıklığına konularak tekrardan ölçüm alınmıştır. Bu sistemde kullanılan objektif, 50mm f/1.4 lensidir ve görünür bölgede çalışabilen bir ışık kaynağı altında 400nm-900nm aralığında görüntü alabilen bir Basler kameraya takılmıştır.

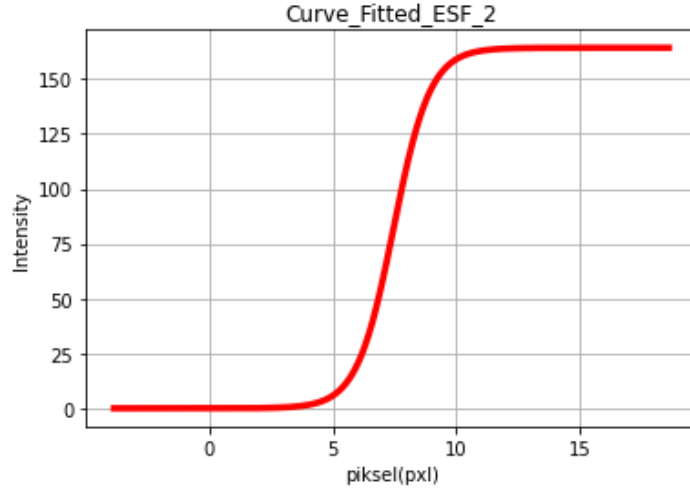


Şekil 3.3 Küre ve teflon malzeme ile aydınlatılan düzenek

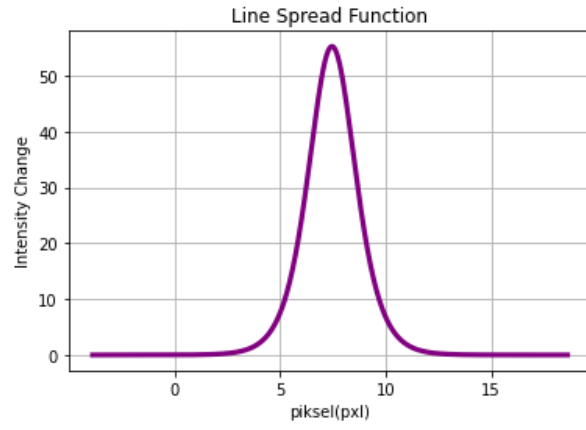


Şekil 3.4 Teflon malzemenin görüntü üzerindeki etkisi

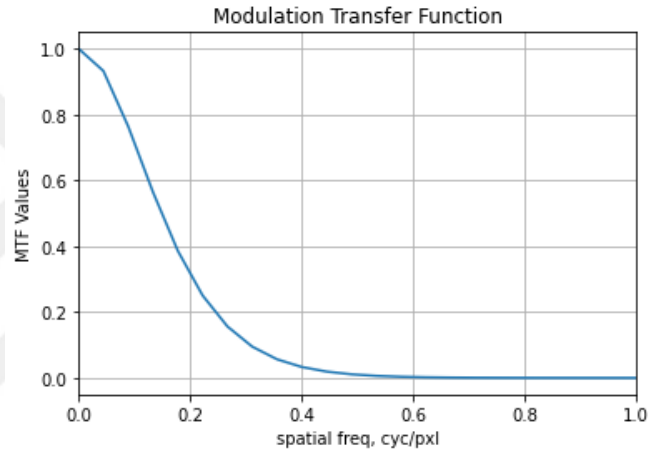
Şekil 3.4 incelendiğinde parlaklığın azalmış olduğu görülmesine rağmen halka deseninin etkisi ortadan kaybolmuştur. Şekil 3.3 ile kurulan düzeneğe basler kamera ve önüne takılan objektif ile birlikte tüm sistemin performansı incelenmiştir ve kodların farklı türdeki deneysel düzeneelerde çalışıp çalışmadığı kontrol edilmiştir. Bu bağlamda Şekil 3.4'te verilen görüntünün MTF'i yazılan kodlar tarafından çıkarılmış ve Şekil 3.5, Şekil 3.6 ve Şekil 3.7'de sunulmuştur. Devamında düzeneğin MTF'ini etkileyen önemli optik ve optomekanik bileşenlere yer verilmiştir (Şekil 3.8 ve Şekil 3.9).



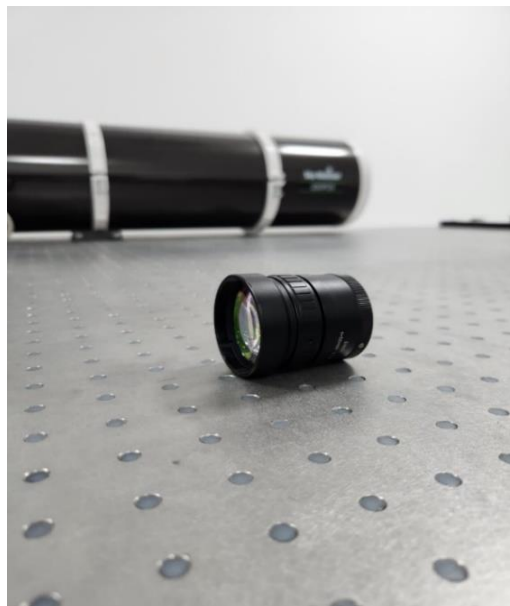
Şekil 3.5 Kenar yayılım fonksiyonu (ESF)



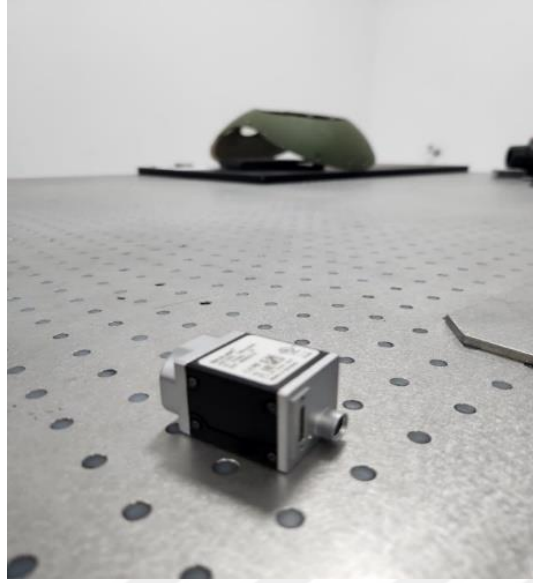
Şekil 3.6 Çizgi yayılım fonksiyonu (LSF)



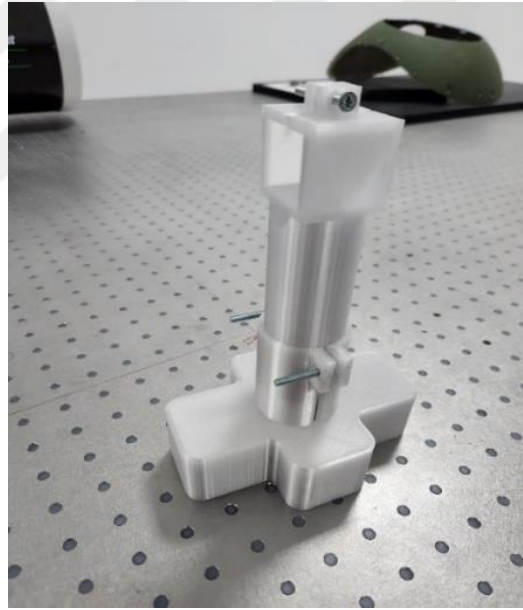
Şekil 3.7 Modülasyon transfer fonksiyonu (MTF)



Şekil 3.8 50mm f/1.4 objektif



Şekil 3.9 Basler kamera



Şekil 3.10 Kamera tutucu

Şekil 3.10’da verilen kamera tutucu biri alt tabla ve diğeri de yükseklik olacak şekilde iki ayrı parça olarak Solidworks ortamında çizilmiştir. Daha sonrasında bu parçalar üretime verilmiştir. Üretim işlemi 3-D Printer (3 boyutlu yazıcı) tarafından gerçekleştirilmiştir ve deneysel düzende kullanılmadan önce üretilen parçaların birleştirilmesi (montajı) yapılmış olup kullanıma hazır hale getirilmiştir (Şekil 3.11)

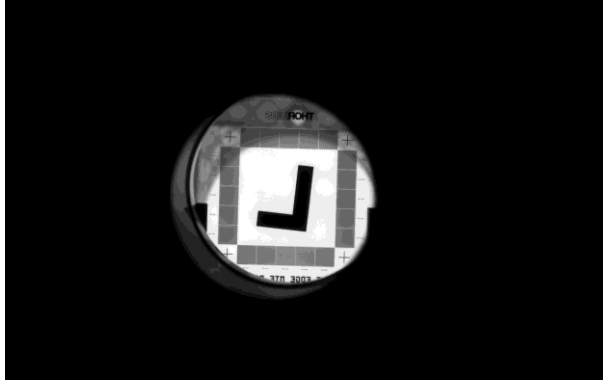


Şekil 3.11 Basler kameranın monte edilmiş hali

Yeni bir ışık homojenleştirici küresi tasarlanmış olup deneysel aydınlatma düzeneğinde kullanılmıştır ve alınan görüntü devamında sunulmuştur (Şekil 3.12 ve Şekil 3.13).

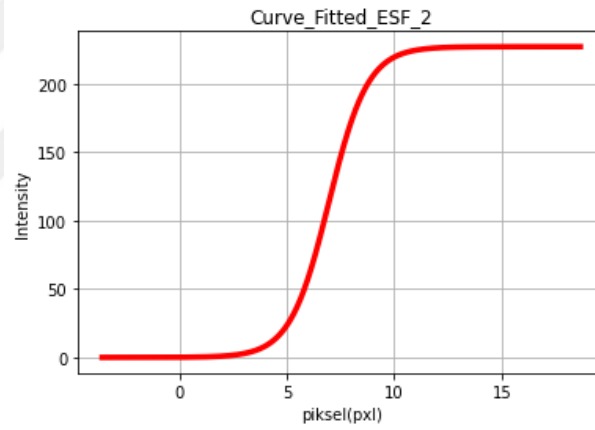


Şekil 3.12 Karanlık ortam laboratuvar düzeneği

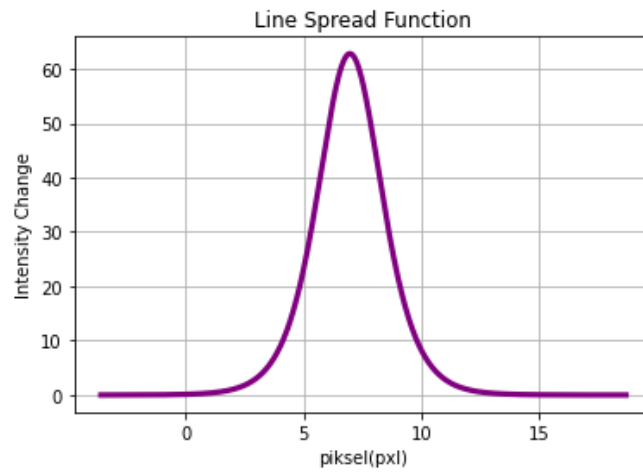


Şekil 3.13 Yeni küre ile aydınlatılan görüntü

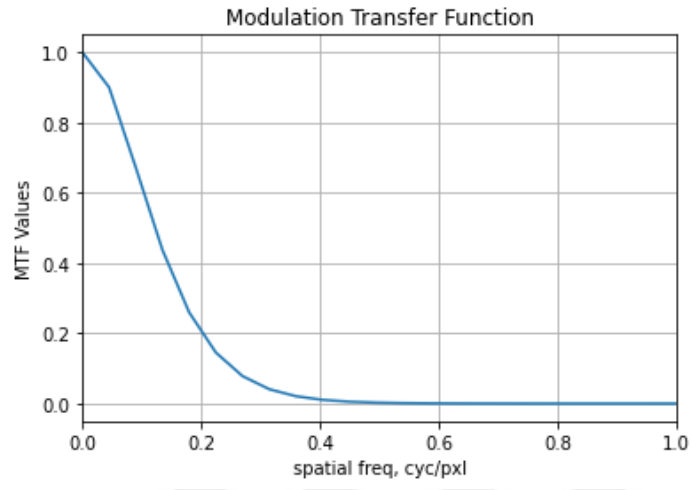
Şekil 3.13 üzerinden çıkarılan ESF, LSF ve MTF grafikleri sırasıyla Şekil 3.14, Şekil 3.15 ve Şekil 3.16’da verilmiştir.



Şekil 3.14 Yeni küre ile kenar yayılım fonksiyonu



Şekil 3.15 Yeni küre ile çizgi yayılım fonksiyonu



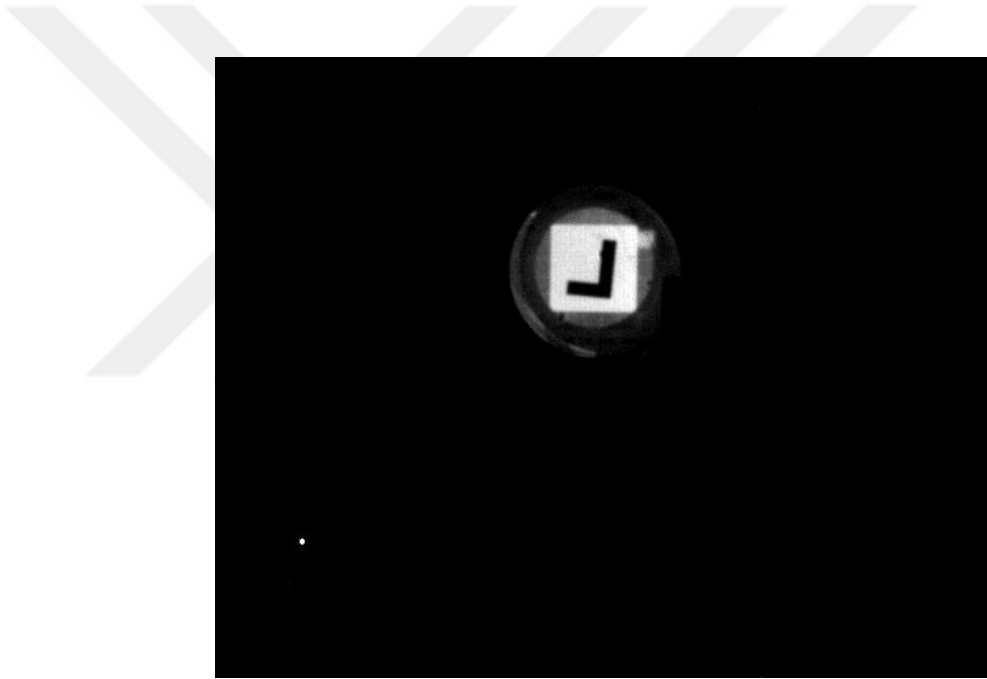
Şekil 3.16 Yeni küre ile modülasyon aktarım fonksiyonu



Şekil 3.17 SWIR kamera ile kurulan deneysel düzenek

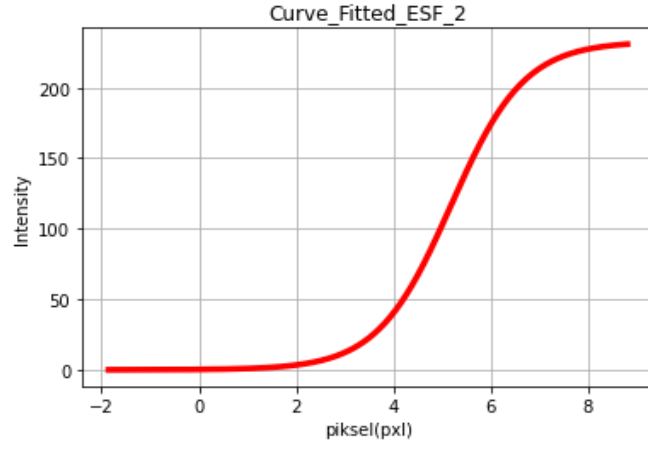
Şekil 3.17’de ışık kaynağından çıkan ışık, eğik kenar hedefine gelmeden önce beyaz bir küre içerisine girip burada çoklu yansımalarla uğradıktan sonra tekdüze ve homojene yakın bir şekilde hedefe ulaşmaktadır. Eğik kenar hedefi ışık kolimatörü olarak kullanılan teleskopun odağına yerleştirilmiştir. Hedeften geçerek gelen ışınlar teleskopun birinci düzlem aynasından yansdıktan sonra ikinci parabolik aynadan yansıyarak uzaktan gelen bir ışık ışını gibi kolime olurlar ve kamera objektifine bu şekilde ulaşırlar.

Burada ölçülecek olan MTF, kamera ile birlikte objektif de dâhil olacak şekilde tüm sistemin MTF’idir. Kameradan alınan görüntü bir USB bağlantısı ile arayüze aktarılır, sonra da görüntüyü analiz etmek için yazılmış olan Python kodlarına verilir (Şekil 3.18).

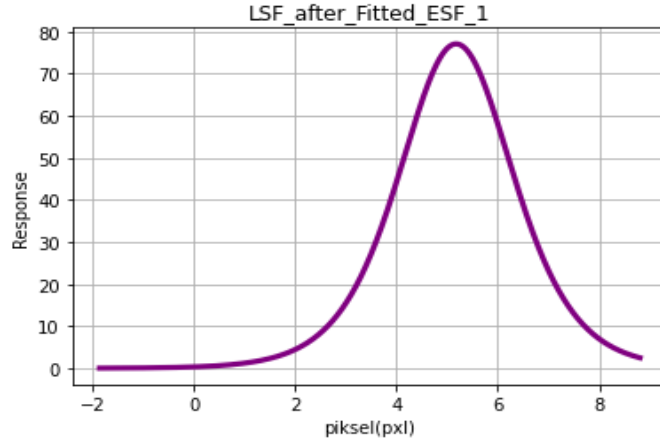


Şekil 3.18 Kameradan elde edilen görüntü

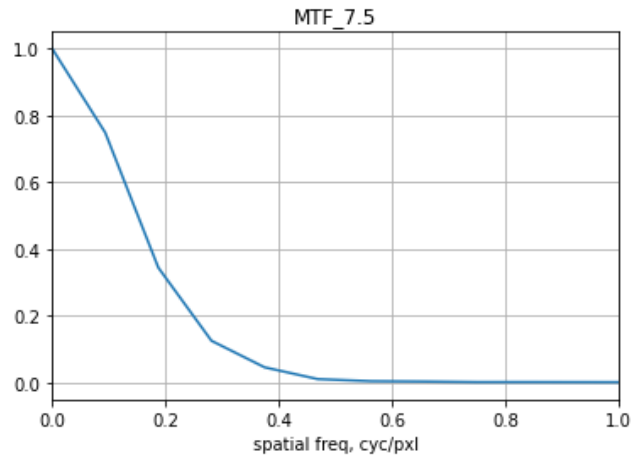
Şekil 3.18’de verilen deneysel görüntüden elde edilen ESF, LSF ve MTF grafikleri sırasıyla Şekil 3.19, Şekil 3.20 ve Şekil 3.21’de sunulmuştur.



Şekil 3.19 Kenar yayılım fonksiyonu



Şekil 3.20 Çizgi yayılım fonksiyonu



Şekil 3.21 Modülasyon aktarım fonksiyonu

Bu noktaya kadar gösterilmiştir ki MTF ölçümü için geliştirilen algoritma ve yazılan kodlar SWIR veya Basler kamera fark etmeksizin belirlenen 50mm f/1.4 objektifle birlikte deneysel olarak çalışabilmektedir. Yani deneysel görüntüler üzerinden herhangi bir satürasyon olmadan ve gürültü etkilerini minimize edecek şekilde sırasıyla ESF, LSF MTF grafikleri elde edilebilmektedir. Bir sonraki adımda karşılaştırmalı analizler yapılacaktır.

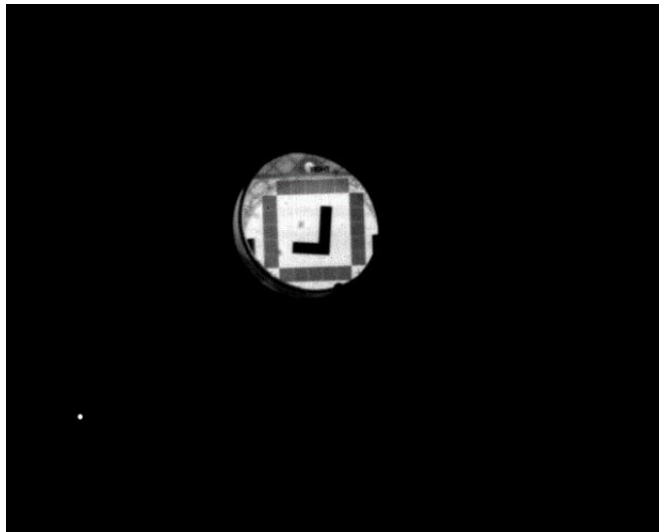
3.1 Karşılaştırmalı MTF Analizi

Elde edilen deneysel bulguların doğruluğunu kontrol etmenin en temel metotlarından bir tanesi, birbiriyle ilintili olmayan yöntemler ile sonuçları kıyaslamaktır. Tez kapsamında teleskoptan kolime bir şekilde gelen ışınları analiz ederek MTF ölçümü yapan SWIR kamera sistemi kurulmuştur (Şekil 3.22). 50mm f/1.4 objektifi kullanılmıştır. Düzenekteki optik elemanlar ve eğik kenar hedefi değiştirilmeden sadece görünür bölge kamerası yerine SWIR detektör kullanılmıştır. Yeni kullanılan kameranın USB 3.0 bağlantısı kurularak bilgisayar ortamına aktarılan arayüzü üzerinden görüntü “.png” dosyası olarak kaydedilip analiz edilmek üzere MTF Python kodlarına girdi olarak verilmiştir. Karşılaştırmalı MTF analizi yaparken değişmemesi gereken en temel araç detektör sistemi ve objektiftir; çünkü hem detektör hem de objektifle birlikte tüm sistemin MTF’i ölçülmektedir. Burada iki farklı optik aydınlatma yöntemi kullanılmış ve sonuçları analiz edilmiştir.

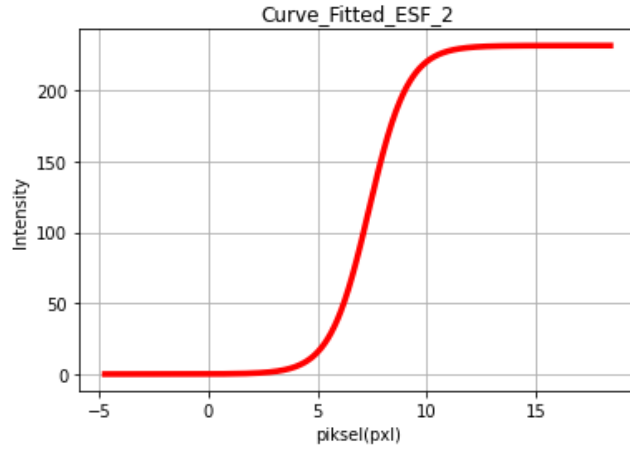


Şekil 3.22 SWIR kamera ölçüm düzeneği

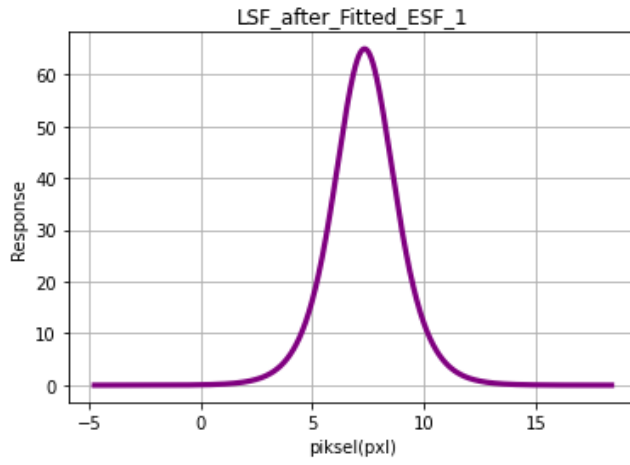
Şekil 3.22’de verilen sistemden alınan görüntü ve ölçülen fonksiyonlar sırasıyla Şekil 3.23, Şekil 3.24, Şekil 3.25 ve Şekil 3.26’da sunulmuştur.



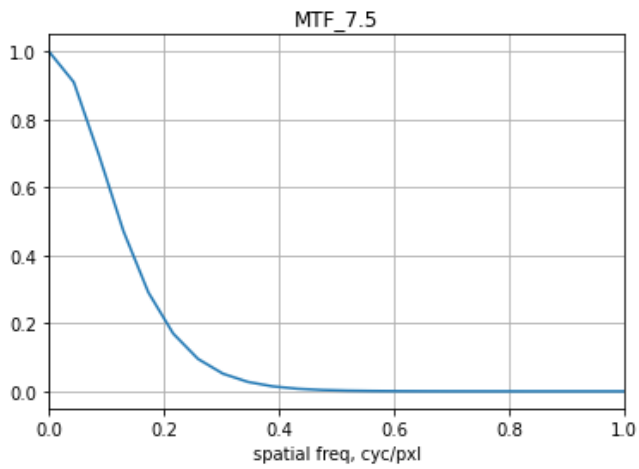
Şekil 3.23 Deneysel görüntü



Şekil 3.24 SWIR kamera kenar yayılım fonksiyonu (ESF)



Şekil 3.25 SWIR kamera çizgi yayılım fonksiyonu (LSF)



Şekil 3.26 SWIR kamera modülasyon aktarım fonksiyonu (MTF)

Optik bileşenler değiştirilmeden aynı düzenek duvar aydınlatması için kurulmuş olup tekrar MTF ölçümü yapılmıştır (Şekil 3.27 ve 3.28). Buradaki önemli konulardan biri eğik kenar hedefinden sonra gelen optik bileşenlerin MTF’i hangi oranda etkilediğinin ölçülmesidir. İdeal durumda MTF’in değişmemesi beklenir ve bu durumu test etmek amacıyla arada hiçbir optik bileşenin olmadığı sadece duvara asılan bir hedefin MTF’inin ölçüldüğü bir düzenek kurulmuştur. Tekdüzeye (uniform) yakın bir ışık aydınlatması yapılarak deneysel görüntü alınmış ve kaydedilerek analiz edilmiştir.

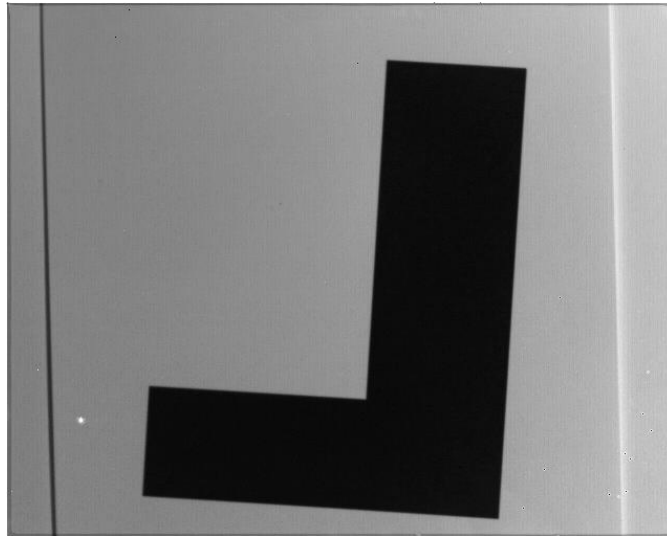


Şekil 3.27 Duvara asılı eğik kenar hedefi ve SWIR kamera



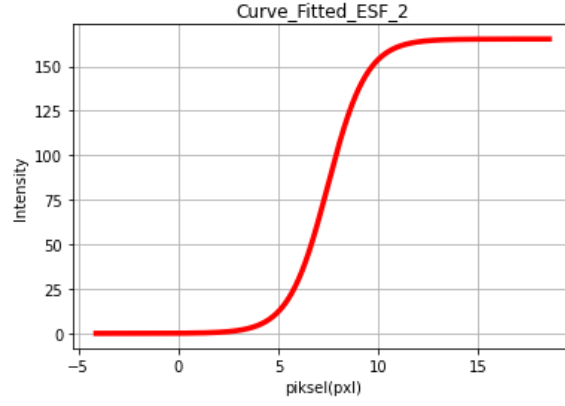
Şekil 3.28 Duvar aydınlatması ve MTF ölçümü

Şekil 3.27 ve 3.28’de gösterilen düzende alınan görüntü Şekil 3.29’daki gibidir:

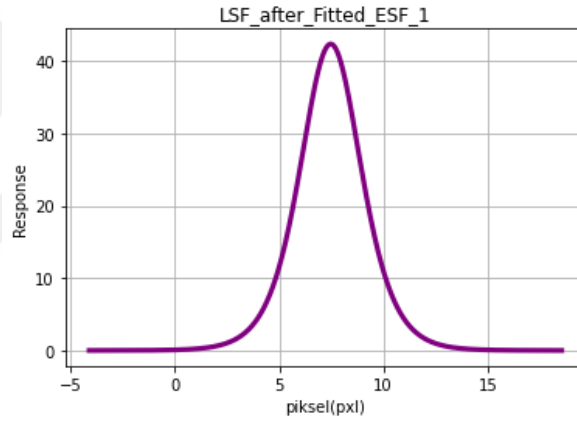


Şekil 3.29 Duvara asılı deneysel görüntü

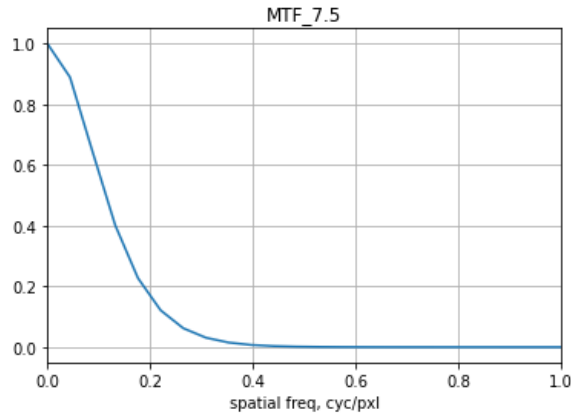
Şekil 3.29'dan çıkarılan ESF, LSF ve MTF fonksiyonları sırasıyla Şekil 3.30, Şekil 3.31 ve Şekil 3.32'de sunulmuştur.



Şekil 3.30 Duvar aydınlatması kenar yayılım fonksiyonu (ESF)

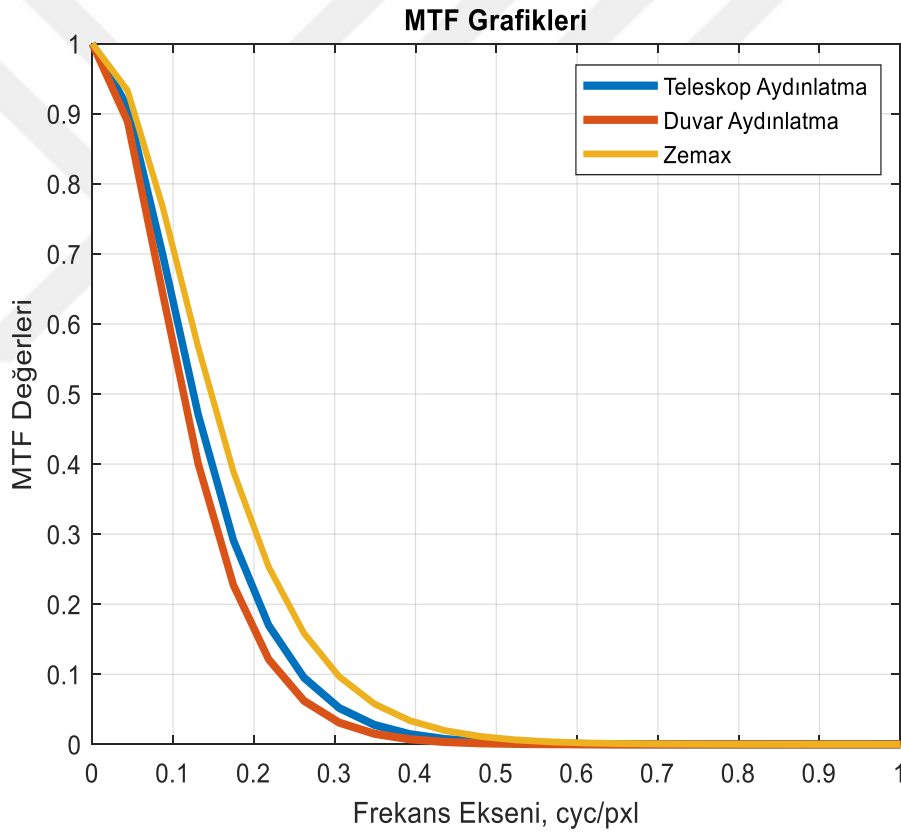


Şekil 3.31 Duvar aydınlatması çizgi yayılım fonksiyonu (LSF)



Şekil 3.32 Duvar aydınlatması modülasyon aktarım fonksiyonu (MTF)

Kurulan her iki düzenden elde edilen sonuçları bilimsel olarak kıyaslamak amacıyla MTF grafiklerinin her iki eksenindeki değerler, yapılan her bir analizin sonucunda ayrı birer “.txt” dosyasına kaydedilmiştir. Bu dosyalarda sırasıyla birinci sütunda MTF değerleri ve ikinci sütunda karşılık gelen ilgili frekans değerleri bulunmaktadır. Bu dosyalar işlenmek üzere MATLAB kodlarına verilmiştir. Ek olarak, Şekil 2.59’ da verilen Zemax OptikStudio görüntü simülasyonu çıktısına ait görüntünün de MTF’i çıkarılıp sonuçlar MATLAB kodlarına verilmiştir. Böylelikle birbiriyle ilintili olmayan 3 farklı yöntem ile MTF sistemi ve kodlar test edilmiş, sonuçlar bilimsel bir grafikte sunulmuştur (Şekil 3.33).

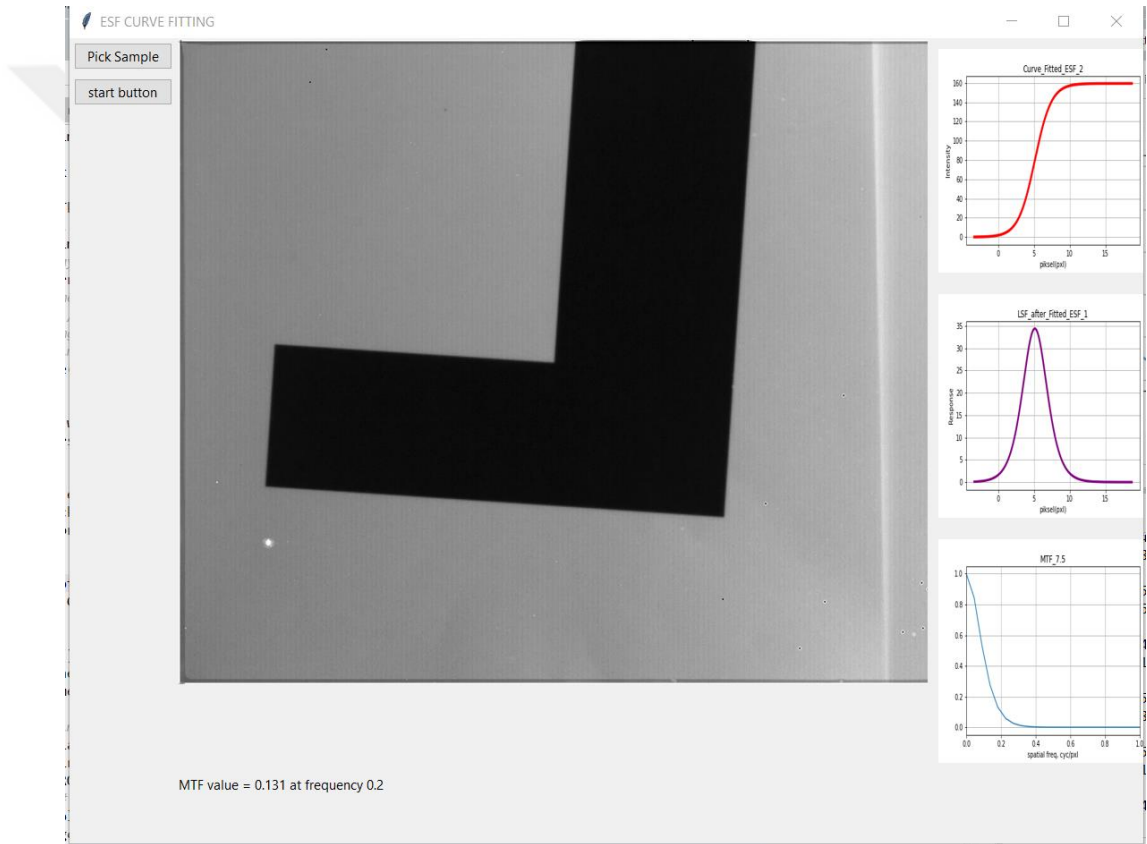


Şekil 3.33 Birbirleriyle ilintili olmayan 3 farklı yöntemin çıktıları

Şekil 3.33’ten de görüldüğü üzere tamamen birbirinden bağımsız ölçüm ve benzetim metotlarıyla yapılan MTF ölçümlerinin çıktıları aynı grafik üzerinde çizdirildiğine birbiriyle uyumlu dağılımlar gözlenmektedir. Bu analiz, yazılan kodların test edilmesi açısından önemli bir bilimsel metriktir.

3.2 Python ile Arayüz Uygulaması

Bilgisayara ait bir webcam üzerinden veya harici USB 3.0 bağlantısı yoluyla video akışını ekrana yansıtan ve üzerinde bulunan butonlar sayesinde akan video üzerinden örnek bir ROI seçerek sırasıyla kenar yayılım fonksiyonu, çizgi yayılım fonksiyonu ve modülasyon aktarım fonksiyonlarını ekrana bastıran ve istenilen özel bir frekanstaki MTF değerini ekranın sol alt kısmında gösteren bir kullanıcı arayüzü (GUI: Graphical User Interface), Python “Tkinter” ve “OpenCv” kütüphanesi aracılığıyla yazılmıştır.



Şekil 3.34 MTF ölçüm sistemi arayüzü (GUI)

Şekil 3.34’te verilen arayüz ekranının ana-orta kısmında MikroTasarım firmasına ait SWIR kamera ve 50mm f/1.4 objektifle alınan duvar aydınlatması görünümü yer almaktadır. Ekranın sol üst köşesindeki “start button” yazan butona tıklanıldığında arayüz ekranı üzerinde görüntü akışı sağlanmaya başlanılmaktadır. Pick Sample butonuna basıldığında ise otomatik kenar tespit algoritması devreye girmekte, siyahtan beyaza

geçişin en keskin olduğu kenar (edge) bulunmaktadır. Yazılan algoritma, tespit edilen kenarın orta noktası üzerinden bir koordinat seçimini yaparak ve bu koordinatları merkez alıp kuzey-güney ve doğu-batı yönlerinde olacak şekilde 10'ar piksellik taramalar yaparak 20x20'lik bir ROI seçimi gerçekleştirir. Seçilen ROI ekrana bastırılır ve klavyeden “q” harfine basıp ROI ekranı kapatıldıktan sonra nümerik hesaplar devreye girer. İlk sonuçlar ekranın sağ kısmındaki bölüme yazdırılır ve daha sonrasında kameradan alınan her yeni çerçeve (frame) için hesaplamalar yapılarak elde edilen fonksiyonlar (ESF, LSF ve MTF) ekrana bastırılır.

3.2.1 Otomatik kenar tespit algoritması

Tez kapsamında yazılmış olan ve Şekil 3.34'te sunulan kullanıcı arayüzü (GUI: Graphical User Interface) için yazılmış görüntü üzerindeki kenarları otomatik tespit eden bir algoritma mevcuttur. Bu algoritma yazılırken hem Python OpenCV kütüphanesinden yararlanılmış hem de görüntü işleme teknikleri kullanılmıştır. Kod, temelde ve başlangıçta kendisine analiz için verilen görüntü üzerindeki siyah-beyaz geçiş noktalarını tespit etmektedir. Bu operasyon için Sobel filtresi kullanılmıştır.

Sobel filtre görüntülerdeki objelerin sınırlarını tespit etmek için kullanılan bir filtreleme yöntemidir. Sobel filtreleme yöntemiyle görüntü üzerinde hem yatay hem de dikey olarak kenar belirleme yapılabilmektedir. Sobel filtresinin iki adet kernel (çekirdek) matrisi bulunmaktadır. Bu matrislerden biri x yönünde uygulanırken diğeri ise y yönünde uygulanmaktadır. Daha sonrasında her bir yönden elde edilen pikseller birleştirilmektedir.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \rightarrow \text{Yatay yön için}$$

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \rightarrow \text{Dikey yön için}$$

$$G = \sqrt{G_x^2 + G_y^2} \rightarrow \text{Birleřtirme İřlemi}$$

Yüksek çözünürlüklü resimlerde veya gereken durumlarda kernel boyutu değıştirilebilir. Bazı uygulamalarda yatay ve dikeyde elde edilen kenarların birbirinden bağımsız olarak kullanılma ihtiyacı olabilmektedir. Bu tür durumlarda birleřtirme işlemleri yapılmayabilir.

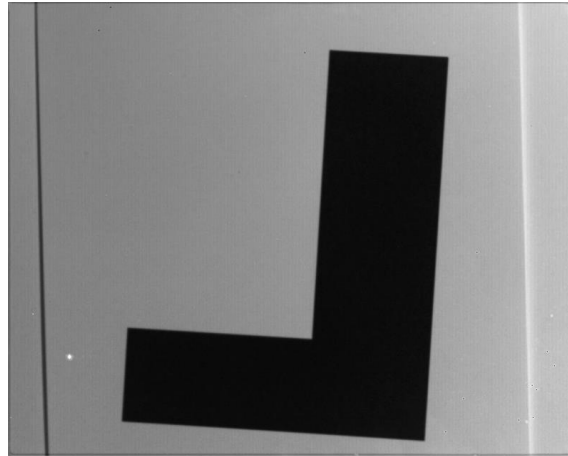
Sobel kernel filtrelerinin anlamlı yoğunluğun olduđu alana uygulanması ile elde edilen değerler oldukça yüksek çıkarken sabit yoğunlukta bir bölgeye uygulanması sonucunda elde edilen değerler sıfıra oldukça yakın çıkacaktır. Uygulamaya geçildiđi zamanda ise öncelikle üzerinde çalışılmak istenilen görüntü Python Spyder derleyici ortamında açılır (Şekil 3.35).

```
img_1 = cv.imread("Duvar_SWIR_50mm.png",0)

print("first_image:",img_1.shape, img_1.dtype) # Görüntü özelliklerini yazdırıyor
cv.imshow('original_image_0', img_1) # İlk görüntüyü gösteriyor.
print(np.max(img_1), np.min(img_1))
cv.waitKey(50000)
```

Şekil 3.35 Duvar aydınlatmasından gelen görüntü

Yukarıdaki kodlar çalıştığında SWIR ışık kaynağıyla duvarın aydınlatıldığı görüntü ekranda belirecektir:

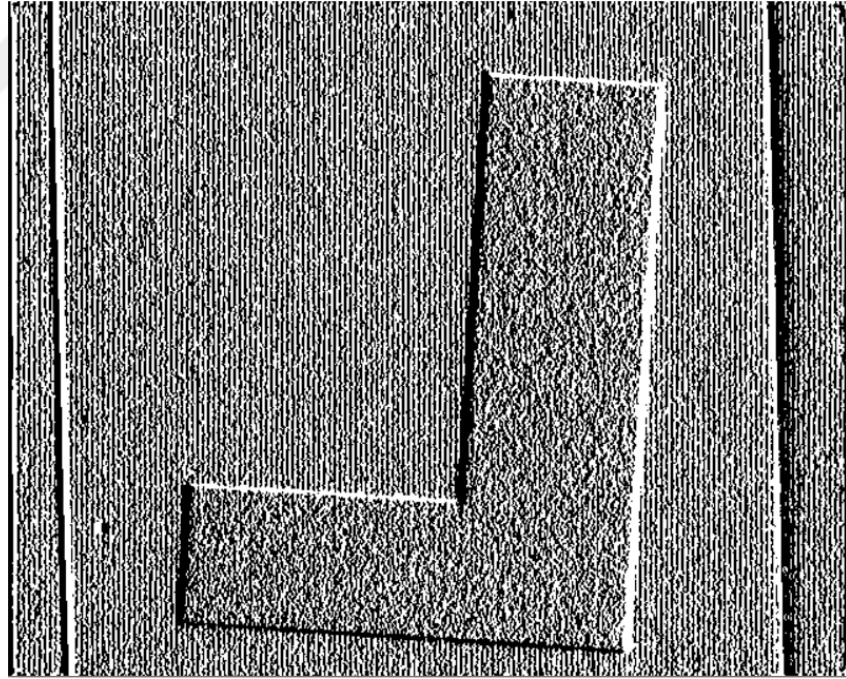


Şekil 3.36 Duvar aydınlatması

Şekil 3.36’da verilen görüntüye öncelikle sobel filtresi uygulanıp değişimin/yoğunluğun olduğu bölgeler tespit edilecektir. Python’da “cv.Sobel()” fonksiyonun parametreleri girilecektir. Bunun için bu fonksiyona verilen görüntüye ilaveten görüntünün kaç bitlik bir derinlikte okunacağını belirten görüntü derinliği parametresine, hangi yönde Sobel filtresinin uygulanacağına (ilgili örnekte x yönünde uygulanacaktır) ve kernel büyüklüğüne (standart olarak 3 verilir) ihtiyaç duyulmaktadır (Şekil 3.37).

```
image_trial = img_1 # Kodun devamında "ROI_Image" olarak işlenecektir.  
  
sobel_x = cv.Sobel(img_1, cv.CV_64F, 1, 0, ksize=3)  
cv.imshow("sobel x", sobel_x)  
print(np.max(sobel_x), np.min(sobel_x))  
cv.waitKey(50000)
```

Şekil 3.37 Sobel filtre uygulaması



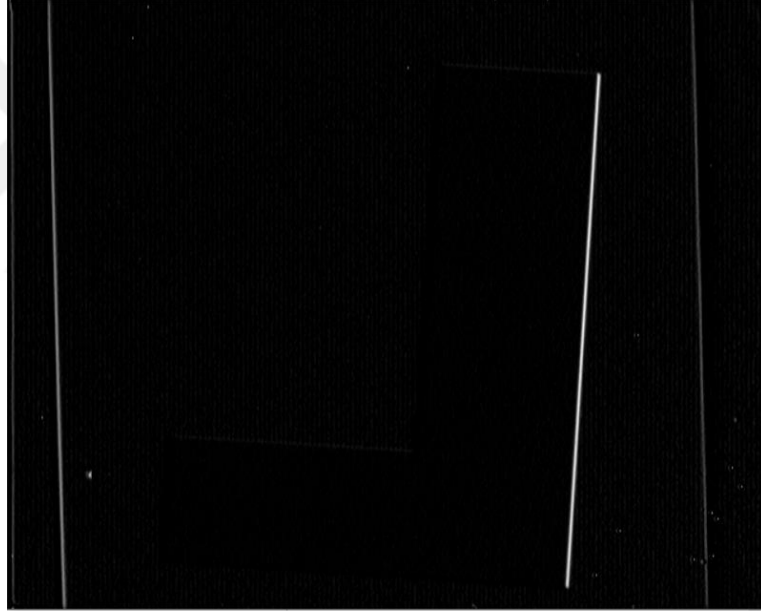
Şekil 3.38 Sobel filtre uygulaması sonraki görüntü

Bir sonraki aşamada Şekil 3.38’de verilen sobel filtresinden geçmiş görüntü normalize edilecektir, yani piksel değerleri 0 – 255 aralığına çekilecektir. Burada üzerinde çalışılan ana girdi görüntüsünün 2 boyutlu gri-skala (gray scale) olduğu unutulmamalıdır.

Öncelikle sobel görüntüsündeki negatiflikler yok edilir çünkü normalizasyon buna göre tasarlanmıştır (0-255 aralığı). Daha sonrasında Sobel görüntüsü matrisi en büyük elemanına bölünüp 255 ile çarpıldığında bir tür ölçekleme işlemi yapılmış olunur (Şekil 3.39).

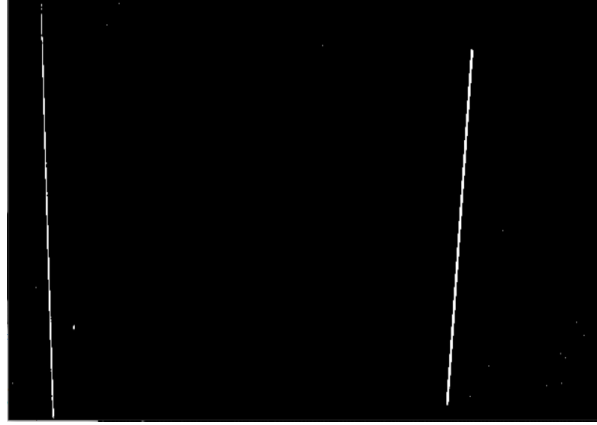
```
sobel_x[sobel_x<0]=0 # eksi değerler sıfır olmalı, normalization ona göre tasarlandı, minimum 0 olmalı  
sobel_x = np.uint8(255*sobel_x/np.max(sobel_x))  
print(np.max(sobel_x), np.min(sobel_x))  
cv.imshow("normalized x", sobel_x)  
cv.waitKey(50000)
```

Şekil 3.39 Görüntü normalizasyonu



Şekil 3.40 Normalize edilmiş Sobel görüntüsü

Şekil 3.40’da verilen görüntüye bir filtreleme işlemi daha uygulanacaktır. Bu işlemde 100 değerinin aşağısındaki piksel değerleri 0’a çekilirken 100 ve yukarısındaki değerler de 255’e çekilmektedir. Bu işlemin temel mantığı, en yüksek geçişlerin olduğu bir diğer deyişle en keskin köşelerin olduğu bir görüntüyü elde etmektir (Şekil 3.41).



Şekil 3.41 Eşik algoritması uygulanması sonucu elde edilen görüntü

Bu aşamada devreye daha kompleks bir görüntü işleme algoritması devreye girmektedir. Bu fonksiyon OpenCV kütüphanesinin `cv.connectedComponentsWithStats()` fonksiyonudur. Bu metot görüntü üzerinde birbiriyle aynı piksel değerine sahip ve geometrik olarak birbirine bağlı bölgeleri tespit eder ve bu bölgeler hakkında istatistiksel bilgiler döner (Şekil 3.42). Bu fonksiyona sobel filtresinin çıktısı ve connectivity parametresi girilmiştir. Girilen connectivity parametresinin değeri 8 olarak seçildiğinde görüntü üzerinde sadece kuzey-güney ve doğu-batı yönlerinde değil, çapraz yolların da analiz edilmesi istenilmiş olunur. Fonksiyonun birinci çıktısı `nb_components` değişkeninde tutulmaktadır ve toplamda görüntü üzerindeki bağlantılı bölgelerin sayısını vermektedir. Output değişkeni her pikselin hangi bileşene ait olduğunu gösteren etiketli bir görüntüyü temsil etmektedir. Statss değişkeninde her bileşen için istatistikleri içeren bir matris (sol üst köşe, genişlik, yükseklik ve alan gibi bilgiler) tutulmaktadır. Son olarak “centroids” değişkeninde her bileşenin ağırlık merkezleri bulunmaktadır.

```
nb_components, output, statss, centroids = cv.connectedComponentsWithStats(sobel_thresh, connectivity=8)
sizes = statss[:, -1]

max_label = 1
max_size = sizes[1]
for i in range(2, nb_components):
    if sizes[i] > max_size:
        max_label = i
        max_size = sizes[i]
```

Şekil 3.42 Bağlı bileşen analizi

Şekil 3.42 incelendiğinde öncelikle “statss” değişkeninin en son sütununa ulaşıldığı ve bunun da “sizes” isimli bir değişkene atandığı görülmektedir. Burada “statss” değişkeninde tutulan değerler, tespit edilen bağlantılı bölge kümelerinin veya bir diğer deyişle bileşenlerin alanlarıdır. Yani bölgelerin içerdikleri piksel sayılarıdır. Yazılan “for döngüsüyle” 0 piksel değeri için tespit edilen bağlantılı bölgenin alanı hariç maksimum sayıda piksel içeren bağlantılı bölgenin piksel sayısına ve ilgili indeksine ulaşılır. Nihayetinde ihtiyaç duyulacak parametreler görüntü üzerinde ROI seçilmesini mümkün kılacak merkez koordinatlar olduğundan öncelikle output görüntüsünün boyutlarında bir sıfır matrisi üretilir ve gerçekten output etiket matrisi üzerinde “max_label” değerini içeren piksellerin değeri 255 yapılır (Şekil 3.43). Buradan 255 değerini içeren piksellerin koordinatlarına ulaşılır ve bunlar bir matriste tutulur. ROI belirlemek için ise bu koordinat değerlerinin ortalamasına ihtiyaç duyulmaktadır, böylece görüntü üzerine bir merkez koordinat saptanmış olur.

```
max_edge = np.zeros(output.shape)
max_edge[output == max_label] = 255
#cx,cy=centroids[max_label,:].astype(int)
x_coords,y_coords=np.where(max_edge==255)
cx=round(np.mean(x_coords))
cy=round(np.mean(y_coords))
```

Şekil 3.43 ROI merkez koordinatların belirlenmesi



Şekil 3.44 Maksimum kenar görüntüsü

4. SONUÇ VE TARTIŞMA

Bu tez çalışmasında optik ve elektro-optik sistemlerin görüntüleme kapasitelerinin bir ölçütü olan ve görüntünün kalite fonksiyonu olarak da bilinen modülasyon aktarım fonksiyonu (MTF) üzerinde çalışılmıştır. MTF ölçümü yapabilmek için öncelikle yapay (sentetik) bir görüntü üzerinde uygulanmak üzere Python ortamında MTF analiz kodları yazılmıştır. Bu aşamada ayırık kenar yayılım fonksiyonu verisini işleyebilmek için iki farklı yöntem uygulanmış ve her iki yöntemden de gelen sonuçların birbiriyle uyumlu olduğu gösterilmiş olup, ayrıca IMATEST v24.3 ticari görüntü analiz yazılımı ile de sonuçlar karşılaştırılmıştır. Teorik açıdan kodların çalışmasının doğrulanmasından sonra deneysel aşamaya geçilmiştir. Deneysel düzenek kurulurken istenilen boyutlarda parabolik ayna endüstriyel olarak temin edilemediğinden ışık kolimatörü olarak bir Newtonian türünde teleskop kullanılmıştır. Elektromanyetik spektrumun görünür ve SWIR bölgesi olmak üzere iki farklı bölgede çalışan detektörlerle deneysel düzenekler kurulmuş, optik aydınlatma yapılmış ve aynı tek bir objektifle görüntüler alınmıştır. Elde edilen deneysel görüntüler yazılmış olan Python kodlarına verildiğinde kodların deneysel görüntüler üzerinde de başarılı bir şekilde çalışabildiği gözlemlenmiştir. Bu aşamadan sonra kurulan farklı optik düzeneklerin MTF'yi hangi ölçüde etkilediği analiz edilmek istenilmiştir. İdeal durumda MTF'nin değişmemesi beklenir ve birbiriyle ilinti olmayan (tamamen bağımsız) yöntemler ile ölçümler yapıp sonuçlar karşılaştırıldığı zaman bilimsel bir metrikle MTF kodları test edilmiş olunur. Bu durumda eğik kenar görüntüsünün SWIR ışıkla aydınlatıldığı bir sistemde teleskoptan alınan görüntünün MTF grafiği, sadece duvar aydınlatmasının yapıldığı durumda kameradan alınan görüntünün MTF grafiği ve tüm sistemin Zemax ortamında çizilip görüntü simülasyonundan alınan MTF grafiği üst üste çizdirildiğinde uyumlu sonuçlar gözlenmiştir ve bu sonuçlar kodların doğruluğunu önemli ölçüde test etmiştir. Tezin devamındaki çalışmalar ise yine birbiriyle ilintili olmayan bir yöntem daha kullanıp MTF ölçümü alarak sonuçları kıyaslamaktır. Örneğin, bir dalga cephesi sensörü de sisteme eklenip görüntü alındığında elde edilen sonucun önceki MTF grafikleriyle ne ölçüde uyumlu olup olmadığı bir sonraki araştırma konusu olarak hedeflenmektedir.

KAYNAKLAR

- Boreman Glenn, D. 2001. Modulation Transfer Function in Optical and Electro-Optical Systems. SPIE, Washington.
- Fiete, R. 2010. Modeling the Imaging Chain of Digital Cameras. SPIE Press, Bellingham, Washington.
- Li, H., Yan, C. ve Shao, J. 2016. Measurement of the modulation transfer function of infrared imaging system by modified slant edge method. Journal of Optical Society of Korea, 20, 381-388
- Masaoka, K., Yamashita, T., Nishida, Y., Sugawara, M. 2014. Modified slanted-edge method and multidirectional modulation transfer function estimation. Optics Express, 22, 6040-6046.
- Robinet, F.V., Helder, D., Fraise, R., Newbury, A., Bergh, F., Lee, D., Saunier, S. 2018. Comparison of MTF Measurement using edge method: towards reference data set. Optics Express, 26, 33625-33648.
- Saleh, B.E.A, Teich, M.C. 1991. Fundamentals of Photonics. John Willey and Sons, New York.
- Smith, W. 1990. Modern Optical Engineering: The Design of Optical Systems. McGraw-Hill, New York.
- Tzannes, Alexis P., Mooney Jonathan M. 1995. Measurement of the modulation transfer function of infrared cameras. Optical Engineering, 34(06), 1808-1817.
- Zhang, H., Li, C., Duan, Y. 2018. Modified slanted-edge method to measure the modulation transfer function of camera. Optik, 1157, 635-643.

EKLER

EK1 Veri Yumuşatması ile MTF Kodları

```
img_1 = cv.imread("Slanted_Edge.png",0)

print("first_image:",img_1.shape, img_1.dtype) # writing out image properties

cv.imshow('original_image_0', img_1) # Showing the first image.

cv.waitKey(50000);

cv.destroyAllWindows();

image_trial = img_1 # Kodun devamında "ROI_Image" olarak işlenecektir.

#####

# Part 2:

# Determining the ROI from the entire image:

from copy import copy, deepcopy

ROI_Image = deepcopy(image_trial[150:550, 350:700])

cv.imshow("ROI_OS", ROI_Image);

cv.waitKey(80000);

cv.destroyAllWindows()

ROI_Image = ROI_Image.astype(np.float64)

#####

###

# Part 3:

### Taking Derivative in x Direction:

# Kernel Definition of Derivating Matrix in x - direction:

Kx = np.array([[-1,1],[-1,1]])
```

```
Kx = (1/2)*Kx
```

```
# Convolution Operation:
```

```
Fx = cv.filter2D(ROI_Image, -1, Kx)
```

```
Fx = abs(Fx)
```

```
### Taking Derivative in y Direction:
```

```
# Kernel Definition of Derivating Matrix in y - direction:
```

```
Ky = np.array([[1,1],[-1,-1]])
```

```
Ky = (1/2)*Ky
```

```
# Convolution Operation:
```

```
Fy = cv.filter2D(ROI_Image, -1, Ky)
```

```
Fy = abs(Fy)
```

```
theta = math.atan(np.sum(Fy)/np.sum(Fx)) # in radians
```

```
theta_in_degrees = 180*(theta/np.pi); # in degrees
```

```
print(theta_in_degrees)
```

```
# Part 4:
```

```
rows, cols = (ROI_Image.shape[0], ROI_Image.shape[1]);
```

```
Value_Matrix = np.zeros((rows*cols,2))
```

```
# ESF Hesaplama Algoritması:
```

```
i = 0;
```

```
for k in range(0,rows):
```

```

for m in range(0, cols):

    Value_Matrix[m + i,0] = (m*1.0 + math.tan(theta)*k)*math.cos(theta)

    Value_Matrix[m + i,1] = ROI_Image[k,m] # Intensity Alignment

    i = i + cols

Value_Matrix_Trial_2 = np.copy(Value_Matrix)

ind = np.lexsort((Value_Matrix_Trial_2[:,1], Value_Matrix_Trial_2[:,0]))

ESF_Matrix = Value_Matrix_Trial_2[ind] # ESF'nin son hali

plt.scatter(ESF_Matrix[:,0], ESF_Matrix[:,1], c="blue")

plt.title('ESF_1')

plt.grid()

plt.xlabel('Pixel')

plt.ylabel('Intensity')

#plt.xlim([193,202])

plt.show()

y_values = ESF_Matrix[:,1] # Intensity Values of ESF Data

x_values = ESF_Matrix[:,0]

#####

# Part 5:

# DISCRETE ESF Data'sının (ESF_TRIAL -- mavi grafik) çizdirilmesinden sonra
izlenilecek 2 yol vardır:

# 1. 1/4xpixel sampling yapılarak ESF datası "smooth" hale getirilir.

# 2. Data'ya hiç dokunmayarak uygun bir fonksiyonla (Örn: Fermi Function) bir
curve fitting yapılır ve öyle devam edilir.

#####

```

1. Yol'dan devam edilirse: --> Smoothing the ESF

```
range_size = 0.10;
```

```
def frange(start, stop, step=1.0):
```

```
    i = start
```

```
    while i < stop:
```

```
        yield i
```

```
        i += step
```

```
x_max = x_values.max();
```

```
x_min = x_values.min()
```

```
print(x_max)
```

```
empty_array = np.array([[0, 0]]) # Array Initialization
```

```
for i in frange(x_min, x_max, range_size):
```

```
    solutions = np.argwhere(np.logical_and(x_values >= i, x_values < (i + range_size)))
```

```
# Index'lere ulaşıyoruz !!
```

```
    sum = 0.0; # Intensity'leri toplamadan önceki initialization
```

```
    for index in solutions:
```

```
        sum += y_values[index] # İlgili indeks'lerdeki Intensity değerlerini topluyoruz !!
```

```
    if solutions.size != 0:
```

```
        bin_middle_point = (2*i + range_size) / 2; # Bin'in ortalaması
```

```
        bin_averaged_intensity = sum[0] / solutions.size; # Intensity Değerlerinin
```

```
Ortalaması
```

```
        empty_array = np.append(empty_array, [[bin_middle_point,
```

```
bin_averaged_intensity]], axis=0) # Her ikisini birlikte tutan array
```

```
#####
```

```

plt.figure()

plt.plot(x2,y2)

plt.xlim([193, 202])

plt.title('Smoothed_ESF_1')

plt.grid()

plt.xlabel('Pixel')

plt.ylabel('Intensity')

plt.show()

#####

# LSF Derivation:

LSF = abs(np.gradient(y2,x2))

plt.figure()

plt.plot(x2,LSF,linewidth = 3.2, color ='purple' )

plt.xlim([193, 202])

plt.title('Line Spread Function (LSF)')

plt.grid()

plt.xlabel('Pixel')

plt.ylabel('Intensity Change')

plt.show()

#####

#

hamming = np.hamming(len(LSF))

plt.figure()

plt.plot(x2,hamming,linewidth = 3.2, color ='purple' )

```

```

plt.title('Hamming window Utilized')

plt.grid()

plt.show()

#####
#

after_hamming = np.multiply(hamming,LSF)

plt.figure()

plt.plot(x2, after_hamming, linewidth = 3.2, color ='purple' )

plt.title('LSF_after hamming window ')

plt.xlabel('Pixel')

plt.ylabel('Intensity Change')

plt.xlim([193,202])

plt.grid()

plt.show()

#####
###

# Part 6:

delta_x = np.mean(np.diff(x2))

sampling = delta_x

def my_fourier(signal, target_frequency, sampling_rate):

    N = len(signal)

    F_target = np.sqrt(2.8)*np.sum(signal * np.exp(-2j * np.pi * target_frequency *
np.arange(N) * sampling_rate))/N

    return F_target

```

```

target_frequency = 0.15 # in cycles per pixel

deneme = np.abs(my_fourier(LSF, target_frequency, sampling))

print(f'Manuel olarak tek bir frekansta hesaplanan Fourier Dönüşümü at
f={target_frequency}: {deneme}')

# Bu aşamda ise LSF(Line Spread Function)'ın DFT(Discrete Fourier Transform)' u
hesaplanmaktadır.

# Fourier Transformu alınmış data normalize edilerek MTF fonksiyonu çizdirilmiştir.

from scipy.fft import fftfreq,rfft

N_1 = len(x2)

f3 = fftfreq(len(x2),np.mean(np.diff(x2))) ## -- >> Cycles per pixel

x2_FFT = abs(rfft(after_hamming)) ## Taking Fourier Transform !!

norm_x2_FFT = x2_FFT/x2_FFT[0] # By dividing zero frequency component,
normalizing is done.

plt.figure()

plt.plot(f3[:N_1//2], norm_x2_FFT[:N_1//2])

plt.xlabel('spatial freq, cyc/pxl')

plt.ylabel('MTF Values')

plt.title('MTF')

plt.grid()

plt.xlim([0,1])

plt.show()

a3 = norm_x2_FFT[:N_1//2] ## Normalize MTF değerleri

a4 = f3[:N_1//2] ## Frekans eksenini.

MTF_Matrix = np.zeros((a4.shape[0],2))

```

```

MTF_Matrix[:,0] = a3

MTF_Matrix[:,1] = a4

# MTF_Matrix'in ikinci sütununda k değerine en yakın olan frekansı bulma

#k = target_frequency # Örnek olarak belirli bir frekans değeri verilmiş

frekanslar = MTF_Matrix[:, 1] # MTF_Matrix'in ikinci sütunu (frekans eksen)

mtf_degerleri = MTF_Matrix[:, 0] # MTF_Matrix'in birinci sütunu (MTF değerleri)

# k'ye en yakın frekansın indeksini bulma

en_yakin_indeks = np.argmin(np.abs(frekanslar - target_frequency))

# Bu indekste yer alan MTF değerini alma

en_yakin_mtf_degeri = mtf_degerleri[en_yakin_indeks]

en_yakin_frekans = frekanslar[en_yakin_indeks]

print(f'hedef frekans değerine en yakın frekans: {en_yakin_frekans}')

print(f'En yakın MTF değeri: {en_yakin_mtf_degeri}')

```

EK2 Eğri Uydurma ile MTF Kodları

```

img_1 = cv.imread("ImageSimulation_1.jpg",0)

print("first_image:",img_1.shape, img_1.dtype) # writing out image properties

cv.imshow('original_image_0', img_1) # Showing the first image.

cv.waitKey(500000)

cv.destroyAllWindows()

image_trial = img_1 # Kodun devamında "ROI_Image" olarak işlenecektir.

#####

###

coordinate_1 = []

```

```

coordinate_2 = []

def click_event(event, c1, c2, flags, params):

    # checking for left mouse clicks

    if event == cv.EVENT_LBUTTONDOWN:

        # displaying the coordinates

        # on the Shell

        print(c1, ' ', c2)

        coordinate_1.append(c1)

        coordinate_2.append(c2)

        #return c1,c2

        # displaying the coordinates

        # on the image window

        font = cv.FONT_HERSHEY_SIMPLEX

        cv.putText(img, str(c1) + ' ' +

                    str(c2), (c1,c2), font,

                    1, (255, 0, 0), 2)

        cv.imshow('image', img)

# driver function

if __name__=="__main__":

    img = cv.imread("ImageSimulation_1.jpg",0)

    # displaying the image

    cv.imshow('image', img)

```

```

# setting mouse handler for the image

# and calling the click_event() function

cv.setMouseCallback('image', click_event)


# wait for a key to be pressed to exit

cv.waitKey(50000)

# close the window

cv.destroyAllWindows()

#####
###

x1 = int(np.array(coordinate_1))

y1 = int(np.array(coordinate_2))

ROI_Image = deepcopy(image_trial[y1-10:y1+10, x1-10:x1+10])

cv.imshow("ROI_OS", ROI_Image);

cv.waitKey(80000);

cv.destroyAllWindows();

ROI_Image = ROI_Image.astype(np.float64)

#####
###

### Taking Derivative in x Direction:

# Kernel Definition of Derivating Matrix in x - direction:

Kx = np.array([[ -1, 1], [ -1, 1]])

Kx = (1/2)*Kx

# Convolution Operation:

```

```

Fx = cv.filter2D(ROI_Image, -1, Kx)

Fx = abs(Fx)

#####

##

### Taking Derivative in y Direction:

# Kernel Definition of Derivating Matrix in y - direction:

Ky = np.array([[1,1],[-1,-1]])

Ky = (1/2)*Ky

# Convolution Operation:

Fy = cv.filter2D(ROI_Image, -1, Ky)

Fy = abs(Fy)

#####

###

theta = math.atan(np.sum(Fy)/np.sum(Fx)) # in radians

theta_in_degrees = 180*(theta/np.pi); # in degrees

print(theta_in_degrees)

#####

###

rows, cols = (ROI_Image.shape[0], ROI_Image.shape[1]);

Value_Matrix = np.zeros((rows*cols,2))

# ESF Calculation Algorithm:

i = 0;

for k in range(rows-1,-1,-1):

    for m in range(0, cols):

```

```

    Value_Matrix[m + i,0] = (m*1.0 + math.tan(theta)*(k-(rows-1)))*math.cos(theta)
# math.cos(theta) # Index Alignment

    Value_Matrix[m + i,1] = ROI_Image[k,m] # Intensity Alignment

    i = i + cols

Value_Matrix_Trial_2 = np.copy(Value_Matrix)

ind = np.lexsort((Value_Matrix_Trial_2[:,1], Value_Matrix_Trial_2[:,0]))

ESF_Matrix = Value_Matrix_Trial_2[ind] # Last Form of ESF is defined here.

plt.scatter(ESF_Matrix[:,0], ESF_Matrix[:,1], c="blue")

plt.title('ESF_TRIAL')

plt.grid()

#plt.xlim([100,120])

plt.show()

#####
#####

y = ESF_Matrix[:,1] # Intensity Values of ESF Data

x = ESF_Matrix[:,0] # New and oversampled indices in the unit of 'pixel' from the ESF
Data

#####
##

def fermi_f(x,a,b,c):

    result = a/(np.exp(-1*((x-b)/c)) + 1)

    return result

#####

from scipy.optimize import curve_fit

#popt, pcov = curve_fit(fermi_f, x, y, p0 = [255, 197.5, 0.2])

```

```

popt, pcov = curve_fit(fermi_f, x, y)

print(popt)

a_opt, b_opt, c_opt = popt # curve_fit fonksiyonunun döndüğü optimal parametreler

x_model = np.linspace(min(x), max(x), 100000)

#y_model = fermi_f(x_model, a_opt, b_opt, c_opt)

y_model = fermi_f(x_model, *popt)

plt.figure()

plt.scatter(x,y) # ESF DATA

plt.plot(x_model,y_model, linewidth = 3.6, color = 'r') # Fitted Function

#plt.xlim([100, 120])

plt.title('Curve_Fitted_ESF_1')

plt.xlabel('piksel(pxl)')

plt.ylabel('Intensity')

plt.grid()

plt.show()

#####
###

# Fitted ESF Function:

plt.figure()

plt.plot(x_model,y_model, linewidth = 3.6, color = 'r') # Fitted Function

#plt.xlim([100, 120])

plt.title('Curve_Fitted_ESF_2')

plt.xlabel('piksel(pxl)')

plt.ylabel('Intensity')

```

```

plt.grid()

plt.show()

#####

###

LSF = abs(np.gradient(y_model,x_model))

plt.figure()

plt.plot(x_model,LSF,linewidth = 3.2, color ='purple' )

#plt.xlim([100, 120])

plt.title('Line Spread Function')

plt.xlabel('piksel(pxl)')

plt.ylabel('Intensity Change')

plt.grid()

plt.show()

# Part 6:

from scipy.fft import fftfreq,fft

x = x_model

N = len(x)

f2 = fftfreq(len(x),np.mean(np.diff(x))) ## -- >> Cycles per pixel

x1_FFT = abs(fft(LSF)) ## Taking Fourier Transform !!

norm_x1_FFT = x1_FFT/x1_FFT[0] # By dividing zero frequency component,
normalizing is done.

plt.figure()

plt.plot(f2[:N//2], norm_x1_FFT[:N//2])

plt.xlabel('spatial freq, cyc/pxl')

```

```

plt.ylabel('MTF Values')

plt.title('Modulation Transfer Function')

plt.grid()

plt.xlim([0,1])

plt.show()

a3 = norm_x1_FFT[:N//2] ## Normalize MTF değerleri

a4 = f2[:N//2] ## Frekans ekseni.

MTF_Matrix = np.zeros((a4.shape[0],2))

MTF_Matrix[:,0] = a3

MTF_Matrix[:,1] = a4

# MTF_Matrix'in ikinci sütununda k değerine en yakın olan frekansı bulma

k = target_frequency # Örnek olarak belirli bir frekans değeri verilmiş

frekanslar = MTF_Matrix[:, 1] # MTF_Matrix'in ikinci sütunu (frekans ekseni)

mtf_degerleri = MTF_Matrix[:, 0] # MTF_Matrix'in birinci sütunu (MTF değerleri)

# k'ye en yakın frekansın indeksini bulma

en_yakin_indeks = np.argmin(np.abs(frekanslar - k))

# Bu indekste yer alan MTF değerini alma

en_yakin_mtf_degeri = mtf_degerleri[en_yakin_indeks]

en_yakin_frekans = frekanslar[en_yakin_indeks]

print(f'k değerine en yakın frekans: {en_yakin_frekans}')

print(f'k değerine en yakın MTF değeri: {en_yakin_mtf_degeri}')

```

EK3 Otomatik Kenar Tespit Algoritması

```
img_1 = cv.imread("Duvar_SWIR_50mm.png",0)
print("first_image:",img_1.shape, img_1.dtype) # Görüntü özelliklerini yazdırıyor..
cv.imshow('original_image_0', img_1) # İlk görüntüyü gösteriyor.
print(np.max(img_1), np.min(img_1))
cv.waitKey(50000)
image_trial = img_1 # Kodun devamında "ROI_Image" olarak işlenecektir.

sobel_x = cv.Sobel(img_1, cv.CV_64F, 1, 0, ksize=3)
cv.imshow("sobel x", sobel_x)
print(np.max(sobel_x), np.min(sobel_x))
cv.waitKey(50000)
sobel_x[sobel_x<0]=0 # eksi değerler sıfır olmalı, normalization ona göre tasarlandı,
minimum 0 olmalı

sobel_x = np.uint8(255*sobel_x/np.max(sobel_x))
print(np.max(sobel_x), np.min(sobel_x))
cv.imshow("normalized x", sobel_x)
cv.waitKey(50000)

_, sobel_thresh = cv.threshold(sobel_x, 100,255, cv.THRESH_BINARY)
print(np.max(sobel_x), np.min(sobel_x))
cv.imshow("threshold x", sobel_thresh)
cv.waitKey(50000)

nb_components, output, statss, centroids =
cv.connectedComponentsWithStats(sobel_thresh, connectivity=8)

sizes = statss[:, -1]
```

```

max_label = 1
max_size = sizes[1]
for i in range(2, nb_components):
    if sizes[i] > max_size:
        max_label = i
        max_size = sizes[i]
max_edge = np.zeros(output.shape)
max_edge[output == max_label] = 255
cx,cy=centroids[max_label,:].astype(int)
x_coords,y_coords=np.where(max_edge==255)
cx=round(np.mean(x_coords))
cy=round(np.mean(y_coords))
print(cx,cy)

max_edge[cx,cy]=128
cv.imshow("max_edge", max_edge)

cv.imwrite("edge_center.png", max_edge)
cv.waitKey(20000)
cv.destroyAllWindows();
from copy import copy, deepcopy
x1 = cy
y1 = cx
ROI_Image = deepcopy(image_trial[y1-10:y1+10, x1-10:x1+10])
cv.imshow("ROI_OS", ROI_Image);
cv.waitKey(80000);
cv.destroyAllWindows();
ROI_Image = ROI_Image.astype(np.float64)
#####
###

```

```

# Part 3:

### Taking Derivative in x Direction:

# Kernel Definition of Derivating Matrix in x - direction:

Kx = np.array([[ -1,1],[ -1,1]])

Kx = (1/2)*Kx

# Convolution Operation:

Fx = cv.filter2D(ROI_Image, -1, Kx)

Fx = abs(Fx)

#####

### Taking Derivative in y Direction:

# Kernel Definition of Derivating Matrix in y - direction:

Ky = np.array([[1,1],[ -1,-1]])

Ky = (1/2)*Ky

# Convolution Operation:

Fy = cv.filter2D(ROI_Image, -1, Ky)

Fy = abs(Fy)

#####

# Slanted Edge Angle (theta) Calculation:

theta = math.atan(np.sum(Fy)/np.sum(Fx)) # in radians

#theta = 0.218166156499292

#theta = math.atan(abs(np.sum(Fy)/np.sum(Fx))) # in radians

theta_in_degrees = 180*(theta/np.pi); # in degrees

print(theta_in_degrees)

#####
#####

rows, cols = (ROI_Image.shape[0], ROI_Image.shape[1]);

Value_Matrix = np.zeros((rows*cols,2))

# ESF Calculation Algorithm:

i = 0;

for k in range(rows-1,-1,-1):

```

```

for m in range(0, cols):

    Value_Matrix[m + i,0] = (m*1.0 + math.tan(theta)*(k-(rows-1)))*math.cos(theta)
# math.cos(theta) # Index Alignment

    #print(m*1.0 + math.tan(theta)*k)

    Value_Matrix[m + i,1] = ROI_Image[k,m] # Intensity Alignment

    i = i + cols

Value_Matrix_Trial_2 = np.copy(Value_Matrix)

ind = np.lexsort((Value_Matrix_Trial_2[:,1], Value_Matrix_Trial_2[:,0]))

ESF_Matrix = Value_Matrix_Trial_2[ind] # Last Form of ESF is defined here.

plt.scatter(ESF_Matrix[:,0], ESF_Matrix[:,1], c="blue")

plt.title('ESF_TRIAL')

plt.grid()

plt.show()

#####
###

y = ESF_Matrix[:,1] # Intensity Values of ESF Data

x = ESF_Matrix[:,0] # New and oversampled indices in the unit of 'pixel' from the ESF
Data

#####
#####

# Part 5:

# Fermi Fonksiyonun Tanımlanması:

def fermi_f(x,a,b,c):

    result = a/(np.exp(-1*((x-b)/c)) + 1)

    return result

#####

from scipy.optimize import curve_fit

#popt, pcov = curve_fit(fermi_f, x, y, p0 = [255, 197.5, 0.2])

popt, pcov = curve_fit(fermi_f, x, y)

print(popt)

a_opt, b_opt, c_opt = popt

```

```

x_model = np.linspace(min(x), max(x),100000)
y_model = fermi_f(x_model, *popt)

plt.figure()
plt.scatter(x,y) # ESF DATA
plt.plot(x_model,y_model, linewidth = 3.6, color = 'r') # Fitted Function
plt.title('Curve_Fitted_ESF_1')
plt.xlabel('piksel(px1)')
plt.ylabel('Intensity')
plt.grid()
plt.show()

#####
#####

# Fitted ESF Function:
plt.figure()
plt.plot(x_model,y_model, linewidth = 3.6, color = 'r') # Fitted Function
#plt.xlim([100, 120])
plt.title('Curve_Fitted_ESF_2')
plt.xlabel('piksel(px1)')
plt.ylabel('Intensity')
plt.grid()
plt.show()

#####
#####

# LSF Derivation:
LSF = abs(np.gradient(y_model,x_model))
plt.figure()
plt.plot(x_model,LSF,linewidth = 3.2, color ='purple' )
plt.title('LSF_after_Fitted_ESF_1')
plt.xlabel('piksel(px1)')
plt.ylabel('Response')

```

```

plt.grid()
plt.show()

#####
### from scipy.fft import fftfreq,fft

x = x_model

N = len(x)

f2 = fftfreq(len(x),np.mean(np.diff(x))) ## -- >> Cycles per pixel

x1_FFT = abs(fft(LSF)) ## Taking Fourier Transform !!

norm_x1_FFT = x1_FFT/x1_FFT[0] # By dividing zero frequency component,
normalizing is done.

plt.figure()
plt.plot(f2[:N//2], norm_x1_FFT[:N//2])
plt.xlabel('spatial freq, cyc/pxl')
plt.title('MTF_7.5')
plt.grid()
plt.xlim([0,1])
plt.show()

a3 = norm_x1_FFT[:N//2] ## Normalize MTF değerleri
a4 = f2[:N//2] ## Frekans ekseni.

MTF_Matrix = np.zeros((a4.shape[0],2))

MTF_Matrix[:,0] = a3
MTF_Matrix[:,1] = a4

```