ZONGULDAK BÜLENT ECEVİT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ENDÜSTRİYEL MAKİNALARDA YAPAY ZEKA GÖRÜNTÜ İŞLEME TABANLI İŞ KAZASI DAVRANIŞLARINI ÖNLEMEYE YÖNELİK SİSTEM TASARIMI

ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

YÜKSEK LİSANS TEZİ

SİNAN YÜKSEL

OCAK 2025

ZONGULDAK BÜLENT ECEVİT ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

ENDÜSTRİYEL MAKİNALARDA YAPAY ZEKA GÖRÜNTÜ İŞLEME TABANLI İŞ KAZASI DAVRANIŞLARINI ÖNLEMEYE YÖNELİK SİSTEM TASARIMI

ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

YÜKSEK LİSANS TEZİ

Sinan YÜKSEL

DANIŞMAN : Prof. Dr. Rıfat HACIOĞLU

ZONGULDAK

Ocak 2025

KABUL:

Sinan YÜKSEL tarafından hazırlanan "Endüstriyel Makinalarda Yapay Zeka Görüntü İşleme Tabanlı İş Kazası Davranışlarını Önlemeye Yönelik Sistem Tasarımı" başlıklı bu çalışma jürimiz tarafından değerlendirilerek Zonguldak Bülent Ecevit Üniversitesi, Fen Bilimleri Enstitüsü, Elektrik Elektronik Mühendisliği Anabilim Dalında Yüksek Lisans Tezi olarak oy birliğiyle kabul edilmiştir. 23/01/2025

Danışman: Prof. Dr. Rıfat HACIOĞLU .....
Zonguldak Bülent Ecevit Üniversitesi, Mühendislik Fakültesi, Elektrik-Elektronik Mühendisliği Bölümü

Üye: Doç. Dr. Rukiye UZUN ARSLAN .....
Zonguldak Bülent Ecevit Üniversitesi, Mühendislik Fakültesi, Elektrik-Elektronik Mühendisliği Bölümü

Üye: Dr. Öğr. Üyesi Muhsin Uğur DOĞAN .....
Abant İzzet Baysal Üniversitesi, Bolu Teknik Bilimler MYO, Elektronik ve Otomasyon Bölümü

ONAY:

Yukarıdaki imzaların, adı geçen öğretim üyelerine ait olduğunu onaylarım. ..../..../20....

Prof. Dr. Fikret GÖLGELEYEN
Fen Bilimleri Enstitüsü Müdürü

"Bu tezdeki tüm bilgilerin akademik kurallara ve etik ilkelere uygun olarak elde edildiğini ve sunulduğunu; ayrıca bu kuralların ve ilkelerin gerektirdiği şekilde, bu çalışmadan kaynaklanmayan bütün atıfları yaptığımı beyan ederim."

Sinan YÜKSEL

ÖZET

Yüksek Lisans Tezi

ENDÜSTRİYEL MAKİNALARDA YAPAY ZEKA GÖRÜNTÜ İŞLEME TABANLI İŞ KAZASI DAVRANIŞLARINI ÖNLEMEYE YÖNELİK SİSTEM TASARIMI

Sinan YÜKSEL

Zonguldak Bülent Ecevit Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik Elektronik Mühendisliği Anabilim Dalı

Tez Danışmanı: Prof. Dr. Rıfat HACIOĞLU
Ocak 2025, 73 sayfa

Bu çalışmada yapay zeka ve görüntü işleme yaklaşımı kullanılarak, daire testere ve benzeri tehlikeli makinalarda çalışanların iş kazası sonucu el yaralanmalarını, uzuv kayıplarını önlemek ve doğru çalışma davranışlarını öğretmeyi amaçlayan bir tasarım geliştirilmektedir. Tasarımda eğitilmiş CNN (evrişimsel sinir ağı) vb. derin öğrenme teknikleri ile geliştirilen MediaPipe Hands makina öğrenimi modeli kullanılarak bilgisayarlı görme (Computervision) işlemi yapılmaktadır. Yapay zeka tasarımı Python yazılım ortamında makine öğrenimi kütüphanesi TensorFlow ve görüntü işleme kütüphanesi OpenCV kullanılarak yapılmaktadır. Kameradan alınan canlı görüntü ile insan elinin, tehlikeli bölgede kesici ile arasındaki yaklaşma mesafesi ve hızlı hareketleri takip edilip, ölçümlenir. Buna göre tehlikeli iş makinalarında tehlike ve uyarı bölgeleri belirlenmektedir. Eğer el tehlikeli bölgede kesiciye aşırı yakınsa veya hızlı hareketler sergilerse iş makinası durdurulur. Uyarı bölgesinde ise kullanıcıya gerekli uyarılar yapılarak el kazalarının önüne geçilmesi hedeflenmektedir. MediaPipe Hands modelinin farklı parlaklık ve gürültü ortam durumları incelenerek tespit performansında kıyaslamalar yapılmaktadır. Burada önerilen parlaklık dengeleme ve gürültü önleyici medyan filtre etkileri incelenmektedir.

ÖZET (devam ediyor)

Olumsuz ortam koşullarından kaynaklanan bu performans kaybı filtre kullanımıyla iyileştirme yapılmaktadır. Parlaklık seviyelerindeki değişim ve gürültü etkisinde önerilen filtreler önemli oranda iyileştirme sağlamaktadır. Ek olarak kullanılan model karmaşıklığı değerlendirilerek önerilen sistemin gerçek zaman çalışabilme kabiliyeti incelenmektedir. Yapay zeka tabanlı bu güvenlik sistemi kamera, bilgisayar ve yazılım alanındaki gelişmelere bağlı olarak iş kazalarını en alt seviyeye indireceği değerlendirilmektedir.

Anahtar Kelimeler: El Kesmeyen Testere, El ve Parmak İş Kazalarını Önlemek, El Koruma Sistemi, Testere Kazaları, İSG.

ABSTRACT

M. Sc. Thesis

SYSTEM DESIGN FOR PREVENTING WORK ACCIDENT BEHAVIORS BASED ON ARTIFICIAL INTELLIGENCE IMAGE PROCESSING IN INDUSTRIAL MACHINES

Sinan YÜKSEL

ZonguldakBülent Ecevit University
Graduate School of Natural and Applied Sciences
Department of Electrical Electronics Engineering

Thesis Advisor: Prof. Dr. Rıfat HACIOĞLU
January 2025, 73 pages

In this study, a design is being developed using image processing techniques with artificial intelligence to prevent hand injuries and limb losses due to work accidents in workers working on circular saws and similar dangerous machines, and to teach them correct working behaviors. Computer vision is performed using the MediaPipe Hands machine learning model developed with CNN (convolutional neural network) and similar deep learning techniques. The design is implemented in Python using the TensorFlow machine learning library and the OpenCV image processing library. The system uses live camera images to monitor the human hand's proximity and fast movements in dangerous areas. Based on these measurements, danger and warning zones are identified. If the hand approaches too closely to the cutter or shows rapid movements, the machine is stopped. In the warning zone, necessary alerts are provided to the user to avoid accidents.

ABSTRACT (continued)

The performance of the MediaPipe Hands model is evaluated under different brightness and noise conditions. To reduce performance loss caused by environmental factors, filters are applied. The brightness compensation and noise-cancelling median filter provide significant improvements in varying lighting and noise levels. Additionally, the model's complexity and its real-time operation capabilities are assessed. The system is expected to reduce work accidents significantly, depending on advancements in cameras, computers, and software.

Keywords: Hand Guard Saw, Preventing Hand and Finger Work Accidents, Hand Protection System, Saw Accidents, OHS.

TEŞEKKÜR

Yapay zeka ve görüntü işleme yaklaşımı ile iş güvenliği alanında yenilikçi çözümler üretmenin zorukları oldukça fazlaydı, danışman hocam beni doğru hedeflere yönlendirdi. Bu özgün tez çalışmasında sürecin başından itibaren bana bilgi ve tecrübesiyle yol gösteren, danışmanım sayın Prof. Dr. Rıfat HACIOĞLU hocama en derin saygılarımla teşekkür ederim.

Tez çalışmamın değerlendirme ve denetleme aşamasında katkıda bulunan Doç. Dr. Ali NARİN, Doç. Dr. Rukiye UZUN ARSLAN ve Dr. Öğr. Üyesi Muhsin Uğur DOĞAN hocalarımıza teşekkür ederim.

Yıllardır her sıkıntımda uzakta bile olsa bana destek veren, gönülden yakın bir dost olan değerli arkadaşım Harun ASLANER'e, çalışmalarımda bana ekipman desteği sağlayan iş arkadaşım Mete ÇIVAK'a, çeşitli işlerde ve tez çalışmamda bana atölyesinde uygulama yapma imkanı veren mobilya ustası Erdal ÇAKIROĞLU ve çalışanlarına teşekkür ederim.

Bundan 20 yıl önce mobilya atölyesinde birlikte çalıştığım İsa usta elini kesmeseydi, belki yıllar sonra bu tezin yazılma fikri ortaya çıkmazdı. Bu çalışma, endüstrideki tehlikeli iş makinalarını kullanan insanların güvenliği için yapılmış bilimsel bir çalışmadır.

Hayatım boyunca yanımda olan tüm sevdiklerime ve sevenlerime teşekkür ederim.

Sinan YÜKSEL

Ocak 2025

İÇİNDEKİLER

İÇİNDEKİLER (Devam ediyor)

Sayfa

ŞEKİLLER DİZİNİ

No Sayfa

ŞEKİLLER DİZİNİ (Devam ediyor)

No Sayfa

ÇİZELGELER DİZİNİ

No Sayfa

EK AÇIKLAMALAR DİZİNİ

Sayfa

SİMGELER VE KISALTMALAR DİZİNİ

SİMGELER

°	: Açı Derece
Δ	: Üçgen Bağlantı, Delta işareti
Y	: Yıldız Bağlantı

KISALTMALAR

ABD	**:** Amerika Birleşik Devletleri
AI	**:** Artificial intelligence (Yapay Zeka)
BEUN	**:** Bülent Ecevit Üniversitesi
CNN	**:** Convolutional Neural Network
ÇSGB	**:** Çalışma ve Sosyal Güvenlik Bakanlığı
DC	**:** Direct Current (Doğru Akım)
DSP	**:** Digital Signal Processing(Dijital Sinyal İşleyici, Sayısal İşaret İşleyici)
f	**:** Frekans
F	**:** Farad
FBE	**:** Fen Bilimleri Enstitüsü
FPS	**:** Frame Per Second (Saniye başına kare sayısı)
GPU	**:** Graphics Processing Unit
İSG	**:** İş Sağlığı ve Güvenliği
Hz	**:** Hertz
kHz	**:** kiloHertz
m	**:** metre
OpenCV	**:** Açık kaynak bilgisayarlı görü (Computer Vision)
cm	**:** santimetre
mm	**:** milimetre
m/sn	**:** metre/saniye
ms	**:** milisaniye
RGB	**:** Red Green Blue (Kırmızı Yeşil Mavi)
RPM	**:** Revolutions Per Minute (Dakikadaki Devir Sayısı)
sn	**:** saniye

BÖLÜM 1

GİRİŞ

Bu çalışmada görüntü işleme ve yapay zeka yaklaşımı ile gerçek zamanlı olarak, daire testere vb. tehlikeli makinalarda iş kazası sonucu çalışanların ellerinde meydana gelen yaralanmaları, uzuv kayıplarını önlemek ve hatalı davranışlar yerine doğru çalışma davranışını öğretmeyi amaçlayan bir güvenlik tasarımı geliştirilmiştir. Akademik alanda bir ilk olan bu çalışma, İSG, orman, endüstri, makina, bilgisayar ve birçok mühendislik alanının yıllardır çözmeye çalıştığı iş güvenliği problemlerinin çözümünde yol gösterici olduğundan dolayı önemlidir.

Endüstride işçi sağlığı ve iş güvenliği konularında kanun ve çalıştaylar yapılmaktadır. 20/06/2012 tarihli ve 6331 sayılı İş Sağlığı ve Güvenliği Kanunu gereğince, işverenler ve çalışanlar güvenlik kurallarından sorumludur [1]. Bunun için İSG alanında yetkili sorumlu kişiler ile birlikte hareket etmelidir [2]. Güvenliği en üst düzeye çıkarmak, günümüzde yenilikçi yaklaşımlardan yapay zeka ve görüntü işleme yaklaşımıyla mümkün olacaktır [3, 4]. Özellikle ABD ve Avrupa ülkeleri iş güvenliği ve işçi sağlığı konusunda ileri teknoloji çözümleri yasalaştırmaya ve zorunlu hale getirmeye 1970'li yıllarda başlamıştır [5]. Bu süreçlerle birlikte özellikle mobilya sektöründe fazlaca olan kazaları azalmak için araştırmalar, değerlendirmeler yapılmıştır [6-9]. Benzer şekilde ÇSGB tarafından da mobilya üretiminde tehlikeler ve alınacak önlemler üzerine bir rehber yapılmıştır [10]. Tüm bu çalışmaların asıl amacı, insan ve makina kaynaklı kazaların azaltılması, üretimin kalitesinin ve sürekliliğinin artmasını sağlamaktır. Üretimde artış sağlanırken, insanların uzuv kayıpları ve travmatolojik durumlar yaşamasının da önüne geçilmesi İSG açısından gereklidir.

Endüstriyel ve bireysel atölyelerde makina ile çalışmalarda özellikle ahşap, mobilya ürünleri ve metal malzemelerin işlenmesinde çok fazla iş kazası yaşanması ve işlerin insan eli ile yapılması sebebiyle iş kazalarında çoğunlukla eller yaralanmaktadır. Bu kazalar geri dönüşü olmayan uzuv kayıplarıyla sonuçlanabilmektedir, bu sebeple çalışılan makinaların ve insanların güvenliğinin sağlanması, kullanıcıların bilinçlendirilmesi önemlidir [11-13].

Eller iyileşmesi en zor olan organlardan biridir, el ve parmaklar iş makinası tarafından kesilmesi durumunda, sinir ve tendon hasarlarına uğradığından iyileşmesi zordur, iyileşme süreci uzundur ve eskisi kadar sağlıklı hale dönmesi mümkün değildir, eğer parçalanma yani kopma söz konusu ise iyileşmesi çok daha zordur, Şekil 1.1'deki gibi uzuv kaybı olabilir[14].

Şekil 1.1 Daire testere iş kazası

Ahşap ve mobilya işlerinde SawStop el kesmeyen testere, insan bedeninin iletkenlik ve kapasitans özelliğinden faydalanarak çalışmaktadır [15]. Tasarım gereği elektriksel bir ölçüm yapıldığından kesilen malzemenin yaş, nemli ve iletken olmaması gereklidir. Üretici nemli ahşap ve iletken alüminyum tarzı metal malzemeleri kesmek isteyen kullanıcılar için bypass seçeneğide sunmaktadır, bu durumda güvenlik devreden çıkar. Testere sisteminde; el, kesici testere ile temas ederse koruma fonksiyonu çalıştığından, kaza oluştuğu anda devreye girmektedir, bunun sonucunda hafif ve belirli dereceli yaralanmalar olabilmektedir.

Sawstop endüstriyel çözümünden tamamen farklı bir teknikle, görüntü işleme ve yapay zeka yaklaşımı kullanılarak el yaralanmalarını önlemeye yönelik bir tasarım 2021 yılında Altendof firması tarafından yapıldı [16]. Altendof firması bu tasarıma 'Hand Guard' adını verdi, tasarımın paylaşılan herhangi bir açık kaynak kodu yok ve makina datasheet verisini indirmek için üyelik istemektedir, sitesi dışında internet ortamında henüz herhangi bir bilgi bulunmayan, yeni bir sistemdir. Güvenlik alanında görüntü işleme ve yapay zeka kullanımı yenilikçi bir yaklaşımdır. Genel yaklaşım olarak nesne tespiti ve takibi ile birlikte eğitilmiş yapay zeka model kullanımıyla karar verme mekanizmalarının kullanıldığı değerlendirilmektedir [17-19].

Tez konusuda; ahşap, mobilya, metal vb. alanlarda, insan eli için tehlike arz eden makinalarda el güvenliğini sağlamak ve doğru çalışma davranışını makina kullanıcısına öğretmeyi amaçlamaktadır. Tasarımda görüntü işleme ve yapay zeka yaklaşımlarından faydalanılmıştır. Tasarım Python yazılım ortamında, TensorFlow makine öğrenimi kütüphanesi, OpenCV görüntü işleme kütüphanesi ve eğitilmiş CNN derin öğrenme medotlarıyla geliştirilen,

MediaPipe Hands modeli kullanılarak geliştirilmiştir. MediaPipe Hands, el tabanlı etkileşim ve hareket analizi uygulamalarında kullanılabilir, gerçek zamanlı olarak ellerin konumunu ve parmak hareketlerini tespit eder, ellerin 21 anahtar noktası kullanarak, ellerin izlenmesi ve parmak pozisyonlarının belirlenmesi işlemlerini hızlı ve verimli bir şekilde gerçekleştirilen makine öğrenimi kütüphanesidir [20, 21].

Eğitilmiş CNN (evirşimsel sinir ağı) öğrenme yaklaşımı ile geliştirilen MediaPipe makina öğrenimi modeli vücut şekli, yüz ve el tespiti ile bilgisayarlı görü (Computervision) işlemi yapılmaktadır [22-25]. El üzerindeki 21 anahtar noktayı belirlemeyi hedefleyen MediaPipe Hands modeli işaret dili uygulamalarında yaygın olarak kullanılırken, el hareketlerine göre karar verme mekanizmaları uygulamalarında da kullanılmaktadır [26-30].

Tasarım, yatar daire testereler başta olmak üzere, insan elinin tehlikende olduğu diğer iş makinalarına da benzer şekilde uyarlanabilir. Örneğin; planya, ahşap freze, metal freze, metal ve ahşap torna tezgahı, tehlikeli parçalayıcı, öğütücü makinalara uygulanabilir yapıdadır.

Python ortamında geliştirilen tasarım, kameradan aldığı canlı görüntüyü kullanarak, önceden tanımlı tehlikeli bölgeyi belirler, sonra bu tehlikeli bölgeye yaklaşan insan elini tespit ve takip eder. Birden fazla insan elinin takibi, ellerin tehlikeli bölgeye olan yakınlık mesafesi ve normalden hızlı hareketler yapması durumları anlık olarak işlenir, el tehlikeli alana girmiş veya aşırı hızlı ise sistem çıktı üretir, bu çıktı makinayı durdurur ve kullanıcıyı uyarır.

Kesicinin bulunduğu tehlikeli bölgeyi, yaklaşım mesafesini ve hız sınırını tasarımcı belirler. İzinler dahilinde makina operatörü, iş güvenliği uzmanı veya amir kişi tarafından da belirlenebilir. Bu sistemde tehlikeli bölgeden görüntü alan kamera, canlı güvenlik denetimi yapan bir insanın gözüne benzetilebilir. Kameradan alınan görüntü, insan eli ve makinanın tehlikeli kesici kısmını gerçek zamanlı olarak sürekli izler, insan eli ve tehlikeli bölge arasındaki mesafeyi anlık olarak hesaplarken, aynı zamanda aşırı hızlı hareketler olursa sistemi durdurur. Tüm bu görüntü işleme süreçleri geliştirilen yapay zeka sistemi yönetir.

Tasarlanan gerçek zamanlı koruma sistemi kullanıcının elini erken algılama ile korurken, aynı zamanda el hızı ve konumuna bağlı olarak, kullanıcıya doğru kabul edilen davranışları öğretmeyi, yani makinayı kullanan kişinin davranışsal hatalarını da azaltmayı hedefler. Mantıklı düşünüldüğünde, bir daire testereye olabildiğince uzak durmak gerekir. Örneğin; el kesiciye 3 cm den fazla yaklaşmışsa bu bir tehlikedir ve hatadır. Benzer şekilde makinayı kullanan kişi, kesicinin yakınlarında aşırı hızlı (agresif) hareket ediyorsa bu da tehlikeli ve

hatalı bir davranıştır, bunun için de 1 m/s gibi bir hız sınırı belirlenebilir. Bu sayede tehlike arz eden kesici ile el arasında mesafe ve hız kontrolü denetimi sağlanır. Böylece hem kaza önlenir, hem de kullanıcılara doğru davranış sınırları belirtilerek, güvenli çalışma sınırları alışkanlık haline getirilebilir.

Tasarlanan sistemin farklı ortam parlaklığı ve olumsuz ortam koşullarındaki el tespit değerleri incelendi. MediaPipe Hands modelinin olumsuz ortam koşullarını temsilen, yüksek oranda gürültüler eklenerek, medyan filtresi ile filtreleme yapıldı. Tuz ve biber gürültüleri ortamdaki talaş tozu, metal tozu, vb. temsil etmekte olup, görüntüdeki gürültüler ile modelin zorlu ortam koşullarındaki tespit ve çalışma performansına etkileri ölçüldü [31, 33].

Tezin katkıları incelendiğinde, materyal ve metod bölümünde genel özellikleri tanıtılan MediaPipe Hands modelinin, geliştirilen görüntü işleme ve yapay zeka yaklaşımı ile el koruma güvenlik sistemine entegrasyonu ve çalışma algoritması tanıtılmıştır. Bulgular kısmında, yapılan deneylerde ortam görüntüsünün iyileştirilmesi için geliştirilen medyan ve parlaklık filtreleri ile çalışmalar yapılmış, el görüntüsü yakalama tespit değerleri, tehlikeli konum ve ön durdurma hesapları, sistemin giriş ve çıkış cevap süresi incelenmiştir. Son olarak değerlendirmelere yer verilmiştir.

BÖLÜM 2

MATERYAL ve METOD

2.1 El Kesmeyen Testere Sistemi ve Daire Testerelerin Çalışması Hakkında

Endüstriyel anlamda dünya üzerinde "El kesmeyen testere" olarak bilinen ilk çözüm, SawStop firma adıyla bilinmektedir. Tasarımın patentli sahibi Dr. Steve Gass isimli fizikçidir, kendisi aynı zamanda mobilya işleri ile uğraşırken, insan bedeninin elektriği iletmesinden yola çıkarak 1999 yılında çalışmalarını başlattığı el kesmeyen testere sistemi, 2005 yılında seri üretime geçer, günümüzde aynı temel prensiple çalışan, farklı isimlerde çok sayıda patentli kopyası vardır ve patent tartışmaları halen devam etmektedir.

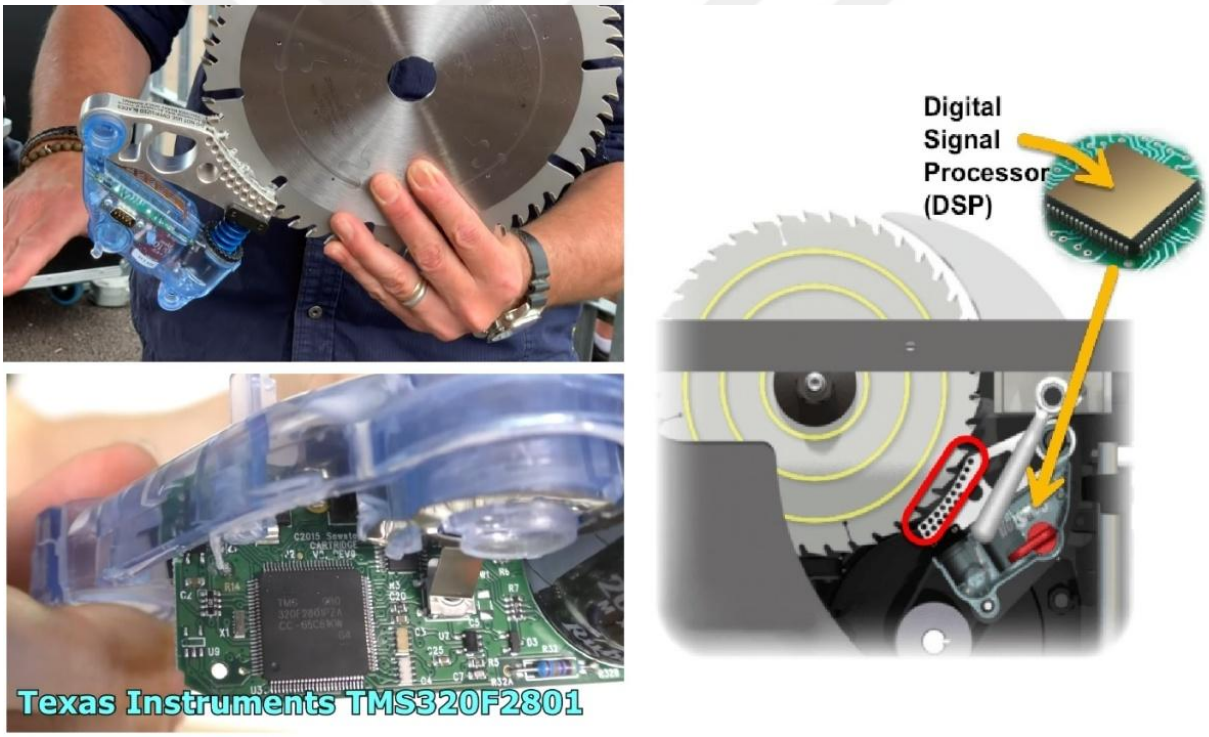
Sistemin en önemli birimi, Şekil 2.1'de gösterilen, DSP (Dijital Sinyal İşleyici) kısmıdır. Fren kartuşu DSP'lerin yüksek sayısal veri işleme hızından faydalanmak suretiyle geliştirilmiştir. Sawstop sisteminde bir osilatör devresinden daire testereye, 12 volt gibi düşük gerilim seviyesinde, saniyede 100-200 kHz sinyaller gönderilir, insan bedeninin elektriği iletme direnci kuru ahşap malzemelere göre daha düşüktür, testereye temas halinde sinyalde düşme olur, bu düşüş DSP ile anlık olarak takip edilir, orjinal ürünün temas halinde algılama ve durdurma süresi 0.005 sn kadar olduğu bilinmektedir.

4500 RPM ile dönen bir daire testere, saniyede 4500 devirden 75 tur atmaktadır. Bir daire testerenin üzerinde 50 diş bulunması durumunda, saniyede 3750 diş darbesi yapar, testere üzerinde 100 diş varsa iki katı kadar yani saniyede 7500 diş vurması anlamına gelir. Daire testerelerin dönme hareketi esnasında kesilen nesneyi çekme veya fırlatması gibi durumlarda olabilir, insan eli gibi yuvarlak hatlara sahip nesneleri genelde çekme eğilimindedir, yani kişi kaza anında elini kesiciden çekmek istesede, elini makinaya kaptırabilir.

Günümüzde çift devirli daire testereler halen çok fazla kullanılmaktadır, makina içinde çift devirli dahlender asenkron motoru kullanılmaktadır, bu motorların kutup sayısı Δ/Y olarak 4 kutup üçgen / 2 kutup yıldız olarak değiştirilerek teorikte 1500 / 3000 RPM olur, kayış ve makaradan daireye aktarılan dönme hızı ise iki katına çıkar, düşük devir 3000 RPM, yüksek devir 6000 RPM olur. Çizicili modeller ise normalde sabit 6000 RPM hızlarda çalışmaktadır.

Bu durumda hızlı devir ve 50 diş sayılı daire üzerinden hesap yapılacak olursa, 6000/60 yani saniyede 100 tur sayısı, 1 dairede 50 diş varsa; 1 saniyede 100*50 = 5000 diş darbesi yapar. Osilatörden 100 kHz frekans ile gönderilen sinyallerden bu durumda insan eline temas eden 1 diş üzerine 20 tane sinyal düşmektedir. Elektriksel olarak algılama 0.00001sn iken, mekaniksel olarak sistemin durması 0.005 sn sürdüğünden dolayı, elektriksel algılama ile mekaniksel durma süreleri arasında olarak 500 kat hız farkı vardır. DSP'ler ise 50-100 MHz gibi frekanslarda çalıştığından dolayı bu verileri çok rahatlıkla işleyebilmektedir.

1 saniyede 5000 diş vuran bir daire testere, 0.005 saniyede 25 diş vurmaktadır, bu süreyi kısaltmak için Şekil 2.1'de gösterilen koruma kartuşu üzerindeki alüminyum fren takozu kullanılır, burada durdurmayı hızlı hale getiren unsur önce alüminyum fren takozudur, sonra daire testerenin gizlenmesidir. Diğer önemli unsur ise kaza anında elin temas hızıdır, yani makinayı kullanan kişi ne kadar hızlı elini kaptırırsa, el o kadar fazla hasar alır.



Şekil 2.1 El kesmeyen testere sisteminin genel yapısı

Temas ile çalışan mecvut tasarımdaki gibi el 1 m/sn hızla hareket ederse, 0.005 sn mekanik durma süresi üzerinden hesaplanırsa, eldeki kesik 0.005 m (5mm) olmaktadır.

Testere sisteminin (SawStop) toplam durdurma süresi 0.005s olduğunu bilinmektedir, bu durdurma süresinde alüminyum fren takozunun etkisi büyüktür, çünkü 0.1 m yükseklikteki

bir daire testereyi gizlemek için geçen sürede, fren takozu sadece 0.01-0.02 m uzaklıkta olduğundan, mekaniksel olarak 3-5 kat daha kısa sürede fren devreye girmektedir ve sonra tezgahın altına gizleme olmaktadır.

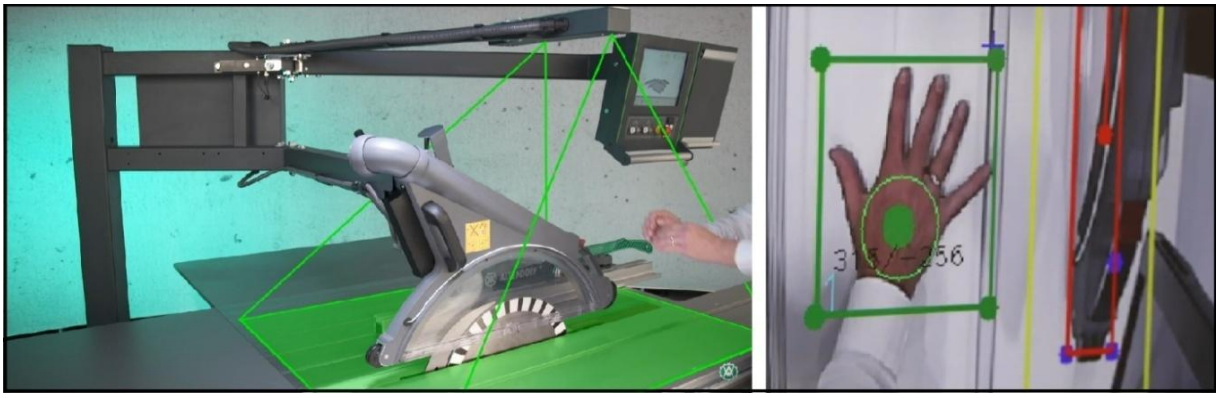
Alüminyum fren takozu ile durdurma yönteminde kaza sonrası, fren kartuşunun kendisi ve daire testere kullanılamaz hale gelir. Dolayısıyla yeni bir daire testere ve fren kartuşu ile değiştirilmesi gerekir. Fren kartuşu olmazsa durdurma süresi 0.015 sn kadar uzamaktadır.

Yapılan hesaplama ve kaza süreci incelendiğinde, kazanın kesiciye temas ile olduğu, temas hızının ve koruma sisteminin cevap hızının önemli olduğu net şekilde anlaşılmaktadır. Bu sistemde temas mesafesinin çok kısa olması, yani el ile kesici daire testerenin teması halinde eriyen bir telin serbest bıraktığı fren takozunun devreye girmesine bağlı olduğundan, durdurma süresi kısa gibi görünsede hafif yaralanmalar olmaktadır, daire testere ve fren kartuşu kullanılamaz hale geldiğinden dolayı her kazada daire testere ve fren kartuşu yenilenmelidir, bu da ek masrafların olması anlamına gelir.

2.2 Görüntü İşleme ve Yapay Zeka Yaklaşımları Kullanılarak Koruma Sistemi

Gerçek zamanlı görüntü işleme ve yapay zeka yaklaşımı ile insan elinin tanıyan, takip eden, kesici daire testerenin bulunduğu tehlikeli bölge ile arasındaki mesafe ve hız hesaplamalarını yaparak makinayı durdurdurup, kazaları önlemek için bir sistem tasarlanabilir.

Benzer teknikle **Altendorf** firması, 2021 yılı itibariyle **“Hand Guard”** ismiyle duyurduğu, Şekil 2.2’deki sistemi piyasaya sürdü. Altendof firmasının ürettiği Hand Guard sisteminin üretim tekniğiyle ilgili internet üzerinde herhangi bir teknik bilgi henüz bulunmamaktadır.



Şekil 2.2 Görüntü işleme ve yapay zeka yaklaşımıyla geliştirilen Altendof Hand Guard sistem

2.3 MediaPipe Hands Modeli

MediaPipe Hands Google tarafından geliştirilmektedir. Gerçek zamanlı el koruma tasarımında sistemde en zor kısım, insan elini görüntüsünden tanıyan bir model geliştirmektir, çünkü insan eli kişiden kişiye, zamana ve ortam koşullarına bağlı olarak, değişkenlik gösteren farklı renk ve anatomik yapılara sahiptir. İnsan elini tanımlamak, sıradan bir eşya, otomobil, uçak veya vücut bütünlüğü bulunan canlı nesneleri tanımaktan çok daha zordur, ayrıca el kolun devamı olan organ olduğundan sınırlarını belirlemek de zordur. İnsan eli, çok sayıda eklemden oluştuğu için duruşu sürekli hızlı değişkenlik gösteren, hareketli bir organdır. Ayrıca elin kişiden kişiye değişen doğal rengi, ortamın aydınlık seviyesine göre de değiştiğinden, geliştirilen model bu değişikliklere de cevap verebilmelidir. Tüm bu sorunları çözmek için öncelikle başarılı bir el tanıma ve takip modeline ihtiyaç vardır.

El tanıma amacıyla, önceden eğitilmiş CNN yapay sinir ağları ile geliştirilen MediaPipe Hands makine öğrenim modeli, Python yazılım ortamında kullanılarak, insan elini tanımlama yani bilgisayarlı görü (CV: Computervision) işlevi kazandırılır.

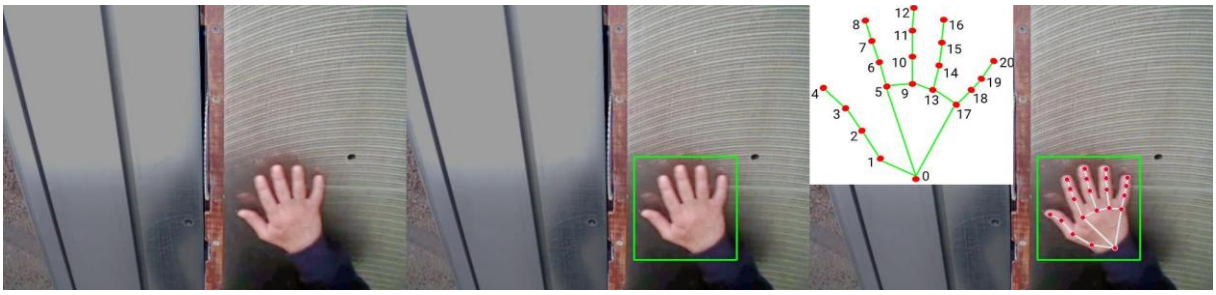
Tasarımda Python yazılım ortamında kütüphane olarak, makine öğrenmesi için TensorFlow ve bilgisayarlı görme için OpenCV kütüphanesinden faydalanılır.

2.3.1 MediaPipe Hands Modeli Nasıl Çalışır?

MediaPipe Hands görüntüdeki elin bulunduğu bölgeyi avuç iç ve dış yapısından palm detection özelliği ile tanır, sonra bir sınır kutusu (bounding box) çizerek elin tam alanını belirler, bu küçük alanda veri işlemesi yapar. Model geliştirilirken 30.000 üzerinde el verisinden faydalanılmıştır ve doğru tespit oranı % 96 civarıdır. Model eğitilirken kullanılan büyük veriler, optimizasyondan geçirilir ve küçük veriler ile düşük sistemlerde çalışabilecek hale getirilir. MediaPipe Hands, 21 anahtar noktadan (landmark) oluşan bir set kullanır. Bu anahtar noktalar, elin çeşitli parçalarının (parmaklar, avuç içi, vb.) konumlarını tanımlar. El dedektörü yakaladığı görüntüyü işlerken **192x192** boyutlarına indirger, el takip halinde ise **224x224** boyutlarına görüntü işleme yapar, böylece grafik yükü azalır ve tespit hızı artar.

El tespiti mimarisi ellerin nerede bulunduğunu kabaca tespit eden bir avuç içi tespiti ve ardından elin ve parmakların farklı kısımlarını daha kesin bir şekilde yerelleştiren bir el işareti tespiti olarak iki aşamadan oluşur. Tek atışlı bir dedektöre dayanan ilk aşama avuç içi tespiti modelini oluşturur. Bu model, tam görüntünün piksel değerlerini girdi olarak alır ve avuç içlerinin görüntüde nerede olabileceğini açıklayan sınırlayıcı kutuları ve bu kutuların her biri

için güven puanlarını çıkarır. Burada kullanılan, örneğin çapa kutularını (sınıflandırılmadan önce ağda oluşturulan öneriler) kutu görüntülerle sınırlamak gibi bazı hileler bulunmaktadır. İkinci kısım olan el işareti modeli, avuç içi tespitinin sınırlayıcı kutusundaki kırpılmış görüntüyü girdi olarak alan ve elin 21 anahtar noktasının 3B koordinatlarını döndüren bir regresyon modelidir [34, 35].



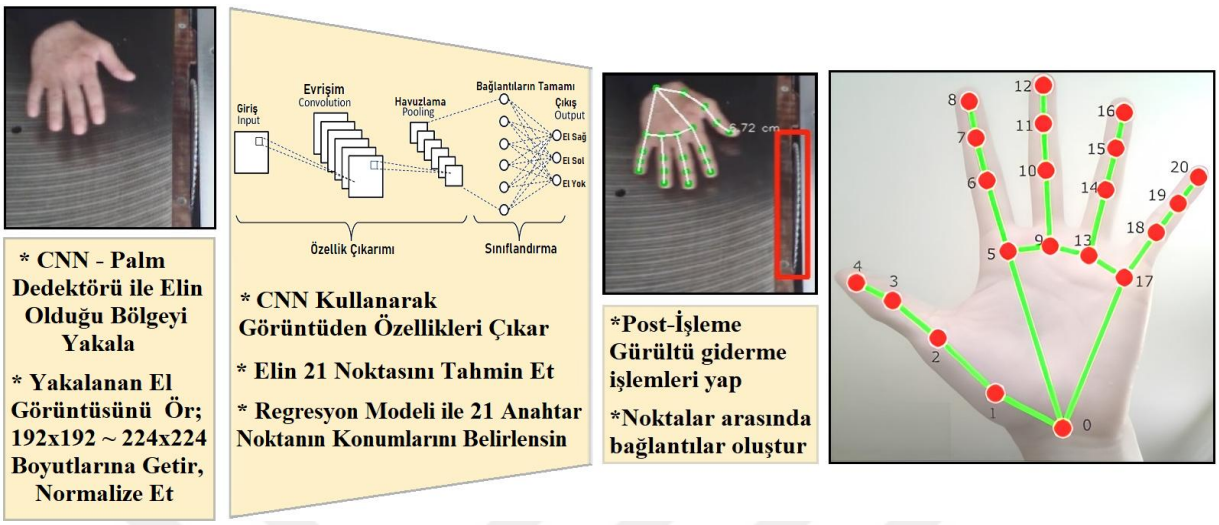
Şekil 2.3 MediaPipe Hands modeli avuç tespiti, çapa kutu oluşumu ile görüntü boyutuazaltılması ve 21 anahtar nokta tespiti gelişim aşamaları

Tasarımda kullandığımız model kameradan aldığı canlı görüntüyü yapay sinir ağları (CNN) metotlarıyla işler ve makine öğrenmesi yeteneği kazandırmak için ham anahtar noktaların özelliklerini çıkartır. Çıkartılan bu veriler, elin pozunu ve hareketini daha anlamlı hale getirmek için modelin tasarımına bağlı olarak, çeşitli görüntü işleme teknikleriyle işlenir. Örneğin, anahtar noktalar arasındaki ilişkiler kullanılarak elin 3D pozisyonu tahmin edilebilir. Bu durumda elin konumuna, hızına ve hareket yönüne göre karar verme mekanizmaları çalıştırılabilir. Mediapipe Hands modeli statik görüntüler üzerinde çalışabildiği gibi dinamik, gerçek zaman görüntüler üzerinde çalışabilmektedir. Her iki durum da modele girdi seçeneği olarak verilirken görüntüdeki el sayısı, el tespit/takip minimum güven katsayısı, model karmaşıklığı da girilebilmektedir. Ek olarak model 21 anahtar noktanın konum bilgisi ile birlikte el tespit sonucunun güven puanını da çıktı olarak üretmektedir.

MediaPipe Hands modelinin ilk tespit ve izleme sonuçlarının görsel optimizasyonu önemlidir. Bu süreç, ham verilerin düzeltilmesi ve anlamlandırılması gibi adımları içerir ve nihai tespit edilen elin görüntüsünden doğru ve kullanışlı veriler sağlanır.

Görüntü işleme ve yapay zeka yaklaşımı ile geliştirilen, gerçek zamanlı olarak çalışan el güvenlik sisteminde kullandığımız MediaPipe Hands modelin asıl kullanım alanı işaret dilidir. Eğitilmiş MediaPipe Hands modeline CNN (evrişimsel sinir ağı) derin öğrenme ve makine öğrenimi veri tabanı, modelin üretim aşamasında öğretilir. Bu modeller, geniş veri setleri üzerinde eğitilir ve belirli görevler için optimize edilir. MediaPipe'ın aynı zamanda el şekli

modelinden hariç, yüz ve vücut şekli tanıma yapabilen, bilgisayarlı görü modelleri de mevcuttur [36].



Şekil 2.4 MediaPipe Hands modelinin 21 anahtar noktasının işlenmesi

MediaPipe Hands, el izleme görevi için derin öğrenme ile bilgisayarlı görü (computervision) tekniklerini bir araya getirir. Bilgisayarlı görme, bilgisayar sistemlerinin dijital görüntüler üzerinden bilgi kazanma, analiz etme ve yorumlar geliştirmesi için kullanılan bir alandır. MediaPipe Hands bu özellikleri ile özel çalışmalara entegre edilerek, görüntü işleme, makine öğrenimi ve yapay zeka uygulamalarında kullanılabilir.

MediaPipe Hands modeli el tanıma işleminde, önceden eğitilmiş derin öğrenme yapılarından olan CNN (Convolutional Neural Network) konvolüsyonel sinir ağları yapılarını kullanarak, görüntüden el nesnesini tanıma ve görsel veri işleme yapar. CNN el görselindeki özellikleri yakalamak için "konvolüsyon" adı verilen işlemi kullanır, görüntünün küçük parçalarını analiz eder ve daha karmaşık özelliklerini çıkartır. CNN, özellikle görsel nesne tanıma görevlerinde kullanılan bir derin öğrenme sinir ağı modelidir. Bu modeller, geniş veri setleri üzerinde eğitilir ve belirli bir görevi gerçekleştirmek için optimize edilir. Eğitim aşamasında, model el tespiti için 30.000 üzerinde sentetik olmayan gerçek insan eli görseli ile eğitilmiştir ve bu görüntüler üzerindeki insan elinin tüm özellikleri ve desenlerinin haritaları öğretilmiştir. Bu süreç, genellikle büyük miktarda hesaplama gücü gerektirir, yüksek güçlü GPU'lar veya özel donanımlar kullanılarak gerçekleştirilir. Model, eğitim aşamasından sonra kendi kendini geliştiremez, ancak modelin derin öğrenme ve çalışma algoritmasının geliştirilmesine bağlı tespit performansı artabilir, model zamanla güncellenebilmektedir.

MediaPipe Hands makine öğrenimi (Machine Learning - ML) modeli, derin öğrenme (Deep learning - DL), yapay zeka (Artificial Intelligence - AI) teknolojileri ile birlikte kullanır. TensorFlow kütüphanesi ise Google tarafından geliştirilen bir açık kaynaklı makine öğrenimi kütüphanesidir, CNN tarzındaki derin öğrenme modellerini geliştirmek için kullanılır. Görüntü işleme uygulamalarında görüntü işleme kütüphanesi olarak OpenCV kullanılmıştır. OpenCV açık kaynak bilgisayarlı görü (CV: Computervision) kütüphanesidir. Derin öğrenme modelleri, büyük miktarda veriler üzerinde eğitilir ve veri setindeki örüntüleri belirli bir görevi yerine getirmek için öğrenir. Yapay zeka sistemi bir webcam üzerinden alınan canlı görüntü veya video içindeki eli tespit eder ve eli izleme işlemlerini gerçekleştirir. MediaPipe Hands, el tespiti ve izleme görevi için derin öğrenme ile bilgisayarlı görü (computervision) tekniklerini kullanır. Bilgisayarlı görme, bilgisayar sistemlerinin dijital görüntüler üzerinden bilgi kazanmaları, analiz etmeleri ve yorumlamaları için kullanılan bir alandır.

MediaPipe Hands modeli çeşitli platformlarda (Android, iOS, macOS, Linux, Windows) çalışabilen geniş bir altyapısı vardır, geliştiricilere el izleme özelliklerini uygulamalarına entegre etme imkanı vermektedir. Örneğin; MediaPipe Hands modelini kullanarak, el kazalarını önlemek için tasarladığımız güvenlik sistemi, Windows işletim sisteminde, Python yazılım ortamında, Pycharm arayüzü kullanılarak tasarlanmıştır.

2.3.2 Görüntü İşleme ve Yapay Zeka Yaklaşımı Kullanılarak Modelin Geliştirilmesi

Literatürde bu alanda daha önce yapılmış bilimsel makale veya tez çalışmasına rastlanmamıştır, yapılan tez ve makale çalışması bu sebeple alanında bir ilktir [37-38]. Tezde yapay zeka ile görüntü işleme tekniklerini kullanarak Python yazılım ortamında, MediaPipe Hands makine öğrenimi modeli üzerinde bazı geliştirmeler yaparak model daha işlevsel hale getirildi. Daha sonra kesicinin türüne bağlı olarak dikdörtgen veya dairesel tehlikeli alanlar belirlendi. Bu tehlikeli alanların matematiksel modellemesi yapıldı. Örneğin; ileri düzeyde testere yüksekliği ve açısına bağlı tehlikeli alanın otomatik değişimi gösterildi, tasarlanan kodlamalar da tezin ek açıklamalar kısmında yer verilmiştir.

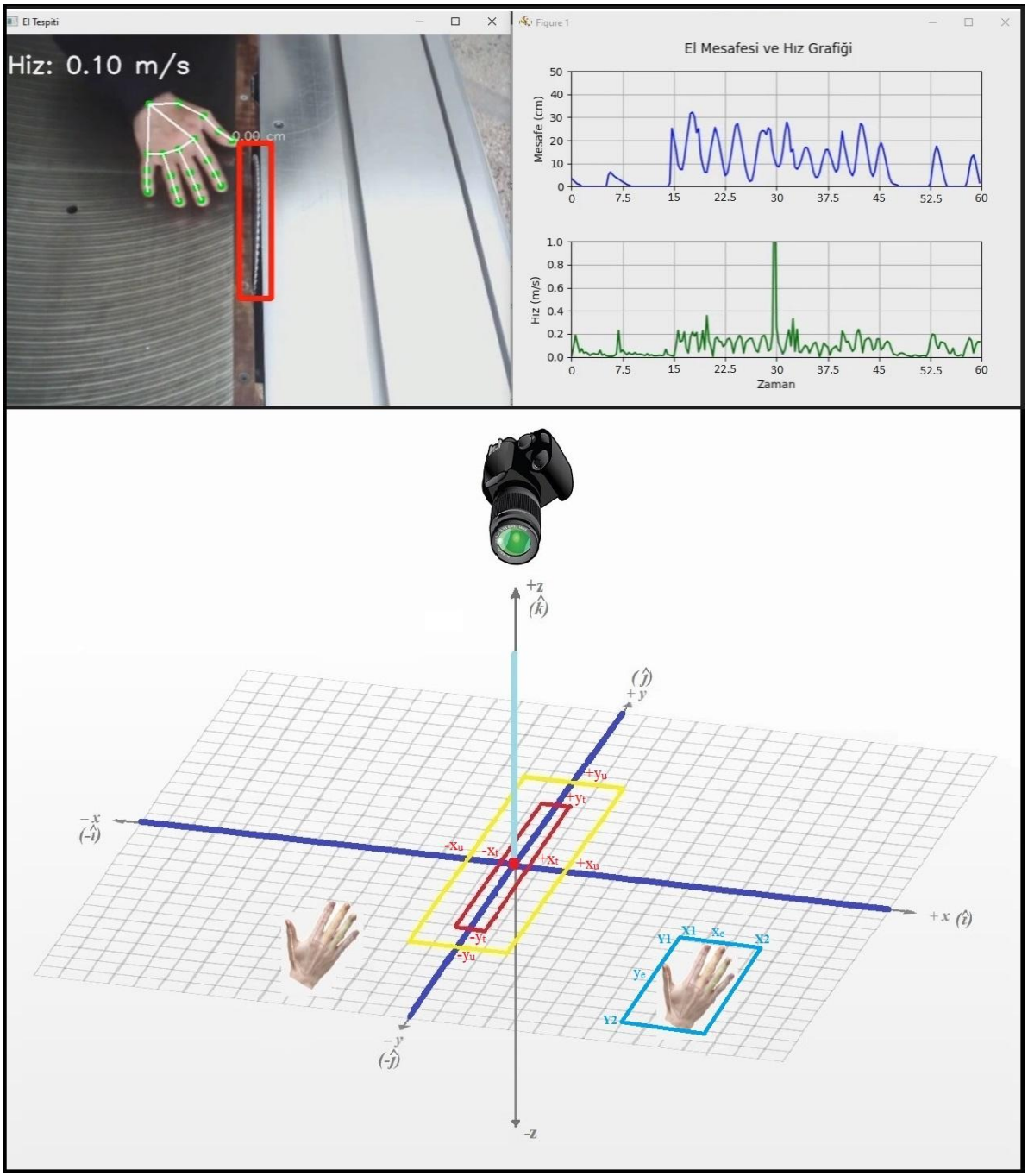
Tez çalışmasında görüntü işleme ve yapay zeka yaklaşımları kullanılarak, kesici daire testerenin bulunduğu tehlikeli bölgeyi tanımlamak, insan elini tanımak ve takibini yapmak, tehlikeli bölgeye el yaklaştığında uyarmak, çok yaklaştığında veya hızlı hareketlerde bulunduğunda makinayı durdurup, kazaları önlemek ve kullanıcıyı uyarıp, doğru davranışa yönlendiren bir sistem tasarımı üzerinde durulmuştur.

Çalışmada kullanılan kamera 30 FPS ve 640x480 pikseldir, çalışma alanı (makina tezgahı) ise 100x75 cm olarak belirlenmiştir, oransal olarak 100/640 veya 75/480 değerlerinin gerçek dünyada her pikselin karşılığı yaklaşık 0.16 cm olmaktadır, bu değer kamera pikseline ve görüntülenen alanın gerçek dünyadaki değerine göre değişiklik gösterir. Şekil 2.5'de kameradan alınan canlı görüntünün tasarlanan kodlama ile elin tanınması, kartezyen koordinat sisteminde tehlike ve uyarı bölgesinde elin kesici ile arasındaki mesafe ve hız değişiminin genel gösterimi yapılmıştır.

Kartezyen koordinat sisteminde tehlikeli iş makinasının kesicisi sıfır noktası kabul edilmektedir. Ek olarak kesici boyutları ($\pm x_k$, $\pm y_k$) ile tanımlanmakta olup yükseklik değeri z_k ile ifade edilmektedir. Buna göre tehlike ve uyarı bölgesi sırasıyla ($\pm x_t$, $\pm y_t$) , ($\pm x_u$, $\pm y_u$) ile tanımlanmaktadır. Elin konumu MediaPipe Hands makine öğrenimi modelinden alınan (x_{ei} , y_{ei}), i=0,…,20 için tehlikeli iş makinasıdaki kesiciye olan uzaklığı;

$$d_i = \begin{cases} |x_{ei} - x_k| , & |y_{ei}| \le y_k \\ \sqrt{(x_{ei} - x_k)^2 + (y_{ei} - y_k)^2}, & |y_{ei}| > y_k \end{cases} \quad i = 0, \dots, 20 \tag{2.1}$$

ile elde edilen 21 anahtar nokta mesafesinin en düşük değeri olarak hesaplanmaktadır. Denklem (2.1)'den en kısa mesafede d_e için elin konumu (x_e , y_e) olarak bulunur ki tehlike ve uyarı bölgesi içinde yer alması durumları incelenmektedir. Burada x_k değeri oldukça küçük olduğun için sıfır olarak değerlendirilecektir. Ek olarak elin hızının hesaplanmasında, birim zamanda 21 anahtar noktanın konum değişiminde en büyük olan değeri alınmakta olup v_e olarak değerlendirilmektedir. Burada z_e elin yüksekliğinin z_k kesici yükseklik değerinden düşük olması durumunda hesaplama yapılmaktadır. Büyük olması durumunda ise el tehlikeli bölgenin üzerinden, güvenli sayılabilecek bir mesafeden geçmektedir ki el denetimi devre dışı kalmaktadır.

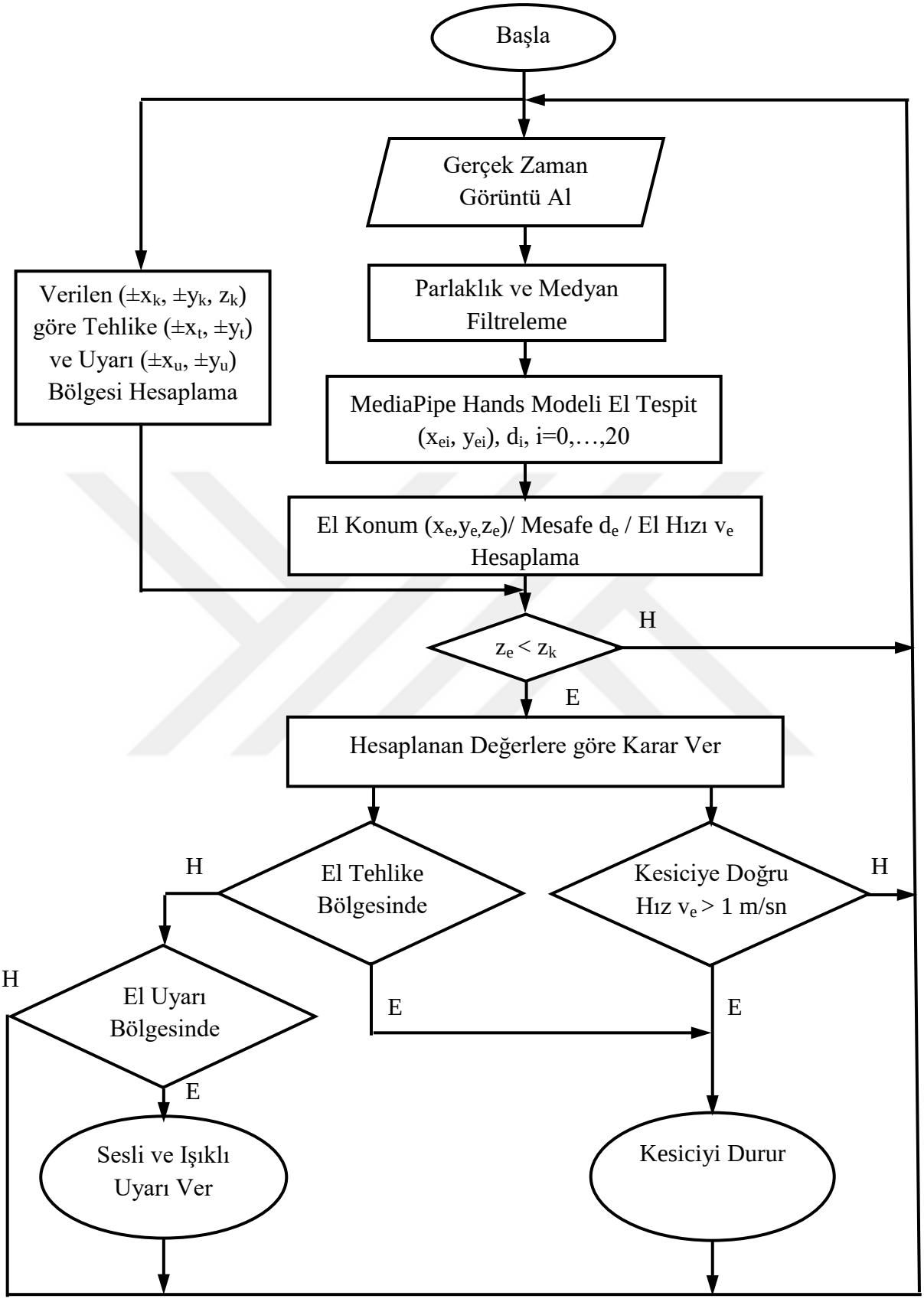


Şekil 2.5 Önerilen sistem MediaPipe Hands modeli ile el tespiti

Tasarımda, kameradan alınan görüntünün adım adım nasıl işlendiği Şekil 2.6'daki algoritma akış şeması ile verilmektedir. Burada görüntü alınmasıyla öncelikle performans iyileştirmek amacıyla parlaklık düzenleme ve medyan filtreleri gerekli durumlarda uygulanır. Bir sonraki görüntü alınmasına kadar işlemcinin gerçek zaman çalışabilmesi için tüm süreci tamamlaması gerekmektedir. Tehlike ve uyarı bölgeleri tespitinin hesaplanması ile birlikte MediaPipe Hands modeli kullanılarak el tespiti gerçekleştirilir. Elin görüntü üzerindeki konumuna göre gerekli durumlarda uyarı şiddeti artırılarak yapılır veya kesici durdurulur.

Çalışmada 1. derece tehlikeli kırmızı bölge ($\pm x_t$, $\pm y_t$) kesiciyi kapsayan bölge olup, sarı bölge ($\pm x_u$, $\pm y_u$) ise uyarı bölgesi 2. derece tehlikeli bölgedir. Bu bölge yaklaşım mesafesinden hesaplanır ki örnek olarak 0.10 m eşik değeri aşılmışsa sesli ve ışıklı uyarı verilir. Eğer kullanıcının eli 1. derece tehlikeli durdurma bölgesine geçerse, sistem tamamen durdurulur. Benzer şekilde eller noktasal olarak 1 m/sn'den daha hızlı tehlikeli bölgeye doğru hareket ederse sistem durdurulur. Elin tehlikeli bölgeye girme durumu hesaplanarak, erken uyarı ile makinanın öncelikle durdurulması, sesli ve ışıklı vb. uyarı verilmesi amaçlanır. Uyarı bölgesi ise 2. derece tehlikeli bölgeyi temsil eder ve tehlikeli bölgeden önce sesli ve ışıklı vb. uyarılar verilerek makina operatörünün tehlikeli bölgeye yaklaşmadan önce uyarır.

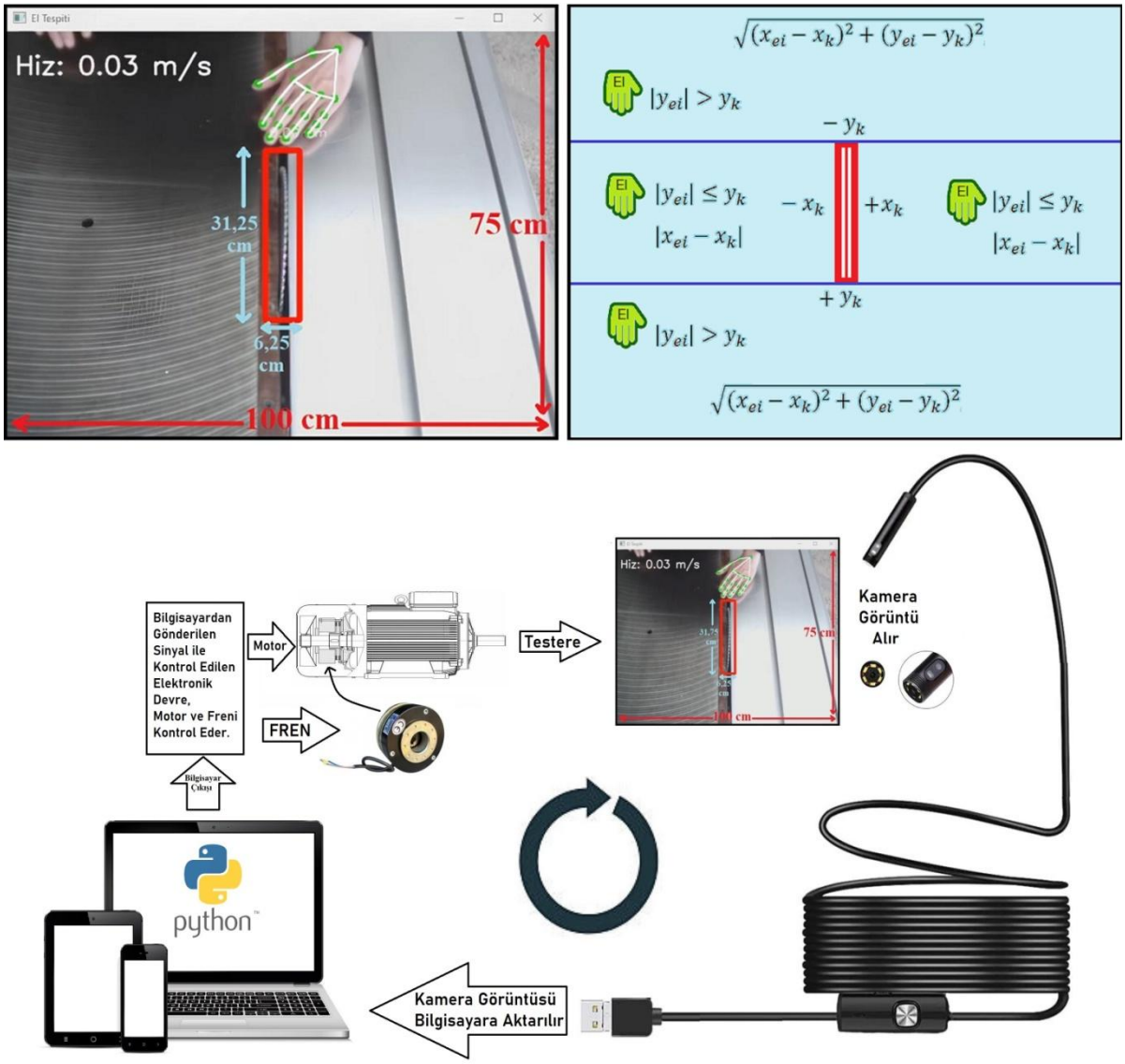
Önerilen sistemin gerçek zamanlı çalışabilmesi işlem hızı ile birlikte görüntü alma hızına bağlıdır. Burada her bir görüntü için ortalama tespit ve toplam işlem süresi mevcut donanım için Python yazılımının toplam veri işleme ve çıktı verme süresi yeterli olmalıdır. Sistemden üretilen tetikleme çıktıları, daire testere vb. tehlikeli makinaların acil durdurma sistemini etkilemektedir. Dr. Steve Gass tarafından geliştirilen sistemin durma süresi 0.005 sn olduğu belirtilmektedir, ancak bu süre testereye temas ile başlamakta ve alüminyum fren kartuşu ile kısalmaktadır. Fren kartuşu sistemi tekrar kullanılamaz hale getirebilmektedir, dolayısıyla daire testere ve kartuş yenilenmelidir. Fren kartuşu olmazsa bu süre 0.015 sn civarındadır ki bu süre testere gizlenme sisteminin eli koruması için geçen süredir [15, 16]. Burada ek bilgi olarak elin 1 m/sn hız ile testere teması halinde 0.005 sn sürede 0.005 m kesik, fren kartuş olmadan 0.015 sn sürede ise 0.015 m kesik olmaktadır, görüntü işleme ile korunacak olan el tehlikeli kesici bölgesine girmeden durdurma işlemi yapılacak olup, erken durdurma yapılmakta ve el kesici ile hiç temas etmeden durdurmayı amaçlamaktadır. Elin hızı ile durdurma sisteminin hızını birlikte değerlendirilerek uyarı ve tehlike bölgesi hesaplanmaktadır. Tespit sistemi kameradan aldığı canlı görüntüyü işleyerek, çıkış birimlerine tetikleme sinyali göndermektedir. Burada durdurma sistemi süresi $\mathbf{t_d}$ ve veri işleme süresi $\mathbf{t_v}$ olarak tanımlanacak olursa, tehlike bölgesi sınırları için el hızı $\mathbf{v_e}$ için $\mathbf{v_e*(t_d+t_v)}$ tehlike bölgesi sınırlarını belirlememizi sağlayacaktır. El hızı 1 m/sn için durdurma sistemi 0.005-0.015 sn olması durumunda veri işlem süresi 0.015-0.030 sn seviyesinde olması 0.03-0.05 m tehlike bölgesi sınırını oluşturmaktadır. Bu süreler donanımın ve tasarımın performansına bağlı olarak kısalabilir veya uzayabilir. Uyarı bölgesi sınırı ise tehlike bölgesi sınırının iki katı kadar alınmaktadır.



Şekil 2.6 Yapay zeka ile el kazalarını önlemek için tasarlanan sisteminin algoritma akış şeması

Sitemin tasarımında çalışma ortamını, tehlikeli bölgeyi, insan elini görüntülemek için kullanılan kamera ve görüntünün aktarılacağı bilgisayar (tasarıma göre telefon, tablet gibi diğer mobil cihazlarda olabilir) gereklidir. Bilgisayara aktarılan kamera görüntüsü bir yazılım ortamında işlenmelidir, işlenen görüntüdeki insan eli tanımlanmalı ve takibi yapılmalıdır, tehlikeli durumların çıktısı üretilmeli, bilgisayarın çıkış birimleri üzerinden makinanın kesme işlemini yapan testereyi tahrik eden elektrik motoru ve durdurucu fren tahrik mekanizması anlık kontrol edilmelidir, bu çevrim sürekli olarak devam etmelidir.

Tasarlanan sistemde fren mekanizması olarak, motor sargılarının enerjisi kesildikten sonra sargılara uygun seviyede DC uygulayarak dinamik ani frenleme sağlanabilir, buna ek olarak elektromekanik frenleme ve sistemi komple gizleyen mekanik frenleme sistemleri yapılabilir.

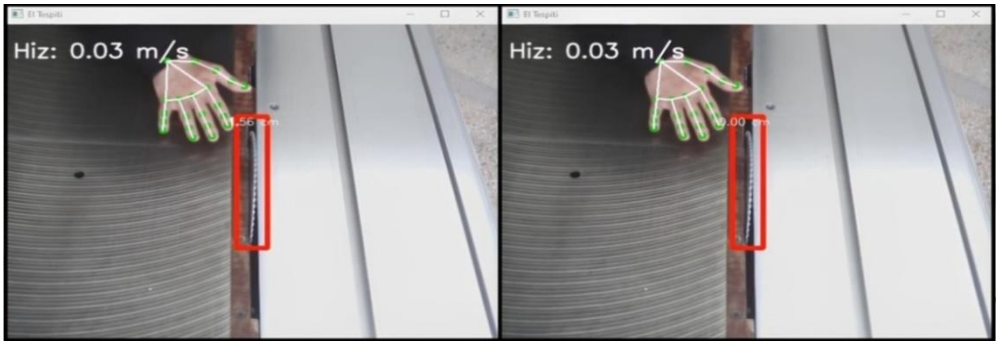


Şekil 2.7 Sisteminin genel yapısı ve çevrimi

2.3.3 MediaPipe Hands Modelinin Güvenlik Sistemi için Uygun Hale Getirilmesi, Hız ve Mesafe Hesaplamaları

MediaPipe Hands modeli elin 21 noktasını referans alır, noktaların gerçek dünyadaki konumları doğru hesaplanmalıdır, aksi halde mesafe ölçümünde konumsal hatalara sebep olabiliyor, bu sebeple el modeli tasarlanan sistemde doğrudan kullanılamaz, bazı düzenlemelerden sonra kullanılabilir. El modelin referans aldığı noktalar orta noktalardır, örneğin; parmağın orta noktasını referans alır, dolayısıyla parmakların tehlikeli bölge ile arasındaki mesafenin hesaplanmasında yanlış sonuç verir, asıl tehlikeli nokta parmakların orta noktası değil, dış kenarlarıdır. Bunu şöyle düşünmek gerekir, örneğin; iri bir insanın baş parmak kalınlığı yaklaşık 3 cm ise bunun orta noktası 1.5 cm yapar, bu da mesafe ölçümünün tüm noktalar için yanlış yapılması anlamına gelir. Bu sorunu çözmek için tehlikeli bölge ile elin noktaları arasına ekstra teğet mesafesi eklendi. Tehlikeli bölge ve elin en yakın noktası arasındaki mesafe, piksellerin konumları üzerinden belirlenir. Tasarımda 1. derece tehlikeli kırmızı bölge ile parmakların orta noktasından 3 cm teğet mesafesi eklendi, böylece temas durumundaki hatalar giderildi.

El modelindeki noktalar tehlikeli bölgeyi çevreleyen kırmızı dikdörtgen çizgilerinden etkilenebilmektedir, noktalar dikdörtgen temasından kaçma eğilimi gösterebilmektedir, bu da eli savunmasız hale getirebilir, dolayısıyla bu sorun da teğet mesafesi ekleyerek çözülmektedir. Aşağıdaki Şekil 2.8'de görüleceği üzere elin kırmızı dikdörtgene en yakın olan noktası soldaki görselde temas etmesine rağmen, sistemin uyarıya geçmesi için 1.56 cm mesafe varken, sağdaki görselde ise teğet mesafesi eklenmiş ve parmak kırmızı dikdörtgene yaklaştığında sıfır değerine ulaştığı için sistem uyarıya geçer. Teğet mesafesi bir tolerans gibi düşünülebilir, buna ek olarak ($\pm x_u$, $\pm y_u$) ön frenleme yada erken durdurma mesafesi eklenerek, kullanıcıya erken uyarı ile hatalı davranışlardan korunmasını öğretebilir.



a) b)

Şekil 2.8 a) Teğet mesafesi yok, b) Teğet mesafesi var

Elin tehlikeli bölgeye yaklaşma mesafesinin hesaplanması; pikseller cinsinden hesaplanıp, santimetre cinsinden mesafe (distance) dönüşümü ve ekstra ön durdurma frenleme mesafesi eklenmektedir. Örnek yazılım kodu EK A'da açıklamalarıyla verilmektedir.

$$d = \sqrt{dx^2 + dy^2} \tag{2.2}$$

MediaPipe Hands modelini kullanarak elin tehlikeli bölgeye ne kadar yakın olduğunu tespit edebiliriz, ancak güvenliği sağlanması gereken el sayısı 1 den fazladır. Bu durumda ellerden ve parmaklardan hangisi tehlikeli bölgeye en yakınsa, en yakın nokta takip edilmelidir. Ancak hız ölçümünde durum farklıdır, tehlike arz eden iş makinasında çalışan operatörün 2 eli vardır, eğer çalışan sayısı 2 kişi ise el sayısı 4 olur, bu sayı daha da artabilir. Dolayısıyla aynı anda 2, 4, hatta daha fazla elin hızını ölçümlemek gereklidir, dahil olan her yeni el işlem yükünü arttırmaktadır, dolayısıyla tespit süreleri değişebilmektedir, bunun önüne geçmek için iyi optimize edilmiş bir tasarım, işlem yükünü kaldırabilecek bir bilgisayar, görüntü kalitesi iyi ve yüksek FPS değerli bir kameralar gerekmektedir. Özellikle gerçek zamanlı çalışmada hız ölçümünde görüntü kareleri (bir önceki kare) arasındaki farklar ile ölçüm yapıldığı için işlem sürelerinin kısa olması mesafe değişimine bağlı hız ölçümünü daha doğru hale getirmektedir, ölçüm hatalarını azaltmaktadır. Detaylar EK A'da açıklamalarıyla verilmiştir.

Hız hesaplamasında piksellerin öklit mesafesi; $d\Delta = \sqrt{dx^2 + dy^2}$ (2.3)

Gerçek zamanlı çalışmada kareler arası işlem süresi; $t\Delta$ (2.4)

Hız ise mesafenin zamana oranıdır yani; $v = \frac{d\Delta}{t\Delta}\ m/s$ (2.5)

Tasarlanan sistem, gerçek zamanlı olarak makinayı kullanan operatörün elini çalışma esnasında oluşan tehlikeli durumlara karşı korurken, diğer taraftan da işin verimli şekilde aksamadan devam etmesi gereklidir. Bunun için makinanın kendisi ve işlenen malzemelerin içinden elin tanınması ve korunması, denetlenmesi sağlanmalıdır. Bunun için öncelikle tehlikeli bölge belirlenir, sonra insan eli belirlenir, sonra insan elinin tehlikeli bölgeye olan 2 ve 3 boyutlu konumunun belirlenmesi gerekir. Ancak bu da yeterli değildir, tehlikeli bölgenin dışındaki çalışma alanındayken aşırı hızlı (agresif) el hareketleri varsa, trafik denetimi gibi hız kontrolü yapılmalı ve kesiciye el teması olmadan önce makina durdurulmalıdır. Ayrıca bu tasarımda ölçülen hız değeri, kesiciye yaklaşırken üstel olarak artarken, kesicinin uzağında ise azalan bir tasarıma da yer verilmiştir ki makinayı kullanan kişiye kesicinin yakınında ise hızını düşürmesi öğretilir. Örnek yazılım kodu EK A'da açıklamalarıyla verilmektedir.

Gerçek zamanlı hız denetiminde bilgisayarın işlem yükünü azaltmak için, hız ölçümünde MediaPipe Hands modelinin 21 noktasının tamamını takip etmek yerine hareketlilik oranı en fazla olan 7 noktayı takip etmek, anlık hız hesaplamasında işlem yükünü 1/3 oranında azaltmakta ve tasarımın daha hızlı tespit yapmasını sağlamaktadır.

Takip edilen en aktif ve hızlı noktalar (0, 4, 8, 9, 12, 16, 20) noktalarıdır.

0: Sol yada sağ (wrist) elin bilek noktası.
4: Başparmak (thumb_tıp) eklemi, başparmağın en uç noktası (parmak ucu).
8: İşaret parmağı (index_finger_tıp) eklemi, işaret parmağının en uç noktası (parmak ucu).
9: Orta parmak (middle_finger_mcp) birinci eklem başlangıcı (elin ortası).
12: Orta parmak (middle_finger_tıp) eklemi, orta parmağın en uç noktası (parmak ucu).
16: Yüzük parmağı (ring_finger_tıp) eklemi, yüzük parmağının en uç noktası (parmak ucu).
20: Serçe parmak (pinky_tıp) eklemi, serçe parmağının en uç noktası (parmak ucu).

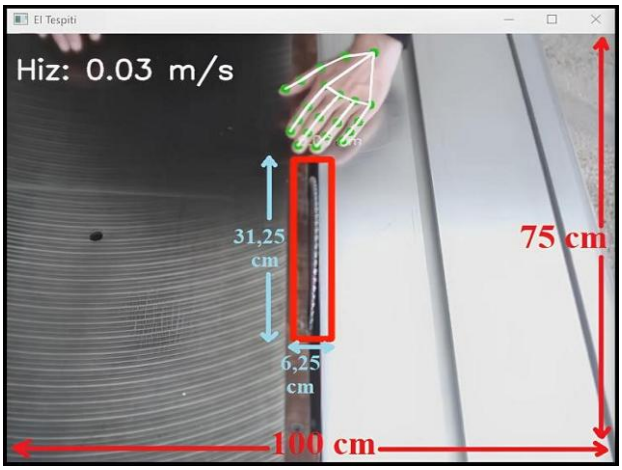
2.4 Çalışma Alanının Gerçek Değerleri ve Tehlikeli Alan

Çalışma alanı genel bir ifadeyle operatörün çalıştığı makinanın tablasının alanıdır. Tehlikeli alan ise kesicinin alanıdır. Alan belirlemede en önemli unsur kameranın gördüğü alanın, gerçek ölçülerini tespit etmektir. Örneğin; kamera tehlikenin olduğu bölgeyi yukarıdan, tıpkı güvenlik denetimi yapan bir insanın gözü gibi takip etmelidir. Tasarımda kullanılan kamera, 640 x 480 piksel çözünürlüğünde olup, kameradan alınan görüntünün 640 piksele denk gelen kısmı 100 cm (1 m) denk gelirken, 480 piksele denk gelen kısmı 75 cm'e denk gelmektedir. Buradan her 1 pikselin kenar uzunluğu yaklaşık olarak 1,6 mm dir. Bu değerler farklı kamera çözünürlüklerine ve denetim yapılacak alanın boyutlarına göre değişebilir.

2.4.1 X - Y Eksenleri ve Tehlikeli Bölgenin Belirlenmesi

Kartezyen koordinat sisteminde tehlikeli iş makinasının kesici testeresi merkez sıfır noktası kabul edilmektedir. Kesici boyutları ($\pm x_k$, $\pm y_k$) ile tanımlanmaktadır. Buna göre tehlike ve uyarı bölgesi sırasıyla ($\pm x_t$, $\pm y_t$) , ($\pm x_u$, $\pm y_u$) ile tanımlanmaktadır. Örnek olarak çözünürlüğü 640x480 olan kamera için tehlikeli bölgenin köşe noktalarını belirlerken, Şekil 2.9'daki gibi merkezi bir noktada X apsisisi yönünde 6.25 cm, Y ordinatı yönünde 31.25 cm uzunluklarında bir dikdörtgen çizilmesi durumunda, görüntünün orta merkez noktası belirlenmesinde, X ekseni için 320. piksel (640/2), Y ekseni için 240. piksel (480/2) bulunur. Gerekli durumda X ve Y eksen değerlerini farklı ihtiyaçlara göre değiştirilebilir. Benzer

uygulamada cm başına düşen piksel oranının sabit olması sebebiyle (pixel_per_cm_x(y)) belirlenmesi ile doğrudan cm cinsinden de yapılabilir, tamamen tasarıma bağlıdır.

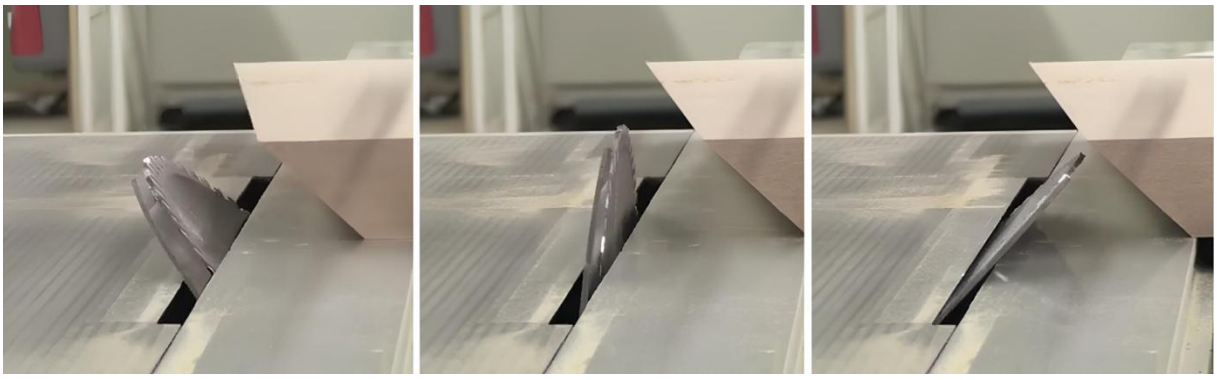


Şekil 2.9 Kameranın gördüğü alanın gerçek değerleri

Tehlikenin engellenmeye çalışıldığı makina özelliğinden dolayı "Yatar daire testere" olarak adlandırılmaktadır, makina normalde düz 90° iken, yatay olarak 89°- 45° sağ yöne, yeni modellerde 91° - 135° sol yöne yani 45° - 135° aralığında yatabilmektedir. Buradan anlaşılacağı üzere, daire testerenin yüksekliği ve açısı tehlikeli alanı belirlemede 2 durumu ortaya çıkartır;

a) Dairenin yüksekliğine bağlı olarak tehlikeli alanın Y yönünde uzayıp ve kısalması.
Dairenin testerenin yüksekliği 1 cm iken, korunan alanın uzunluğu standart 18.75 cm kabul edilsin, daire testerenin yüksekliği 9 cm daha yükseltilir ve 10 cm olursa tehlikeli bölgenin Y ordinatı yaklaşık 20 cm daha ekstra uzamalı, 38.75 cm olmadır. Benzer şekilde $\pm y_k$ daire yüksekliği ile orantılı olarak, tehlikeli bölge $\pm y_t$ ve $\pm y_u$ değerleri değişebilir olmalıdır.

b) Dairenin kesme açısına göre sağ 45° ve sol 135° bir alanda X yönünde uzayıp ve kısalması.



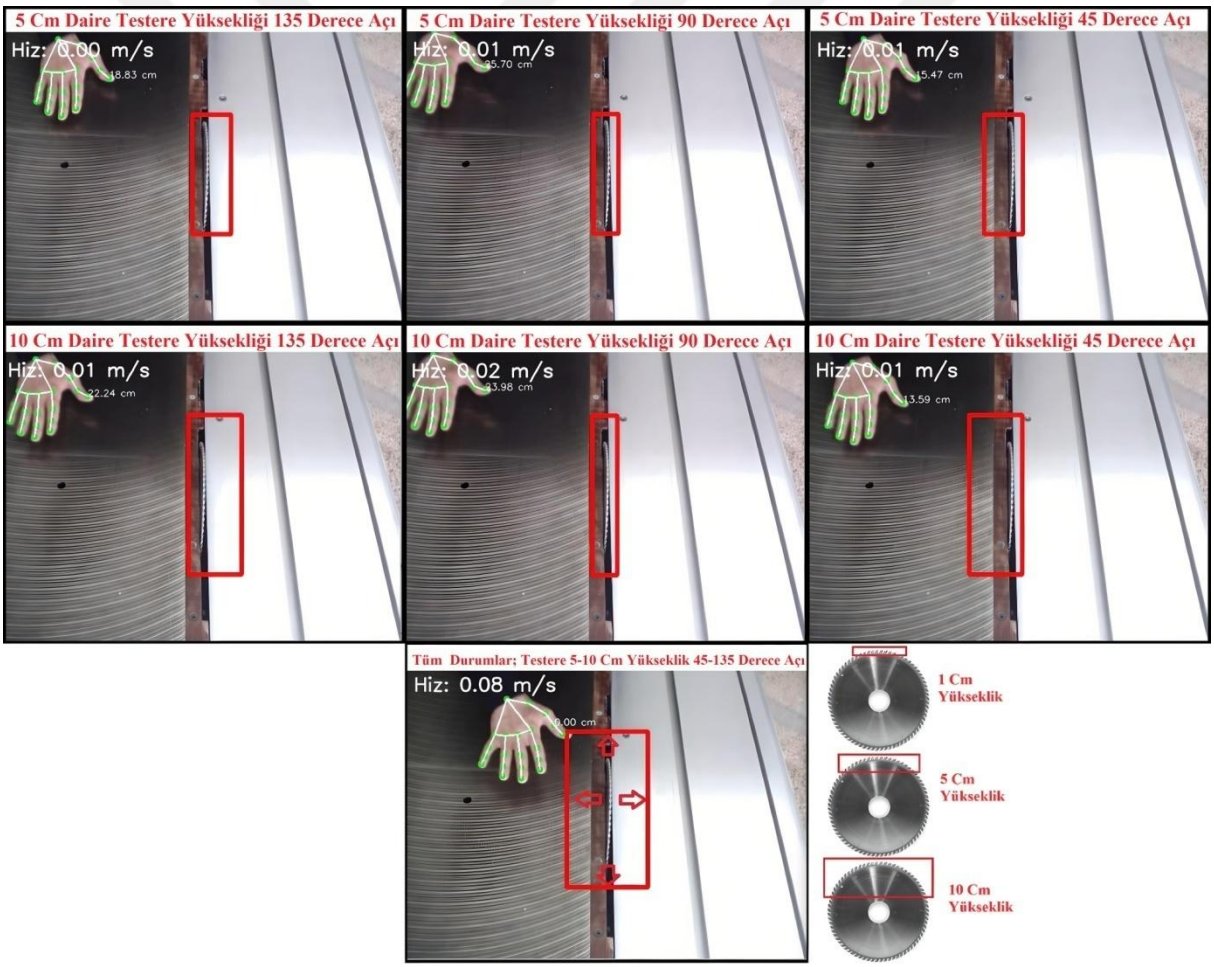
Şekil 2.10 Daire testerenin sol 135°, dik 90 °, sağ 45 ° açılı duruşu ile tehlikeli alan değişimi

Kesici testerenin yüksekliğine ve 45° açı ile sağa veya sola yatması durumuna bağlı olarak, 6.25 cm kabul edilen $\pm x_t$ tehlikeli alanı, kesici yüksekliğine bağlı olarak $5\sqrt{2}$ kadar genişlemelidir. Örneğin; 10 cm daire yüksekliği ve 45° açı ile $\pm x_t$ tehlikeli alanın genişlemesi;

$$10\text{Cos}(45) = 5\sqrt{2} \tag{2.8}$$

Dolayısıyla tehlikeli alanı belirleyen 2 temel unsur; daire testerenin yüksekliği ve açısıdır. Burada sisteme girilen 10 cm daire yüksekliği değerleri (pixel_per_cm_x ve pixel_per_cm_y) piksel/cm olarak normalize edilir, daha sonra bu alan mevcut koruma alanına eklenerek genişletilir. Örnek yazılım kodu EK A'da açıklamalarıyla verilmektedir.

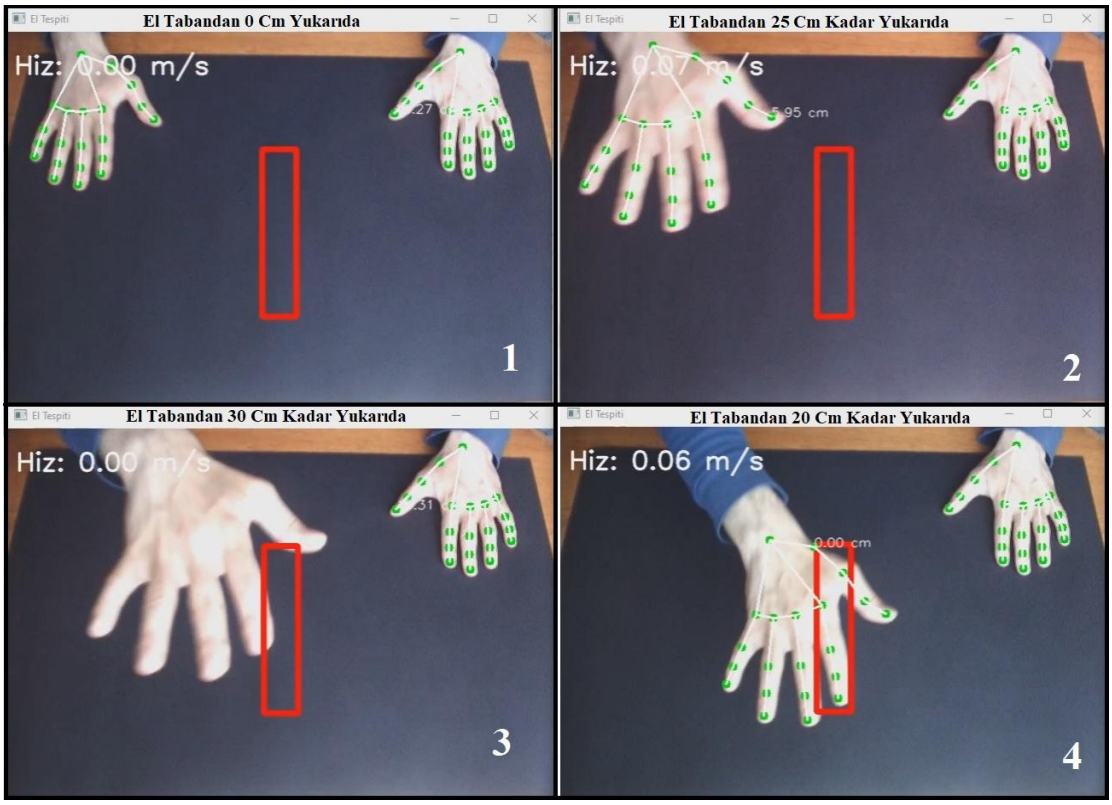
Testerenin yükseklik ve açısına bağlı olarak değişen tehlikeli kırmızı bölgeler.



Şekil 2.11 Testerenin aşağı-yukarı, sola 135° ve sağa 45° hareketinde değişen tehlikeli bölge

2.4.2 Z Ekseni ve Kameraya Olan Uzaklık Belirlenmesi

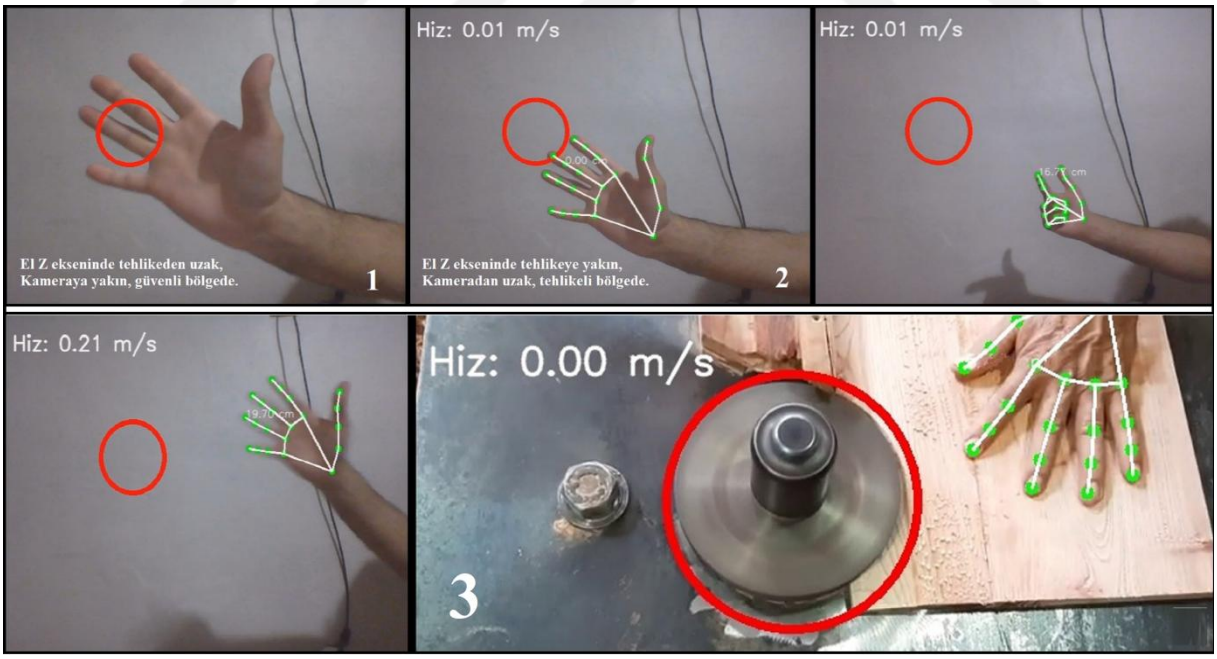
Tek kamera ile elin $\pm x_e$, $\pm y_e$ boyutları üzerinden z_e ekseninide kullanabiliriz (çift kamera kullanarak daha detaylı üç boyutlu ölçümlerde yapılabilir, ancak bu tasarımda tek kamera tercih edilmiştir). Örneğin; el kesicinin 30 cm üzerinde iken, tehlikeli bölgenin üzerindedir, ancak kesiciden uzak, kameraya ise yakındır, kameraya olan yakınlık - uzaklık mantığını kullanarak, el için bir z_e boyutu belirlenebilir, bu sayede el kesicinin üzerinden güvenli sayılabilecek bir mesafeden geçirilebilir. El z ekseninde kesiciye yukarıdan aşağı 30 cm kadar yaklaştığında tekrar güvenlik sistemi devreye girer. Bunun için el ölçüsünü belirleriz, örneğin; 20 cm uzunluğu olan bir elin, kesicinin 30 cm üzerinde 1.5 kat büyüme ile 20 cm den 30 cm ortalama değerlere büyüyeceği belirlenerek yaklaşık bir şekilde z_k bölgesinin üzerinden güvenli sayılan bölgede olur, böylece el tehlikeli bölgenin güvenli sayılabilecek kadar üzerinden dikkatlice geçirilebilir ve z koordinatında işlemler yapılabilir. Aşağıdaki Şekil 2.12'de 3 numaralı görüntüde el kameraya çok yaklaşmış, kesiciden uzaklaşmıştır, boyut filtresi devreye girdiğinden MediaPipe Hands modeli devre dışı kalmış, 21 noktalı model takibi bırakmıştır. El zemine tekrar 30 cm kadar yaklaştığında kameradan uzaklaşacak, küçülecektir ve 4 numaralı görüntüdeki gibi el koruma fonksiyonu tekrar devreye girip, koruma işlevini yapacaktır.



Şekil 2.12 Elin kameraya olan mesafesinden faydalanarak Z ekseninin belirlenmesi

Tasarımda elin tehlikeli alanın üzerinden geçme işlemini, elin x ve y ölçüsü belirlenen boyut değerini aşarsa el takipten çıkar, bu elin tehlikeli alandan uzaklaşıp, kameraya yaklaştığını gösterir. Bu şekilde 10 cm yükseklikteki bir daire testere tehlikeli alanının 30 cm üzerinden güvenli şekilde geçilebilir, eğer daha alçaktan geçilirse koruma devreye girer.

Tehlikeli bölge yatayda dairesel bir alan ise belirlenen x ve y kordinatlarına r yarıçaplı bir çember çizdirilerek, tehlikeli alanı dairesel olarak belirlemek de mümkündür. Ayrıca z ekseninde Şekil 2.13'de kameraya yaklaşıldığında koruma bilinçli şekilde devre dışı bırakılabilir, yani tehlikeli kesicinin üzerinden güvenli sayılabilecek, yüksek bir mesafeden geçmeye imkan sağlar, el tekrar kameradan uzaklaşıp, kesiciye yaklaşırsa, tehlikeli bölgeye girdiğinden el boyutu küçülür ve koruma devre girer. Şekil 2.13 freze tezgahı için tasarlanmıştır, 1 numaralı görselde el tehlikeli bölgenin üzerinde ancak, kameraya z ekseninde yakın, kesiciden ise uzak güvenli bölgededir. 2 numaralı görselde el kameradan uzaklaşmış ve tehlikeli bölgeye yukarıdan aşağıya doğru yaklaşmıştır, tekrar koruma devreye girmiştir. 3 numaralı görsel ise gerçek çalışma görüntüsüdür. Örnek yazılım kodu EK B'de açıklamalarıyla verilmektedir.



Şekil 2.13 Freze makinası için dairesel güvenli alan belirleme

2.5 Medyan ve Parlaklık Filtreleri ile Ortam Görüntüsünün İyileştirilmesi

MediaPipe Hands modelinin farklı parlaklık ve olumsuz ortam koşullarındaki performansının kullanıma uygunluk açısından deneyler ile test edilmesi ve iyileştirme önerileri gereklidir. Çalışmanın tasarımında Şekil 2.6'daki algoritmanın "Medyan ve Parlaklık Filtreleme" bloğunun MediaPipe Hands modeli üzerindeki etkileri incelenmelidir. MediaPipe Hands modelinin varsayılan ayarlarındaki özellikleri ile değişken ortam parlaklığı koşullarında, makina-atölye ortamında deneysel sonuçlar incelenmelidir. Bu deneyde olumsuz ortam koşulları simüle edilerek tespit performansındaki düşüşler izlenmekte ve iyileştirmeler için tasarlanan parlaklık ve medyan filtrelerinin tespit performansına etkileri ölçümlenmekdir. Deneyin standart ve tekrarlanabilir olması açısından 60 saniye 30 FPS bir gerçek dünya çalışma video kaydı oluşturulmuş olup, ortam koşulları farklı parlaklık seviyelerinde ölçümler ile tekrarlanabilir şekilde yapılmaktadır. Yapılan testte video görüntüsüne **%5 tuz**, **%5 biber** olmak üzere **%10 gürültü** eklenmiştir.

i-Filtresiz: MediaPipe Hands modelinin herhangi bir filtre olmadan, doğrudan tespit sonuçları.

ii-Parlaklık Filtresi: MediaPipe Hands modelinin el tespitini iyileştirmek için tasarıma eklediğimiz parlaklık dengeleme filtresi ile tespit sonuçları.

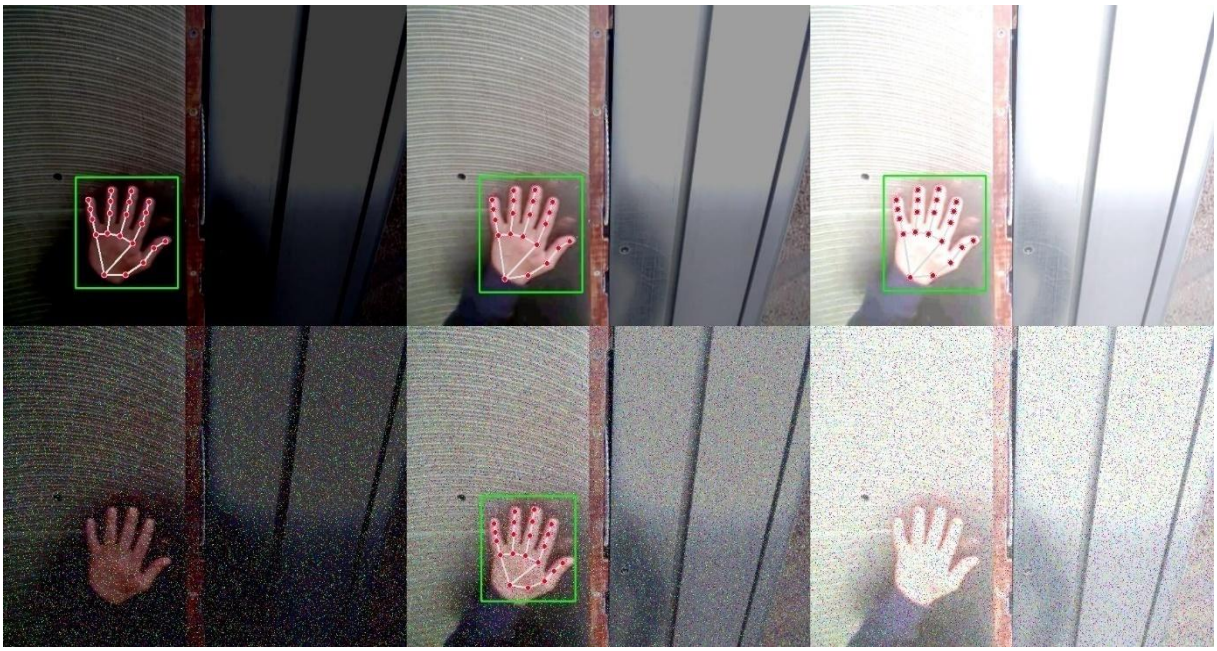
iii-Medyan Filtresi: MediaPipe Hands modeli el tespitini iyileştirmek için tasarıma eklenen gürültü temizleme filtresidir ki bu filtre yumuşatma yaparak tuz ve biber gürültülerini filtreler. Gerçek hayatta bu gürültülere toz, talaş ve çapaklar örnek gösterilebilir. Kameradan alınan görüntünün her pikseli RGB 3 farklı 256 değerden oluşan 8 bitlik renk düzeninde, tuz gürültüsü görüntüde bulunan beyaz noktalardır. Beyaz 8 bitlik renk skalasında 255 değeri ile tüm renklerdeki gürültüleri temsil ederken, siyah 0 değeri ile yokluğu yada kör noktaları temsil eder. Beyaz ve siyah noktalar görüntüdeki insan elini tespit etmeyi zorlaştırdığından, medyan filtresi ile iyileştirme yapılacaktır. Buradaki amacımız özellikle MediaPipe Hands modelinin olumsuz ortam koşullarındaki tespit performansını incelemek ve iyileştirmektir.

101	102	103	104	105
101	102	103	104	105
101	102	255	104	105
101	102	103	104	105
101	102	103	104	105

Şekil 2.14 Medyan filtre matrisi

Şekil 2.14'de örnek olarak verilen 5x5 lik bir görüntünün sayısal değerleri incelendiğinde; medyan filtresinde 24 değerin küçükten büyüğe doğru sıralandığında en ortada yer alan değeri 103 yapmaktadır ki ortadaki **255** değeri görüntüye eklenen beyaz tuz gürültüsüdür. Medyan filtre özelliği ile değeri aritmetik orta değer 103'e sabitler. Doğrusal olmayan medyan filtresi bu mantıkla aykırı olan tüm değerleri kaldırır ki bunun sonucunda gürültü temizlenirken görüntüde filtrenin boyutuna bağlı yumuşama gerçekleştirilir. Filtrenin boyutu temelde gürültünün yoğunluğuna bağlıdır ve yoğunluk arttıkça filtrenin (n x n) boyutu da artırılmalıdır.

iv-Parlaklık ve Medyan Filtresi: MediaPipe Hands modelinin hem parlaklık dengeleme hem de görüntüdeki gürültüleri ortadan kaldıran medyan filtre birlikte çalışma durumudur. Yapılan testte video görüntüsüne **%5 tuz**, **%5 biber** olmak üzere **%10 gürültü** eklenmiştir. MediaPipe Hands modeli ile yapılan tasarımın iyileştirilmesi için görüntü anlık olarak **5x5 medyan filtreden** geçirildi, gürültüler temizlendi ve sonra parlaklık dengelenmesi sağlandı, son olarak MediaPipe Hands modeli ile iyileştirilen görüntü işlenerek el tespiti yapılmıştır. Testte %10 gibi yüksek bir gürültünün **3x3** ve **5x5** boyutlu filtreler ile kontrolleri yapıldığında **5x5** medyan filtrenin gürültüyü tamamen temizleyebildiği görüldü ve sonuçlara bulgular kısmında Çizelge 3.2'de yer verildi.



Şekil 2.15 MediaPipe Hands modeli ile 21 anahtar nokta tespitinde parlaklık ve gürültü etkisi

BÖLÜM 3

BULGULAR ve TARTIŞMA

3.1 Gürültüsüz Ortam Testleri

MediaPipe Hands modelinin varsayılan ayarlarındaki filtreleme özellikleri ile değişken ortam parlaklığı koşullarında, makina-atölye ortamında deneysel sonuçlar incelenmektedir. Bu deneyde amaç atölye ortamında simüle edilmiş olumsuz ortam koşullarında MediaPipe Hands modelinin performans kayıplarını incelemek ve tespit performansını iyileştirmek için tasarlanan parlaklık ve medyan filtrelerinin tespit performansına etkileri ölçümlenmekdir.

Deneyin standart ve tekrarlanabilir olması için 60 saniyelik bir çalışma görüntü kaydı (30 FPS görüntü alma hızı) oluşturulmuş olup ortam koşulları farklı parlaklık seviyelerinde ölçümler tekrarlanabilir şekilde yapılmaktadır. Çalışmada kullanılan kamera 640x480 piksel görüntü almakta olup çalışma alanı ise 1 x 0.75 m olarak belirlenmiştir ki her pikselin karşılığı 0.16 cm kadardır. El hızı en fazla 1 m/sn için durdurma sistemi ile veri işlem süresi 0.03 sn seviyesinde olması 0.04 m tehlike bölgesi sınırı seçilmiş olup uyarı bölgesi sınırı ise tehlike bölgesi sınırının iki katı alınmaktadır. Elin kesiciye doğru hız sınırını aşması durumunda da durdurma ve uyarı işlemi yapılmaktadır. Varsayılan parlaklık değerinin renk sıcaklığı RGB 8 bit renk düzeninde [0, 255] aralığında 256 değerin orta noktası 128 olduğu kabul edilmiştir. Parlaklık seviyesi orta noktası 128 (varsayılan) kabul edilerek 50, 100 fazla (fazla ve çok fazla) ve 50, 100 az (düşük ve çok düşük) seviyelerde kayıtlar bulunmaktadır. Ek olarak çalışma görüntü kaydına %5 tuz, %5 biber olmak üzere %10 gürültü eklenerek sonuçlar kıyaslanmaktadır. Performans kıyaslaması tamamında en az bir el görüntüsü bulunan çalışma kaydında el tespitinin yüzdesel değişimi ile yapılmaktadır.

Mediapipe Hands modeli kullanımında gerçek zaman görüntüler üzerinde çalıma seçimi yapılırken görüntüdeki el sayısı 1, el tespit/takip minimum güven katsayısı 0.5, model karmaşıklığı da 1 olarak seçilmiştir. En fazla el sayısının 1 seçimi gerçek zaman çalışabilme durumu ile işlemci yükünün azaltılması hedeflenmektedir. Ek olarak model 21 anahtar noktanın konum bilgisi ile birlikte el tespit sonucunun güven puanını da çıktı olarak değerlendirilmektedir.

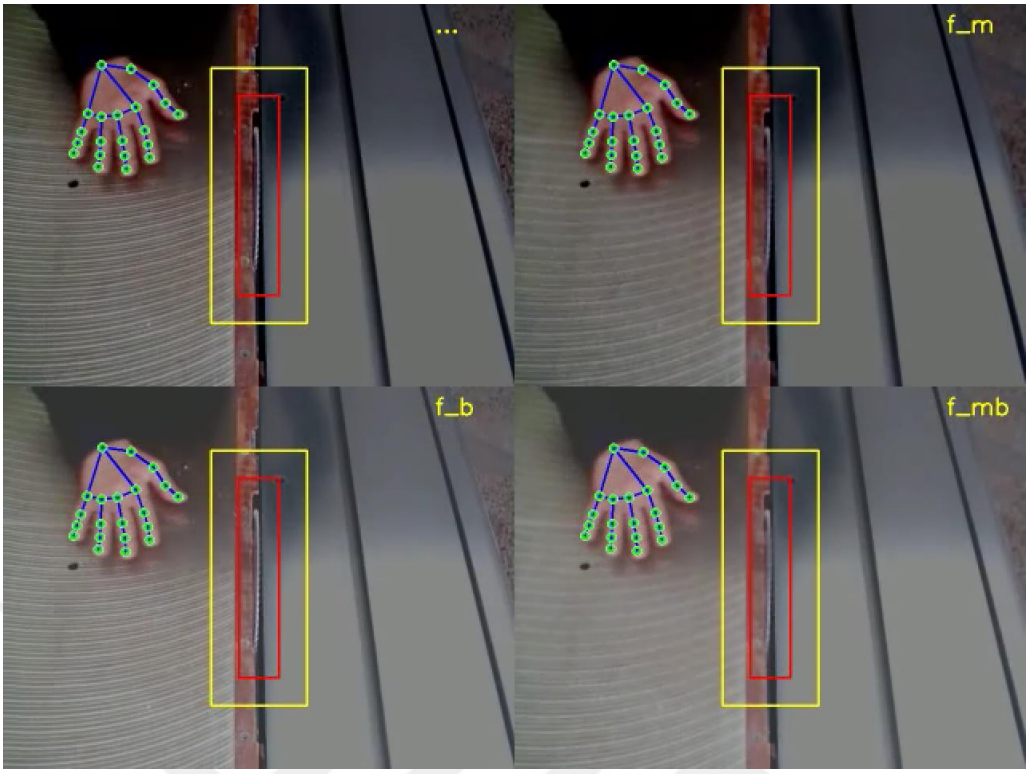
Bu çalışmada kullanılan model %95 güven düzeyinde elde edilen güven puanı sonuçları da kıyaslanmaktadır.

Çizelge 3.1'de gürültüsüz ortamda ve farklı aydınlık koşullarında el tespit oranları verilmektedir. Ek olarak gerçek zaman çalışmalarını etkileyecek işlem süreleri de verilmektedir. Burada tek bir el tespiti için çalışma yapılmakta olup iki el çalışması için yapay zeka modeli işlem süresi yaklaşık iki kat artmaktadır. Gürültüsüz durum incelendiğinde, MediaPipe Hands modeli tespit performansına kıyasla, ek parlaklık dengeleme ve medyan filtreleri fazla bir etki gösteremediği değerlendirilmektedir. Parlaklık seviyesinin çok düşük ve çok fazla olması az da olsa performansı düşürmektedir. Ek olarak not etmek gerekir ki işlem süresi kullanılan parlaklık dengeleme filtresi ile bir saniye, medyan filtresi ile beş saniye kadar artmaktadır. Bununla birlikte işlem süresi görüntü alma hızının yarısı kadar olup gerçek zaman çalışma yapılabilmektedir.

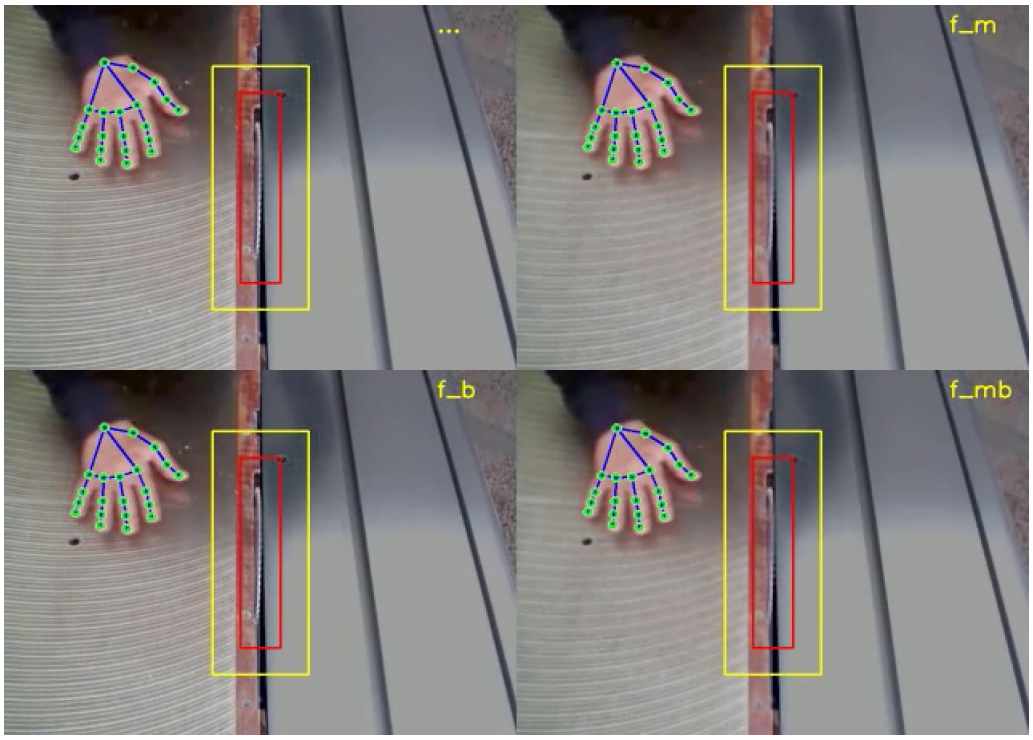
Çizelge 3.1 Gürültüsüz ortamda, farklı parlaklık koşullarında önerilen sistem el tespit performansı ve işlem süreleri

Parlaklık Oranı	Filtresiz	Parlaklık Filtresi	Medyan Filtresi	Parlaklık ve Medyan Filtreleri	İşlem Süresi (sn)
Çok Düşük (28)	%99,94	%100,00	%100,00	%100,00	(25,63 - 30,96)
Düşük (78)	%100,00	%100,00	%100,00	%100,00	(25,51 - 30,78)
Varsayılan (128)	%100,00	%100,00	%100,00	%100,00	(25,03 - 30,45)
Fazla (178)	%99,89	%100,00	%100,00	%100,00	(25,60 - 31,01)
Çok Fazla (228)	%99,83	%99,94	%100,00	%100,00	(25,38 - 30,91)

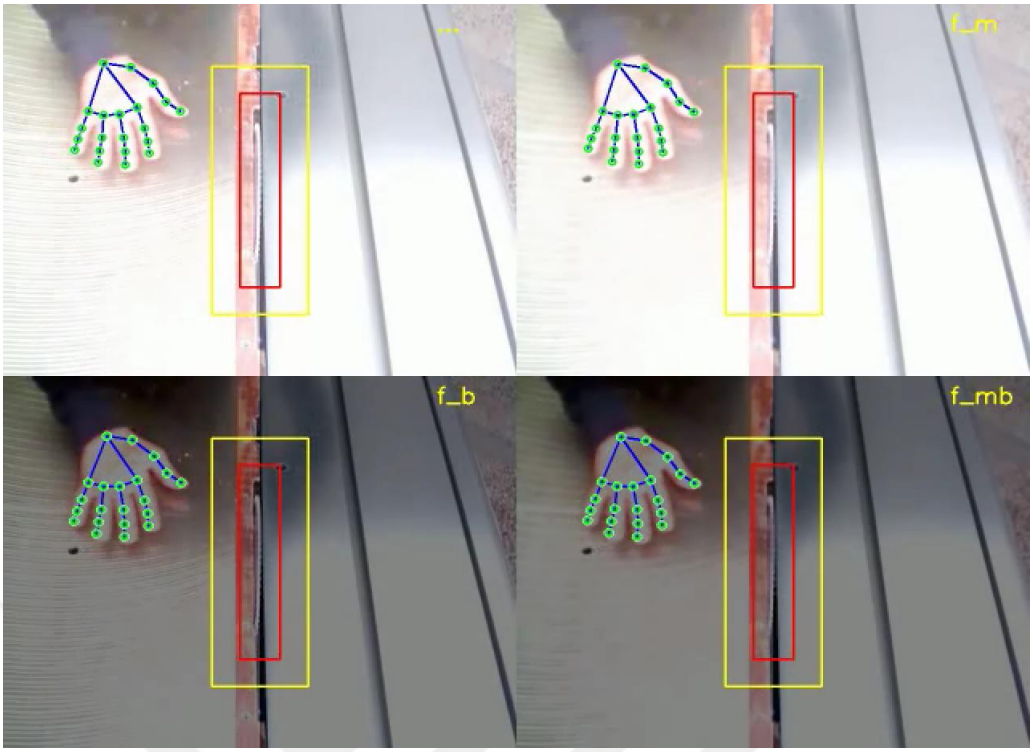
Parlaklık dengeleme ve medyan filtrelerin gürültüsüz durumda çok az, varsayılan ve çok fazla parlaklık seviyelerinde el tespit sonuçları Şekil 3.1-3.3'de sırasıyla verilmektedir. Burada önerilen sistem tüm koşullarda el tespiti yapmakta olup görüntülerde iyileştirmeler değerlendirilmektedir. Tespit edilen 21 anahtar noktanın iş makinası kesiciye yakınlığına göre tehlike ve uyarı işlemleri gerçekleştirilmektedir.



Şekil 3.1 Varsayılan parlaklıktan 100 birim daha düşük karanlığa yakın ortam parlaklığı el tespit sonuçları (filtresiz, f_b parlaklık, f_m medyan, f_mb medyan-parlaklık filtreleri)

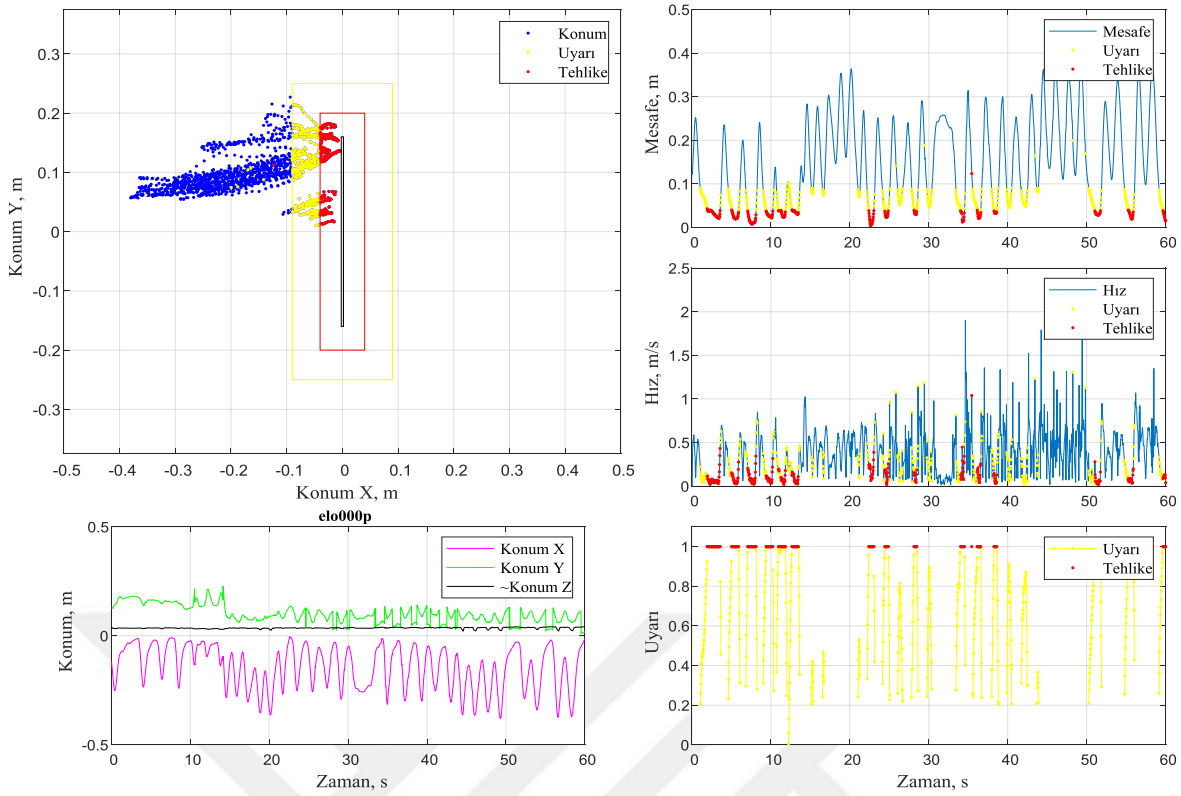


Şekil 3.2 Varsayılan parlaklıkta el tespit sonuçları (filtresiz, f_b parlaklık, f_m medyan, f_mb medyan-parlaklık filtreleri)

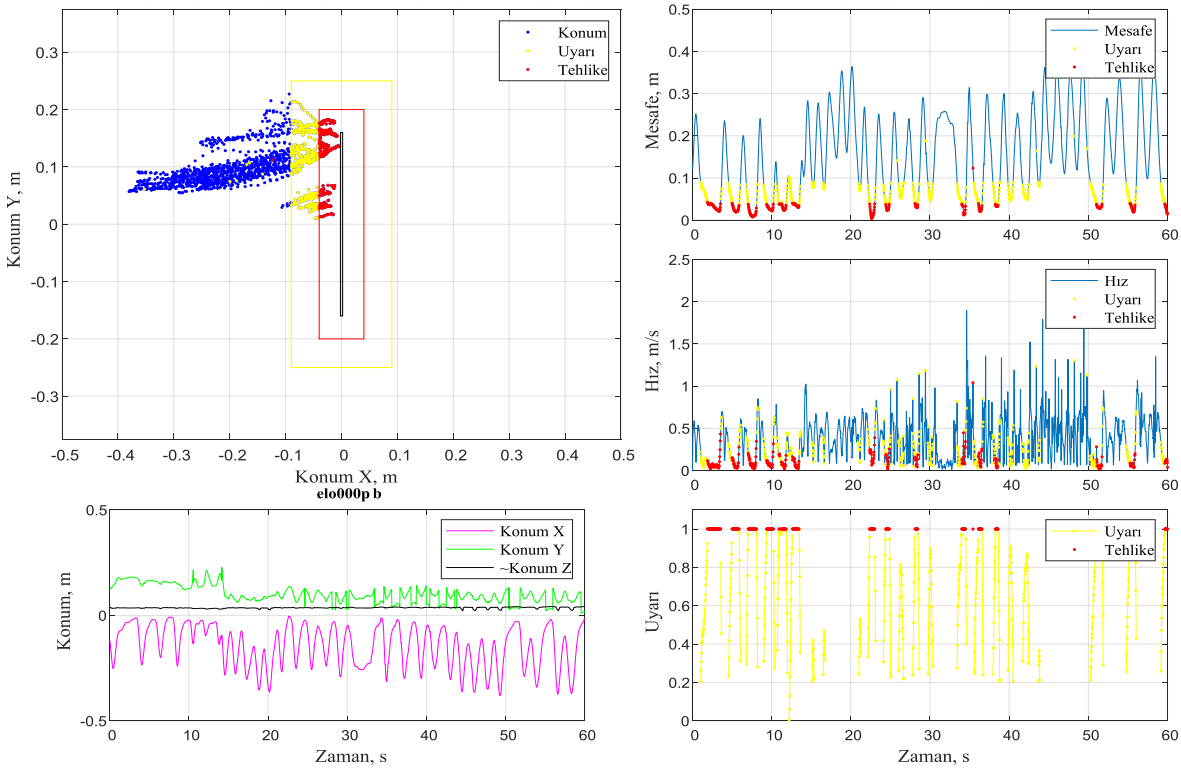


Şekil 3.3 Varsayılan parlaklıktan 100 birim daha fazla ortam parlaklığı el tespit sonuçları (filtresiz, f_b parlaklık, f_m medyan, f_mb medyan-parlaklık filtreleri)

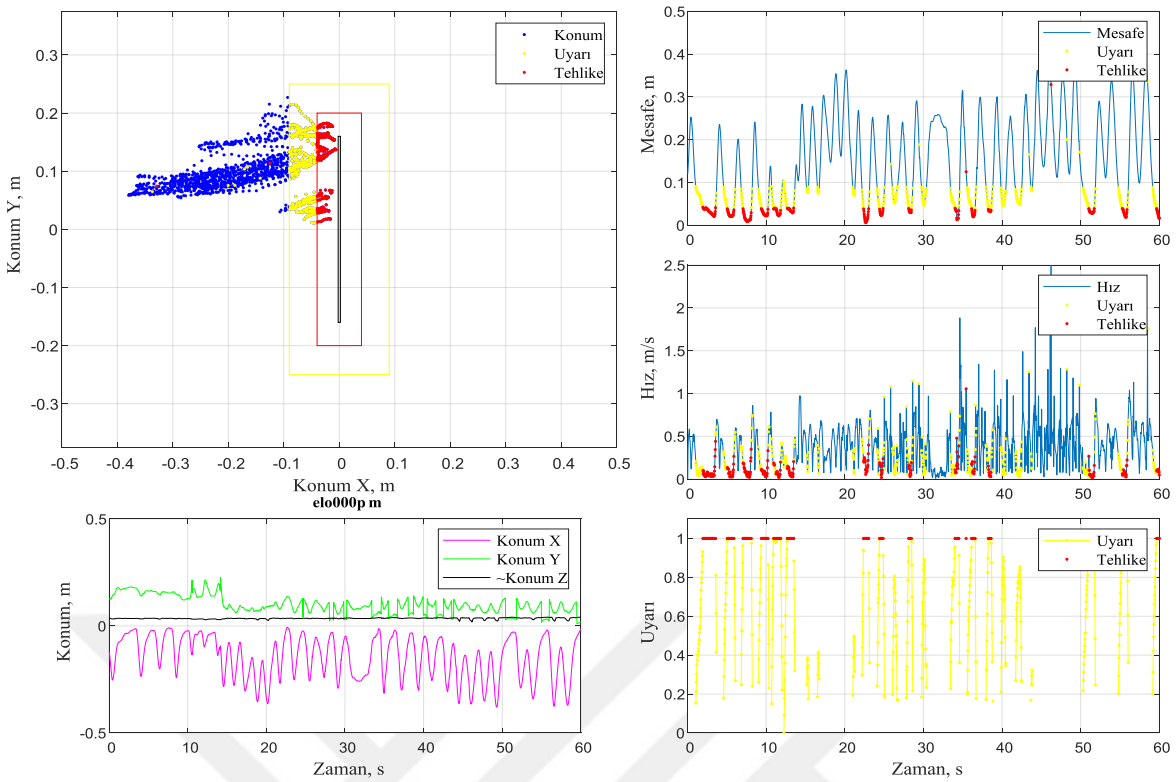
Burada tehlike ve uyarı bölgesinde olma durumu kıyaslamaların detaylı analizi sırasıyla filtresiz, parlaklık dengeleme, medyan ve medayan-parlaklık filtreleriyle Şekil 3.4-3.7'de verilmektedir. Burada deney süresince tespit edilen elin mesafe ve hız değişimi verilmekte olup tehlike ve uyarı bölgesi durumları değerlendirilmektedir. Ek olarak X-Y düzleminde konum değişimi ile tehlike ve uyarı bölgesinde olma durumları da gözlemlenmektedir. Burada elin hızının tehlike bölgesine doğru sınır 1 m/s üzerinde olması durumunda da sistem iş makinasının durdurulmasını sağlamaktadır. Ek olarak uyarı ve durudurma işlemlerinin zamana göre değişimi de değerlendirilmektedir.



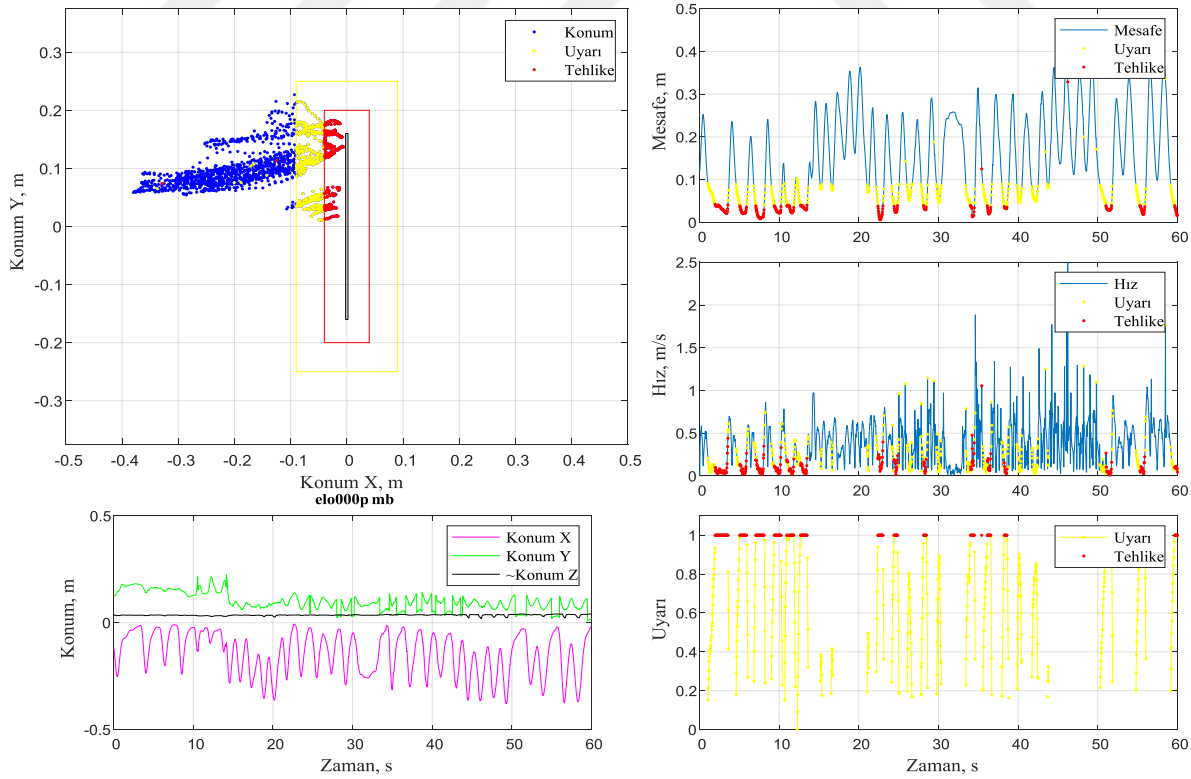
Şekil 3.4 Varsayılan parlaklıkta filtresiz el tespit sonuçları; konum, mesafe, hız, uyarı ve konum değişimi



Şekil 3.5 Varsayılan parlaklıkta parlaklık filtresi el tespit sonuçları; konum, mesafe, hız, uyarı ve konum değişimi



Şekil 3.6 Varsayılan parlaklıkta medyan filtresi el tespit sonuçları; konum, mesafe, hız, uyarı ve konum değişimi



Şekil 3.7 Varsayılan parlaklıkta medyan-parlaklık filtresi el tespit sonuçları; konum, mesafe, hız, uyarı ve konum değişimi

3.2 Gürültülü Ortam Testleri

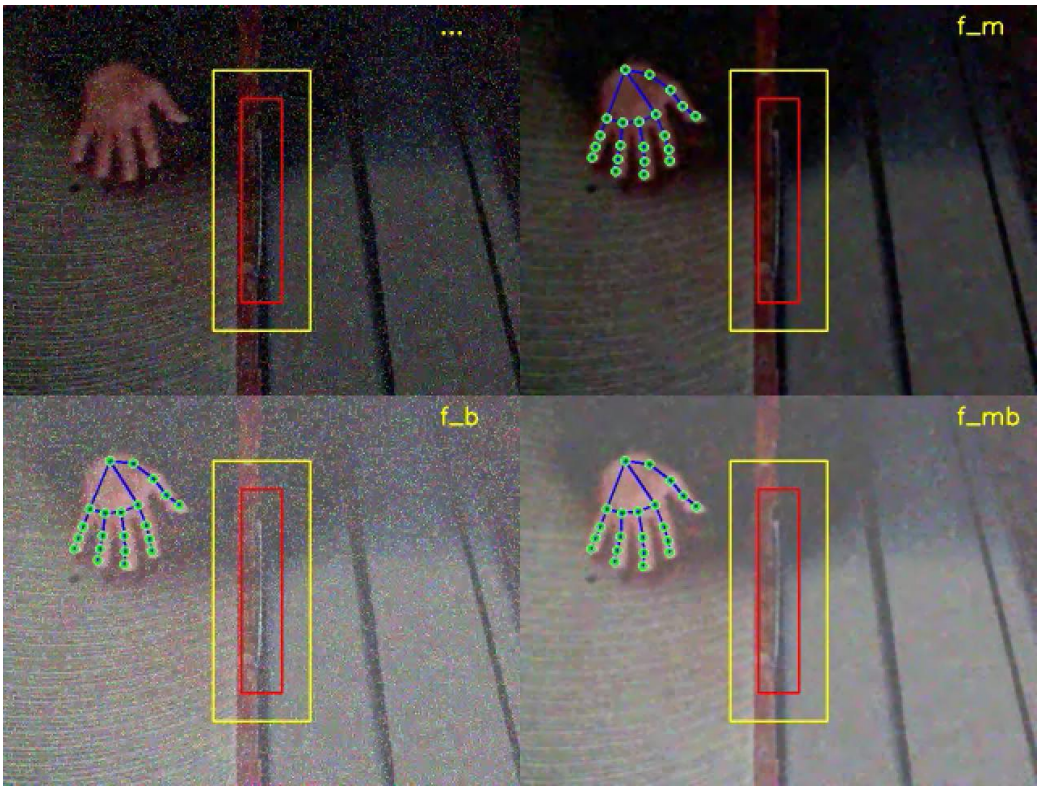
Çizelge 3.2'de gürültülü ortam ve farklı parlaklık seviyelerinde yapılan deneysel çalışmada ise tespit performansında düşüşlerin olduğu, bu düşüşleri medyan ve parlaklık filtresi ile belirli oranda iyileştirildiği el tespit oranlarındaki değişim ile görülmektedir. Deneyin kıyaslanabilir ve tekrarlanabilir olması için beş farklı rasgele gürültü her bir parlaklık seviyesine eklenerek filtre etkisi incelenmiş ve ortalama değerler Çizelge 3.2'de sunulmuştur. Burada %10 gürültünün MediaPipe Hands modeli performansını nasıl düşürdüğü gözlemlenmektedir. Sadece parlaklık dengeleme filtresi gürültüden dolayı çok başarılı olamadığı, filtresiz duruma göre sadece medyan filtresi tespit sonuçlarını iyileştirdiği görülmektedir. Parlaklık dengeleme filtresinin ise bazı durumlarda tespit sayısında çok düşük oranda artışı sağladığı gözlemlenmektedir. İşlem sürelerinin de gürültünün etkisi ile birkaç saniye artış gösterdiği değerlendirilmektedir. Not etmek gerekir ki model karmaşıklığına ek olarak parlaklık filtresi %4, medyan filtresi %11 ve her ikisi %18 işlem sürelerinde artış göstermekte olup mevcut durumu ile gerçek zaman çalışabilmektedir.

Çizelge 3.2 Gürültülü ortamda, farklı parlaklık koşullarında önerilen sistem el tespit performansı ve işlem süreleri

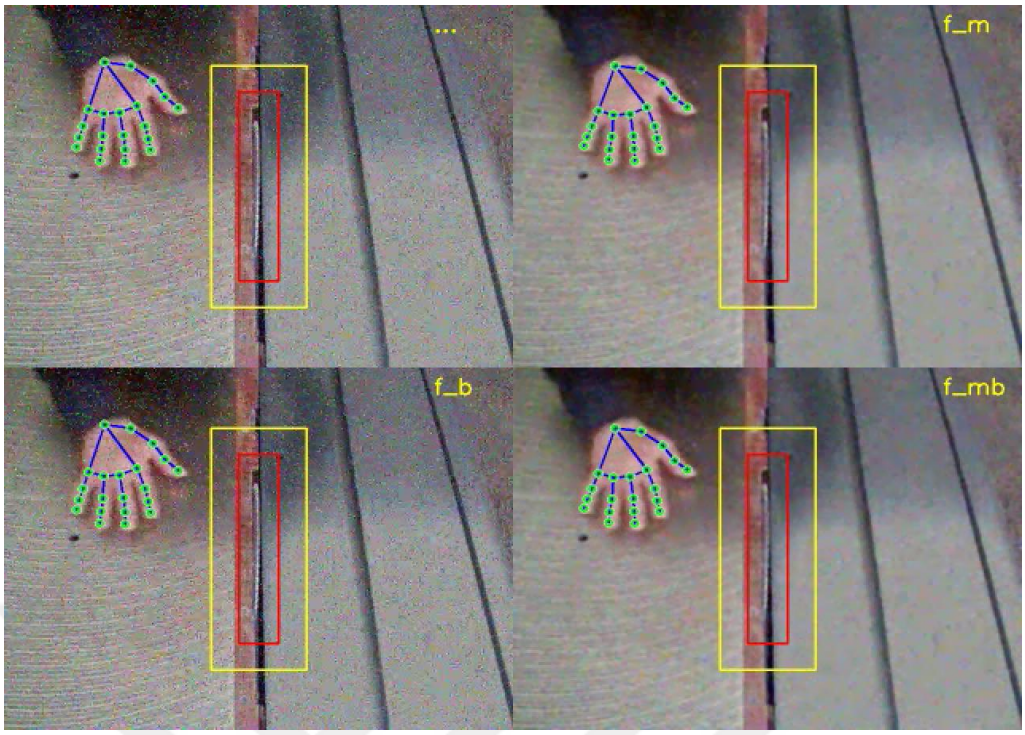
Parlaklık Oranı	Filtresiz	Parlaklık Filtresi	Medyan Filtresi	Parlaklık ve Medyan Filtreleri	İşlem Süresi (sn)
Çok Düşük (28)	%97,84	%97,93	%99,92	%99,97	(27,59 - 31,82)
Düşük (78)	%98,71	%98,71	%99,99	%100,00	(27,49 - 31,39)
Varsayılan (128)	%99,66	%99,93	%100,00	%100,00	(27,61 - 30,94)
Fazla (178)	%93,73	%98,53	%99,98	%100,00	(27,91 - 31,11)
Çok Fazla (228)	%47,83	%75,57	%90,86	%96,33	(27,99 - 31,37)

Gürültülü ortamda çalışan tespit sisteminin el görüntüleri Şekil 3.8-3.10'da verilmekte olup tuz ve biber gürültü etkisi görünmektedir. Burada sırasıyla çok az, varsayılan ve çok fazla parlaklık durumları verilmektedir. Her bir şekilde filtresiz, parlaklık, medyan ve medyan-parlaklık filtre kullanım performansı kıyaslanmakta olup önerilen filtreler ile tespit sisteminin

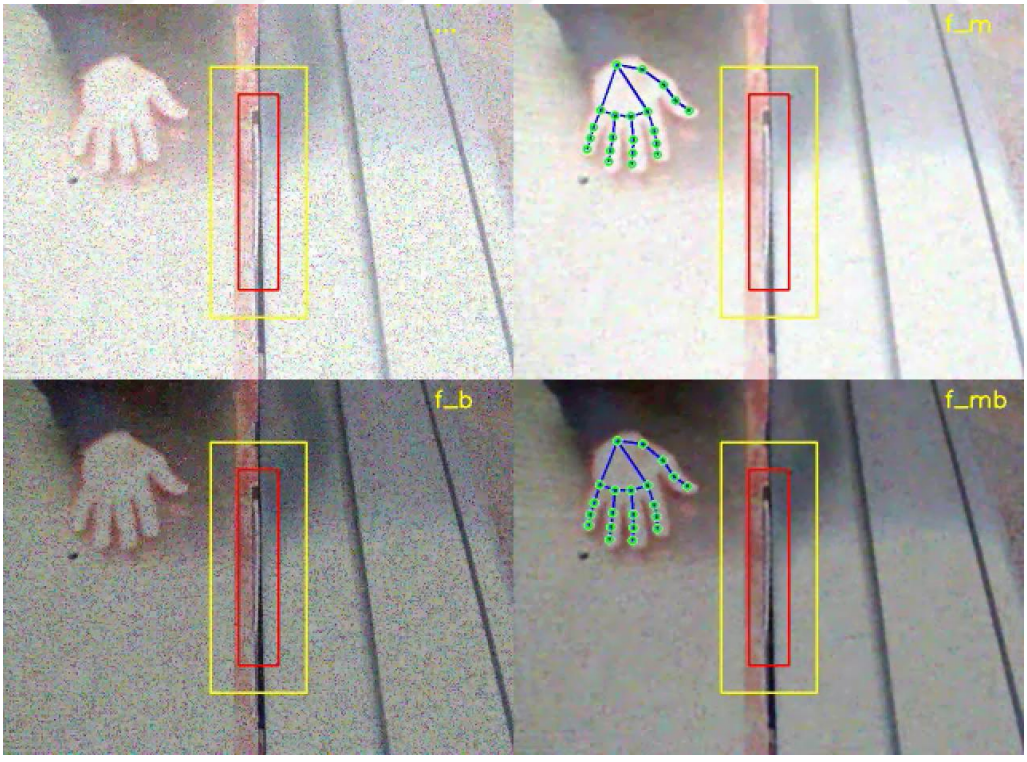
tuz ve biber gürültü etkisi azalttığı görülmektedir. Tespit sisteminin performans iyileştirmesindeki etkileriyle el pozisyon değişimi ve hızı kestirimi ile uyarı ve durdurma sisteminin çalışması tuz ve biber gürültü etkisi altında bile çalışabilmektedir. Bu durumda tespit sisteminin yakalanan el pozisyon değişimi ve hızı bulunmakta olup uyarı ve durdurma sistemi etkin bir şekilde çalıştırılmaktadır.



Şekil 3.8 Gürültülü ortam çok düşük parlaklık el tespit sonuçları (filtresiz, f_b parlaklık, f_m medyan, f_mb medyan-parlaklık filtreleri)

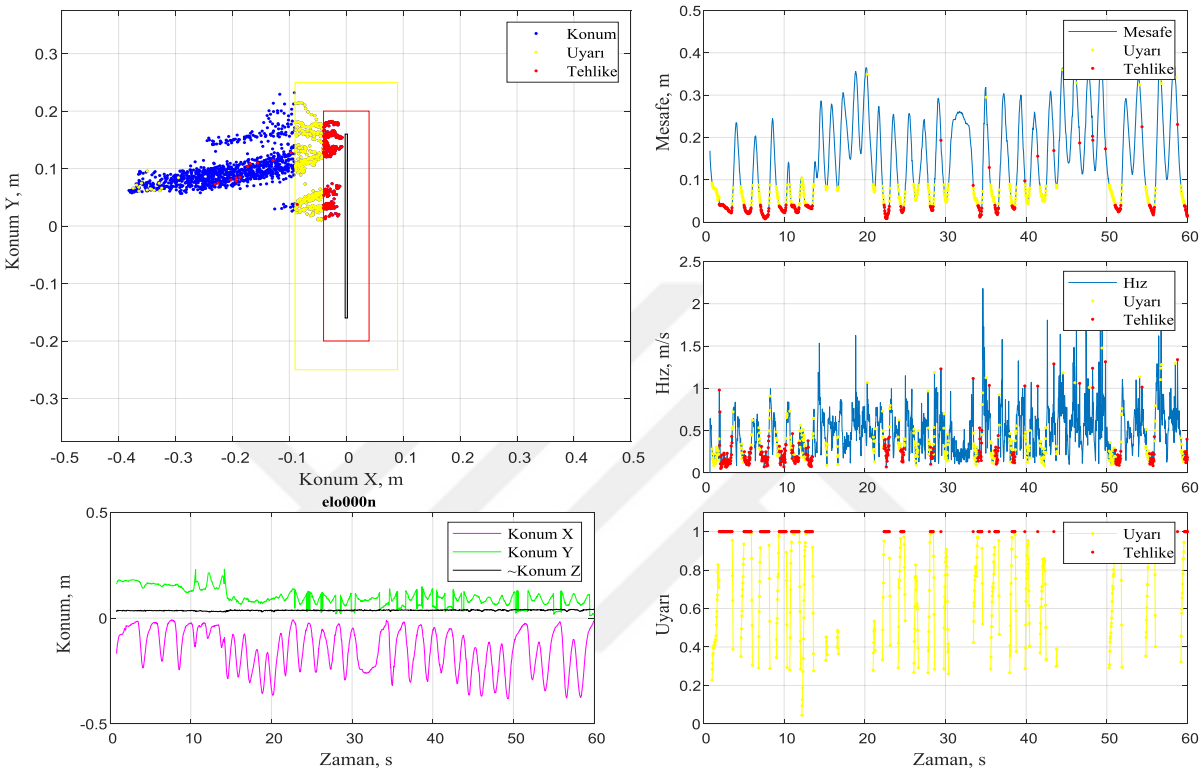


Şekil 3.9 Gürültülü ortam varsayılan parlaklık el tespit sonuçları (filtresiz, f_b parlaklık, f_m medyan, f_mb medyan-parlaklık filtreleri)

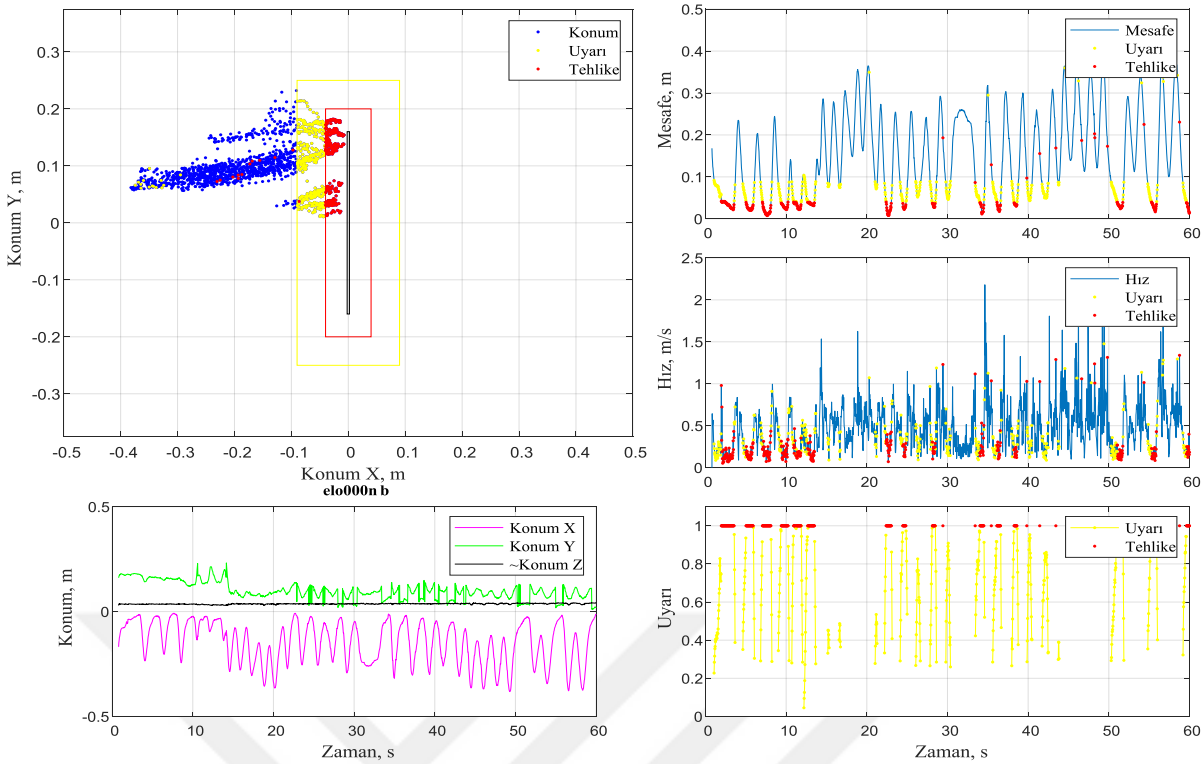


Şekil 3.10 Gürültülü ortam çok fazla parlaklık el tespit sonuçları (filtresiz, f_b parlaklık, f_m medyan, f_mb medyan-parlaklık filtreleri)

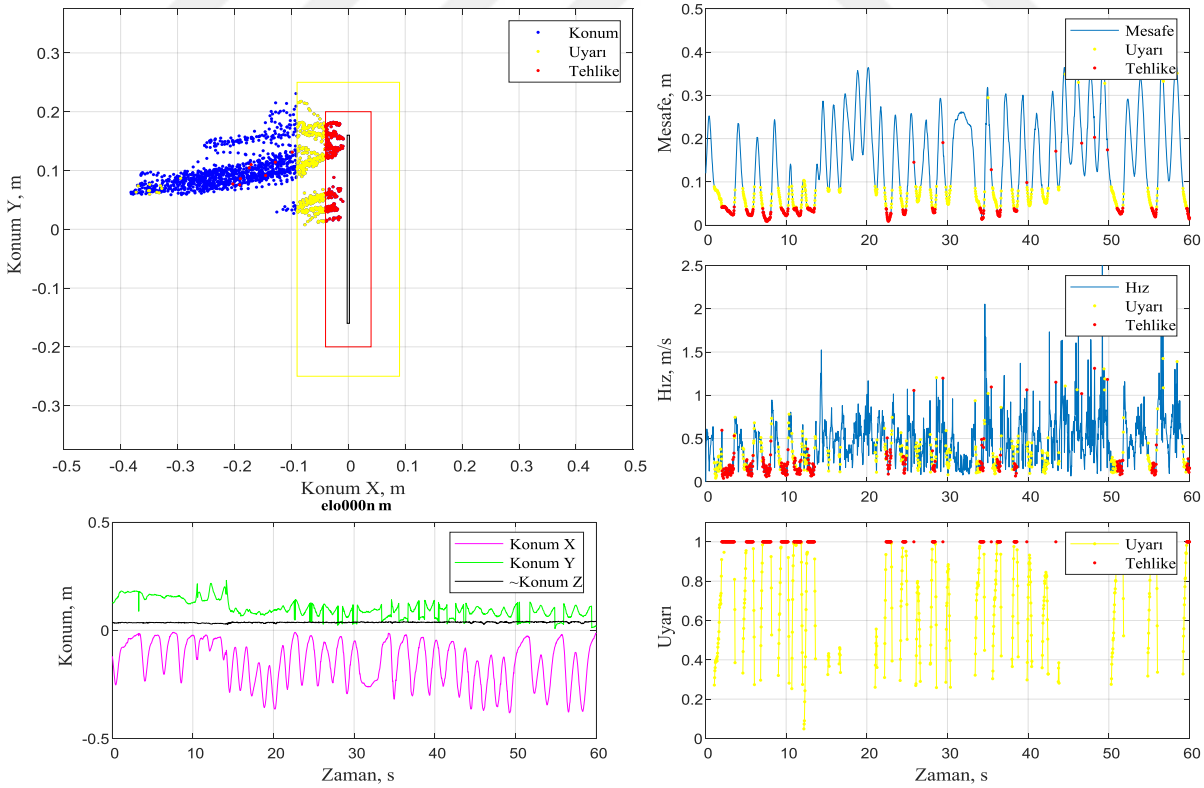
Tespit sisteminin performans iyileştirmesindeki etkileriyle el pozisyon değişimi ve hızı kestirimi ile uyarı ve durdurma sisteminin çalışması tuz ve biber gürültü etkisi altında bile çalışabilmektedir. Bu durumda tespit sisteminin yakalanan el pozisyon değişimi ve hızı bulunmakta olup uyarı ve durdurma sistemi etkin bir şekilde çalıştırılma sonuçları Şekil 3.11-3.14'de verilmektedir.



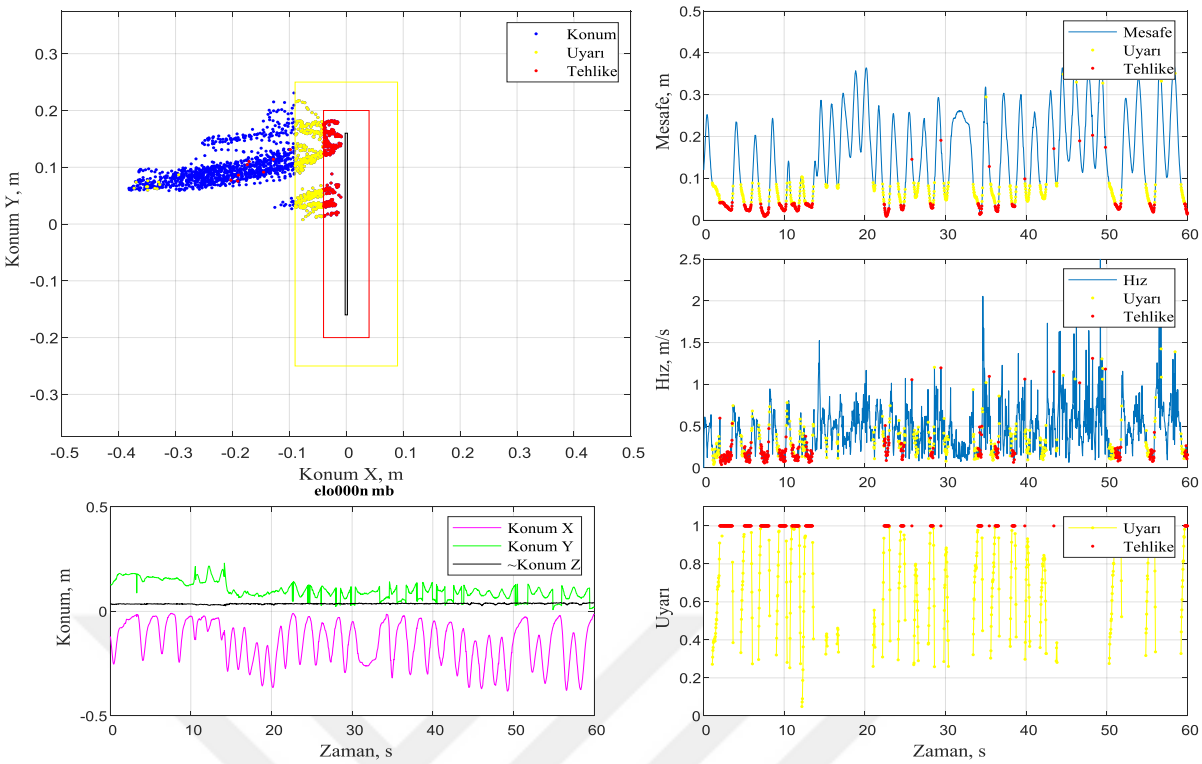
Şekil 3.11 Gürültülü ve varsayılan parlaklıkta filtresiz el tespit sonuçları; konum, mesafe, hız, uyarı ve konum değişimi



Şekil 3.12 Gürültülü ve varsayılan parlaklıkta parlaklık filtresi el tespit sonuçları; konum, mesafe, hız, uyarı ve konum değişimi



Şekil 3.13 Gürültülü ve varsayılan parlaklıkta medyan filtresi el tespit sonuçları; konum, mesafe, hız, uyarı ve konum değişimi



Şekil 3.14 Gürültülü ve varsayılan parlaklıkta medyan-parlaklık filtresi el tespit sonuçları; konum, mesafe, hız, uyarı ve konum değişimi

Kullanılan MediaPipe Hands modeli önerilen iyileştirme filtreleri ile birlikte bu çalışmada hedeflenmektedir. Her ne kadar farklı el tespit modelleme çalışmaları bununsa da burada beş farklı parlaklık seviyesinde gürültü etkisi ile birlikle filtreleme etkisi değerlendirilmektedir. Ek olarak mevcut donanım ile gerçek zaman çalışabilme durumu incelenmektedir. Buna göre model karmaşıklığına ek olarak önerilen filtrelerinin Çizelge 3.1-3.2'de verildiği gibi işlem sürelerini %4 ile %18 oranında artırmış olsa dahi gerçek zaman çalışabileceği görülmektedir.

Bu çalışmada önerilen MediaPipe Hands model güven puanı gürültü olmadığı durumda Çizelge 3.3'de ve gürültü durumunda Çizelge 3.4'de kıyaslanmaktadır. Yüzdesel güven puanı güven aralıkları ile birlikte değerlendirilirken beş farklı parlaklık seviyesinde filtre kullanım etkisi kıyaslanmaktadır. Burada model giriş parametreleri arasında yer alan el tespit/takip katsayılarının %50 olarak seçilmesi ile birlikte gürültü olamadığı durumda en düşük %73.81±0.63 güven seviyesinde el tespitinin ve takibinin yapıldığı görülmektedir. Normal parlaklık seviyelerinde ise %89 seviyelerine ulaşmaktadır. Gürültü etkisi ile birlikte %71.99±0.58 değerine kadar düşmekle birlikte parlaklık seviyesin bağlı olarak %80 değerine ulaşmaktadır. Filtrelerin etkisi ile güven puanının artışı genel olarak gözlemlenmektedir.

Çizelge 3.3 Gürültüsüz ortam MediaPipe Hands model güven puanı kıyaslaması

Parlaklık Oranı	Filtresiz	Parlaklık Filtresi	Medyan Filtresi	Parlaklık ve Medyan Filtreleri
Çok Düşük (28)	%83,68±0,55	%83,11±0,56	%85,45±0,50	%85,46±0,51
Düşük (78)	%87,37±0,46	%85,42±0,49	%89,24±0,38	%89,25±0,38
Varsayılan (128)	%87,61±0,46	%87,61±0,46	%89,39±0,38	%89,39±0,38
Fazla (178)	%86,44±0,51	%87,02±0,45	%87,98±0,45	%88,15±0,40
Çok Fazla (228)	%73,81±0,63	%76,33±0,53	%74,31±0,62	%75,78±0,53

Çizelge 3.4 Gürültü etkisi ile MediaPipe Hands model güven puanı kıyaslaması

Parlaklık Oranı	Filtresiz	Parlaklık Filtresi	Medyan Filtresi	Parlaklık ve Medyan Filtreleri
Çok Düşük (28)	%75,89±0,60	%71,99±0,58	%74,58±0,60	%73,58±0,60
Düşük (78)	%81,09±0,59	%76,94±0,62	%80,83±0,61	%80,16±0,61
Varsayılan (128)	%78,76±0,61	%78,76±0,61	%80,00±0,60	%80,00±0,60
Fazla (178)	%75,88±0,59	%76,73±0,59	%75,83±0,60	%78,73±0,61
Çok Fazla (228)	%73,89±0,60	%72,81±0,56	%72,89±0,56	%72,93±0,57

Deneysel çalışmada yer alan 60 sn uzunluğunda işlenen 1800 görüntüde tehlike ve uyarı bölgesi model el tespit sayıları hız aşımı durumu ile birlikte Çizelge 3.5'de verilmektedir. Parlaklık seviyelerinde değişim ile birlikte gürültü etkisi de kıyaslanmaktadır. Şekil 3.4-3.7'de bu sayıları konum bilgisi ile gözlemlemek mümkündür. Örneğin varsayılan parlaklık değerinde filtresiz durumda 319 tehlike bölgesinde 378 uyarı bölgesinde yer alırken gürültü etkisiyle 317 tehlike bölgesi 372 uyarı bölgesinde yer almaktadır. Ayrıca hız aşımı nedeni ile tehlikeli 1 görüntü bulunurken uyarı olması gereken 5 görüntü yer almaktadır ki gürültü etkisi ile bu durum 2 tehlike, 5 uyarı bölgesinde karşılaşılmaktadır. Parlaklık etkisi ile bu değerlerde sapma olduğununu da vurgulamak gerekir.

Çizelge 3.5 Tehlike ve uyarı bölgesi MediaPipe Hands model el tespit sayıları

Parlaklık Oranı	Filtresiz	Parlaklık Filtresi	Medyan Filtresi	Parlaklık ve Medyan Filtreleri
Çok Düşük (28)	318+1, 374+3 315+3, 367+6	316+1, 372+4 320+1, 372+3	318+1, 371+5 319+1, 376+2	318+1, 374+3 318+2, 376+4
Düşük (78)	319+1, 379+3 317+1, 370+6	318+1, 376+4 321+1, 369+5	318+2, 373+5 316+2, 370+4	318+2, 376+3 316+2, 371+3
Varsayılan (128)	319+1, 378+5 317+2, 372+5	319+1, 378+5 318+1, 372+5	318+2, 378+5 318+2, 373+5	318+2, 378+5 318+1, 374+5
Fazla (178)	317+1, 380+2 317+2, 367+5	322+2, 374+4 316+3, 368+4	318+1, 375+3 317+2, 370+4	323+3, 372+5 318+1, 371+5
Çok Fazla (228)	317+2, 376+5 287+4, 172+4	319+2, 374+6 311+5, 342+4	315+1, 379+3 314+4, 361+5	317+2, 373+4 318+2, 368+5

3.3 Uyarı ve Durdurma Mesafesinin Belirlenmesi

Gerçek zamanlı ve videodan veri işlemede, el tespit süresi; takip edilen el sayısına, gerçek ortam koşullarına, kayıt yapan kameranın, bilgisayarın ve tasarlanan yazılımın performansına bağlı olarak değişiklik göstermektedir. Yapılan deneyde tasarlanan örnek kodlama ile tespit edilen ellerin her frame için veri işlenme süresi bulunmuştur, bu değerler Çizelge 3.1 ve 3.2'de işlem süresi olarak gösterilmiştir.

Tasarımda durdurma bölgesi, daire testere kesicisinin bulunduğu tehlikeli bölgeyi temsil eder, elin bu bölgeye girme durumu hesaplanarak, erken uyarı ile makinanın durdurulması, sesli ve ışıklı vb. uyarı verilmesi düşünülmektedir, bu şekilde makina operatörünün uyarılması ve elinin korunması amaçlanır. Uyarı bölgesi ise ikinci derece tehlikeli bölgeyi temsil eder ve durdurma bölgesinden önce sesli ve ışıklı vb. uyarı verilerek makina operatörünün tehlikeli bölgeye yaklaşmadan uyarır, uyarı mesafesi durdurma mesafesinin iki katı kadar seçilebilir.

Tasarımada hıza bağlı durdurma şartı ellerden herhangi birisinin, noktasal olarak kesicinin yakınlarında, örneğin 1 m/sn hıza çıkması durumunda riskli kabul edilip, makinanın durdurulmasıdır. Burada ellerin saniyedeki azami hızının 1 m/sn olduğu kabul edilir.

El hızı 1 m/sn için durdurma sistemi 0.005-0.015 sn olması durumunda veri işlem süresi 0.015-0.030 sn seviyesinde olması 0.03-0.05 m tehlike bölgesi sınırını oluşturmaktadır. Bu süreler donanımın ve tasarımın performansına bağlı olarak kısalabilir veya uzayabilir. Elin kesiciye en yakın noktasının tehlikeli bölge mesafesi belirlenirken, sistemin veri işleme süresi dikkate alınır. Uyarı bölgesi sınırı ise tehlike bölgesi sınırının iki katı kadardır.

Örnek: Veri işleme süresi 0.025 sn, testere durdurma (gizleme) süresi 0.015 sn alınarak hıza bağlı durdurma mesafesi;

v_e = Elin hızı (Maksimun 1 m/s)
X_d = Testere güvenli yaklaşım mesafesi (m)
t_t = Testere toplam durdurma süresi ($\boldsymbol{t_d}$: testere durma süresi, $\boldsymbol{t_v}$: veri işleme çıktı süresi)

$t_t = t_d + t_v$
t_t = 0.015 + 0.025 = 0.040 sn bulunur.
$\boldsymbol{X_d = v_e * t_t}$
X_d = 1 * 0.040 = 0.04 m = 4 cm bulunur. (4 cm güvenli durma mesafesi, 8 cm uyarı mesafesi)

Elin kesiciye yaklaşma mesafesi 4 cm olduğunda sistem güvenli şekilde durmalıdır veya elin azami hızı 1 m/s aşarsa yine sistem durmalıdır ve el güvenli şekilde korunmalıdır. Uyarı bölgesi ise bu mesafenin iki katı 8 cm seçilebilir.

Örnek: Toplam durdurma süresi t_t = 0.050 sn alınarak hıza bağlı durdurma mesafesi;

$\boldsymbol{X_d = v_e * t_t}$
X_d = 1 * 0.050 = 0.05 m = 5 cm bulunur.
Güvenli durma mesafesi 5 cm, uyarı mesafesi 10 cm seçilmesi uygundur.

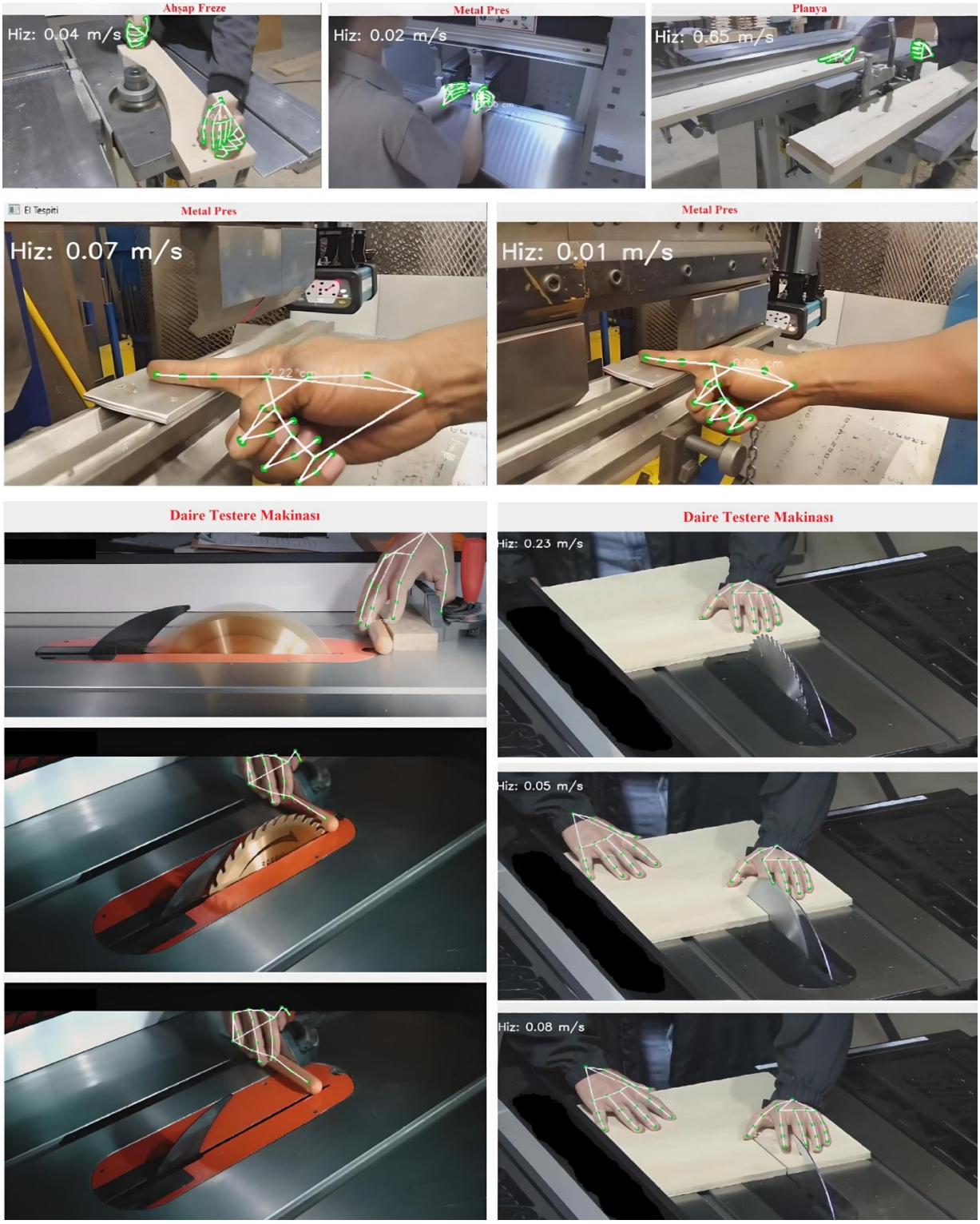
Örnek: Toplam durdurma süresi t_t = 0.050 sn için hız sınırını aşmadan tehlike bölgesine 0.5 m/s ve 0.25 m/s hızlarda giren el için algılama sonrası, elin kesiciye en yakın noktasının alacağı mesafe kaç cm'dir ?

0.5 m/s hız için; X_d = 0.5 * 0.050 = 0.025 m = 2.5 cm bulunur**.**
2.5 ≤ 5 cm olduğundan, el güveli şekilde korunur.

0.25 m/s hız için; X_d = 0.25 * 0.050 = 0.0125 m = 1.25 cm bulunur.
1.25 ≤ 5 cm olduğundan, el güveli şekilde korunur.

3.4 Örnek El Takip Uygulamaları

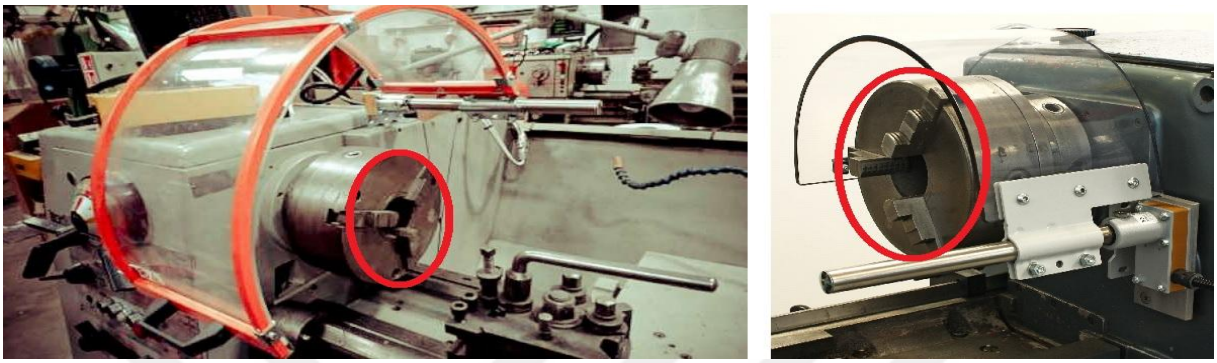
Tasarımı farklı olan ahşap, metal işleyen makinalara, tehlike arz eden çeşitli kesici, delici, sıkıştırıcı(pres), karıştırıcı, öğütücü makinalara bu sistemi uyarlamak mümkündür. Ancak bu sistemlerin Ar-Ge çalışmalarını yaparak, el korumasının geliştirilmesi gereklidir.



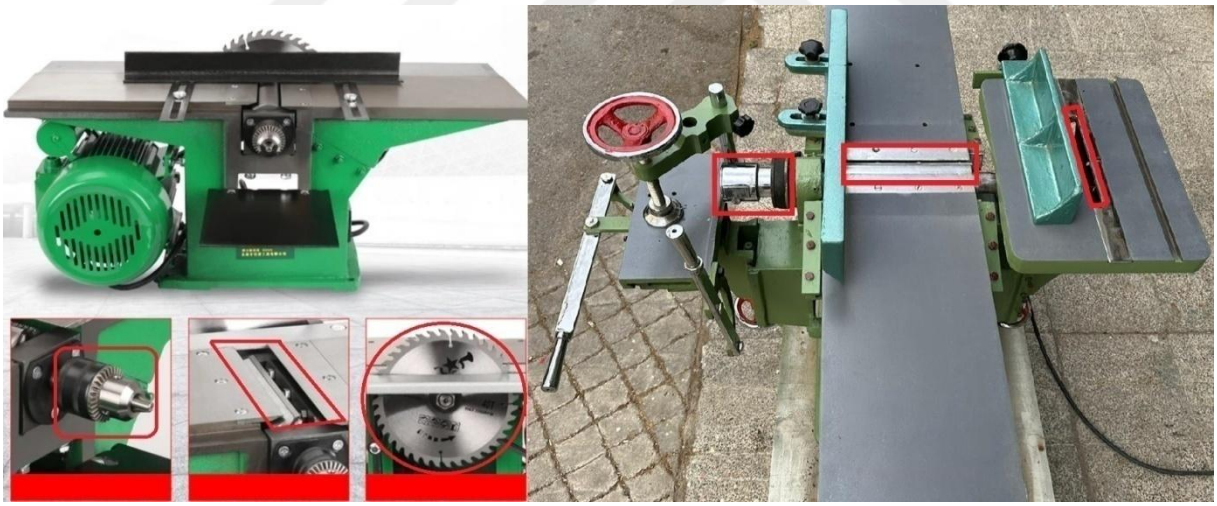
Şekil 3.15 Freze, pres, planya,daire testere el takip örnekleri

3.5 Örnek Tehlikeli Makina Bölgeleri

Şekillerdeki kırmızı bölgeler temsilidir, uygun kamera açısına ve kodlamadaki değişikliklere bağlı olarak, elin korunmasını istediğimiz tehlikeli bölgeyi belirlemek mümkündür, tasarım bu örnekler ile sınırlı değildir, benzer mantıkla tüm tehlike arz eden makinalara uygulanabilir ve el korunabilir.



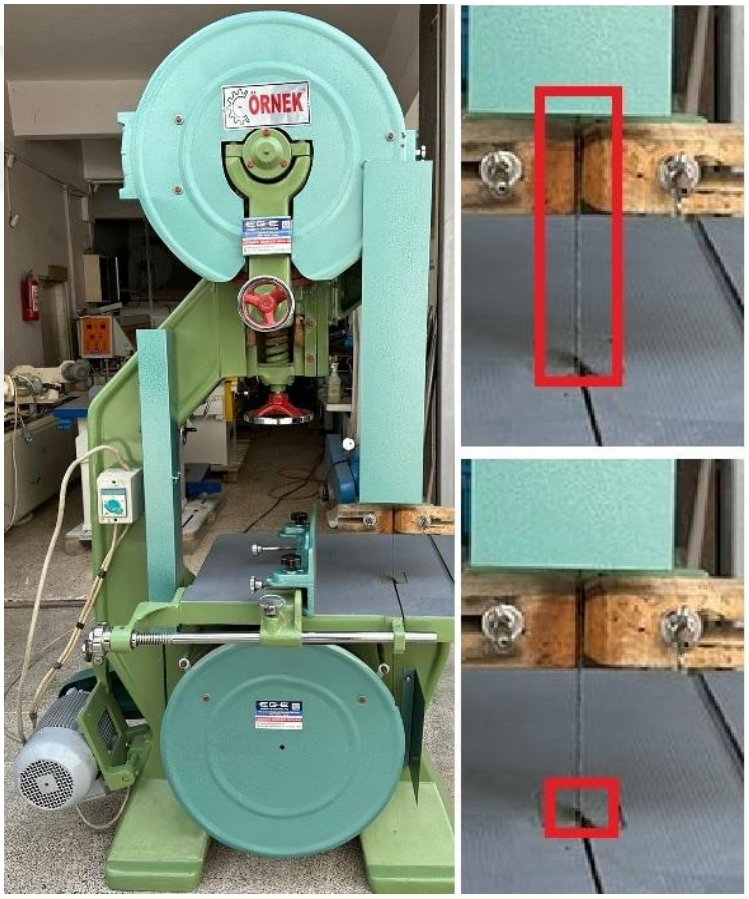
Şekil 3.16 Metal torna tezgahı örnek tehlikeli bölge



Şekil 3.17 Planya tezgahı örnek tehlikeli bölge

Şekil 3.18 Öğütücüler örnek tehlikeli bölge

Şekil 3.19 Dairesel ve dikdörtgen tehlikeli bölgeler freze, planya



Şekil 3.20 Şerit testere için tehlikeli kırmızı bölge

BÖLÜM 4

SONUÇ ve ÖNERİLER

Bu çalışmada görüntü işleme tabanlı el güvenlik sistemi yapay zeka algoritmaları ile tasarlanmış olup görüntü kalitesini etkileyen faktörler ile deneysel sonuçlar değerlendirilmiştir. Eğitilmiş MediaPipe Hands model kullanımı önerisiyle işçi sağlığı ve iş güvenliği alanında kazaları önlemek ve doğru çalışma davranışlarının sınırlarını belirlemeyi hedefleyen bu çalışma, parlaklık etkisinin ve gürültü azaltma önerileri ile iyileştirme sonuçlarını ortaya koymaktadır. El pozisyon tespiti ile sesli uyarı sistemi ve iş makinasını durdurma denetim sistemi bu çalışmada önerilmekte olup, tehlikeli iş makinaları ile çalışanların uyarı sistemi sayesinde eğitildiği de düşünülmektedir. Önerilen tasarım mevcut durumu ile gerçek zaman çalışabilmektedir. Çalışmada kullanılan kamera özelliklerine bağlı olarak model olumlu ve olumsuz ortam koşullarında değerlendirilmiş olup geliştirilen filtreler ile el tespit performansının büyük oranda iyileştirebildiği gözlemlenmiştir. Gömülü sistemlere uyumuyla MediaPipe Hands modeli yapay zeka algoritmalarının ve kameraların görüntü kalitesine bağlı olarak çok daha hızlı el tespit ve takip yapması beklenmektedir.

MediaPipe Hands modeline benzer şekilde, derin öğrenme yöntemleri ve makine öğrenimi ile bilgisayarlı görü yaparak tehlikeli nesneleri tanımlayabilen tehlikeli anlan dedektörü eğitilebilir. Tasarlanan sistem ilerleyen zamanda çoklu kameralar, akıllı gözlükler ile tehlikeli alanı otomatik tanıma ve koruma sistemleri geliştirilebilir. Akıllı gözlükler nesne tanıma işlemlerinde, iş ve trafik güvenliğinde kullanılabilir.

Tasarlanan her sistem gibi bu sisteminde cevap alamadığı kör noktalar olabilir, makinayı kullanan kişi kameranın önünü kapatacak şekilde çalışmalar yaparsa korumayı aksatabilir, bu durumda benzer veya farklı paralel koruma sistemleri birlikte çalışacak şekilde geliştirilerek koruma denetimi sağlanabilir. Tehlikeli alan zaten konum olarak belirli olduğundan, bu konuma yaklaşan kişinin elini birden fazla kamera, ileri teknoloji termal kamera veya akıllı gözlükler ile denetlemek çözüm olabilir.

KAYNAKLAR

[1] İSG 6331 Sayılı İş Sağlığı Ve Güvenliği Kanunu (2012) Resmi Gazete, Adres: URL <*https://www.resmigazete.gov.tr/eskiler/2012/06/20120630-1.htm*>, Ziyaret tarihi: 04.03.2024.

[2] Demir B ve Demir N (2016) Kamu sektöründe 6631 sayılı iş sağlığı güvenliği yasasının uygulanması ve mevcut yükümlülükler. *İstanbul Aydın Üniversitesi Dergisi*, 8(29): 167-194.

[3] Başaran G ve Cagıl G (2021) Koruyucu gözlük kullanımının görüntü işleme yöntemiyle tespit edilmesi. *El-Cezeri*, 9(1): 86-95.

[4] Köstekli C, Koçer H ve Altunok M (2023) Bilgisayar görüşü tabanlı masa başı çalışanlara yönelik medıapıpe ile postür analizi. *Mühendislikte, 1. Mühendislikte Yenilikçi Yaklaşımlar-2*

[5] Engür M O (2017) Mobilya endüstrisinde iş sağlığı ve güvenliği mevzuatına uyumun kontrol listeleri ile sağlanması üzerine bir çalışma. *Mühendislik Bilimleri ve Tasarım Dergisi*, 5: 283-292.

[6] İlhan A, Koşar G, Karapınar A ve Gedik T (2013) Sakarya ili mobilya imalatında iş kazası ve meslek hastalıklarının ortaya çıkış nedenlerinin analizi. *Kastamonu University Journal of Forestry Faculty*, 13(2): 202-210.

[7] Gedik T ve İlhan A (2014) Sakarya ili mobilya imalatçılarında iş sağlığı ve iş güvenliği üzerine bir inceleme. *Turkish Journal of Forestry*, 15(2): 123-129.

[8] Demirci S (2018) Mobilya imalatında kullanılan malzeme ve makinelerin iş sağlığı ve güvenliği yönünden değerlendirilmesi. *Hastane Öncesi Dergisi*, 3(2): 103-119.

[9] Aksoy E ve Keskin H (2019) Mobilya endüstrisinde iş sağlığı ve güvenliği ile ilgili risk değerlendirmesi: Begonya mobilya imalat işletmesi örneği. *Mobilya ve Ahşap Malzeme Araştırmaları Dergisi*, 2(1): 46-60.

[10] ÇSGB (2017) Mobilya Sektörü İş Sağlığı ve Güvenliği Yönetim Sistemi, T.C. Çalışma ve Sosyal Güvenlik Bakanlığı. Adres: URL <*https://www.csgb.gov.tr/Media/x5efu2g0/mobilya-sektörü-iş-sağlığı-ve-güvenliği-yönetim-sistemi-rehberi.pdf*>, Ziyaret tarihi: 02.12.2024.

[11] Smith T P (2011) Human factors evaluatıon of technology ıntended to address blade-contact ınjurıes wıth table saws.

[12] Ulusoy H, Atılgan A ve Peker H (2018) Mobilya endüstrisinde kullanılan makinelerde çalışma güvenliği. *Türk Bilimsel Derlemeler Dergisi*, 11(1): 70-81.

KAYNAKLAR (devam ediyor)

[13] Current, Richard S, Main, Bruce W, Main and McKinnon Main (2020) Saw safety risk in the real world. *Professional safety*, 65(11): 24-32.

[14] Chung, K C and Shauver, M J (2013) Table saw injuries: epidemiology and a proposal for preventive measures. *Plastic and reconstructive surgery*, 132(5): 777e-783e.

[15] SawStop (2021) Sawstop safety system ICS owner menual. Adres: URL <*https://www.sawstop.com/wp-content/uploads/2021/10/M_ICS_WEB.pdf*>, Ziyaret tarihi: 02.12.2024.

[16] Hand Guard (2022) Altendorf sliding table saws Hand Guard the safety system. Adres: URL <*https://www.altendorfgroup.com/en/machines/altendorf-hand-guard/*>, Ziyaret tarihi: 02.12.2024.

[17] Tan F G, Yüksel A S, Aydemir E ve Ersoy M (2021) Derin Öğrenme teknikleri ile nesne tespiti ve takibi üzerine bir inceleme. *Avrupa Bilim ve Teknoloji Dergisi*, (25): 159-171.

[18] Özdemir D B ve Kınacı A C (2018) Görüntülerdeki araba nesnelerinin belirlenmesi için derin öğrenme ile bir model eğitilmesi, *20. Akademik Bilişim Konferansı*.

[19] Daş R, Polat B ve Tuna G (2019) Derin öğrenme ile resim ve videolarda nesnelerin tanınması ve takibi. *Fırat Üniversitesi Mühendislik Bilimleri Dergisi*, 31(2): 571-581.

[20] Sarkar D, Bali R and Ghosh T (2018) Hands-on transfer learning with Python: ımplement advanced deep learning and neural network models using TensorFlow and Keras. *Packt Publishing Ltd.*

[21] MediaPipe (2023) MediaPipe solutions guide. Adres: URL <*https://ai.google.dev/edge/mediapipe/solutions/guide*>, Ziyaret tarihi: 02.12.2024.

[22] Yıldırım B ve Cagıl G (2020) Bir montaj parçasının derin öğrenme ve görüntü işleme ile tespiti. *Journal of Intelligent Systems: Theory and Applications*, 3(2): 31-37.

[23] Garg S, Saxena A and Gupta R (2023) yoga pose classification: a CNN and Mediapipe inspired deep learning approach for real-world application. *Journal of Ambient Intelligence and Humanized Computing*, 14(12): 16551-16562.

[24] Sánchez-Brizuela G, Cisnal A, de la Fuente-López E, Fraile J C and Pérez-Turiel J (2023) Lightweight real-time hand segmentation leveraging Mediapipe landmark detection. *Virtual Reality*, 27(4): 3125-3132.

[25] Latreche A, Kelaiaia R, Chemori A and Kerboua A (2023) Reliability and validity analysis of mediapipe-based measurement system for some human rehabilitation motions. *Measurement*, 214: 112826.

KAYNAKLAR (devam ediyor)

[26] Çelik Ö ve Odabaş A (2020) Sign2Text: Konvolüsyonel sinir ağları kullanarak türk işaret dili tanıma. *Avrupa Bilim ve Teknoloji Dergisi*, (18): 923-934.

[27] Bora, Jyotishman, Dehingia S, Boruah A, Chetia, Anuj A and Gogoi D (2023) Real-time assamese sign language recognition using Mediapipe and deep learning. *Procedia Computer Science*, 218: 1384-1393.

[28] Kumar R, Bajpai A and Sinha A (2023) Mediapipe and CNNs for real-time ASL gesture recognition. *arXiv preprint arXiv:2305.05296*.

[29] Haji Mohd M N, Mohd Asaari M S, Ping O L and Rosdi B A (2023) Vision-based hand detection and tracking using fusion of kernelized correlation filter and single-shot detection. *Applied Sciences*, 13(13): 7433.

[30] Amprimo G, Masi G, Pettiti G, Olmo G, Priano L and Ferraris C (2024) Hand tracking for clinical applications: validation of the Google MediaPipe Hand (GMH) and the depth-enhanced GMH-D frameworks. *Biomedical Signal Processing and Control*, 96: 106508.

[31] Erkan U ve Gökrem L (2017) Tuz-biber gürültüsünde tekrarsız medyan filtre. *Gaziosmanpaşa Bilimsel Araştırma Dergisi*, 6(2): 11-19.

[32] Küpeli C ve Bulut F (2020) "Görüntüdeki tuz biber ve gauss gürültülerine karşı filtrelerin performans analizleri." *Haliç Üniversitesi Fen Bilimleri Dergisi*, 3(2): 211-239.

[33] Erkan U, Enginoğlu S, Thanh D N and Hieu L M (2020) Adaptive frequency median filter for the salt and pepper denoising problem. *IET Image Processing*, 14(7): 1291-1302.

[34] Ghanbari S, Ashtyani Z P and Masouleh M T (2022) User identification based on hand geometrical biometrics using Media-Pipe. In *2022 30th International Conference on Electrical Engineering (ICEE)*, (pp. 373-378). IEEE.

[35] Lin Y, Jiao X and Zhao L (2023) Detection of 3d human posture based on improved MediaPipe. *Journal of Computer and Communications*, 11(2); 102-121.

[36] Kim J W, Choi J Y, Ha E J and Choi J H (2023) Human pose estimation using mediapipe pose and optimization method based on a humanoid model. *Applied sciences*, 13(4); *2700.*

[37] Yüksel S ve Hacıoğlu R (2024) Görüntü işleme tabanlı MediaPipe algoritması ile tehlikeli iş makinalarında el kazalarını önleme. *Zonguldak Bülent Ecevit Üniversitesi, 100. Yıl Mühendislik Kongresi* (pp. 7).

[38] Yüksel S ve Hacıoğlu R (2024) Görüntü işleme tabanlı MediaPipe algoritması ile tehlikeli iş makinalarında el kazalarını önleme. *Karaelmas Fen ve Mühendislik Dergisi,* (Kabul Edildi)

EK AÇIKLAMALAR

EK A: Gerçek Zamanlı Çalışan El Koruması Güvenlik Tasarımı Python Kodu

```
import cv2
import mediapipe as mp
import numpy as np
import matplotlib.pyplot as plt
from collections import deque
import math
import time
import threading
import pyaudio

# Kamera çözünürlüğü
H = 640
V = 480

# Makina tablası santim cinsinden
X = 100
Y = 75

# Medyan filtresini açma/kapama değişkeni
median_filter_on = 0  # 1: Açık, 0: Kapalı

# Parlaklık normalleştirme açma/kapama değişkeni
brightness_filter_on = 0  # 1: Açık, 0: Kapalı

# Medyan filtresini uygulama
def apply_median_filter(frame, kernel_size=5):
    if median_filter_on:
        return cv2.medianBlur(frame, kernel_size)
    return frame

# Parlaklık normalleştirme
def normalize_brightness(frame, target_brightness=128):
    if brightness_filter_on:
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        mean_brightness = cv2.mean(gray_frame)[0]
        adjustment_factor = target_brightness /
mean_brightness
        adjusted_frame = cv2.convertScaleAbs(frame,
alpha=adjustment_factor, beta=0)
        return adjusted_frame
    return frame
```

```
# MediaPipe Hands modelini yükleme
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(max_num_hands=4,
min_detection_confidence=0.5, min_tracking_confidence=0.5)

# Kamerayı başlat
webcam = cv2.VideoCapture(0)

# Kameranın gördüğü alanın piksel başına santimetre cinsinden
değerleri
pixel_per_cm_x = H / X  # 640 piksel genişlik, 100 cm genişlik
pixel_per_cm_y = V / Y  # 480 piksel yükseklik, 75 cm
yükseklik

# Ses çalma fonksiyonu
def play_sound(frequency, volume):
    p = pyaudio.PyAudio()
    fs = 44100  # Ses örnekleme hızı
    duration = 5  # Sinyal süresi, 5 saniye
    samples = (np.sin(2 * np.pi * np.arange(fs * duration) *
frequency / fs)).astype(np.float32)
    stream = p.open(format=pyaudio.paFloat32,
                    channels=1,
                    rate=fs,
                    output=True)
    stream.write(volume * samples)
    stream.stop_stream()
    stream.close()
    p.terminate()

# Alarm fonksiyonunu tanımlama
def activate_alarm(distance, speed, distance_threshold_1,
distance_threshold_2, speed_threshold):
    global alarm_active

    # Hız 1 m/s'yi geçtiğinde alarm çal
    if speed > speed_threshold:  # Hız eşiği m/s'yi geçerse
        frequency = 500  # Hız aşımı için 500 Hz frekansta
alarm
        volume = 1.0  # Alarmın sesi yüksek
    elif distance <= distance_threshold_1:
        frequency = 500  # 0 cm için 500 Hz alarm
        volume = 1.0
    elif distance <= distance_threshold_2:
        frequency = 3000  # 10 cm için 3000 Hz alarm
        volume = 0.1
    else:
        frequency = None  # Alarmı durdur

    # Alarm aktif değilse ve bir alarm frekansı belirlenmişse,
alarmı etkinleştir
```

```
    if not alarm_active and frequency:
        sound_thread = threading.Thread(target=play_sound,
args=(frequency, volume))
        sound_thread.start()
        alarm_active = True
    # Alarm durdurulacaksa (mesafe veya hız tekrar eşikleri
geçerse)
    elif alarm_active and (distance > distance_threshold_2 or
speed <= speed_threshold):
        alarm_active = False

# Dosyayı yazma modunda açma
with open("el_konumu.txt", "w") as file:
    # Mesafe ve hız verilerini saklamak için deque kullanarak
boş liste
    mesafe_data = deque(maxlen=60) # 60 deque değeri
ax2.set_xlim değeri ile ölçüm hızına göre değişir.
    hiz_data = deque(maxlen=60)

    # Mesafe ve hız grafiği için iki subplot oluştur
    fig, (ax1, ax2) = plt.subplots(2, 1)
    plt.ion()  # Etkileşimli modu etkinleştir
    fig.tight_layout(pad=3.0)
    fig.suptitle('El Mesafesi ve Hız Grafiği')
    ax1.set_ylabel('Mesafe (cm)')
    ax2.set_ylabel('Hız (cm/s)')
    ax2.set_xlabel('Zaman')

    # Önceki karenin el konumu
    prev_point = None
    prev_landmarks = []
    prev_distances = []  # Mesafeleri tutan liste
    # Kareler arası bekleme zamanı yok
    delay_counter = 0

    # Alarmın sınırlar normale döndüğünde susması için alarm
etkinlik durumu
    alarm_active = False

    # Kullanıcı tarafından belirlenen mesafe ve hız eşikleri
(cm ve m/s cinsinden)
    distance_threshold_1 = 0  # 0 cm
    distance_threshold_2 = 4  # 4 cm
    speed_threshold = 1  # Hız eşik değeri

    # Daire testere dikdörtgen tehlikeli alan hesaplamaları
    height, width = V, H  # Kameranın çözünürlüğü
    R = 5  # Örnek daire testere yüksekliği yarıçap yüksekliği
    C = -math.radians(90)  # Dereceyi radyana çevirme

    rect_width = int(pixel_per_cm_x * (6.25 + R *
```

```
abs(math.cos(C))))  # Dikdörtgen genişliğini özel çarpan ve
açıya göre ayarlama
    rect_height = int(pixel_per_cm_y * (20 + R * 2))  #
Dikdörtgen yüksekliğini özel çarpana göre ayarlama

    x_center = int(H/2) - int(R * pixel_per_cm_x * math.cos(C)
/ (100 / 50))  # Örnek olarak dikdörtgen x merkezi ve kaydırma
ayarı
    y_center = int(V/2)  # Örnek olarak y merkezi

    rect_x = x_center - rect_width // 2  # Dikdörtgenin sol
üst köşesinin x koordinatı
    rect_y = y_center - rect_height // 2  # Dikdörtgenin sol
üst köşesinin y koordinatı

    while True:
        start_time = time.perf_counter()  # Döngünün başında
işlem başlangıç zamanını al

        # Kamera görüntüsünü oku
        ret, frame = webcam.read()

        # Median filtresi uygula
        median_frame = apply_median_filter(frame)

        # Parlaklık normalleştirme
        adjusted_frame = normalize_brightness(median_frame)

        # Ekranın istenilen bir noktasına bir dikdörtgen çizme
        cv2.rectangle(adjusted_frame, (rect_x, rect_y),
(rect_x + rect_width, rect_y + rect_height), (0, 0, 255), 5)

        # Görüntüyü RGB formatına dönüştürme (MediaPipe, RGB
formatını bekler)
        frame_rgb = cv2.cvtColor(adjusted_frame,
cv2.COLOR_BGR2RGB)

        # MediaPipe Hands modelini kullanarak el tespiti yap
        results = hands.process(frame_rgb)

        # En yakın el noktasının dikdörtgene olan mesafesini
sakla
        min_distance = float('inf')

        # En yakın nokta
        nearest_point = None

        # El tespiti sonuçlarını işle
        if results.multi_hand_landmarks:
            for hand_landmarks in
results.multi_hand_landmarks:
```

```
                # El noktalarının piksel koordinatlarını
içeren bir liste oluştur
                hand_points = [(int(landmark.x * width),
int(landmark.y * height)) for landmark in
                               hand_landmarks.landmark]

                # Elin boyutunu kontrol et
                bounding_box =
cv2.boundingRect(np.array(hand_points))
                bounding_box_width, bounding_box_height =
bounding_box[2], bounding_box[3]

                Ze = 20  # Elin X ve Y ekseninde uzunluğu 20
cm, Z ekseni için kullanılır.
                Zk = 1.5   # Elin Z eksenindeki güvenli
yüksekliği, kesici yüksekliği 30 Cm.
                if bounding_box_width > (Ze * Zk *
pixel_per_cm_x) or bounding_box_height > (Ze * Zk *
pixel_per_cm_y):
                    continue

                for landmark in hand_landmarks.landmark:
                    # Her bir elin tespit edilen noktalarını
işle
                    x = int(landmark.x * width)
                    y = int(landmark.y * height)

                    # El noktalarını çiz
                    cv2.circle(adjusted_frame, (x, y), 5, (0,
255, 0), -1)

                    T = 4  # Elin kırmızı dikdörtgene olan
teğet ön durdurma mesafesi
                    # Elin kırmızı dikdörtgene olan mesafesini
hesapla (Ekstra yaklaşma mesafesi ekle)
                    dist_x = max(0,abs(x - (rect_x +
rect_width // 2)) - rect_width // 2 - int((T + 3) *
pixel_per_cm_x))
                    dist_y = max(0, abs(y - (rect_y +
rect_height // 2)) - rect_height // 2 - int((T + 3) *
pixel_per_cm_y))
                    distance = (dist_x ** 2 + dist_y ** 2) **
0.5

                    # En küçük mesafeyi ve en yakın noktayı
güncelle
                    if distance < min_distance:
                        min_distance = distance
                        nearest_point = (x, y)

                # El noktaları arasında çizgileri çiz
```

```
                connections = mp_hands.HAND_CONNECTIONS
                for connection in connections:
                    point1 = connection[0]
                    point2 = connection[1]
                    cv2.line(adjusted_frame,
hand_points[point1], hand_points[point2], (255, 255, 255), 2)

        # Elde edilen mesafeyi ve süreyi dosyaya yaz
        if nearest_point:
            current_time = time.perf_counter()  # Zaman
damgasını al
            elapsed_time = current_time - start_time  # Zaman
aralığını hesapla
            file.write(f"{min_distance / pixel_per_cm_x:.2f},
{elapsed_time:.2f} seconds\n")

        # El konumunu en yakın noktanın üzerine yaz
        if nearest_point:
            cv2.putText(adjusted_frame, f"{min_distance /
pixel_per_cm_x:.2f} cm", nearest_point,
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,
255, 255), 1, cv2.LINE_AA)

            # El Hız hesaplama işlemleri
            if results.multi_hand_landmarks and delay_counter
== 0:
                max_speed = 0
                curr_distances = []
                # Takip etmek istediğimiz noktalar (0, 4, 8,
9, 12, 16, 20 numaralı noktalar)
                points_to_track = [0, 4, 8, 9, 12, 16, 20]  #
Elin en aktif noktaları

                # Bu landmark noktaları için mesafe hesapla
                for hand_landmarks in
results.multi_hand_landmarks:
                    distances = []
                    for point in points_to_track:
                        landmark =
hand_landmarks.landmark[point]
                        x, y = int(landmark.x * width),
int(landmark.y * height)

                        # Normalizasyon: X (-4, -3, ..., 3, 4)
ve Y (-3, -2, ..., 2, 3) koordinat
                        normalized_x = (x - x_center) / (H /
2) * 4 # 4 normalizasyon değişkeni
                        normalized_y = (y - y_center) / (V /
2) * 3 # 3 normalizasyon değişkeni
                        # Merkeze olan mesafeyi hesaplama
(Öklid mesafesi)
```

```
                        distance_from_center =
math.sqrt(normalized_x ** 2 + normalized_y ** 2)

                        # Hız faktörünü hesapla: Merkeze doğru
gidildikçe hız üstel(exp) olarak artacak
                        speed_factor = math.exp(-
distance_from_center)
                        distances.append(distance_from_center)
                    curr_distances.append(distances)

                # Hız hesaplansın (Eğer önceki mesafeler varsa
hesaplansın)
                if prev_distances:
                    for curr, prev in zip(curr_distances,
prev_distances):
                        for curr_dist, prev_dist in zip(curr,
prev):
                            # Mesafe değişimini hesapla ve
hızı hesapla
                            distance_change = abs(curr_dist -
prev_dist)
                            if distance_change > 0:
                                speed = (distance_change /
elapsed_time) * speed_factor  # V=(d/t).sf
                                max_speed = max(max_speed,
speed)  # Maksimun hızı güncelle
                # Gerçek zamanlı çalışmadaki Hız (En yüksek
hızlı noktadır)
                real_speed_m_s = max_speed  # hız (m/s)
cinsinden

                # Hızı ekranda göster (metre/saniye olarak)
                cv2.putText(adjusted_frame, f"Hiz:
{real_speed_m_s:.2f} m/s", (10, 50), cv2.FONT_HERSHEY_SIMPLEX,
1,
                            (255, 255, 255), 2, cv2.LINE_AA)

                # Hızı kaydet
                hiz_data.append(real_speed_m_s)
                ax2.clear()
                ax2.plot(hiz_data, color='green')
                ax2.set_ylim(0, 1)
                ax2.set_xlim(0, 60) # 60 deque değeri değeri
ile kayıt hızına göre değişir.
                ax2.grid(True)
                ax2.set_ylabel('Hız (m/s)')
                ax2.set_xlabel('60 Saniye Ölçekli Zaman
Aralığı')

            # Alarmı tetikle
            activate_alarm(min_distance / pixel_per_cm_x,
```

```
real_speed_m_s, distance_threshold_1, distance_threshold_2,
speed_threshold)

            prev_distances = curr_distances

        if min_distance != float('inf'):
            mesafe_data.append(min_distance / pixel_per_cm_x)
            ax1.clear()
            ax1.plot(mesafe_data, color='blue')
            ax1.set_ylim(0, 50)
            ax1.set_xlim(0, 60)
            ax1.grid(True)
            ax1.set_ylabel('Mesafe (cm)')

        cv2.imshow('El Tespiti', adjusted_frame)
        plt.pause(0.001)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

# Kamera yakalama cihazını serbest bırak
webcam.release()

# Pencereleri kapat
cv2.destroyAllWindows()
```

EK B: Gerçek Zamanlı Dairesel Tehlikeli Alanlarda Güvenlik Tasarımı Python Kodu

```
import cv2
import mediapipe as mp
import numpy as np
import matplotlib.pyplot as plt
from collections import deque
import math
import time
import threading
import pyaudio

# Kamera çözünürlüğü
H = 640
V = 480
# Makina tablası santim cinsinden
X = 100
Y = 75

# Medyan filtresini açma/kapama değişkeni
median_filter_on = 0  # 1: Açık, 0: Kapalı
# Parlaklık normalleştirme açma/kapama değişkeni
brightness_filter_on = 0  # 1: Açık, 0: Kapalı
```

```
# Medyan filtresini uygulama
def apply_median_filter(frame, kernel_size=5):
    if median_filter_on:
        return cv2.medianBlur(frame, kernel_size)
    return frame

# Parlaklık normalleştirme
def normalize_brightness(frame, target_brightness=128):
    if brightness_filter_on:
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        mean_brightness = cv2.mean(gray_frame)[0]
        adjustment_factor = target_brightness /
mean_brightness
        adjusted_frame = cv2.convertScaleAbs(frame,
alpha=adjustment_factor, beta=0)
        return adjusted_frame
    return frame

# MediaPipe Hands modelini yükleme
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(max_num_hands=1,
min_detection_confidence=0.5, min_tracking_confidence=0.5)

# Kamerayı başlat
webcam = cv2.VideoCapture(0)

# Kameranın gördüğü alanın piksel başına santimetre cinsinden
değerleri
pixel_per_cm_x = H / X  # 640 piksel genişlik, 100 cm genişlik
pixel_per_cm_y = V / Y  # 480 piksel yükseklik, 75 cm
yükseklik

# Ses çalma fonksiyonu
def play_sound(frequency, volume):
    p = pyaudio.PyAudio()
    fs = 44100  # Ses örnekleme hızı
    duration = 5  # Sinyal süresi, 5 saniye
    samples = (np.sin(2 * np.pi * np.arange(fs * duration) *
frequency / fs)).astype(np.float32)
    stream = p.open(format=pyaudio.paFloat32,
                    channels=1,
                    rate=fs,
                    output=True)
    stream.write(volume * samples)
    stream.stop_stream()
    stream.close()
    p.terminate()

# Alarm fonksiyonunu tanımlama
def activate_alarm(distance, speed, distance_threshold_1,
```

```
distance_threshold_2, speed_threshold):
    global alarm_active

    # Hız 1 m/s'yi geçtiğinde alarm çal
    if speed > speed_threshold:  # Hız eşiği m/s'yi geçerse
        frequency = 500  # Hız aşımı için 500 Hz frekansta
alarm
        volume = 1.0  # Alarmın sesi yüksek
    elif distance <= distance_threshold_1:
        frequency = 500  # 0 cm için 500 Hz alarm
        volume = 1.0
    elif distance <= distance_threshold_2:
        frequency = 3000  # 10 cm için 3000 Hz alarm
        volume = 0.1
    else:
        frequency = None  # Alarmı durdur

    # Alarm aktif değilse ve bir alarm frekansı belirlenmişse,
alarmı etkinleştir
    if not alarm_active and frequency:
        sound_thread = threading.Thread(target=play_sound,
args=(frequency, volume))
        sound_thread.start()
        alarm_active = True
    # Alarm durdurulacaksa (mesafe veya hız tekrar eşikleri
geçerse)
    elif alarm_active and (distance > distance_threshold_2 or
speed <= speed_threshold):
        alarm_active = False

# Dosyayı yazma modunda açma
with open("el_konumu.txt", "w") as file:
    # Mesafe ve hız verilerini saklamak için deque kullanarak
boş liste
    mesafe_data = deque(maxlen=60) # 60 deque değeri
ax2.set_xlim değeri ile ölçüm hızına göre değişir.
    hiz_data = deque(maxlen=60)

    # Mesafe ve hız grafiği için iki subplot oluştur
    fig, (ax1, ax2) = plt.subplots(2, 1)
    plt.ion()  # Etkileşimli modu etkinleştir
    fig.tight_layout(pad=3.0)
    fig.suptitle('El Mesafesi ve Hız Grafiği')
    ax1.set_ylabel('Mesafe (cm)')
    ax2.set_ylabel('Hız (cm/s)')
    ax2.set_xlabel('Zaman')

    # Önceki karenin el konumu
    prev_point = None
    prev_landmarks = []
    prev_distances = []  # Mesafeleri tutan liste
```

```
    # Kareler arası bekleme zamanı yok
    delay_counter = 0

    # Alarmın sınırlar normale döndüğünde susması için alarm
etkinlik durumu
    alarm_active = False

    # Kullanıcı tarafından belirlenen mesafe ve hız eşikleri
(cm ve m/s cinsinden)
    distance_threshold_1 = 0  # 0 cm
    distance_threshold_2 = 4  # 4 cm
    speed_threshold = 1  # Hız eşik değeri

    # Daire tehlikeli alan hesaplamaları
    height, width = V, H  # Kameranın çözünürlüğü
    x_center = int(H/2)  # Örnek olarak 300 piksel
    y_center = int(V/2)  # Örnek olarak 150 piksel
    R = 10
    radius = int(R * pixel_per_cm_x)  # Çemberin yarıçapı
    color = (0, 0, 255)    # Çemberin çizileceği renk
    thickness = 3    # Çemberin kalınlığı

    while True:
        start_time = time.perf_counter()  # Döngünün başında
işlem başlangıç zamanını al

        # Kamera görüntüsünü oku
        ret, frame = webcam.read()

        # Median filtresi uygula
        median_frame = apply_median_filter(frame)

        # Parlaklık normalleştirme
        adjusted_frame = normalize_brightness(median_frame)

        # Çember halkasını çizme
        cv2.circle(adjusted_frame, (x_center, y_center),
radius, color, thickness)

        # Görüntüyü RGB formatına dönüştürme (MediaPipe, RGB
formatını bekler)
        frame_rgb = cv2.cvtColor(adjusted_frame,
cv2.COLOR_BGR2RGB)

        # MediaPipe Hands modelini kullanarak el tespiti yap
        results = hands.process(frame_rgb)

        # En yakın el noktasının dikdörtgene olan mesafesini
sakla
        min_distance = float('inf')
```

```
        # En yakın nokta
        nearest_point = None

        # El tespiti sonuçlarını işle
        if results.multi_hand_landmarks:
            for hand_landmarks in
results.multi_hand_landmarks:
                # El noktalarının piksel koordinatlarını
içeren bir liste oluştur
                hand_points = [(int(landmark.x * width),
int(landmark.y * height)) for landmark in
                               hand_landmarks.landmark]

                # Elin boyutunu kontrol et
                bounding_box =
cv2.boundingRect(np.array(hand_points))
                bounding_box_width, bounding_box_height =
bounding_box[2], bounding_box[3]

                Ze = 20  # Elin X ve Y ekseninde uzunluğu 20
cm, Z ekseni için kullanılır.
                Zk = 1.5   # Elin Z eksenindeki güvenli
yüksekliği, kesici yüksekliği 30 Cm.
                if bounding_box_width > (Ze * Zk *
pixel_per_cm_x) or bounding_box_height > (Ze * Zk *
pixel_per_cm_y):
                    continue

                for landmark in hand_landmarks.landmark:
                    # Her bir elin tespit edilen noktalarını
işle
                    x = int(landmark.x * width)
                    y = int(landmark.y * height)
                    # El noktalarını çiz
                    cv2.circle(adjusted_frame, (x, y), 5, (0,
255, 0), -1)

                    T = 4  # Elin kırmızı çembere olan teğet
ön durdurma mesafesi
                    # Elin çembere olan mesafesini hesaplama
(Merkeze olan mesafeden yarıçapı çıkararak)
                    distance_to_center = np.sqrt((x -
x_center) ** 2 + (y - y_center) ** 2)
                    distance = max(0, distance_to_center -
radius - ((T+2) * pixel_per_cm_x)) #Ekstra teğet ve yaklaşma
mesafesi

                    # En küçük mesafeyi ve en yakın noktayı
güncelle
                    if distance < min_distance:
                        min_distance = distance
```

```
                        nearest_point = (x, y)

                # El noktaları arasında çizgileri çiz
                connections = mp_hands.HAND_CONNECTIONS
                for connection in connections:
                    point1 = connection[0]
                    point2 = connection[1]
                    cv2.line(adjusted_frame,
hand_points[point1], hand_points[point2], (255, 255, 255), 2)

        # Elde edilen mesafeyi ve süreyi dosyaya yaz
        if nearest_point:
            current_time = time.perf_counter()  # Zaman
damgasını al
            elapsed_time = current_time - start_time  # Zaman
aralığını hesapla
            file.write(f"{min_distance / pixel_per_cm_x:.2f},
{elapsed_time:.2f} seconds\n")

        # El konumunu en yakın noktanın üzerine yaz
        if nearest_point:
            cv2.putText(adjusted_frame, f"{min_distance /
pixel_per_cm_x:.2f} cm", nearest_point,
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,
255, 255), 1, cv2.LINE_AA)

            # El Hız hesaplama işlemleri
            if results.multi_hand_landmarks and delay_counter
== 0:
                max_speed = 0
                curr_distances = []
                # Takip etmek istediğimiz noktalar (0, 4, 8,
9, 12, 16, 20 numaralı noktalar)
                points_to_track = [0, 4, 8, 9, 12, 16, 20]  #
Elin en aktif noktaları

                # Bu landmark noktaları için mesafe hesapla
                for hand_landmarks in
results.multi_hand_landmarks:
                    distances = []
                    for point in points_to_track:
                        landmark =
hand_landmarks.landmark[point]
                        x, y = int(landmark.x * width),
int(landmark.y * height)

                        # Normalizasyon: X (-4, -3, ..., 3, 4)
ve Y (-3, -2, ..., 2, 3) koordinat
                        normalized_x = (x - x_center) / (H /
2) * 4 # 4 normalizasyon değişkeni
                        normalized_y = (y - y_center) / (V /
```

```
2) * 3 # 3 normalizasyon değişkeni
                        # Merkeze olan mesafeyi hesaplama
(Öklid mesafesi)
                        distance_from_center =
math.sqrt(normalized_x ** 2 + normalized_y ** 2)

                        # Hız faktörünü hesapla: Merkeze doğru
gidildikçe hız üstel(exp) olarak artacak
                        speed_factor = math.exp(-
distance_from_center)
                        distances.append(distance_from_center)
                    curr_distances.append(distances)

                # Hız hesaplansın (Eğer önceki mesafeler varsa
hesaplansın)
                if prev_distances:
                    for curr, prev in zip(curr_distances,
prev_distances):
                        for curr_dist, prev_dist in zip(curr,
prev):
                            # Mesafe değişimini hesapla ve
hızı hesapla
                            distance_change = abs(curr_dist -
prev_dist)
                            if distance_change > 0:
                                speed = (distance_change /
elapsed_time) * speed_factor  # V=(d/t).sf
                                max_speed = max(max_speed,
speed)  # Maksimun hızı güncelle
                # Gerçek zamanlı çalışmadaki Hız (En yüksek
hızlı noktadır)
                real_speed_m_s = max_speed  # hız (m/s)
cinsinden

                # Hızı ekranda göster (metre/saniye olarak)
                cv2.putText(adjusted_frame, f"Hiz:
{real_speed_m_s:.2f} m/s", (10, 50), cv2.FONT_HERSHEY_SIMPLEX,
1,
                            (255, 255, 255), 2, cv2.LINE_AA)

                # Hızı kaydet
                hiz_data.append(real_speed_m_s)
                ax2.clear()
                ax2.plot(hiz_data, color='green')
                ax2.set_ylim(0, 1)
                ax2.set_xlim(0, 60) # 60 deque değeri değeri
ile kayıt hızına göre değişir.
                ax2.grid(True)
                ax2.set_ylabel('Hız (m/s)')
                ax2.set_xlabel('60 Saniye Ölçekli Zaman
Aralığı')
```

```
            # Alarmı tetikle
            activate_alarm(min_distance / pixel_per_cm_x,
real_speed_m_s, distance_threshold_1, distance_threshold_2,
speed_threshold)

            prev_distances = curr_distances

        if min_distance != float('inf'):
            mesafe_data.append(min_distance / pixel_per_cm_x)
            ax1.clear()
            ax1.plot(mesafe_data, color='blue')
            ax1.set_ylim(0, 50)
            ax1.set_xlim(0, 60)
            ax1.grid(True)
            ax1.set_ylabel('Mesafe (cm)')

        cv2.imshow('El Tespiti', adjusted_frame)
        plt.pause(0.001)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

# Kamera yakalama cihazını serbest bırak
webcam.release()

# Pencereleri kapat
cv2.destroyAllWindows()
```

EK C: Tekrarlanabilir Deneyler için Videodan Veri İşleme Tasarımı Python Kodu

```
import cv2
import mediapipe as mp
import numpy as np
import matplotlib.pyplot as plt
from collections import deque
import math
import time
import threading
import pyaudio

# Video çözünürlüğü
H = 640
V = 480
# Makina tablası santim cinsinden
X = 100
Y = 75

# Medyan filtresini açma/kapama değişkeni
median_filter_on = 0  # 1: Açık, 0: Kapalı
```

```
# Parlaklık normalleştirme açma/kapama değişkeni
brightness_filter_on = 0  # 1: Açık, 0: Kapalı

# Medyan filtresini uygulama
def apply_median_filter(frame, kernel_size=5):
    if median_filter_on:
        return cv2.medianBlur(frame, kernel_size)
    return frame

# Parlaklık normalleştirme
def normalize_brightness(frame, target_brightness=128):
    if brightness_filter_on:
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        mean_brightness = cv2.mean(gray_frame)[0]
        adjustment_factor = target_brightness /
mean_brightness
        adjusted_frame = cv2.convertScaleAbs(frame,
alpha=adjustment_factor, beta=0)
        return adjusted_frame
    return frame

# MediaPipe Hands modelini yükleme
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(max_num_hands=1,
min_detection_confidence=0.5, min_tracking_confidence=0.5)

# Video dosyasını oku
video_path = "video.mp4"  # Video dosyasının yolu
video = cv2.VideoCapture(video_path)

# Kameranın gördüğü alanın piksel başına santimetre cinsinden
değerleri
pixel_per_cm_x = H / X  # 640 piksel genişlik, 100 cm genişlik
pixel_per_cm_y = V / Y  # 480 piksel yükseklik, 75 cm
yükseklik

# Ses çalma fonksiyonu
def play_sound(frequency, volume):
    p = pyaudio.PyAudio()
    fs = 44100  # Ses örnekleme hızı
    duration = 5  # Sinyal süresi, 5 saniye
    samples = (np.sin(2 * np.pi * np.arange(fs * duration) *
frequency / fs)).astype(np.float32)
    stream = p.open(format=pyaudio.paFloat32,
                    channels=1,
                    rate=fs,
                    output=True)
    stream.write(volume * samples)
    stream.stop_stream()
    stream.close()
    p.terminate()
```

```
# Alarm fonksiyonunu tanımlama
def activate_alarm(distance, speed, distance_threshold_1,
distance_threshold_2, speed_threshold):
    global alarm_active

    # Hız 1 m/s'yi geçtiğinde alarm çal
    if speed > speed_threshold:  # Hız eşiği m/s'yi geçerse
        frequency = 500  # Hız aşımı için 500 Hz frekansta
alarm
        volume = 1.0  # Alarmın sesi yüksek
    elif distance <= distance_threshold_1:
        frequency = 500  # 0 cm için 500 Hz alarm
        volume = 1.0
    elif distance <= distance_threshold_2:
        frequency = 3000  # 10 cm için 3000 Hz alarm
        volume = 0.1
    else:
        frequency = None  # Alarmı durdur

    # Alarm aktif değilse ve bir alarm frekansı belirlenmişse,
alarmı etkinleştir
    if not alarm_active and frequency:
        sound_thread = threading.Thread(target=play_sound,
args=(frequency, volume))
        sound_thread.start()
        alarm_active = True
    # Alarm durdurulacaksa (mesafe veya hız tekrar eşikleri
geçerse)
    elif alarm_active and (distance > distance_threshold_2 or
speed <= speed_threshold):
        alarm_active = False

# Dosyayı yazma modunda açma
with open("el_konumu.txt", "w") as file:
    # Mesafe ve hız verilerini saklamak için deque kullanarak
boş liste
    mesafe_data = deque(maxlen=60) # 60 deque değeri
ax2.set_xlim değeri ile ölçüm hızına göre değişir.
    hiz_data = deque(maxlen=60)

    # Mesafe ve hız grafiği için iki subplot oluştur
    fig, (ax1, ax2) = plt.subplots(2, 1)
    plt.ion()  # Etkileşim
    fig.tight_layout(pad=3.0)
    fig.suptitle('El Mesafesi ve Hız Grafiği')
    ax1.set_ylabel('Mesafe (cm)')
    ax2.set_ylabel('Hız (cm/s)')
    ax2.set_xlabel('Zaman')

    # Önceki karenin el konumu
    prev_point = None
```

```
    prev_landmarks = []
    prev_distances = []  # Mesafeleri tutan liste
    # Kareler arası bekleme zamanı yok
    delay_counter = 0

    # Alarmın sınırlar normale döndüğünde susması için alarm
etkinlik durumu
    alarm_active = False

    # Kullanıcı tarafından belirlenen mesafe ve hız eşikleri
(cm ve m/s cinsinden)
    distance_threshold_1 = 0  # 0 cm
    distance_threshold_2 = 4  # 4 cm
    speed_threshold = 1  # Hız eşik değeri

    # Daire testere dikdörtgen tehlikeli alan hesaplamaları
    height, width = V, H  # Kameranın çözünürlüğü
    R = 5  # Örnek daire testere yüksekliği yarıçap yüksekliği
    C = -math.radians(90)  # Dereceyi radyana çevirme

    rect_width = int(pixel_per_cm_x * (6.25 + R *
abs(math.cos(C))))  # Dikdörtgen genişliğini özel çarpan ve
açıya göre ayarlama
    rect_height = int(pixel_per_cm_y * (20 + R * 2))  #
Dikdörtgen yüksekliğini özel çarpana göre ayarlama
    x_center = int(H/2) - int(R * pixel_per_cm_x * math.cos(C)
/ (100 / 50))  # Örnek olarak dikdörtgen x merkezi ve kaydırma
ayarı

    y_center = int(V/2)  # Örnek olarak y merkezi
    rect_x = x_center - rect_width // 2  # Dikdörtgenin sol
üst köşesinin x koordinatı
    rect_y = y_center - rect_height // 2  # Dikdörtgenin sol
üst köşesinin y koordinatı

    while True:
        start_time = time.perf_counter()  # Döngünün başında
işlem başlangıç zamanını al
        for ii in range(0,1,1):
            ret, frame = video.read()  # Video dosyasından bir
kare oku
        if not ret:
            break  # Video bittiğinde döngüyü sonlandır

        # Kamera görüntüsünü oku
        ret, frame = video.read()

        # Median filtresi uygula
        median_frame = apply_median_filter(frame)
        # Parlaklık normalleştirme
        adjusted_frame = normalize_brightness(median_frame)
```

```
        # Ekranın istenilen bir noktasına bir dikdörtgen çizme
        cv2.rectangle(adjusted_frame, (rect_x, rect_y),
(rect_x + rect_width, rect_y + rect_height), (0, 0, 255), 5)
        # Görüntüyü RGB formatına dönüştürme (MediaPipe, RGB
formatını bekler)
        frame_rgb = cv2.cvtColor(adjusted_frame,
cv2.COLOR_BGR2RGB)
        # MediaPipe Hands modelini kullanarak el tespiti yap
        results = hands.process(frame_rgb)
        # En yakın el noktasının dikdörtgene olan mesafesini
sakla
        min_distance = float('inf')
        # En yakın nokta
        nearest_point = None

        # El tespiti sonuçlarını işle
        if results.multi_hand_landmarks:
            for hand_landmarks in
results.multi_hand_landmarks:
                # El noktalarının piksel koordinatlarını
içeren bir liste oluştur
                hand_points = [(int(landmark.x * width),
int(landmark.y * height)) for landmark in
                               hand_landmarks.landmark]

                # Elin boyutunu kontrol et
                bounding_box =
cv2.boundingRect(np.array(hand_points))
                bounding_box_width, bounding_box_height =
bounding_box[2], bounding_box[3]

                Ze = 20  # Elin X ve Y ekseninde uzunluğu 20
cm, Z ekseni için kullanılır.
                Zk = 1.5   # Elin Z eksenindeki güvenli
yüksekliği, kesici yüksekliği 30 Cm.
                if bounding_box_width > (Ze * Zk *
pixel_per_cm_x) or bounding_box_height > (Ze * Zk *
pixel_per_cm_y):
                    continue

                for landmark in hand_landmarks.landmark:
                    # Her bir elin tespit edilen noktalarını
işle
                    x = int(landmark.x * width)
                    y = int(landmark.y * height)

                    # El noktalarını çiz
                    cv2.circle(adjusted_frame, (x, y), 5, (0,
255, 0), -1)

                    T = 4  # Elin kırmızı dikdörtgene olan
```

```
teğet ön durdurma mesafesi
                    # Elin kırmızı dikdörtgene olan mesafesini
hesapla (Ekstra yaklaşma mesafesi ekle)
                    dist_x = max(0,abs(x - (rect_x +
rect_width // 2)) - rect_width // 2 - int((T + 3) *
pixel_per_cm_x))
                    dist_y = max(0, abs(y - (rect_y +
rect_height // 2)) - rect_height // 2 - int((T + 3) *
pixel_per_cm_y))
                    distance = (dist_x ** 2 + dist_y ** 2) **
0.5

                    # En küçük mesafeyi ve en yakın noktayı
güncelle
                    if distance < min_distance:
                        min_distance = distance
                        nearest_point = (x, y)

                # El noktaları arasında çizgileri çiz
                connections = mp_hands.HAND_CONNECTIONS
                for connection in connections:
                    point1 = connection[0]
                    point2 = connection[1]
                    cv2.line(adjusted_frame,
hand_points[point1], hand_points[point2], (255, 255, 255), 2)

        # Elde edilen mesafeyi ve süreyi dosyaya yaz
        if nearest_point:
            current_time = time.perf_counter()  # Zaman
damgasını al
            elapsed_time = current_time - start_time  # Zaman
aralığını hesapla
            file.write(f"{min_distance / pixel_per_cm_x:.2f},
{elapsed_time:.2f} seconds\n")

        # El konumunu en yakın noktanın üzerine yaz
        if nearest_point:
            cv2.putText(adjusted_frame, f"{min_distance /
pixel_per_cm_x:.2f} cm", nearest_point,
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255,
255, 255), 1, cv2.LINE_AA)

            # El Hız hesaplama işlemleri
            if results.multi_hand_landmarks and delay_counter
== 0:
                max_speed = 0
                curr_distances = []
                # Takip etmek istediğimiz noktalar (0, 4, 8,
9, 12, 16, 20 numaralı noktalar)
                points_to_track = [0, 4, 8, 9, 12, 16, 20]  #
Elin en aktif noktaları
```

```
                # Bu landmark noktaları için mesafe hesapla
                for hand_landmarks in
results.multi_hand_landmarks:
                    distances = []
                    for point in points_to_track:
                        landmark =
hand_landmarks.landmark[point]
                        x, y = int(landmark.x * width),
int(landmark.y * height)

                        # Normalizasyon: X (-4, -3, ..., 3, 4)
ve Y (-3, -2, ..., 2, 3) koordinat
                        normalized_x = (x - x_center) / (H /
2) * 4 # 4 normalizasyon değişkeni
                        normalized_y = (y - y_center) / (V /
2) * 3 # 3 normalizasyon değişkeni
                        # Merkeze olan mesafeyi hesaplama
(Öklid mesafesi)
                        distance_from_center =
math.sqrt(normalized_x ** 2 + normalized_y ** 2)

                        # Hız faktörünü hesapla: Merkeze doğru
gidildikçe hız üstel(exp) olarak artacak
                        speed_factor = math.exp(-
distance_from_center)
                        distances.append(distance_from_center)
                    curr_distances.append(distances)

                # Hız hesaplansın (Eğer önceki mesafeler varsa
hesaplansın)
                if prev_distances:
                    for curr, prev in zip(curr_distances,
prev_distances):
                        for curr_dist, prev_dist in zip(curr,
prev):
                            # Mesafe değişimini hesapla ve
hızı hesapla
                            distance_change = abs(curr_dist -
prev_dist)
                            if distance_change > 0:
                                speed = (distance_change /
elapsed_time) * speed_factor  # V=(d/t).sf
                                max_speed = max(max_speed,
speed)  # Maksimun hızı güncelle
                # Gerçek zamanlı çalışmadaki Hız (En yüksek
hızlı noktadır)
                real_speed_m_s = max_speed  # hız (m/s)
cinsinden

                # Hızı ekranda göster (metre/saniye olarak)
```

```
                cv2.putText(adjusted_frame, f"Hiz:
{real_speed_m_s:.2f} m/s", (10, 50), cv2.FONT_HERSHEY_SIMPLEX,
1,
                            (255, 255, 255), 2, cv2.LINE_AA)

                # Hızı kaydet
                hiz_data.append(real_speed_m_s)
                ax2.clear()
                ax2.plot(hiz_data, color='green')
                ax2.set_ylim(0, 1)
                ax2.set_xlim(0, 60) # 60 deque değeri değeri
ile kayıt hızına göre değişir.
                ax2.grid(True)
                ax2.set_ylabel('Hız (m/s)')
                ax2.set_xlabel('60 Saniye Ölçekli Zaman
Aralığı')

            # Alarmı tetikle
            activate_alarm(min_distance / pixel_per_cm_x,
real_speed_m_s, distance_threshold_1, distance_threshold_2,
speed_threshold)

            prev_distances = curr_distances

        if min_distance != float('inf'):
            mesafe_data.append(min_distance / pixel_per_cm_x)
            ax1.clear()
            ax1.plot(mesafe_data, color='blue')
            ax1.set_ylim(0, 50)
            ax1.set_xlim(0, 60)
            ax1.grid(True)
            ax1.set_ylabel('Mesafe (cm)')

        # Yakalanan el görüntüsünü 'el' klasörüne kaydetme
        if nearest_point:
            x, y = nearest_point
            hand_image = adjusted_frame[y - 100:y + 100, x -
100:x + 100]  # 200x200 boyutunda alanı yakala
            if hand_image.size != 0:  # Eğer el görüntüsü boş
değilse
                cv2.imwrite(f"el/el_{time.time()}.jpg",
hand_image)

        cv2.imshow('El Tespiti', adjusted_frame)
        plt.pause(0.001)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
# Video yakalama cihazını serbest bırak
video.release()
# Pencereleri kapat
cv2.destroyAllWindows()
```

ÖZGEÇMİŞ

Sinan YÜKSEL, ilköğrenimini Samsun'da tamamladıktan sonra marangoz ve mobilya dekorasyon zanaatları üzerine eğitim aldı. 2010 yılında Bafra Endüstri Meslek Lisesi Elektrik Elektronik bölümünden mezun oldu. 2012 yılında Gaziosmanpaşa Üniversitesi Mekatronik bölümünden, bölüm birincisi ve onur öğrencisi olarak mezun oldu. 2014 yılında Anadolu Üniversitesi İşletme bölümünden lisans derecesiyle mezun oldu. 2021 yılında Amasya Üniversitesi Elektrik Elektronik Mühendisliği bölümünden, onur öğrencisi olarak mezun oldu. 2022 yılında Zonguldak Bülent Ecevit Üniversitesi, Elektrik Elektronik Mühendisliği Anabilim Dalı'nda tezli yüksek lisans eğitimine başladı.