AMELİYAT SİMÜLATÖRLERİ İÇİN ÖZGÜN ÜÇ BOYUTLU KONUM BELİRLEME SİSTEMİ GELİŞTİRİLMESİ

HİLAL ATICI

YÜKSEK LİSANS TEZİ

Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Prof. Dr. Yaşar HOŞCAN

Eskişehir

Anadolu Üniversitesi

Fen Bilimleri Enstitüsü

Mayıs, 2017

JÜRİ VE ENSTİTÜ ONAYI

Hilal ATICI'nın **"Ameliyat Simülatörleri İçin Özgün Üç Boyutlu Konum Belirleme Sistemi Geliştirilmesi"** başlıklı tezi 23/05/2017 tarihinde aşağıdaki jüri tarafından değerlendirilerek "Anadolu Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliği"nin ilgili maddeleri uyarınca, **Bilgisayar Mühendisliği** Anabilim dalında Yüksek Lisans Tezi olarak kabul edilmiştir.

	Unvanı-Adı Soyadı	İmza
Üye (Tez Danışmanı)	: Prof. Dr. Yaşar HOŞCAN	.....
Üye	: Yrd. Doç. Dr. Alper BİLGE	.....
Üye	: Yrd. Doç. Dr. Mehmet KOÇ	.....

Prof. Dr. Nedim DEĞİRMENCİ

Enstitü Müdürü

ÖZET
AMELİYAT SİMÜLATÖRLERİ İÇİN ÖZGÜN ÜÇ BOYUTLU KONUM BELİRLEME SİSTEMİ GELİŞTİRİLMESİ

Hilal ATICI

Bilgisayar Mühendisliği Anabilim Dalı

Anadolu Üniversitesi, Fen Bilimleri Enstitüsü, Mayıs, 2017

Danışman: Prof. Dr. Yaşar HOŞCAN

Ameliyat simülatörleri, hasta, kadavra veya hayvana ihtiyaç duymadan tıbbi uzmanları eğitmek amacıyla cerrahi işlemleri taklit etmek için geliştirilen bir bilgisayar teknolojisidir. Tıbbi işlemler hayati önem taşıdığı için, bu işlemleri yapacak uzmanların eğitimi önemlidir. Bu teknoloji ile tıbbi işlem yapacak olan uzmanlar eğitilmektedir. Ameliyat simülasyonlarında görsel gerçekçilik ve gerçek zamanlı etkileşimler önemlidir. Gerçek zamanlı etkileşim, uzmanın herhangi bir tıbbi işlemde, işlem yaptığı organdan anlık bir yanıt üretmesini gerektirir. Uzmanın algıladığı anlık yanıtlar yani haptik geri bildirim ameliyat simülatörlerinde büyük öneme sahiptir. Haptik geri bildirim ve sanal ortamın doğruluğu ise ameliyat aletinin doğru konumlandırılmasına bağlıdır. Ameliyat simülatörü uygulamalarında kullanılan pozisyon tahmini çalışmalarında, uygulamanın gerçek zamanlı olması kritik öneme sahiptir. Bu tez çalışmasında düşük maliyetle, piyasadan alınan kameralar kullanılarak gerçek zamanlı uygulama yapılmıştır. Ameliyat simülatörleri için 3 boyutlu pozisyon tahmini için farklı yöntemler denenmiş ve incelenmiştir. Öncelikle tek kamera ile küresel koordinat sisteminde pozisyon tespiti yapılmıştır. Daha sonra iki kamera ile stereo görme prensibi ve kartezyen koordinat sisteminde pozisyon tahmini yapılmıştır. Uygulamanın kullanıcıda gerçek ortam hissi uyandırması için saniyede 30 görüntü alınmalı ve işlenmelidir. Kullanılan metodların istenilen hata oranlarıyla pozisyon tahmini yaptığı görülmüştür. Piyasadan alınan web kameralar ile saniyede 30 görüntüye ulaşılamamış ve uygulanan metodların ortam şartlarından (ışık, arka plan vb.) etkilendiği görülmüştür. Bu nedenle saniyedeki görüntü sayısını (FPS) arttırmak ve ortam şartlarından etkilenme problemini ortadan kaldırmak için Raspberry Pi kamera modülleri kullanılmıştır. Kullanılan bu modüller ile ortam şartlarında etkilenmeden pozisyon tahmini yapıldığı görülmüştür. Aynı zamanda işlem hızı arttırılmış ve istenilen FPS değerine ulaşılmıştır.

Anahtar Sözcükler: Ameliyat Simülatörü, 3D, Pozisyon Tahmini

ABSTRACT
DEVELOPMENT OF ORIGINAL THREE DIMENSIONAL POSITIONING SYSTEM FOR OPERATING SIMULATORS

Hilal ATICI

Department of Computer Engineering

Anadolu University, Graduate School of Sciences, May, 2017

Supervisor: Prof. Dr. Yaşar HOŞCAN

Surgical simulators are a computer technology developed to simulate surgical procedures to train medical specialists without the need for patients, cadavers or animals. As medical procedures are vital, the training of specialists who perform these procedures is important. With this technology, specialists are trained to perform medical procedures. Visual realism and real-time interactions are important in surgical simulations. Real-time interaction requires the expert to produce an instantaneous response to the process in any medical procedure. The instantaneous responses that the expert perceives, so haptic feedback, have great importance in surgical simulators. Haptic feedback and the accuracy of the virtual environment depend on the correct positioning of the surgical instrument. In the position estimation exercises used in surgical simulator applications, The real-time implementation of the application is critical. In this thesis study, the real time application was made at low cost, using the cameras taken from the market. Different methods have been tried and studied for 3D position estimation for surgical simulators. Firstly with a single camera position determination was made in the spherical coordinate system. Then, with two cameras, position determination is made in the prenciple of stereo vision and the cartesian coordinate system. 30 images must be received and processed in order for the application to give the user a sense of real environment. It is seen that the methods used make the position estimation with the desired error rates. With web cams taken from the market, 30 images per second could not be reached and it has been observed that the applied methods are affected by the ambient conditions. For this reason, Raspberry Pi camera modules have been used to increase the number of images and to eliminate the problem of environmental influences. With these modules used, it is seen that position estimation is done without being affected by ambient conditions. At the same time, the processing speed has been increased and the desired FPS value has been reached.

Keywords: Operation Simulator, 3D, Position Estimation

TEŞEKKÜR

Yüksek lisans tezim süresince benden yardımlarını esirgemeyen, karşılaştığım sorunlarda bilgi ve deneyimlerini benimle paylaşan değerli hocam ve tez danışmanım Prof. Dr. Yaşar HOŞCAN'a teşekkür ederim.

Yüksek lisans öğrenimim boyunca beni destekleyen sevgili babam İsrafil ATICI ve aileme sonsuz teşekkürler.

Hilal ATICI

Mayıs,2017

23/05/2017

ETİK İLKE VE KURALLARA UYGUNLUK BEYANNAMESİ

Bu tezin bana ait, özgün bir çalışma olduğunu; çalışmamın hazırlık, veri toplama, analiz ve bilgilerin sunumu olmak üzere tüm aşamalarında bilimsel etik ilke ve kurallara uygun davrandığımı; bu çalışma kapsamında elde edilemeyen tüm veri ve bilgiler için kaynak gösterdiğimi ve bu kaynaklara kaynakçada yer verdiğimi; bu çalışmanın Anadolu Üniversitesi tarafından kullanılan "bilimsel intihal tespit programı"yla tarandığını ve hiçbir şekilde "intihal içermediğini" beyan ederim. Herhangi bir zamanda, çalışmamla ilgili yaptığım bu beyana aykırı bir durumun saptanması durumunda, ortaya çıkacak tüm ahlaki ve hukuki sonuçlara razı olduğumu bildiririm.

Hilal ATICI

İÇİNDEKİLER

Sayfa

TABLOLAR DİZİNİ

ŞEKİLLER DİZİNİ

Sayfa

GÖRSELLER DİZİNİ

Sayfa

SİMGELER VE KISALTMALAR DİZİNİ

b : Paralaks

DLT : Doğrudan Lineer Dönüşüm

EPnP : Etkili Perspektif N Nokta Problemi

FPS : Saniyedeki Görüntü Sayısı

f : Kamera Lenslerinin Odak Uzunluğu

HSV : Hue, Saturation, Value

HSB : Hue, Saturation, Brightness

OpenCV : Open Source Computer Vision

PLY : Polygon File Format

PnP : Perspektif N Nokta Problemi

P3P : Perspektif 3 Nokta Problemi

RANSAC : Random Sample Consensus

r : Yarıçap

RGB : Kırmızı, Yeşil, Mavi Renk Uzayı

3D : 3 Boyutlu

2D : 2 Boyutlu

(X, Y, Z) : 3 Boyutlu Koordinat

θ : Teta

ϕ : Fi

1. GİRİŞ

Ameliyat simülatörü, hasta, kadavra veya hayvana ihtiyaç duymadan tıbbi uzmanları eğitmek amacıyla cerrahi işlemleri taklit etmek için geliştirilen bir bilgisayar teknolojisidir. Ameliyat simülatörleri, ilkel plastik mankenlerden gömülü teknolojiye sahip makinelere ve son zamanlarda gerçekçi fizyolojik ve hasta senaryoları oluşturma kapasitesine sahip bilgisayarlar ile hızlı bir şekilde gelişmektedir. Tıbbi işlemler hayati önem taşıdığı için, bu işlemleri yapacak uzmanların eğitimi önemlidir. Örneğin; bir epidural enjeksiyon, uygulaması zor bir prosedür olduğundan ve hata yapılırsa hasta için bir takım ciddi riskleri içerdiğinden, bu işlem esnasında oluşacak hataların sayısını azaltmak için ameliyat simülatörleri kullanılmaktadır. Pek çok simülatör türü geliştirilmiş ve piyasaya sürülmüştür. Bu simülatörler gerçek bir cerrahi senaryoyu üretir ve kullanıcıya ameliyatla gerçekçi olabilen farklı ara yüzler aracılığıyla anatomi ile etkileşime girme imkânı sunar ve bu etkileşim sonucunda kullanıcıya bir tür haptik geri bildirim uygular. Haptik geri bildirim olarak bilinen bu işlem, kullanıcıya sanal ortamdaki işlem sonuçlarının kullanıcının fiziksel ortamda hissedebileceği bir formata dönüştürmektir. Ameliyat simülatörleri fiziksel ve sanal ortam etkileşiminden oluşmaktadır. Fiziksel ve sanal ortam etkileşimi simülatöre eklenerek, uzmanların bu etkileşimi hissetmesi ve gerçekçi bir işlem yapması sağlanmaktadır. Ameliyat simülatörleri, prosedürü öğrenmek için güvenli ve risksiz bir ortam sağlar ve daha yüksek bir hasta güvenliği sağlar.

Ameliyat simülatörlerinde gerçeklik hissinin sağlanabilmesi için yeterli tekrarlama hızına ulaşılması gerekmektedir. Bu işleme gerçek zamanlı etkileşim denilmektedir. Örneğin, görsel duyuların gerçeklik hissi alabilmesi için görüntünün oluştuğu sahnenin saniyede 30 defa yenilenmesi gerekmektedir. Ameliyat simülasyonlarında görsel gerçekçilik ve gerçek zamanlı etkileşimler önemlidir. Gerçek zamanlı etkileşim, uzmanın herhangi bir eyleminin, geometrisinin karmaşıklığı ne olursa olsun, uyarılmış organdan anlık bir yanıt üretmesini gerektirir. Sonuç olarak fiziksel etkileşimin gerçekçiliği ameliyat simülasyonunda önemli noktadır. Ameliyat simülasyonunda, fiziksel etkileşimin entegrasyonu, sanal etkileşim kadar öneme sahiptir. Simülasyon güçlerin kesin hesaplamaları ile birleştiğinde, cerrahın gerçeğe yakın fiziksel duyguları hissetmesi mümkün olabilir. Fiziksel etkileşim ve sanal ortam etkileşim doğruluğu ise ameliyat aletinin doğru konumlandırılmasına bağlıdır.

Konumlandırma pozisyon tahmini problemi olarak ifade edilebilir. Bu tezde ameliyat aletinin pozisyon tahmini için farklı yöntemler denenmiş ve incelenmiştir.

Pozisyon tahmini, ameliyat simülatörlerinde temel sorunlardan birisidir ve pozisyon tahmini problemi uzun yıllardır bir araştırma alanı olmuştur. Ameliyat simülatörü uygulamalarında kullanılan pozisyon tahmini çalışmalarında, uygulamanın gerçek zamanlı olması kritik öneme sahiptir. Pozisyon tahmini, birçok yöntem ile yapılmasına karşın, genel olarak görsel tabanlı metotlar kullanılmaktadır. 3D pozisyon tahmini için birçok güvenilir yöntem önerilmiştir. Son yıllarda görüntü tabanlı izleme sistemleri, 3D nesne pozisyon takibinde iyi bir seviyeye ulaşmıştır.

Geliştirilecek simülasyon programlarının genel amacı; öğrenciye gerçek laboratuvarların bir modelini sunabilen sanal laboratuvarlarda kullanılabilmek olarak belirlenmiştir. Böylece öğrenci laboratuvar imkanlarının kısıtlı olduğu ya da doğal ortamlarda inceleme ve gözleme olanağı bulamadığı olay ya da olguları sanal laboratuvarlar aracılığıyla inceleme olanağı bulabilecektir. Seçilen konunun farklı şekillerde gözlemleyebilme imkânı, konunun daha iyi öğrenilmesini ve algılanmasını sağlamış olacaktır.

Ameliyat simülatörlerinin eğitim amaçlı kullanılma oranı gün geçtikçe artmaktadır. Eğitim ve öğretimin teknikleri yıllardır değişmektedir. Tıp öğrencilerinin pratik yapabilmeleri için yeterli bir ortam bulunmamaktadır. Ameliyat simülatör uygulamaları medikal alanda yeni bir trendin doğmasını sağlamıştır. Söz konusu teknoloji sayesinde öğrenciler her konuyla ilgili defalarca pratik yapabilmektedirler. Uygulamalardaki hata oranlarının azaltılması amaçlanmaktadır.

Piyasada ameliyat simülatörü çalışmasının birçok örneği vardır. TechRepublic™, VRS©, WMG©, Techrunch©, Medelita™, Vrphobia©, Wired©, Topphysio® ve Ispr© firmaları sanal gerçeklik ile birçok alanda çalışmalara sahiptir. Korkuları tedavi etmek için sanal gerçeklik ile korkularla yüzleştirme tedavisi, travma sonrası stres bozukluğu tedavisi, ağrı ve acı tedavisi, cerrahi eğitim, beyin hasarı değerlendirme ve rehabilitasyon, otizmli bireyler için sosyal biliş eğitimi ve meditasyon gibi birçok çalışma bu firmalar tarafından yapılmıştır [1]. Yapılan çalışmalar yüksek maliyetli çalışmalardır. Söz konusu çalışmalar yüksek gerçeklikli ve yüksek fiyat etiketli çalışmalar olduğu için genel kullanıcıya hitap etmekte sınırlı kalmıştır.

Görsel 1.1. *Simülatör Uygulaması [1]*

Orta gerçeklik seviyesindeki simülatörler ile, simülatör eğitimlerinden elde edilmesi hedeflenen kazanımların büyük oranda elde edildiği gözlemlenmiştir.

Orta gerçeklikli ve orta maliyetli kullanıcı dostu bir simülatör, sanal ortamlarda tıbbi işlemler için görüntü işleme ve bilgisayar görüntüleme yoluyla spesifik eğitim görevlerini taklit etmek üzere tasarlanmıştır.

Bu tez çalışmasında orta maliyetle, piyasadan alınan kameralar kullanılarak gerçek zamanlı uygulama yapılmıştır. Ameliyat simülatörleri için 3D pozisyon tahmini için farklı yöntemler denenmiş ve incelenmiştir. Ameliyat aletinin konum tespiti için renk algılama yöntemi kullanılmıştır. Renk algılama işlemi OpenCV kütüphanesi ile gerçekleştirilmiştir. Yapılan çalışma ile gerçek zamanlı uygulamalarda olması gereken hız konusu üzerinde durulmuştur. Yapılan araştırmalarda doktorların ortalama 1 mm hassasiyet ile çalıştıkları görülmüştür. Aynı zamanda kullanıcını gözünde gerçek hisler oluşması için saniyede 30 görüntü gerektiği görülmüştür. Bu nedenle ameliyat simülatörlerinde pozisyonun 1 mm hassasiyet ve saniyede 30 görüntü ile tespit edilmesi amaçlanmaktadır. Saniyede 30 görüntü verilmesi için görüntü işleme kısmının ortalama 33 ms'de konum bilgisi göndermesi gerekmektedir. Çalışmanın esası nesnenin pozisyon tespitinin yanı sıra bu işlemin FPS (saniyede görüntü sayısı) değerini arttırmaktır. FPS değeri arttıkça alınan görüntü sayısı ile birlikte daha gerçek bir uygulama sağlanacaktır.

1.1. Tezin Yapısı

Bu tez, 5 bölümden oluşmaktadır. Bölüm 2'de tezde kullanılacak olan yöntemler ile ilgili olan genel tanımlamalara ve formüllere yer verilmiştir. Tek kamera ve çift kamera ile yapılan işlemler ayrıntılı bir şekilde anlatılmıştır. Literatürde yapılmış olan pozisyon tespiti ile ilgili çalışmalar hakkında bilgi verilmiştir.

Bölüm 3'de, tek kamera kullanılarak nesnenin pozisyonun tespiti için yapılan işlemler açıklanmaktadır. Açık kaynak kodlu görüntü işlem kütüphanesi olan OpenCV kaynaklarından faydalanılarak doku kaplı nesnenin pozisyon tespiti için yapılan işlemlerin aşamaları ayrıntılı olarak anlatılmıştır. Tek kamera göz önüne alınarak katı nesneler için bir 3D pozisyon tahmin algoritmasının uygulanmasını sunulmuştur. Devamında küresel koordinat sistemi kullanılarak renkli bir nesnenin pozisyon tahmini için yapılan çalışma anlatılmıştır. Öncelikle, işaretleyici tespiti için yapılan işlemler açıklanmştır. Çalışma kapsamında, renk tabanlı algılama yaklaşımı kullanarak nesnenin yatay ve dikey düzlemdeki konumunun ve bu nesnenin derinlik bilgisinin gerçek zamanlı tespiti ve takibini yapacak yazılım çalışması yapılmıştır. Yazılım kısmı için Windows 10 işletim sistemi altında çalışılmış olup, OpenCV kütüphanesinden yararlanılmıştır. Bu yöntemde nesnenin yatay ve dikeydeki koordinatları belirlenebilirken, z ekseni için ise nesnenin derinlik bilgisine ihtiyaç duyulmuştur. Yapılan çalışma ile nesnenin derinlik bilgisine küçük hata oranlarıyla ulaşılmıştır. Bu konum bilgisi için referans noktası belirlenmiş ve bu referans noktasına göre küresel koordinatlar göz önünde bulundurularak hareketli nesnenin koordinat bilgileri gerçek zamanlı olarak tespit edilmiştir.

Bölüm 4'de, iki kamera ile pozisyon tespiti için yapılan işlemler anlatılmıştır. Kullanılan yöntemler stereo görme prensibi ve kartezyen koordinat sistemidir.

Stereo görme, doğrudan derinlik ölçebilme yeteneği ile üç boyutlu algılama için çok önemli bir çalışma alanıdır. Bilgisayarla görüş makinelerinin görmesini sağlayan teknoloji olarak tanımlanabilir. İnsanlarda bulunan binoküler sistemin incelenmesi ile stereo görüş sistemleri modellenmektedir. Stereo görüşte iki ya da daha fazla kamera ile farklı bakış açılarından ve uzaklıklarından alınan görüntüler kullanılarak üç boyutlu sahne çıkarımı yapılabilmektedir. Bu çalışmada OpenCV kullanılarak stereo görüntülerden derinlik kestiriminin nasıl yapılabileceği üzerinde durulmuş, OpenCV'de tanımlı olan blok eşleme algoritması kullanılması ile elde edilen derinlik kestirimi

sonuçları değerlendirilmiştir. Nesne tespiti için çalışmanın tamamında HSV renk uzayında tanımlı bir renkli nesneye kontur uygulaması kullanıldı. Stereo görme prensibine göre iki kamera birbirlerine paralel olarak yerleştirilmiş ve kalibrasyonları ve düzeltmeleri yapıldıktan sonra nesnenin kameraya olan uzaklığının tespiti yapılmıştır.

Kartezyen koordinat sisteminde kameralar birbirlerine dik olarak biri x eksenine biri y eksenine paralel olarak yerleştirilmiştir. Alınan eş zamanlı görüntülerde kamera parametreleri kullanılarak resim düzeltme işlemi yapılmıştır. Kullanılan renkli üç boyutlu nesnenin projeksiyon ekranlarındaki izdüşümlerine göre pozisyon tahmini yapılmaktadır.

Bölüm 5'de, Raspberry Pi 3 kamera modülü kullanılarak nesnenin pozisyonun tespiti için yapılan işlemler açıklanmaktadır. Bu bölümde kullanılacak olan Raspberry Pi modülü hakkında bilgi verilmiştir. Raspberry Pi için işletim sistemi kurulumu, kullanılacak olan programların ve OpenCV kütüphanesinin kurulumu açıklanmıştır. Kartezyen koordinat sistemi kullanılarak aktif işaretçinin pozisyon tahmini için yapılan çalışma anlatılmıştır. Çalışma kapsamında, renk tabanlı algılama yaklaşımı kullanarak nesnenin yatay ve dikey düzlemdeki konumunun gerçek zamanlı tespiti ve takibini yapacak yazılım çalışması yapılmıştır.

2. TEMEL BİLGİLER

Bu bölümde, tezin geri kalanında kullanılacak olan sistemler için teorik arka plan hakkında genel bilgi vermektedir.

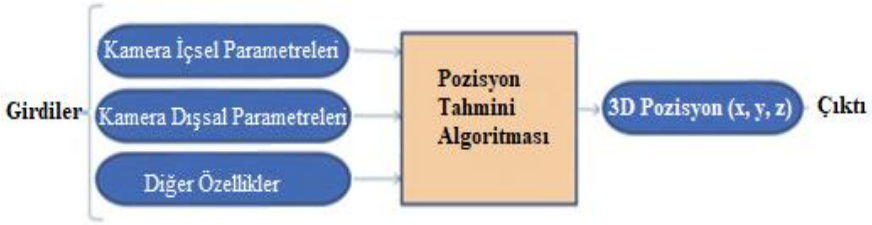
Bir nesnenin görüntülerini kullanarak o nesnenin kameraya göre pozisyon ve yönelim hesaplamasına pozisyon tahmini problemi denir. Pozisyon tahmini, bilgisayarla görü, robotik ve fotogrametri alanlarındaki önemli problemlerden birisidir. Nesne takibi, nesne tanıma, robotların kendi konumlarını bulması gibi uygulamalar pozisyon tahmini probleminin kullanım alanlarına örnek olarak verilebilir.

Pozisyon çıkarımı için yapılacak ilk işlem kullanılacak yöntemin seçilmesidir. Bunlar tek görünüm geometrisi (*ing.* Single View Geometry) ve çoklu görüş geometrisi (*ing*. Multiple View Geometry) yöntemleridir. Bu iki yöntemde de öncelikle kamera kalibrasyonunun yapılması gerekmektedir. Kalibrasyon sonucunda içsel ve dışsal olmak üzere iki çeşit parametre elde edilmektedir.

i. Kamera parametreleri

Seçilen yönteme göre tek veya iki kameralı sistem oluşturularak kalibrasyon yapılmaktadır. Kameralar ölçüleri bilinen bir kalibrasyon nesnesinin farklı açılardan ve uzaklıklardan alınmış görüntüleri kullanılarak kalibre edilir. Kalibrasyon sonrasında içsel ve dışsal olmak üzere iki çeşit parametre elde edilir. İç (*ing*. intrinsic) parametreler lens bozukluklarının düzeltilmesinde kullanılır. Dış (*ing*. extrinsic) parametreler ise kameralar arasındaki birbirlerine göreceli geometrik ilişkiyi göstermektedir.

ii. Kamera parametreleri kullanarak pozisyon çıkarımı



Şekil 2.1. *Pozisyon Tahmini Genel Şeması*

Kamera parametreleri kullanılarak pozisyon tahmini yapılmaktadır. Şekil 2.1.'de pozisyon tahmini genel şeması verilmiştir.

Bölüm 2.1.'de kamera modeli ve radyal objektif bozulması kullanan görüntülerden, perspektif geometriden oluşan tek görünüm geometrisi açıklanmıştır.

Bölüm 2.2.'de ise epipolar geometri, düzeltme ve eşitsizlik (*ing*. disparity) tanımını içeren çoklu görüş geometrisi üzerine temel bilgiler verilmektedir. Bölüm 2.3.'de pozisyon tahmini algoritmaları hakkında bilgi verilmiştir. Bölüm 2.4.'de literatürde bu çalışma ile ilgili yapılmış çalışmalar hakkında bilgi verilmiştir.

2.1. Tek Görünüm Geometrisi

Bu bölümün ana amacı, bir 3 boyutlu görüntünün 2 boyutlu görüntü düzlemine yansıtma olarak adlandırılan, görüntü oluşum sürecini kabaca özetlemektir.

2.1.1. Görüntüler

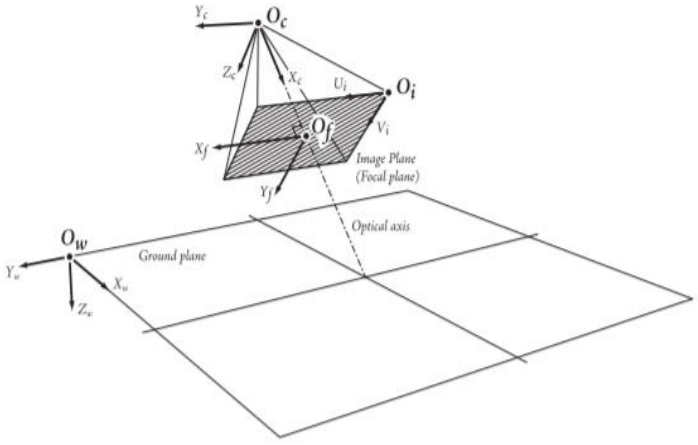
Bu çalışmada bir I görüntüsü, iki boyutlu bir konumu gri tonlamaya veya renk değerine eşleyen bir işlevdir:

$$I: \Omega^2 \rightarrow \mathbb{R}^d \tag{2.1}$$

Sonuç olarak, bir piksel konumu x verildiğinde, renk değeri I (x) olarak ifade edilmektedir.

2.1.2. Kamera modeli

Bir görüntüdeki nesneler arasındaki mesafeleri ölçmek için gerçek kameranın fiziksel özelliklerini doğru bir şekilde taklit eden bir model kameraya ihtiyaç duyulmaktadır. Birkaç kamera modeli vardır, bunlardan en çok kullanılan model olan perspektif projeksiyon modeli bu bölümde anlatılacaktır. Şekil 2.2., bu kamera modelleri için ilgili koordinat sistemlerini göstermektedir [2].



Şekil 2.2. *Kamera Modeli [2]*

• O_w, Dünya koordinatları [x_w, y_w, z_w]: en temel koordinat sistemidir, diğer tüm sistemler bununla bağlantılı olarak tanımlanmaktadır.

• O_c, Kamera koordinatları [x_c, y_c, z_c]: her kamera için benzersizdir, bu dünya koordinat sistemi ile ilişkili olarak açıklanmaktadır.

• O_f, Görüntü düzlemi koordinatları [x_f, y_f]: her kamera için benzersizdir, aslında kamera koordinatlarıyla aynı koordinat sistemidir ancak iki boyuttadır. Görüntü düzlemi koordinatlarının ölçeklendirilmesi her zaman kamera koordinatlarıyla aynıdır.

• O_i, Görüntü koordinatları [u_i, v_i]: görüntü düzlemi koordinatları gibi iki boyutludur. Görüntü koordinatları esasen, farklı ölçeklendirme ve çeviri haricinde, görüntü düzlemi koordinat sistemi ile aynı koordinat sistemidir.

2.1.2.1. *Parametreler*

Bir kamera modeli, kamerayı modellemek için kullanılan bir dizi parametre ile tanımlanır. Parametreler genellikle içsel ve dışsal parametreler olmak üzere iki kategoriye ayrılır.

***İç (ing. Intrinsic) parametreler*:** Önceki bölümde, genellikle kamera kalibrasyon matrisi olarak adlandırılan K, iç kamera kalibrasyon parametreleri ile bir matris olarak tanımlanmıştır. K aşağıdaki gibi tanımlanmaktadır.

$$K=\begin{bmatrix} a_u & s & u_0 \\ 0 & a_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.2)$$

- a_u ve a_v, u ve v yönünde her bir koordinatında tanımlanan ölçek faktörüdür. Bu ölçek faktörleri, kamera odak uzaklığı ile orantılıdır. $a = k_u f$ ve $a= k_v f$, Burada k_u ve k_v, u ve v yönlerinde birim başına toplam piksel sayısıdır.
- $c = [u_0; v_0]^T$ optik eksen ile görüntü düzleminin kesiştiği asıl nokta koordinatlarını temsil eder.
- Eğim açısı olarak bilinen s, u ve v yönlerinin diklik oranını inceler. Modern kameralarda bu değer genellikle sıfırdır.

Dış (ing. Extrinsic) parametreler: [R | t], bir dünya koordinat sisteminden kamera koordinat sistemine öklid dönüşümüne karşılık gelen 3x4 bir matris olarak tanımlanmaktadır. Gerçekte bu matris, rotasyon matrisinin ve kamera pozisyonu olarak tanımlanan dönüşüm vektörünün birleşimini ifade etmektedir.

$$[R \mid t] = \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_1 \\ R_{21} & R_{22} & R_{23} & t_2 \\ R_{31} & R_{32} & R_{33} & t_3 \end{bmatrix} \quad (2.3)$$

$R_{11} = \cos\Psi \cos\theta$

$R_{12} = -\sin\Psi \cos\Phi + \cos\Psi \sin\theta \sin\Phi$

$R_{13} = \sin\Psi \sin\theta + \cos\Psi \sin\theta \cos\Phi$

$R_{21} = \sin\Psi \cos\theta$

$R_{22} = \cos\Psi \cos\Phi$

$R_{23} = -\cos\Psi \sin\Phi + \sin\Psi \sin\theta \cos\Phi$

$R_{31} = -\sin\theta$

$R_{32} = \cos\theta \sin\Phi$

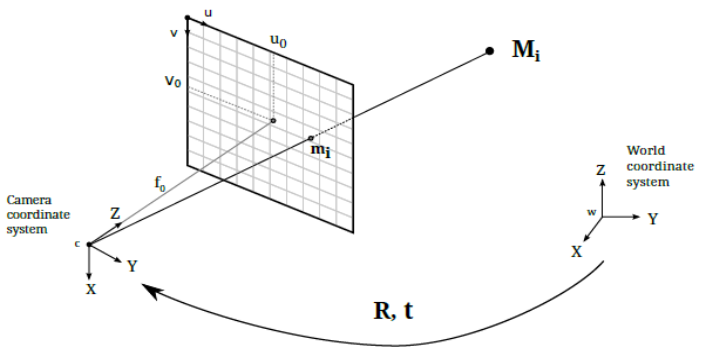
$R_{33} = \cos\theta \cos\Phi$

t = [t1; t2; t3]

Burada Ψ , Φ, θ euler açılarını ifade etmektedir. Φ, x ekseninde dönme açısı, θ, y ekseninde dönme açısı, Ψ, z ekseninde dönme açısını verir. Rotasyon matrisinin parametreleri yukarıdaki gibi tanımlanmaktadır.

2.1.2.2. *Perspektif projeksiyon modeli*

Gerçek dünyada nesneler üç boyutlu algılanırken, bilgisayar ekranında iki boyutlu olarak algılanmaktadır. İki boyutlu ekrandaki görüntünün üçüncü boyutunun bulunması önem taşımaktadır. Örneğin; nesnenin uzaktaki görünüşünün gerçekçi olması için küçük, ekrana yakın olan nesnenin ise daha büyük olması gerekmektedir. Bu işlem perspektif projeksiyon ile açıklanmaktadır.



Şekil 2.3. *Perspektif Projeksiyon Modeli [3]*

Perspektif projeksiyon modelinde, doğrular birbirlerine paralel değildir. Doğrular projeksiyon merkez noktasından çıkmaktadır.

Tanımlanan projeksiyon 2.4 eşitliği ile ifade edilebilir.

$$s\, m_i' = P\, M_i' \tag{2.4}$$

Burada s ölçek faktörü, $m_i' = [u;\ v;\ 1]^T$ ve $M_i' = [X;\ Y;\ Z;\ 1]^T$ m_i ve M_i noktalarının homojen koordinatlarıdır. P ise 3x4 projeksiyon matrisini ifade etmektedir.

Perspektif matrisi şu şekilde ayrılabilir:

$$P = K[R \mid t] \tag{2.5}$$

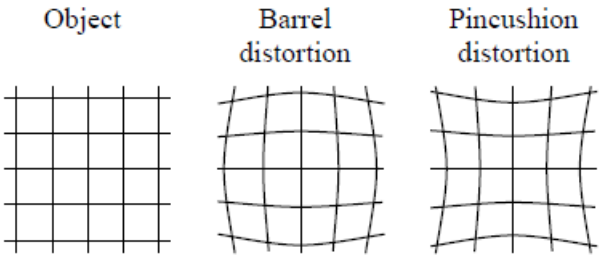
K odak uzaklığı, ölçek faktör ve optik merkez noktası koordinatları gibi kamera kalibrasyon parametrelerinden oluşan 3x3 matristir.

[R | t] bir dünya koordinat sisteminden kamera koordinat sistemine öklid dönüşümüne karşılık gelen matristir. R rotasyon matrisi, t dönüşüm vektörüdür.

2.1.2.3. *Lens bozuklukları*

Önceki bölümdeki kamera modeli noktaların düz çizgili bir projeksiyonu olduğunu varsaymaktadır, yani dünyadaki düz çizgiler görüntü düzleminde düz çizgi olarak yansıtılmaktadır. Çok iyi bir yaklaşım olmasına rağmen gerçek dünya lensleri için bu tam olarak doğru değildir. Lens üreticileri tarafından çok mercekli sistemler kullanılıyor olsa da hala optik sapmalar denen bazı kusurlar vardır. En yaygın olanları radyal ve teğetsel bozukluklar, kromatik ve küresel sapma ve astigmattır.

Lens sisteminin fiziksel özellikleri radyal bozulma olarak adlandırılan bir olayı ortaya çıkarmaktadır. Tipik olarak, kare bir cisim, Şekil 2.4.'de gösterildiği gibi varil (*ing.* barrel) bozukluğu veya iğnelik (*ing.* pincushion) bozulması ile görüntülenecektir. Görüntü, görüntünün merkezinden daha uzakta bozulmaktadır. Lens sisteminin optik ekseni, görüntü sensörü düzleminin normal vektörüyle mükemmel hizalanmadığında görüntüde teğetsel bozukluk ortaya çıkmaktadır.



Şekil 2.4. *Barrel ve Pincushion Bozuklukları [2]*

Radyal ve teğetsel objektif bozulmasına sahip kamera modelleri için nihai görüntü koordinatlarını elde etmek için bozulmuş koordinatlara içsel kamera matrisi K ile bir haritalama uygulanır.

2.1.3. İğne deliği kamera kalibrasyonu

Kameralar uzun süredir çevremizde bulunmaktadır. Bununla birlikte, 20. yüzyılın sonlarında ucuz iğne deliği kameralarının tanıtılması ile, onlar günlük hayatımızda yer almaya başlamıştır. Ne yazık ki, bu ucuz fiyatı ile birlikte önemli bir bozulma gelmektedir. Bu bozulma bazı sabitler, kalibrasyon ve bazı yeniden eşlemeler ile düzeltilmektedir. Ayrıca, kalibrasyon ile kameranın doğal birimleri (pikseller) ile gerçek dünya birimleri (milimetre) arasındaki ilişkiyi de belirlenebilmektedir [4].

Teori: OpenCV bozulmalar için radyal ve teğetsel faktörleri dikkate alır. Radyal faktör için aşağıdaki formül kullanılır:

$$x' = x(1 + k_1 r^2 + \ k_2 r^4 + k_3 r^6) \tag{2.6}$$

$$y' = y(1 + k_1 r^2 + \ k_2 r^4 + k_3 r^6) \tag{2.7}$$

Dolayısıyla giriş görüntüsündeki (x, y) koordinatlarındaki düzeltilmemiş bir piksel noktası için düzeltilmiş çıktı görüntüsündeki konumu (x' , y') olmaktadır. Radyal bozulmanın varlığı "varil" (*ing.* barrel) veya "balık gözü" (*ing.* fish-eye) etkisi şeklinde kendini gösterir.

Teğetsel bozulma, görüntü alma lensleri görüntüleme düzlemine tamamen paralel olmadığı için oluşmaktadır. Formüller aracılığı ile düzeltilebilir:

$$x' = x + [2p_1 xy + \ p_2 (r^2 + 2x^2)] \tag{2.8}$$

$$y' = y + [p_1 (r^2 + 2y^2) + 2p_2 xy] \tag{2.9}$$

Böylelikle OpenCV'de beş sütun içeren bir satır matrisi olarak sunulan beş bozulma parametresi ortaya çıkmaktadır:

$$C = (k_1\ k_2\ p_1\ p_2\ k_3) \tag{2.10}$$

C bozulma katsayısı vektörünü ifade etmektedir. Birim dönüşüm için aşağıdaki formül kullanılmaktadır:

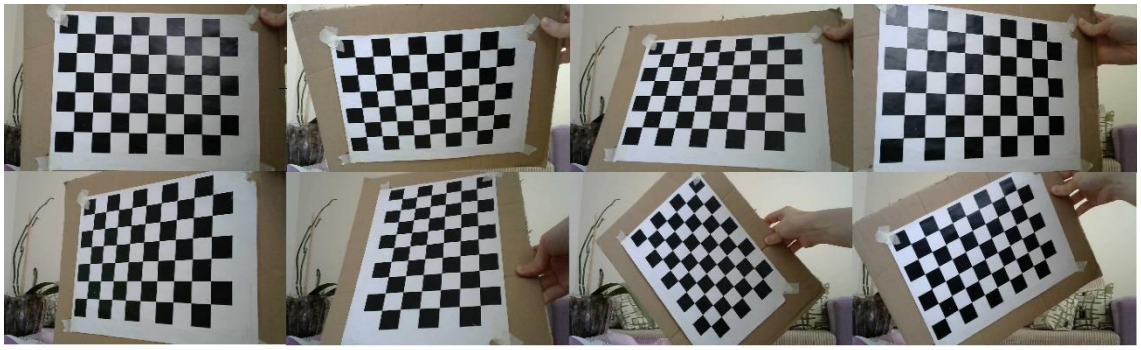
$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.11}$$

Burada w'nin varlığı homografi koordinat sistemi kullanılarak açıklanmaktadır. Bilinmeyen parametreler f_x ve f_y (kamera odak uzaklıkları) ve piksel koordinatlarıyla ifade edilen (c_x, c_y) optik merkezleridir. Her iki eksen için belirli bir en-boy oranı (genellikle 1) ile birlikte ortak bir odak uzunluğu kullanılırsa o zaman $f_y = f_x * a$ ve üst formülde tek bir f odak uzunluğuna sahip olunur. Bu dört parametreyi içeren matriste kamera matrisi denir. Bozulma katsayıları, kullanılan kamera çözünürlüğünden bağımsız olarak aynı olmakla birlikte, kalibre edilmiş çözünürlükteki mevcut çözünürlük ile birlikte ölçeklendirilmelidir.

Bu parametrelerin hesaplanması temel geometrik denklemler vasıtasıyla yapılır. Kullanılan denklemler, seçilen kalibre eden nesnelere bağlıdır. OpenCV, kalibrasyon için üç tür nesneyi desteklemektedir:

- Klasik siyah-beyaz satranç tahtası
- Simetrik daire modeli
- Asimetrik daire modeli

Temelde, kamera ile bu desenlerin fotoğraflarını çekilir ve OpenCV'nin bulmasına izin verilmesi gerekir. Bulunan her desen yeni bir denklemle sonuçlanır. Denklemi çözmek için, iyi pozisyonlandırılmış bir denklem sistemi oluşturmak için önceden belirlenmiş sayıda desen anlık görüntüsüne ihtiyaç duyulmaktadır. Bu sayı satranç tahtası deseni için daha yüksek ve daire olanlar için daha az olmaktadır. Teorik olarak satranç tahtası deseninde en az iki anlık görüntü gerekmektedir. Bununla birlikte, girdi görüntülerinde iyi miktarda gürültü olmaktadır, bu nedenle iyi sonuçlar elde etmek için, giriş deseninin farklı pozisyondaki en az 8 iyi resmi gerekmektedir.



Görsel 2.1. *Kameradan Alınan Resimler*

Görsel 2.2. *Düzeltilmiş Kamera Görüntüleri*

Kameralar ölçüleri bilinen bir satranç tahtası modelinin farklı açı ve uzaklıklardan alınmış görüntüleri kullanılarak kalibre edilir. Kalibrasyon sonrasında elde edilen iç parametreler lens bozukluklarının düzeltilmesinde kullanılır.

Kalibrasyon sonucu elde edilen parametreler kaydedilir. Pozisyon çıkarımı yöntemlerinde bu parametreler kullanılarak kameranın bilgileri kullanılmış olacaktır.

2.2. Çoklu Görünüm Geometrisi

Bir önceki bölümde, perspektif bir projeksiyon kamerası modeli kullanarak karşılık gelen bir dünya noktasının 3 boyutlu bir koordinatını 2 boyutlu bir görüntü koordinat noktasına nasıl eşleştirildiği anlatılmıştır. Bunu yaparak, nesnenin uzaysal konumu kaybedilir. Bu nedenle kaybedilen bu boyut bulunmaya çalışılır. Bu bilgiyi elde etmenin en çok kullanılan araçlarından biri, aynı sahnedeki birden fazla görüntünün bilgisini değerlendirmektir. Temel fikir, dünya noktasının kesin konumunu belirlemek için farklı fakat bilinen bir kamera pozisyonundan gelen bilgileri kullanmaktır. Daha sonra, dünya üzerindeki nokta, kameraların her birinden kaynaklanan iki ışının kesişme noktasında olacak ve bilgisayarlı görme literatüründe üçgenleme veya üç boyutlu yeniden yapılandırma olarak belirtilen bir süreçle hesaplanmaktadır. Yeterli nokta

mevcut ise, gözlemlenen noktaların üç boyutlu yapısına ek olarak kameranın üç boyutlu hareketi tahmin edilebilmektedir. Bu amaca ulaşmak için, resimler arasındaki eşleşmelerin (*ing.* correspondance) nasıl elde edileceği ile ilgili temel sorunun çözülmesi gerekmektedir.

2.2.1. Epipolar geometri

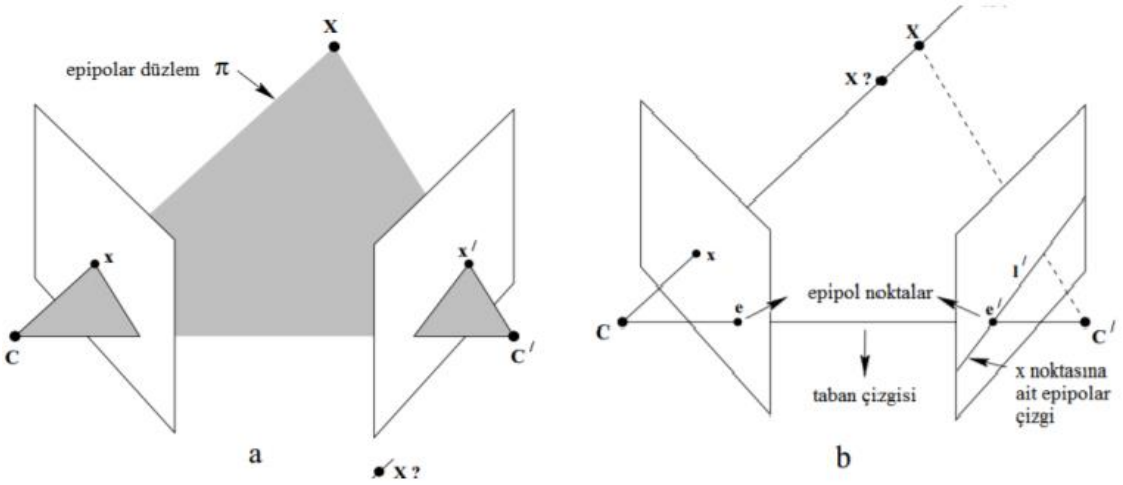
Epipolar geometri, bir sahnenin veya nesnenin iki farklı görüntüsü arasındaki geometrik ilişkinin kurulması olarak ifade edilebilir [5]. Şekil 2.5.'de C ve C' kamera merkezleridir. Görüntüye ait X noktasının birinci ve ikinci görüntüye düşen imge noktaları sırasıyla x = $(x, y, 1)^T$ ve x′ = $(x', y', 1)^T$ vektörleri ile ifade edilmektedir. Bu iki nokta arasındaki ilişkiyi F matrisi kurmaktadır (2.12).

$$x'^T F x = 0 \tag{2.12}$$

Kamera merkezinden başlayarak x ve x' noktalarından geçerek şekilde ışınlar çizildiğinde bu ışınlar X noktasında kesişmektedir. X, x, x' , C ve C' noktalarını içerecek şekilde oluşturulan düzleme epipolar düzlem denilmektedir [6]. İki kamera merkezi taban çizgisi tarafından birleştirilmektedir. Kamera merkezlerini birleştiren taban çizgisi ile görüntü düzlemlerinin kesiştiği noktalar epipol (e, e') adını almaktadır. Epipol ve F matrisinin çarpımı (2.13) sıfıra eşittir [6]. Epipol noktalardan imge noktasından geçecek şekilde çizilen doğruya ise epipolar çizgi denilmektedir [7]. Epipolar düzlem ile görüntü düzleminin kesişimi sonucu da epipolar çizgiler oluşmaktadır. Eş noktalar eş epipolar çizgiler üzerinde yer almaktadırlar. Bu nedenle x noktasının eşdeğer noktası olan x′ noktası, x noktası kullanılarak hesaplanan l' epipolar çizgisi üzerinde (2.14) olmalıdır. l', x′ noktasının aranacağı epipolar çizgidir. Bu nedenle x′ noktası, ikinci görüntünün tamamı içerisinde değil, sadece l' epipolar çizgisi üzerinde aranacağından pencere boyutu düşürülmekte ve arama maliyeti azaltılmaktadır.

$$Fe = 0, \; F^T e' = 0 \tag{2.13}$$

$$l' = Fx, \quad l = F^T x' \tag{2.14}$$



Şekil 2.5. *Epipolar Geometri [6]*

2.2.2. F matrisi hesaplama

F matrisi (*ing.* fundamental matrix) epipolar geometriyi temsil eden temel matristir. Hareket segmentasyonu, 3D model oluşturma, kamera kalibrasyonu, stereo görme gibi birçok uygulama alanında *F* matrisi kullanılmaktadır. *F* matrisinin etkin ve doğru olarak hesaplanması, çoklu görüntü analizinde önemli bir problemdir [6]. Aynı sahnenin birbirine yakın olan farklı açılardan çekilen imgelerinin içerisinde barındırdığı tüm geometrik bilgi *F* matrisi ile hesaplanmaktadır.

İlk resimdeki bir noktadan ikinci resimdeki ilgili nokta epipolar çizgiye eşlenir:

$$F(x) : x \rightarrow l' \tag{2.15}$$

X noktasının 3 boyutlu uzayındaki geri yansıyan ışını X (λ), projeksiyon matrisi P'nin tersi P^+ 'sini kullanarak elde edilebilir:

$$X(\lambda) = P^+ x + \lambda C \tag{2.16}$$

Burada C = X (∞), P'nin optik merkezidir. Daha sonra, x'in epipolar çizgisi l', o çizgiye rasgele bir nokta ile $e' = P'C$ ile birleştirerek hesaplanabilir:

$$l' = (P'C) \times (P' X (\lambda)) \tag{2.17}$$

Burada, (P′C) x (P′C) = 0 olması ilginçtir ve doğrudan:

$$=[e']_x(P' P^+) x \tag{2.18}$$

Bu gösterimde, $x = (x_1, x_2, x_3)^T$ vektörünün eğrilik simetrik matrisi $[x]_x$ kullanılmıştır ve aşağıdaki gösterilmektedir:

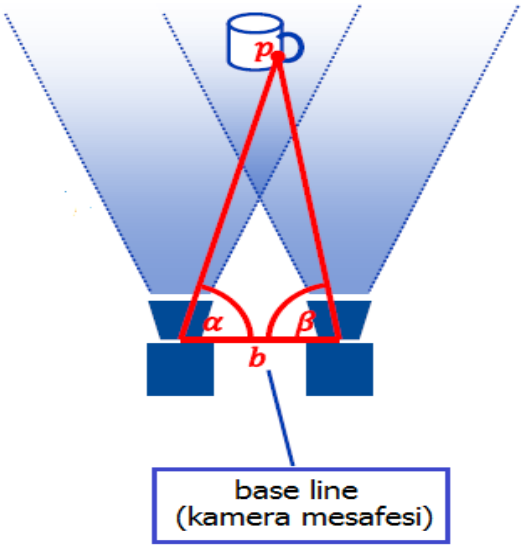
$$[x]_x = \begin{pmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{pmatrix} \tag{2.19}$$

Böylelikle, F matrisi aşağıdaki hesaplanmaktadır:

$$F = [e']_x \, P' \, P^+ \tag{2.20}$$

2.2.3. Stereo görme

Derinlik algılamada temel prensip olan stereo görme aynı nesneye ait, aralarındaki uzaklık bilinen, kalibre edilmiş iki kameradan farklı açılardan alınan resimlerin karşılaştırılması prensibine dayanmaktadır [8].



Şekil 2.6. *Stereo Görme (İkili Görü) Prensibi [8]*

P noktası her iki kamerada bir miktar kaymış olarak görünmektedir. Her iki kamera arasındaki mesafe, açı, odak mesafe biliniyorsa, kaymanın kaç piksel olduğu da hesaplanabilmektedir. Bu kayma değerine eşitsizlik (*ing.* disparity) denir.

2.2.3.1. *Stereo geometri*

Bir stereo geometri, aynı odak uzunluğuna sahip iki iğne deliği kameranın, adı verilen kanonik bir konfigürasyonda yerleştirilmesiyle oluşturulmuştur. Bu, iki

kameranın taban mesafesi olarak adlandırılan bir mesafeye kaydırıldığı anlamına gelir. Kamera görüntü düzlemleri ve optik eksen paraleldir.

Uzayda bir nokta $\bar{p}$ = (X, Y, Z) ^{T} koordinatında düşünülmektedir. Bu durumda, aşağıdaki ilişki doğrudan geometriden gelmektedir.

$$\frac{\frac{b}{2}+X}{Z} = \frac{x_l}{f} \tag{2.21}$$

$$\frac{\frac{b}{2}-X}{Z} = \frac{x_r}{f} \tag{2.22}$$

Bu eşitliklerin birlikte eklenerek ve d = x_l - x_r tanımlanarak aşağıdaki denklem üretilmektedir:

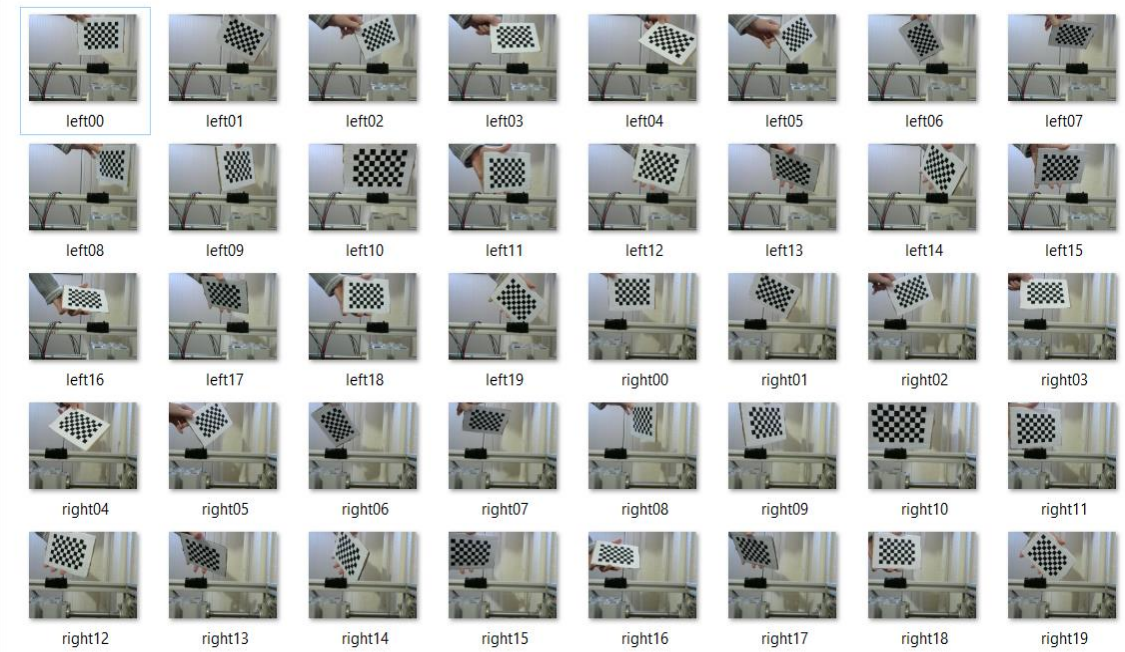
$$Z = \frac{bf}{\mathrm{xl}-\mathrm{xr}} = \frac{bf}{d} \tag{2.23}$$

Dolayısıyla sol resimdeki bir pikselin sağdaki görüntüye ne kadar taşındığını tespit etmek mümkünse, karşılık gelen noktanın derinliğini hesaplamakta basittir.

Bununla birlikte, bu tanımla ilgili birkaç problem var. İlki, tanıma göre, her iki görüntü düzlemi paralel olmalıdır. Bu genellikle pratik ve mümkün değildir. İkincisi, tanımı iğne deliği kamera modeline dayanır, çünkü gerçek bir kamera objektif bozulmasına ve projeksiyon merkezinin yer değiştirmesine maruz kalmaktadır. Bu problemlerin bir çözümüne görüntü düzeltme denir.

2.2.3.2. *Stereo kalibrasyon*

Stereo görüş tekniklerinin doğrulukla uygulanabilmesi için kameraların kalibre edilmesi gerekmektedir. Gerçekte hiçbir lens mükemmel değildir ve görüntüde üretim sürecinden kaynaklanan bozulmalar meydana gelir [9].

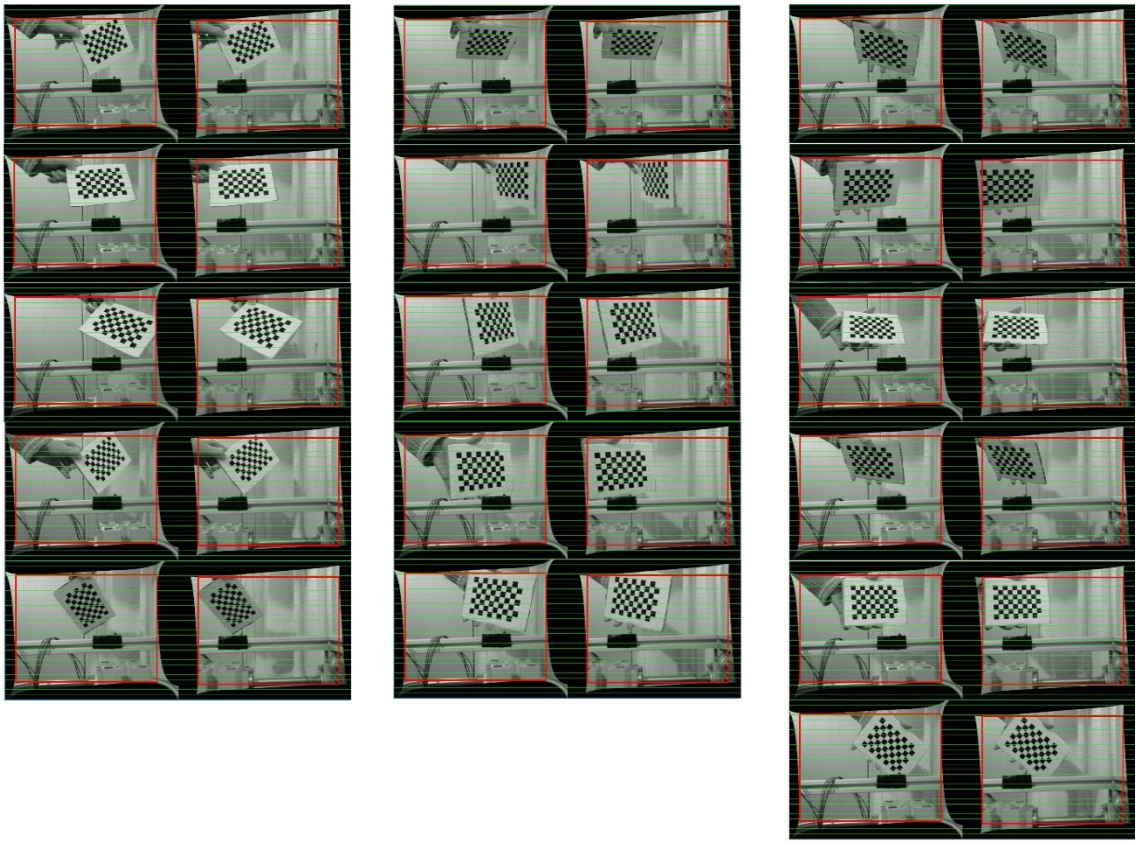


Görsel 2.3. *Sağ ve Sol Kamera Resimleri*

Kameralar ölçüleri bilinen bir kalibrasyon nesnesinin farklı açılardan alınmış görüntüleri kullanılarak kalibre edilir. Kalibrasyon sonrasında elde edilen iç parametreler lens bozukluklarının düzeltilmesinde kullanılır. Dış parametreler ise kameralar arasındaki birbirlerine göreceli geometrik ilişkiyi göstermektedir. Stereo düzeltme işleminin ve derinlik kestiriminin gerçekleştirilebilmesi için kameralar arasındaki geometrik ilişkinin bilinmesi gereklidir.

2.2.3.3. *Düzeltme*

Genel olarak, bir epipolar çizgi, görüntü alanında etkili stereo uygulamalarını engelleyen herhangi bir eğime sahip olabilir. Pratik açıdan, eşleştirmeler (*ing.* correspondance) ararken ilk görüntünün her piksel konumu için epipolar çizgi hesaplanmalıdır. Bu, bir vektör matris çarpımı ve epipolar çizgi boyunca ikinci görüntünün ayrık piksel konumları aracılığıyla önemli bir iterasyon içermektedir. Bir epipolar geometrinin en çok tercih edilen motivasyonundan yola çıkarak, ana fikir her iki görüntüyü de epipolar çizgiler yatay olacak ve görüntüler arasında eşleştirilecek şekilde dönüştürmek düşüncesindedir. Bu, karşılıklı gelen noktaların aynı y koordinatına sahip olduğu anlamına gelir. Bu durum her iki kameranın görüntü düzlemleri aynı düzlemde olduğunda ve taban çizgisi x eksenlerine paralel olduğunda söz konusu olmaktadır.



Görsel 2.4. *Stereo Düzeltme*

Düzeltmenin (*ing.* rectification) arkasındaki temel fikir, bir homografi kullanarak epipolü sonsuzluğa eşleştirmektir. Bu sadece iki serbestlik (*ing.* freedom) derecesini sınırladığından, homografiler genellikle görüntü bozulması mümkün olduğunca küçük olacak şekildedir. Standart yaklaşım, epipolü sonsuza $(1; 0; 0)^T$ noktasına haritalanan ilk resim için bir homografi H′ yaratmaktır. Ardından, ikinci görüntü için, eşdeğer bir dönüşüm H, epipolar çizgileri eşleştirmek için H′ temel alınarak hesaplanır. Dahası, H iki görüntü arasındaki yatay yer değiştirmeleri en aza indirgemek üzere seçilmektedir. Başka bir olasılık, bu noktaların sıfır eşitsizlik farkına sahip olmasını sağlayarak, düzlemdeki noktaları sonsuza kadar kullanmaktır.

Düzeltme işleminden sonra iki kamera aynı yönlendirmeye, aynı içsel parametrelere sahiptir ve optik eksenleri taban çizgisine diktir:

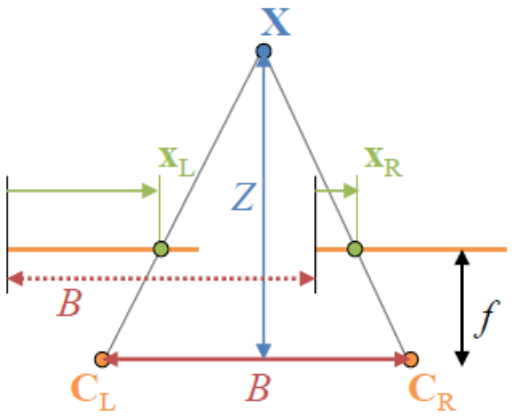
$$P = KR(I \mid -C) \qquad P' = KR(I \mid -C') \tag{2.24}$$

ve

$$R(C' - C) \times (1, 0, 0)^T = 0 \tag{2.25}$$

2.2.3.4. *Eşitsizlik*

Düzeltilmiş kamera ayarı, bir sol ve sağ kameradan oluşur ve karşılık gelen görüntü noktalarının $x_L \leftrightarrow x_R$ koordinatları, yalnızca x koordinatında farklılık gösterir [10]. İki değerin farkına eşitsizlik (*ing.* disparity) denir ve her piksel görüntü eşitsizlik haritasında saklanır: $D(x_L; y_L) = x_L - x_R$.



Şekil 2.7. *Eşitsizlik [10]*

Derinlik ve eşitsizlik arasındaki ilişki: Z derinlikli bir 3 boyutlu X noktası, taban çizgisi B olan bir sol ve sağ kamerasından gözlemlenir. Odak uzaklığı f ve optik merkezler C_L ve C_R'dir. X'in projeksiyonları sol ve sağ kamera için x_L ve x_R ile verilir.

Genel olarak, eşitsizlik nokta derinliği ile ters orantılıdır. $\Delta_Z = \Delta(X\ C_L\ C_R)$ ve $\Delta d = \Delta\ (X\ x_L\ x_R)$ üçgenlerinin uyumlu olduğu fark edilmektedir. Taban çizgisi ve görüntü düzlemlerinin paralel olması nedeniyle, kesişme teoremi uygulanabilir: ΔZ'nin yüksekliği ve temeli doğrudan Z ve B tarafından verilirken, Δd için Z - f ve $(B + x_R) - x_L$ olarak hesaplanmalıdır. Bu yüzden, $d = x_L - x_R$ ile şunu elde ederiz:

$$\frac{B}{Z} = \frac{B-d}{Z-f} \qquad d = \frac{fB}{Z} \tag{2.26}$$

Açıkçası, yakın görüntü noktaları uzak olanlardan daha büyük bir eşitsizliğe sahiptir ve sonsuzluktaki düzlemdeki noktalar sıfır eşitsizliğe sahiptir.

2.2.3.5. *Stereo eşleme ve derinlik kestirimi*

Bir sahnenin 3 boyutlu yapısını elde edebilmek için farklı bakış açılarından çekilmiş stereo görüntülerinde görülen her noktanın uzaydaki koordinatlarının elde

edilmesi gerekir [11]. Bir noktanın 3 boyutlu uzaydaki yerini bulmak için bir görüntü üzerindeki her bir noktanın diğer görüntü üzerindeki eşinin bulunması gereklidir. Stereo eşleme problemi aslında bir arama problemi olarak düşünülebilir.

Stereo eşleştirme algoritmalarının büyük bölümümün amacı genel olarak sol görüntüyü referans görüntü olarak belirleyerek, her bir piksel için bir eşitsizlik kestirimi yapmaktır. Bu aykırılıklar sağ ve sol kameralarda görülen özelliklerin resim düzlemindeki koordinatları arasındaki farktır. Eşitsizlik kestirimleri gözlemlenen nesnelerin ters çevrilmiş uzaklıkları olarak yorumlanır [12].

Bir stereo görüş sisteminin derinlik kestirimi yapabilmesi için yerine getirmesi gereken bazı adımlar vardır. Stereo eşleme yapılabilmesi için kameraların stereo kalibrasyon ve görüntülerin stereo düzeltme işlemlerinin de yapılmış olması gerekmektedir. Ancak kalibre edilmiş ve düzeltilmiş görüntüler üzerinde sağlıklı bir derinlik kestirimi yapılabilir.

2.3. Pozisyon Tahmini (*ing.* Pose Estimation) Algoritmaları

Bu bölümde, bu tez çalışmasında karşılaştırılan algoritmalar daha ayrıntılı olarak açıklanmıştır. Nokta tabanlı pozisyon tahmini yapan doğrudan lineer dönüşüm (DLT) [13], Etkili perspektif-n-nokta (*ing.* efficient perspektif-n-point) (EPnP) [14] algoritmalarıdır.

Pozisyon tahmininde 3D-2D noktaları eşleştirilmesi literatürde yaygın kullanılan bir yöntemdir. Bu yöntemde hedef, kamera pozisyonunun ve kamera kalibrasyon parametrelerinin altı derecesini tahmin etmektir: odak uzaklığı (*ing*. focal length), ana nokta (*ing*. principal point), en-boy oranı (*ing*. aspect ratio) ve eğrilik (*ing*. skew). Bu bilinen algoritmalardan olan Doğrudan Lineer Dönüşüm (DLT) (*ing*. direct linear transform) kullanılarak yapılmaktadır.

DLT'nin bir varyantı, kamera iç parametrelerinin bilinmekte olduğunu varsayan Perspektif-n-nokta (*ing*. perspektif-n-point) problemidir. Sistemi çözmek için birden fazla nokta tanıtıldığında, 3D noktaları ve onların 2D projeksiyonları arasında n eşleştirme kümesi verilen konum ve yönü belirleyen Perspektif-n-nokta problemi ortaya çıkmaktadır. Bu nedenle bu bölümünde kullanılan DLT, EPnp gibi pozisyon tahmini algoritmaları hakkında bilgi verilmiştir.

2.3.1. Doğrudan lineer dönüşüm (DLT)

Doğrudan Lineer Dönüşüm, pozisyon kurtarmada başlangıç noktasıdır. Önce fotogrametristler tarafından kullanılan ve bilgisayar görme topluluğunda tanıtılan bu algoritma, bilinmeyen kamera parametreleriyle doğrusal bir denklem sistemini çözerek projeksiyon matrisini (P) tahmin edilebilmektedir [13].

Her Mi ↔ mi eşleşmesi için, doğrusal olarak bağımsız iki denklem aşağıdaki gibi yazılabilir:

$$u_i = \frac{P11Xi + P12Yi + P13Zi + P14}{P31Xi + P32Yi + P33Zi + P34}$$

$$vi = \frac{P21Xi + P22Yi + P23Zi + P24}{P31Xi + P32Yi + P33Zi + P34}$$

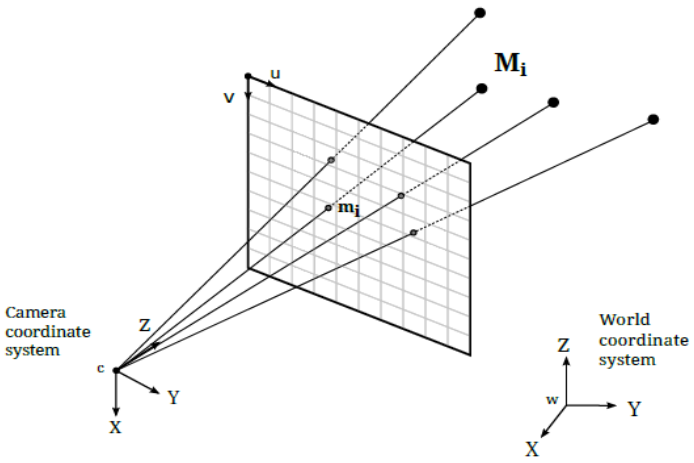
Bu sistem Ap = 0 formunda yazılabilir, burada p, Pij katsayıları ile oluşturulan bir vektördür. Bu sistemin çözümü, öz (*ing.* eigen) vektörü asgari öz değeri ile alarak A'nın Tekil Değer Ayrışımı (*ing.* Singular Value Decomposition) (SVD) 'nden bulunabilir.

Kameranın pozisyonunu düzeltmek için P'den bir ölçek faktörüne aşağıdaki gibi çıkartılması için kamera kalibrasyon matrisi K'ya ihtiyaç duyulmaktadır: $[R \mid t] \sim K^{-1} P$. Son olarak, 3x3 dönüş matrisi, bir düzeltme adımı uygulayarak ilk üç sütundan hesaplanabilir.

2.3.2. Perspektif-n-Nokta problemi

DLT, kamera içselliğini bilmeden ve yalnızca tek bir noktayı kullanarak projeksiyon matrisinin 11 parametresini tahmin ettiğinden çevrimdışı bir süreçte iç parametrelerin tahmin edilmesi gerekir. Bununla birlikte, bu bilgiler sisteme ekstra noktaların eklenmesi ile kullanılabilir ve bu da kameranın pozisyon tahmini işlemini daha sağlam hale getirmektedir [14].

Sistemi çözmek için birden fazla nokta tanıtıldığında, sorun Perspektif N Nokta Problemi (PnP) diye adlandırılan, 3D noktaları ve onların 2D projeksiyonları arasında n eşleştirme kümesi verilen konum ve yönü belirleyen probleme dönüşmektedir.

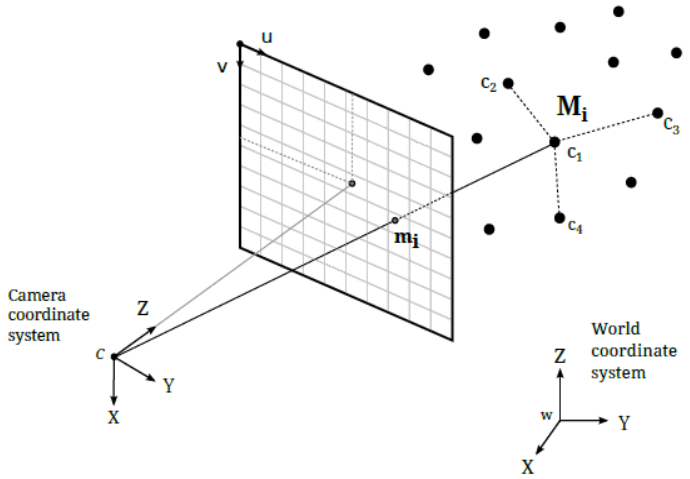


Şekil 2.8. *PnP Problem Şeması [3]*

Referans metot, probleme dört çözüm üreten kosinüs yasasını kullanarak tekrarlamalı olmayan (*ing.* noniterative) perspektif-3-nokta (P3P) problemidir [15].

2.3.2.1. *EPnP*

F.Moreno et al.'s [14] tarafından tanıtılan , "EPnP: An Accurate Non-Iterative O(n) Solution to the PnP Problem" kamera parametrelerini ve dünya koordinat sistemindeki 3D koordinatları ve 2D görüntü projeksiyonları bilindiği varsayılarak kamera pozisyonunu etkili bir şekilde çözmektedir.



Şekil 2.9. *EPnP Problem Şeması [3]*

Problem formülasyonu;

$$P_i^w = \sum_{i=1}^{4} a_{ij}\, c_j^w \tag{2.27}$$

Burada P_i^w = [X^w; Y ^{w}; Z^w] ^{T}, dünya koordinat sisteminde bir 3D nokta, ij homojen ağırlık merkezli koordinatları ve c_j^w = [X^w; Y ^{w}; Z^w] ^{T}, dünya koordinatlarında bir 3D kontrol noktasıdır. Daha sonra, kamera koordinatları c_j^c'deki 4 kontrol noktası, sorunun bilinmeyene dönüşür ve toplam 12 bilinmeyen olur.

DLT'ye benzer şekilde, kontrol noktaları referans çerçevesinde doğrusal bir sistem oluşturmak için gerekmektedir.

$$\forall_i w_i \begin{bmatrix} u_i \\ 1 \end{bmatrix} = K\, p_i^c = K \sum_{i=1}^{4} a_{ij}\, c_j^c \tag{2.28}$$

Burada, w_i skalar projektif parametrelerdir, u_i 2D koordinatlarıdır [u_i; v_i] ^{T}, K kamera parametreleri matrisidir. Bu ifade aşağıdaki gibi yeniden yazılabilir;

$$\forall_i w_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_c \\ 0 & f_v & v_c \\ 0 & 0 & 1 \end{bmatrix} \sum_{i=1}^{4} a_{ij}\, c_j^c \begin{bmatrix} x_j^c \\ y_j^c \\ z_j^c \end{bmatrix} \tag{2.29}$$

2.29'den iki doğrusal bağımsız denklem elde edilmektedir:

$$\sum_{i=1}^{4} a_{ij}\, f_u x_j^c + a_{ij}\,(u_c - u_i)\, z_j^c = 0,$$

$$\sum_{i=1}^{4} a_{ij}\, f_v y_j^c + a_{ij}\,(v_c - v_i)\, z_j^c = 0,$$

Böylelikle, aşağıdaki şekilde doğrusal bir sistem oluşturulur:

$$\mathrm{M}x = 0 \tag{2.30}$$

Burada M, bilinen katsayıları olan bir 2nx12 matrisidir ve x = $[c_1^c, c_2^c, c_3^c, c_4^c]^T$ bilinmeyenlerden oluşan 12-vektörüdür.

Bu sistemin çözümü M'nin boş alanı veya çekirdeği üzerinde uzanmaktadır. Aşağıdaki gibi ifade edilir:

$$x = \sum_{i=1}^{N} \beta_i\, v_i \tag{2.31}$$

Burada v_i, M'nin N boş öz değerlerine tekabül eden, doğru öz vektörleridir. EPnP'nin etkinliği, öz vektörlerin hesaplanmasından önce M'nin 12x12 boyutundaki küçük bir sabit matris M^TM'ye dönüştürülmesine kalmaktadır.

2.31'dan, teoride çözüm her bir N için b_i's ile verilecektir. Ancak, uygulamada çözüm yalnızca N = 1,..,4 için elde edilmektedir.

$$N = 1 : x = \beta_1 v_1$$
$$N = 2 : x = \beta_1 v_1 + \beta_2 v_2$$
$$N = 3 : x = \beta_1 v_1 + \beta_2 v_2 + \beta_3 v_3$$
$$N = 4 : x = \beta_1 v_1 + \beta_2 v_2 + \beta_3 v_3 + \beta_4 v_4$$

Yöntem, kamera koordinat sistemindeki kontrol noktaları arasındaki mesafelerin, dünya koordinat sisteminde hesaplanan mesafelere eşit olması gerektiğini varsaymaktadır:

$$\left\|c_i^c - c_j^c\right\|^2 = \left\|c_i^w - c_j^w\right\|^2 \tag{2.32}$$

Basitlik açısından, yalnızca N = 1 durumu için:

$$\|\beta v^{[i]} - \beta v^{[j]}\|^2 = \left\|c_i^w - c_j^w\right\|^2 \tag{2.33}$$

Daha sonra beta aşağıdaki gibi hesaplanabilir:

$$\beta = \frac{\sum_{(i,j)\in[1;4]} \|v^{[i]} - v^{[j]}\| . \|c_i^w - c_j^w\|}{\sum_{(i,j)\in[1;4]} \|v^{[i]} - v^{[j]}\|^2} \tag{2.34}$$

Bir kere betalar hesaplandıktan sonra, ters işlem yapmak için kamera pozisyonuna ihtiyaç duyulur. Öncelikle, kamera çerçevesi referansındaki kontrol noktalarının koordinatlarını hesaplanır.

İkinci olarak, kamera çerçevesi referansındaki tüm 3D noktaların koordinatlarını hesaplanır ve son olarak, döndürme matrisi R'yi ve çeviri vektörünü t çıkarılır.

2.3.4. Sağlam tahmin (*ing.* Robust estimation)

Sağlam tahmin, kamera pozisyonu kaba hatalarla ortaya çıkartılan gürültülü veriyi olabildiğince kaldırarak hesaplamak için kullanılan bir metodolojidir. Bu sorunu çözmek için yaygın yöntemlerden biri olan ve reprojeksiyon hatasını en aza indirgeyen RANSAC algoritması kullanılmıştır.

Pozisyon tahmininde kameralar kullanıldığından, görüntülerde gürültü oluşması kaçınılmazdır. Benzeşmeleri bulmak için kullanılan yöntemler bazı uyumsuzluklar ve çukurluklar getirebilmektedir. Bu nedenle bu sonunun çözümleri kararsız ve güvenilmez olabilmektedir. Özellikler arasında sağlam eşleştirme kurma ihtiyacına karşın en yaygın çözüm olan RANSAC'ı kullanarak ek bir yinelemeli adım eklenmektedir.

Bilgisayar görmesinde veya robotik alanlarında algılama veya izleme tekniklerini uyguladıktan sonra bazı sensör hataları nedeniyle kötü sonuçlar elde edilmesi yaygın bir problemdir. İlerleyen bölümlerde bu kötü algılamalardan kaçınmak için kullanılan Lineer Kalman Filtresi hakkında bilgi verilmiştir.

2.3.4.1. *Doğrusal olmayan reprojeksiyon hatası*

Sonuçları iyileştirmek için önerilen bu metodoloji, M_i'nin üç noktasının projeksiyonuyla ve onun ölçülen 2D koordinatları arasındaki birikmiş kare uzaklık olan reprojeksiyon hatalarının toplamını en aza indirgeyerek tahmini kamera pozisyonunu yeniden ayarlamaktadır. Bu nedenle:

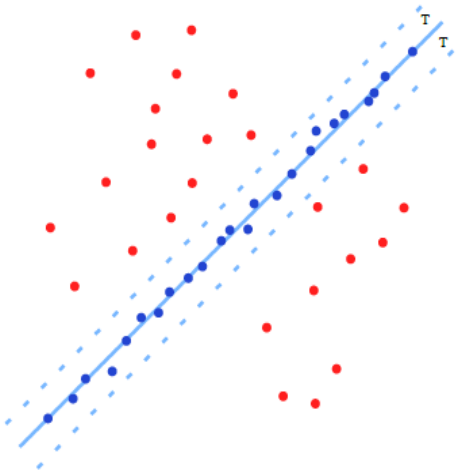
$$[R \mid t] = \underset{R,\,t}{min} \sum_{i=1}^{n} dist^2(PM_i' , m_i) \tag{2.35}$$

her ölçümün bağımsız ve Gauss olduğu gerçeğinden dolayı optimal olarak kabul edilebilir.

2.3.4.2. *RANSAC*

Random Sample Consensus veya RANSAC [16], yinelenmelerin sayısı arttıkça yaklaşık bir sonuç üreten gözlemlenen verilerden matematiksel modelin parametrelerini tahmin eden, deterministik olmayan bir yinelemeli yöntemdir.

Kamera pozisyonu tahmini bağlamında, parametrelerin başlangıcı tahmin edilmesine ihtiyaç olmadığından, uygulanması çok kolaydır. Algoritma rasgele olarak, hipotez adı verilen şeyi oluşturmak için küçük noktaların alt kümelerini çıkarır. Her bir hipotez için, daha sonra reprojeksiyon hatasını hesaplamak için kullanılan bir kamera pozisyonunu kurtarmak için bir PnP yaklaşımı kullanılır. Reprojeksiyonunun 2D noktalarına yeterince yakın olan noktalarına inlierler denilmektedir.



Şekil 2.10. *RANSAC*

RANSAC, bir noktanın reprojeksiyon hatasına dayalı olarak bir inlier olarak kabul edilip edilmeyeceğine karar veren tolerans hatası gibi bazı parametrelere bağlıdır. [16]'de, algoritmanın, hipotezden en az birinin tutarlı bir çözüm olarak başarılı olmasını istediği bir olasılık p'ye göre yapması gereken yineleme sayısını hesaplamak için bir formül önerilir. Formül aşağıdaki gibidir:

$$k = \frac{\log(1-p)}{\log(1-w^n)} \tag{2.36}$$

Burada w, inlierler ile noktaların sayısı arasındaki orandır. Alt kümelerin boyutu arttıkça k değeri artmaya eğilim gösterir.

2.3.4.3. *Bayes izleme (ing. Bayesian Tracking)*

İzleme algoritmaları, olası kamera pozisyonu alanındaki ardışık durum s_t yoğunluğunu tahmin etmek için kullanışlıdır. Kullanılan modele bağlı olarak, durum vektörleri, rotasyon ve çeviri (*ing.* translation) parametrelerini ve çoğunlukla çeviri ve açısal (*ing.* angular) hızlar gibi ilave parametreleri içermektedir. Bayes izlemeler şu şekilde formüle edilebilirler:

$$p(s_t \mid z_t \dots z_0) = \int_{s_{t-1}} p((s_t \mid s_{t-1}) p_{t-1}(s_{t-1}) \tag{2.37}$$

Burada, $\int_{s_{t-1}}$, önceki s_{t-1} durumları için olası değerler seti üzerinden entegrasyonudur. $p(s_t \mid z_t \dots z_0)$ terimi, önceki yoğunluk durumuna $p_{t-1}(s_{t-1})$ hareket modeli uygulayarak yapılan $p_t(s_t)$ üzerinde bir tahmin olarak yorumlanabilir.

2.3.4.3.1. *Kalman filtresi*

Kalman Filtresi, durum uzayı modeli ile gösterilen bir dinamik sistemde, modelin önceki bilgileriyle birlikte giriş ve çıkış bilgilerinden sistemin durumlarını tahmin edilebilen filtredir. Macar asıllı Amerikan matematiksel sistem teoristi Rudolf Kalman tarafından bulunmuştur.

$$s_t = As_{t-1} + w_t \tag{2.38}$$

Burada A, durum geçiş matrisi (*ing.* state transition matrix) olarak adlandırılır ve w_t, sıfır ortalamayla normal olarak dağıldığı varsayılan işlem gürültüsünü (*ing.* noise) temsil eder. Durum vektörü, kamera pozisyonunun 6 parametresine ek olarak çevrilebilir (*ing.* translational) ve açısal hızlar tarafından oluşturulmaktadır.

t zamanda kamera pozisyonu gibi z_t ölçümlerinin, doğrusal bir ölçüm modeli tarafından s_t durumu ile ilişkili olduğu varsayılmaktadır.

$$z_t = Cs_t + v_t \tag{2.39}$$

Burada v_t gürültü ölçüsünü temsil etmektedir.

Her zaman adımında, Kalman Filtresi a önceki (*ing.* priori) durum s_t^- tahmini olarak adlandırılan mevcut durumun birincil tahminini yapar. s_t ve onun kovaryans matrisi S_t , tahmin işlemi sırasında hesaplanmaktadır ve aşağıdaki gibi yazılabilir:

$$s_t^- = As_{t-1}\,, \tag{2.40}$$

$$S_t^- = AS_{t-1}\,A^T + \Lambda_w\,, \tag{2.41}$$

Burada S_{t-1} , önceki zaman adımı için bir posteriori tahmin hatası kovaryansıdır ve Λ_w gerçekle ilgili hareket modelinin kalitesini ölçen süreç kovaryans gürültüsüdür. Ardından, Kalman Filtresi bir "ölçüm güncelleme" ya da düzeltme yapar. Bir tümevarımsal (*ing.* posteriori) durum tahmini s_t ve onun kovaryans matrisi S_t , z_t ölçümlerini ekleyerek üretilmektedir.

$$s_t = s_t^- + G_t(z_t - Gs_t^-), \tag{2.42}$$

$$S_t = S_t^- - G_tCS_t^-\,, \tag{2.43}$$

Burada Kalman kazancı G_t hesaplanır ölçüm kovaryans matrisi olan Λ_v ile hesaplanır:

$$G_t = S_t^- C^T (C S_t^- C^T + \Lambda_v)^{-1} \quad (2.44)$$

3D izleme bağlamında, kamera dış parametrelerini tahmin etmek için a önceki durum s_t^- tahmini kullanılabilir. Bu nedenle, tahmin edilen ölçüm vektörü z_t^- aşağıdaki gibidir:

$$z_t^- = C s_t^- \quad (2.45)$$

Tahmin üzerindeki belirsizlik, belirsizliği yayarak tahmin edilen kovaryans matrisi Λ_z ile gösterilir.

$$\Lambda_z = C S_t^- C^T + \Lambda_v \quad (2.46)$$

2.4. İlgili Çalışmalar

Bölüm 2.1'de kamera modeli ve radyal objektif bozulması kullanan görüntülerden, perspektif geometriden oluşan tek görünüm geometrisi açıklanmıştı. Bölüm 2.2.'de ise epipolar geometri, düzeltme ve eşitsizlik (*ing*. disparity) tanımını içeren çoklu görüş geometrisi üzerine temel bilgiler verilmişti. Bölüm 2.3'de pozisyon tahmini algoritmaları hakkında bilgi verilmişti. Bu bölümde de literatürde bu çalışma ile ilgili yapılmış çalışmalar hakkında bilgi verilecektir.

Bilgisayarla görme literatüründe pozisyon tahmini için farklı yaklaşımlar vardır. Algoritmalar birçok farklı grupta sınıflandırılabilir.

Algoritmada kullanılan özelliklere (noktalar veya çizgiler) göre bir sınıflandırma yapılabilir. Birçok algoritma, pozisyon tahmini için nokta özelliklerini kullanmaktadır, ancak satır özelliklerini de kullanan bazı algoritmalar [17] [18] [19] vardır.

Algoritmalar aynı zamanda giriş gereksinimlerine göre de sınıflandırılmaktadırlar. Birçok algoritma, pozisyon tahmini için 2D-3D nokta karşılıklarını bilinmesini gerektirir [13] [20].

2009'da A. Lipnickas ve A. Knys "A Stereovision System for 3D Perception" adlı çalışmalarında stereo görme tabanlı 3D algılama üzerine genel bilgiler vermişlerdir [21]. Stereo uygulamalar girdi olarak iki boyutlu resim çiftleri almakta ve çakışan noktaları bularak üç boyutlu veriler elde edilmektedir. Bu uygulamaların doğruluğu,

eşitsizlik değerinin doğruluğu, stereo sistem kalibrasyonu, görüntü düzeltme işlemi ve genel sistem inşasına bağlanmaktadır. Birçok kamera kalibrasyonu metodu kalibrasyonu içsel ve dışsal parametreler ile çözmektedir. Birçok yapılmış çalışmada stereo düzeltme metodu iğne deliği kamera modeli ve paralel geometriye dayandırılır. Bu yüzden, verilen iki nokta koordinatları herhangi bir stereo eşleme işlemi tarafından belirlenmektedir. Derinlik noktaların eşitsizlik değerinden veya resim piksel koordinatları içerisinde iki noktanın ayrıştırılması ile tespit edilmektedir. Bu çalışmada piksel koordinatları; sol resim için (X_L, Y_L), sağ resim için (X_R, Y_R), 3D koordinat (X, Y, Z), $d = X_L - X_R$, b = paralaks, f = kamera lenslerinin odak uzunluğu olarak ifade edilmektedir.

$$X = (X_L b) / d \tag{2.47}$$

$$Y = (Y_L b) / d \tag{2.48}$$

$$Z = (f b) / d \tag{2.49}$$

Yukarıdaki formüllerde, eşitsizlik olarak adlandırılan d piksel cinsinden sol ve sağ kameradaki piksellerin farkını, X_L sol kameradaki piksel değerini, X_R sağ kameradaki piksel değerini vermektedir.

Stere görme prensibi kullanılarak yapılmış çalışmalarda genellikle yukarıdaki formüller kullanılarak konum tespiti yapıldığı gözlemlenmiştir.

2012'de Güven Çetinkaya, önceden önerilmiş ve model noktaları ile görüntü noktaları arasındaki ilişkinin bilindiğini varsayan dört pozisyon tahmini algoritması; Orthogonal Iterations, POSIT, DLT ve Efficient PnP, kodlamış ve karşılaştırmıştır [22]. Pozisyon ve eşleştirme problemlerini aynı anda çözen iki bilinen algoritmanın; Soft-POSIT ve Blind-PnP, kodlamış ve karşılaştırmıştır. İlk adımda, gerçek hareket senaryoları kullanılarak sentetik veriler üretilmiş ve algoritmalar bu veriler kullanılarak karşılaştırmıştır. Sonraki adımda, üç boyutlu modeli bilinen bir nesnenin kalibre edilmiş bir kamera ile çekilmiş gerçek görüntülerinden faydalanmıştır. Simülasyon sonuçlarına göre, model noktaları ve görüntü noktaları arasındaki ilişkinin bilindiğini varsayan algoritmalar arasında POSIT algoritmasının en iyi performansı gösterdiği sonucuna varmıştır. Deneyler sonucunda Soft-POSIT algoritmasının Blind-PnP algoritmasına göre daha iyi bir performans gösterdiği sonucuna varmıştır.

Changhyun Choi, Seung-Min Baek ve Sukhan Lee yaptıkları çalışmada gerçek zamanlı 3D pozisyon tahmini problemine bir yaklaşım sunmuşlardır [28]. Bu çalışmada Kante-Lucas-Tomasi (KLT) olarak adlandırılan izleme algoritması kullanılarak 3D pozisyon tahmini yapılmıştır. KLT ile izleme noktalarını takip ederek, izleme noktası dış hatlarını (*ing.* outlier) otomatik olarak kaldırarak ve izleme noktalarını SIFT kullanarak yeniden başlatarak gerçek zamanlı olarak nesnenin 3D pozisyon tahmini yapılmıştır. Bu metot hem tek hem çift kamera tabanlı pozisyon tahminine uygulanmıştır. Sonuç olarak tek kamera modu yüksek kare hızı performansını garanti etmiş ve stereo modu daha iyi pozisyon tahmini sonuçları göstermiştir.

2015'de Edgar Riba Pi, 3D pozisyon tahmin algoritmasının uygulanması üzerine bir tez çalışması yapmıştır [25]. Bu çalışmada tek bir kamera kullanılarak katı nesneler için 3D pozisyon tahmin algoritmasının uygulanmıştır. Doğal özellik noktaları ile dokulu 3D model arasındaki uyuma dayanan algoritma, bir PnP yöntemi kullanarak belirli bir nesnenin 3D pozisyonunu verimli bir şekilde bulmaktadır. Ayrıca, bu proje sırasında bu projenin yöneticileri tarafından yayınlanan UPnP yaklaşımının bir C++ uygulaması gerçekleştirilmiştir.

Kinect sensörü Microsoft™ oyun konsolu XBOX 360 ile kullanılmak üzere geliştirilmiştir. Kinect sensörü, Kinect SDK ile Windows uygulamalarında da kullanılabilmektedir. Kinect SDK ile kameradan görüntü alınabilir, her bir noktanın sensörden uzaklığına erişilebilir, 20 farklı iskelet özelliğinin X, Y ve Z koordinatını elde edilebilir, mikrofon dizisini kullanılabilir ve cihazın açısını kontrol edilebilir. Kinect'in üzerinde 3 adet göz, sıra mikrofonlar ve hareket sağlayıcı bir motor mekanizması bulunmaktadır. Soldaki göz lazer projeksiyonu yapmaktadır ve sağdaki kızılötesi sensör bu ışınların gidiş - geliş süresini hesaplayarak 320x240 çözünürlüğünde her bir noktanın mesafesini bildirmektedir.

2013'de yapılan bir çalışmada [23], Microsoft Kinect Sensör kullanılarak 3D yeniden yapılandırma yapılmıştır. Bu işlem dört adımda yapılmıştır;

1. Veri ön işleme,
2. Sensörün pozisyon tahminini ortaya koymak,
3. Derinlik verilerinin kaynaştırılması,
4. 3D yüzey oluşturma.

İlk olarak nokta bulutu oluşturmak için derinlik verisinin her karesi toplanmıştır. İkinci olarak, ICP algoritması kullanılarak sensörün pozisyonu tahmin edilmiştir.

Üçüncü olarak, volumetrik metod kullanılan derinlik verilerinin tamamı kombinlenmiştir. Son olarak ise global modelden 3D yüzey çıkarılmıştır.

2014'de yapılan bir çalışmada [24], tıbbi uygulamalarda yükseklik-derinlik tahmini için değerlendirme kriterleri tanımlanmıştır. Bir ultra geniş bant kapalı konumlandırma sistemi, bir optik Microsoft Kinect kamera sistemi ve bu kriterlere karşı kendi geliştirdikleri kablosuz bir barometrik sensör karşılaştırılmıştır. Sonuç olarak barometrik iki sensör sisteminin düşük form faktörü ve iyi kullanılabilirlik ile iyi bir doğruluk sağladığını göstermiştir.

Akademik Platform Dergisinde yayınlanmış bir çalışma olan "OpenCV ve Kinect ile Stereo Görüntüden Derinlik Algılama" adlı çalışmada derinlik kestirimi problemi üzerinde durulmuştur [26]. Çalışma kapsamında yapılan uygulamada açık kaynak görüntü işleme kütüphanesi OpenCV ile farklı açılardan çekilmiş iki resimden derinlik bilgisi çıkarılmış, aynı işlem gerçek zamanlı görüntü üzerinden Kinect ile de yapılmıştır. Yapılan çalışma ile stereo görme görüntü alımında Kinect'in OpenCV'ye göre kamera kalibrasonu ve komut kütüphanesinin kullanımı açısından daha avantajlı olduğu görülmüştür. OpenCV'de iki adet kamera kullanımının bellek kullanımı açısından dezavantaj olduğu görülmüştür.

Kinect kamera kullanılarak birçok çalışma yapılmıştır. Bu çalışmalarda farklı yöntemler ile konum tespiti yapılmıştır. Bu çalışmalardan biri olan "Combining Kinect and PnP for Camera Pose Estimation" adlı çalışmada yazarlar Kinect kamera ve Pnp yöntemini kullanarak kamera pozisyonunu tahmin işlemi hakkında bilgi vermişlerdir [27]. Kinect kamera kalibre edilmiş derinlik kestirimi sonucunda konum tespiti yöntemleri uygulanmıştır. Kinect kamera resimleri hizalama işlemi yapılmış ve hizalamadan sonra RGB resmin içinden 2D noktaların her birinin koordinatı derinlik resim içindeki 2D koordinata eşleştirilir. Daha sonra uzaydaki 3D koordinat elde edilmiştir.

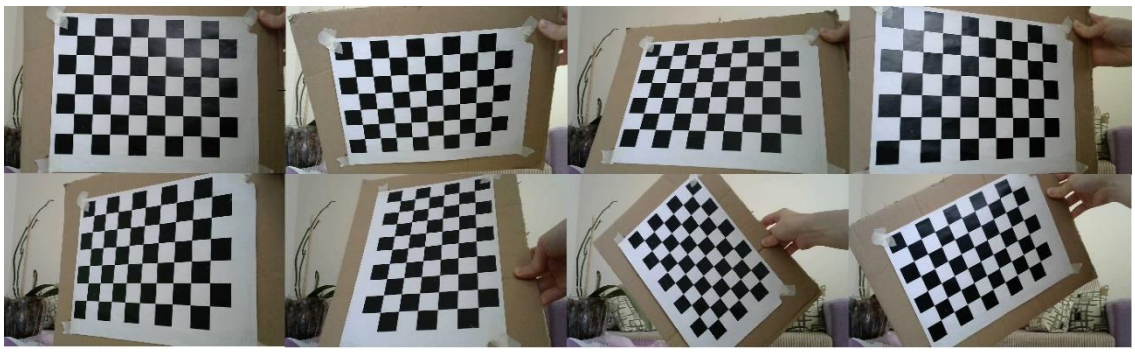
$$X_p / (u - u_0) = Y_p / (v - v_0) = Z_p / f \qquad (2.50)$$

Burada f kızılötesi kameranın odak uzaklığı, (u_0, v_0) kinect'in derinlik resminin merkezi, (u, v) derinlik resmindeki 2D nokta ve (X_p, Y_p, Z_p) 3D koordinatı ifade etmektedir. Yapılan çalışmalarda yukarıdaki formül (1.4) kullanılarak 3D konum tespiti yapılmıştır.

3. TEK KAMERA İLE POZİSYON TAHMİNİ

Bu bölümde, tek kamera kullanılarak nesnenin pozisyonun tespiti için yapılan işlemler açıklanmaktadır. Pozisyon tahmini için gereken ilk işlem kameranın kalibrasyonudur. Bu işlemden elde edilen parametreler sunulan yöntemlerde kullanılacaktır.

Kamera kalibrasyonu:



Görsel 3.1. *Kameradan Alınan Resimler*

Kameralar ölçüleri bilinen bir satranç tahtası modelinin farklı açılardan ve uzaklıklardan alınmış görüntüleri kullanılarak kalibre edilir. Kalibrasyon sonrasında elde edilen iç parametreler lens bozukluklarının düzeltilmesinde kullanılır.

Görsel 3.2. *Düzeltilmiş Kamera Görüntüleri*

Kalibrasyon sonucu elde edilen parametreler kaydedilir. Pozisyon çıkarımı yöntemlerinde bu parametreler kullanılarak kameranın bilgileri kullanılmış olacaktır.

Bölüm 3.1.'de açık kaynak kodlu görüntü işleme kütüphanesi olan OpenCV'den faydalanılarak doku kaplı nesneni pozisyon tespiti için yapılan işlemlerin aşamaları ayrıntılı olarak anlatılmıştır. Tek kamera göz önüne alınarak katı nesneler için bir 3D pozisyon tahmin algoritmasının uygulanması sunulmuştur.

Bölüm 3.2.'de küresel koordinat sistemi kullanılarak renkli bir nesnenin pozisyon tahmini için yapılan çalışma anlatılmıştır. Çalışmada kullanılacak olan işaretçiler ile ilgili bilgi verilmiştir. İki farklı işaretleyici yöntemi ele alınmıştır. İlk işaretçi OpenCV'nin sağladığı Aruco modülü diğeri ise satranç tahtasıdır. İki işaretçi içinde yapılan işlemler bu bölümde anlatılmıştır. Çalışma kapsamında, renk tabanlı algılama yaklaşımı kullanarak nesnenin yatay ve dikey düzlemdeki konumunun ve bu nesnenin derinlik bilgisinin gerçek zamanlı tespiti ve takibini yapacak yazılım çalışması yapılmıştır. Yazılım kısmı için Windows© 10 işletim sistemli, İntel® Core™ i7-4700HQ CPU @2.40GHz işlemciye sahip, 8,00 GB RAM ve 64 bit sisteme sahip bilgisayar altında çalışılmış olup, OpenCV 3.1 kütüphanesinden yararlanılmıştır.

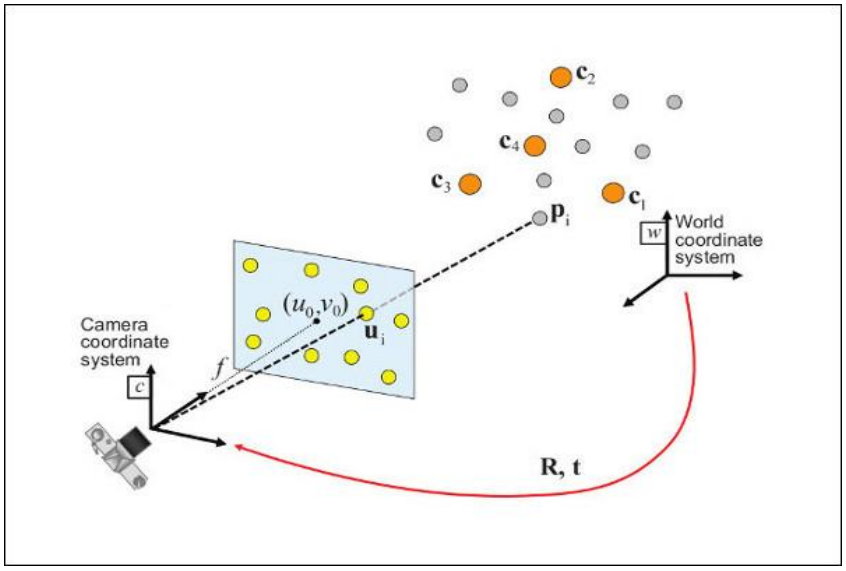
Bu çalışmada kullanılacak olan görüntü işleme konusunda 3.2.3 bölümünde açıklama yapılmıştır. Nesne takibinde kullanılan OpenCV kütüphanesi ve HSV renk uzayı hakkında 3.2.4 bölümünde açıklama yapılmıştır. Yazılımın gerçekleştirilmesi ve alt başlıkları ise 3.2.5 bölümünde detaylı bir şekilde anlatılmıştır.

Bu bölümün sonunda, bu yaklaşımın uygulanması ve deney sonuçları tartışılacaktır.

3.1. Doku Kaplı Bir Nesnenin Pozisyon Tahmini

Bu bölümde, 2 boyutlu görüntüsü ve 3 boyutlu dokulu modeli verilen, altı serbestlik derecesine sahip dokulu bir nesneyi izlemek amacıyla gerçek zamanlı bir uygulama nasıl oluşturulacağı anlatılacaktır. Altı serbestlik derecesi rotasyon ve çeviri vektörlerinden oluşmaktadır.

Bilgisayarlı görmede 3D-2D noktalı eşleştirmelerden kısaca görüntüdeki 2D noktaların 3D nokta bilgilerinin eşleştirmelerinden kamera pozisyonunun tahmin edilmesi temel bir problemdir. Sorunun en genel versiyonu pozisyonun altı serbestlik derecesini tahmin etmeyi ve beş kalibrasyon parametresini gerektirir: odak uzaklığı, temel nokta, en/boy oranı ve eğrilik. Bilinen Doğrudan doğrusal dönüşüm (DLT) algoritması kullanılarak en az altı eşleştirme kurulabilir. Ancak DLT algoritmasının doğruluğunu iyileştiren farklı algoritmalar vardır. Bunlardan en yaygını, perspektif n nokta problemidir.



Şekil 3.1. *Perspektif N Nokta Problemi*

Problem formülasyonu:

Dünya referans çerçevesi kameradan alınan görüntülerdir. Model referans noktaları ise elle kayıt edilen noktalardan oluşmaktadır. Model referans çerçevesi, model referans noktaları verilen noktaların, dünya referanslarında nereye geldiğinin eşleştirilmesidir.

Bir dünya referans çerçevesinde ifade edilen 3 boyutlu p_i noktaları ve onların görüntüler üzerindeki 2 boyutlu projeksiyonları arasındaki bir dizi eşleştirme verildiğinde, kameranın dünya pozisyonu (R ve t) ve odak uzunluğunu bulunabilir.

OpenCV, R ve t'yi döndüren Perspektif n nokta problemini çözmek için dört farklı yaklaşım sunmaktadır. Ardından, aşağıdaki formül kullanılarak 3 boyutlu noktaları görüntü düzlemine yansıtmak mümkündür:

$$s\, m' = A[\, R \mid t]M' \tag{3.1}$$

veya

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.2}$$

Burada:

- **(X, Y, Z),** dünya koordinat alanındaki bir 3B noktasının koordinatlarıdır,
- **(u, v)** piksel olarak projeksiyon noktasının koordinatlarıdır,
- **A**, bir kamera matrisi veya iç parametrelerin bir matrisidir,

- **(c_x, c_y)** genellikle resim merkezindeki temel noktadır,
- **f_x, f_y** piksel birimleri cinsinden ifade edilen odak uzunluklarıdır.

Böylece, kamera görüntüsü bir faktör ile ölçeklendirilirse, bu parametrelerin tümü aynı faktörle (sırasıyla çarpılır/bölünür) ölçeklendirilmelidir. İçsel parametreler matrisi görüntülenen sahneye bağlı değildir. Bu nedenle, bu matris bir kere hesaplandıktan sonra, odak uzaklığı sabitlendiğinde yeniden kullanılabilmektedir. Döndürme – çeviri matrisinin birleşimi [R | t] dışsal parametrelerin bir matrisi olarak adlandırılır. Bu matris durağan bir sahnede kameranın hareketini veya tam tersi, duran bir kameranın önünde bir nesnenin sabit hareketini tanımlamak için kullanılır. Yani, [R | t] bir (X, Y, Z) noktasının koordinatlarını kameraya göre sabitlenmiş bir koordinat sistemine çevirmektedir.

Doku kaplı nesnenin pozisyon tahmini aşağıdaki başlıklar altında toplanmaktadır.

3.1.1. Kamera görüntülerini alma

Bu kullanım durumunun amacı, bir dijital kameradan veya görüntü dizisinden bir görüntüyü yakalamak ve görüntünün her noktası için sayısal değerler görülen bilgilere dönüştürmektir.

3.1.2. Özellikleri hesaplama

Bölüm 2'de, denklemler için ilk önce 2D görüntü ile 3D model arasındaki eşleştirmeleri bulmanın gerekli olduğu açıklanmıştır. Bu nedenle görüntü verisinden doğal özelliklerin ne olduğunu tespit etmek gerekmektedir. Bu doğal özellikler veya anahtar noktalar, genellikle görüntü gradyanlarının hesaplanmasından elde edilen görüntü düzlemindeki tekil (*ing.* singular) konumlardır. Buna ek olarak, bulunan her bir anahtar nokta için bir yerel tanımlayıcı hesaplanır; bu sabit bir boyutta, bilgi çıkarma için kullanılan tekniğe bağlı olan, gradyan oryantasyonu, parlaklık gibi belirli bir konuma ilişkin bazı bilgiler sağlayacak bir vektördür.

3.1.3. Model kaydetme

Model kaydetme bu algoritmanın başarılı olmak için vazgeçilmez bir parçasıdır. Belirli bir nesneyi tanımak için bir modele ihtiyaç duyulduğu için, ilk adım onun

oluşturulmasıdır. Bu nedenle kayıt işlemi tespit işleminden önce ve çevrimdışı yapılmalıdır.

Model, farklı nesneler arasında ayrım yapmak için temel olacak nesneye ilişkin belirli bilgileri içeren bir dizi 2D öznitelik tanımlayıcısı tarafından oluşturulacaktır. Buna ek olarak, her tanımlayıcı, daha sonra kamera pozisyonunu kurtarmak için gereken 2D-3D karşılıklarının setini oluşturmak için kullanılacak nesne referans çerçevesine göre ilişkili bir 3D koordinatına sahip olacaktır.

3.1.4. Elle kaydetme

Nesne modelini oluşturmak için, modelin 2D özelliklerini ve bulunan her özellik için onun 3D konumunu hesaplamak için bir yazılım gerekmektedir. Karmaşık nesneler için hareketten yapılanma tekniğini (*ing.* Structure From Motion Technique) kullanarak yeniden yapılandırma algoritması gerekmektedir, ancak bu uygulamanın odak noktası değildir. Bu nedenle, nesnenin 3D kafes (*ing.* mesh) ve nesnenin bir veya daha fazla perspektif resmi gerektiren düzlemsel yüzeylere sahip nesneler için uygulama geliştirilmiştir. Uygulama 3D kafes bilgisini yükler ve fare ile köşelerin 2D konumlarını işaretlemeyi gerektirir (Görsel 3.3.)

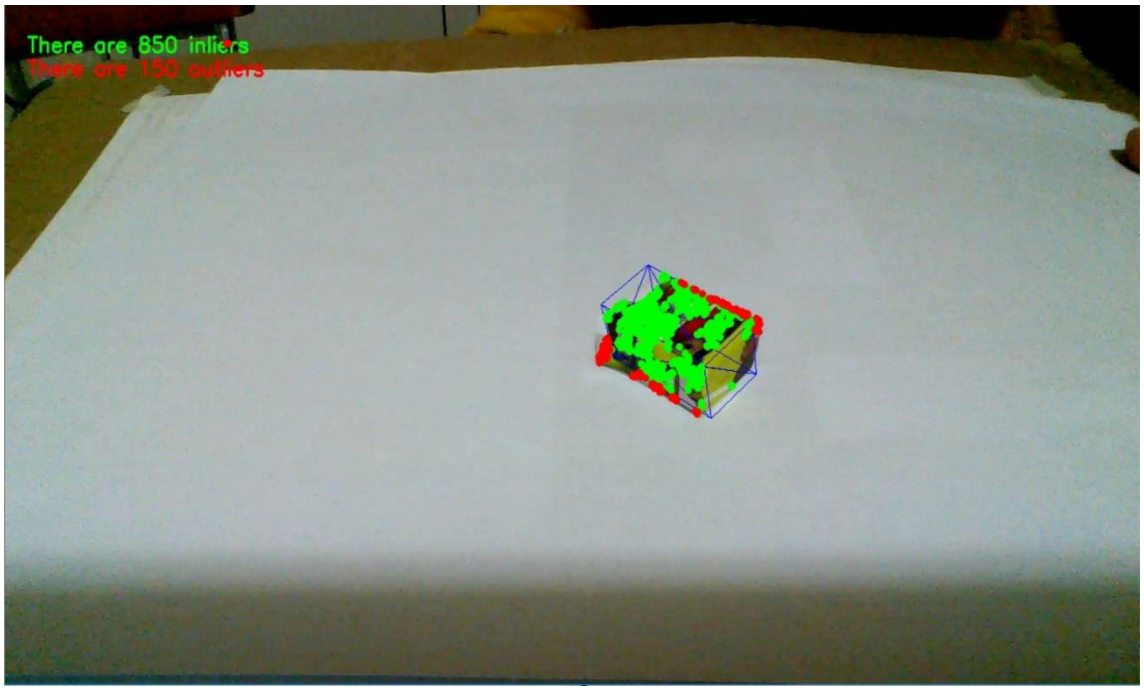
Görsel 3.3. *Fare ile Köşe Noktaları Tıklayarak Nesne Kayıt İşleminin Ekran Görüntüsü. (Kırmızı noktalar işaretlenen noktaları, yeşil noktalar tahmin edilen nokta pozisyonları temsil etmektedir.)*

3.1.5. Geometrik özellikleri çıkartma

Nesne köşeleri tanımlandıktan sonra, kamera pozisyonunu hesaplamak için PnP algoritmasını kullanarak yeterli 2D-3D eşleştirme setine sahip olunmaktadır (Bkz. Bölüm 3.1.7). Bir sonraki adım, 2D özelliklerin hangisinin nesne yüzeyine konumlandığını tespit etmektir. (Bkz. Bölüm 3.1.2)

3D koordinatların çıkarılması ile ilgili olarak, ışın yönü verilen, 3D düzlemde tanımlanan kesişme noktasını hesaplayan Möller-Trumbore intersection [29] algoritması uygulanmıştır. Görsel 3.4.'de elle kayıt işleminin sonucu görsel sonucu gösterilmiştir:

yeşil renkte, nesne yüzeyinde elde edilen 2D özellikler aynı zamanda bu özelliklerin 3D koordinatları ve tanımlayıcıları hesaplanmıştır.



Görsel 3.4. *Kutunun Doku Çıkarımı (Yeşil noktalar tespit edilen modele uygun olan noktalar (ing. inlier), kırmızı noktalar modele uygun olmayan(ing. outlier) noktalardır.)*

3.1.6. Sağlam eşleştirme

Her bir yerel tanımlayıcı için görüntü düzlemindeki bir 2D konumun ilişkili olduğu bir dizi yerel tanımlayıcılar elde edilir. Bununla birlikte, eşleştirmeleri çıkarmak için, 3D konumunu içeren yerel tanımlayıcılara ihtiyaç vardır.

Yüksek güvenilirlikle eşleşen tanımlayıcıları gerçekleştirmek için en yaygın, kümedeki her tanımlayıcı diğer kümedeki tüm tanımlayıcılarla karşılaştırıldığı anlamına gelen brute force tekniğidir. Bununla birlikte, litaratürde, büyük veri setlerinde hızlı bir şekilde en yakın komşu arama için optimize edilmiş bir algoritma koleksiyonu içeren Fast Library for Approximate Nearest Neighbors (FLANN) [30] gibi diğer arama algoritmaları da bulunmaktadır.

3.1.7. Pozisyon tahmini

Eşleştirmeler filtrelendikten sonra, sahne anahtar noktalarından ve elde edilen bir eşleşme listesini kullanan 3D modelden 2D-3D eşleştirmeler bulunmalıdır. Örnek

kodda, daha sonra kamera pozisyonunu çıkarmak için kullanılacak 2D-3D eşleştirmelerin nasıl çıkarılacağı kısaca gösterilmektedir.

```
    vector<Point3f> points3d ;
    vector<Point2f> points2d ;

    for ( size_t match_idx = 0 ; match_idx < matches . size ( ) ;
++match_idx )
    {
       // 3D point from model
      Point3f point3d =
          points3d_model [ matches [ match_idx ] . trainIdx ] ;

      // 2D point from the s c ene
      Point2f point2d_scene =
          keypoints_scene [ matches [ match_idx ] . queryIdx ] . pt ;

          points3d . push_back ( point3d_model ) ; // add 3D point
          points2d . push_back ( point2d_scene ) ; // add 2D point
    }
```

3.1.8. İzleyici güncelleme (*ing.* Update Tracker)

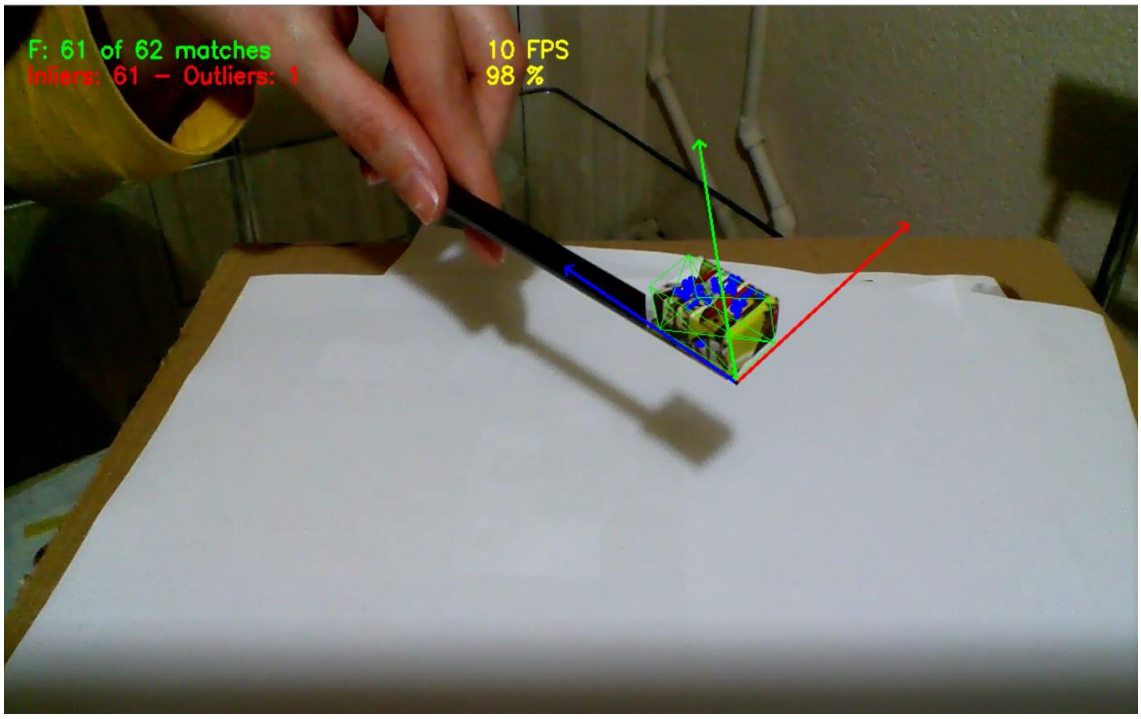
Bilgisayar görüntülerinde veya robotlarda, sonuçların iyileştirilmesi için Bayes izleme algoritmalarını kullanmak yaygın bir işlemdir. Bu uygulamada doğrusal kalman filtresi, modele uygun olan noktaların (*ing.* inlier) belirli bir eşikten düşük olduğu gibi durumlarda nesne pozisyonunun izlenmesini sağlamak için uygulanmıştır. Tanımlanmış durum vektörü aşağıdaki gibidir:

$$X = (x, y, z, \dot{x},\ \dot{y},\ \dot{z}, \ddot{x}, \ddot{y}, \ddot{z}, \psi, \theta, \varphi, \dot{\psi}, \dot{\theta},\ \dot{\varphi}, \ddot{\psi},\ \ddot{\theta}\ , \ddot{\varphi}) \tag{3.3}$$

Burada X vektörü ilk ve ikinci türevleri (hız ve ivme) ile konumsal verileri (x, y, z), ilk ve ikinci türevleri (hız ve ivme) ile euler açılar temsilindeki (ψ, θ, φ) oryantasyon verilerinden oluşmaktadır.

3.1.9. Kafes yeniden izdüşümlendirme (*ing.* Reproject Mesh)

Kamera pozisyonu bulunduktan sonra, küçük bir artırılmış gerçeklik uygulamasıyla elde edilen sonuçları görselleştirilebilir. Bu durumda, Perspektif Projeksiyon Modeli formülünün uygulanması ve nesneni 3D koordinatlarının bilinmesi ile, kafes (*ing.* mesh) görüntü düzlemine geri döndürülür.



Görsel 3.5. *Pozisyon Tahmini ve Kafes Yeniden Projeksiyonu*

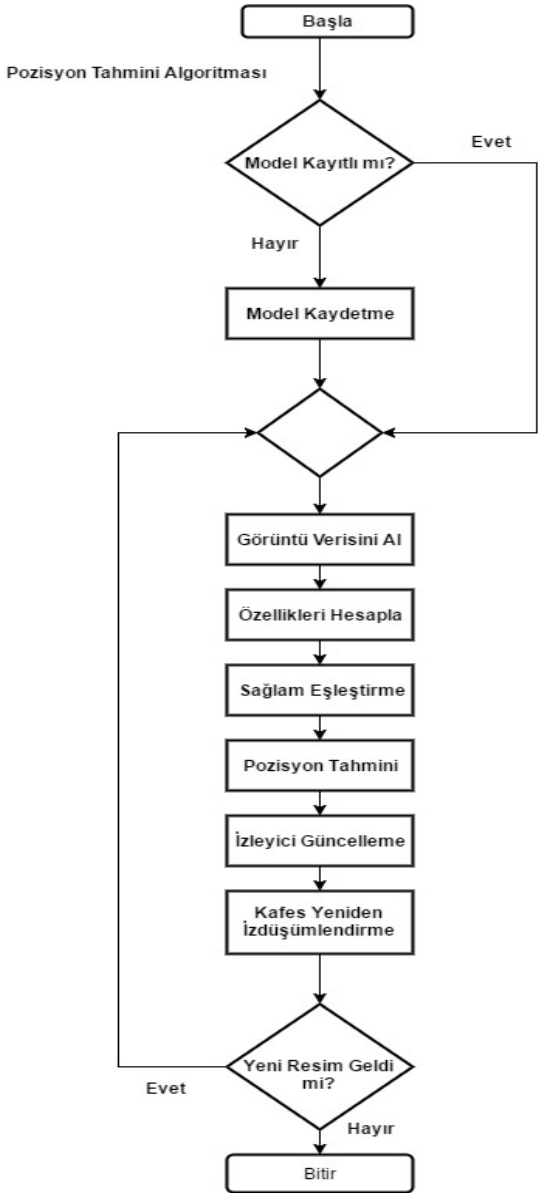
Görsel 3.5.'da nesne kafesi (yeşil renkte) gerçeğe uygun bir şekilde yerleştirildiğinden kamera pozisyonunun iyi tahmin edildiği söylenebilir.

Bu uygulama pozisyon tahmini için C++ ile yazılmış iki ana bölümden oluşmaktadır: model kaydı ve nesne algılama.

İlk olarak, 2D özellikleri ayıklanan bir nesneyi kaydeder ve PLY formatında verilen nesnenin 3D modelin her özelliğinin 3D koordinatlarını hesaplar. Kayıt modülü, algılama modülü tarafından yüklenmek üzere özel hazırlanmış nesne kaydının sonucunu yazmaktadır.

Algılama modülü, kayıt modülü tarafından üretilen belirli bir nesne modelini yükler ve verilen bir görüntü dizisinden, elde edilen sonuçları görselleştirmek için nesnenin kafesi yeniden projeksiyonlandırarak nesneyi izler ve takip eder.

Tüm uygulama OpenCV depolarında bilgisayarla görme geliştiricilerinin bakış açısı ile adım adım açıklanan öğretici metinden bir kaynak olarak dahil edilmiştir.



Şekil 3.2. *Model Kaydı ve Algılama Süreçleri Dahil Olmak Üzere Uygulamanın Akış Şeması*

3.2. Küresel Koordinatlar Kullanılarak Pozisyon Tahmini

Bu bölümde küresel koordinat sistemi kullanılarak renkli bir nesnenin pozisyon tahmini için yapılan çalışma anlatılmıştır. Çalışma kapsamında, renk tabanlı algılama yaklaşımı kullanarak nesnenin yatay ve dikey düzlemdeki konumunun ve bu nesnenin derinlik bilgisinin gerçek zamanlı tespiti ve takibini yapacak yazılım çalışması yapılmıştır.

Öncelikle bölüm 3.2.1'de referans noktalarının elde edilmesi için kullanılan modeller hakkında bilgi verilmiştir. Referans noktaları için ise satranç tahtası ve aruco işaretçisi üzerinde durulmuştur.

Bölüm 3.2.2'de küresel koordinat sistemi hakkında bilgi verilmiştir. Bu bilgiler dahilinde koordinat sistemine derinlik bilgisinin eklenmesi ile pozisyon tahmini gerçekleştirilmiştir.

Derinlik bilgisinin elde edilmesi için sarı renkli bir nesne kullanılmıştır. OpenCV görüntü işleme yöntemleri ile bu nesne tespit edilerek, bu nesnenin yatay ve dikey koordinatları referans noktalarına göre tespit edildikten sonra buna derinlik bilgisi de eklenmiştir.

3.2.1. İşaretçi tespiti

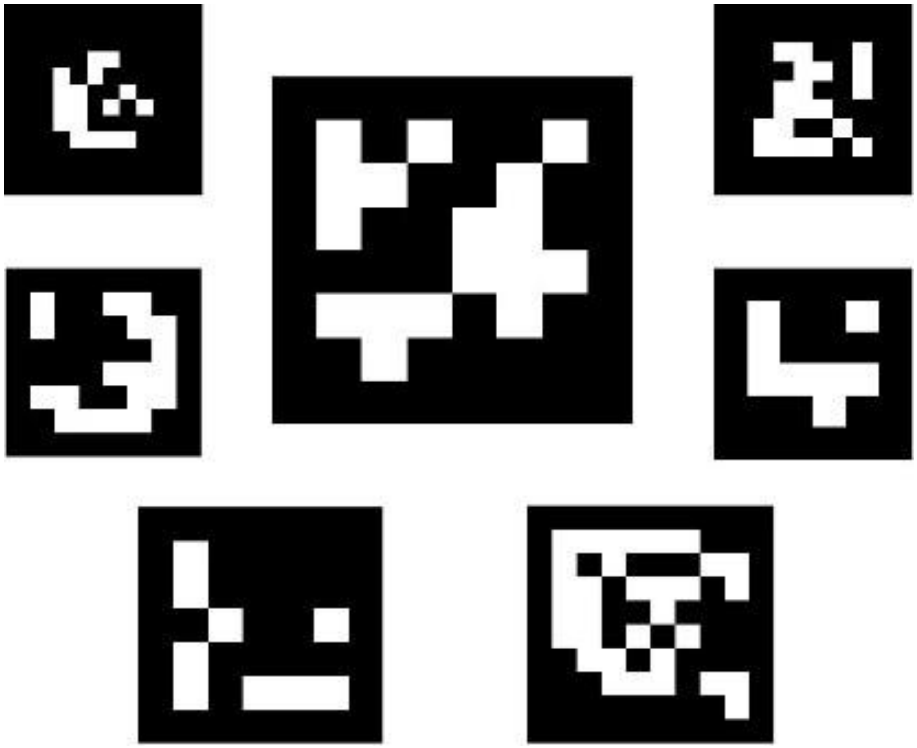
Bu çalışmada işaret (*ing.* marker) tabanlı izleme kullanılmaktadır. İşaret tespiti ile, kullanılan kameranın nerede olduğu tespit edilmektedir. İşaret tabanlı çözüm, basılı işaretleyicileri bulmak için bilgisayar görme teknikleri kullanılmaktadır. Bir işaretleyici basit bir işaret veya bir resim olabilir. Bilgisayar görüntüsü, bu işaretleri gerçek zamanlı olarak algılar. İşaretleri belirlerken, görüntü işleme, model tanıma ve bilgisayar görme teknikleri kullanılır. Bilgisayarla görme (*ing.* computer vision) işaretleyiciyi algılar ve doğru ölçeklendirir ve ayrıca kameranın doğru pozisyonunu tanımlar [2].

Basit işaretlerin kullanılmasının nedeni bu işaretler iyi ve güvenilir algılama sağlamasıdır. İşaretlerin en iyi tespiti siyah beyaz işaretler kullanılarak yapılabilmektedir.

Bilinen dört nokta kameranın pozisyonunu benzersiz bir şekilde hesaplamak için yeterlidir ve bu noktaları elde etmek için en basit yol, siyah kare işaretleyicileri kullanmaktır [13]. Hızlı ve güvenilir algılama nedeniyle birçok sistem siyah beyaz kare işaretleri kullanmaktadır. Bu işaretler, iyi izleme sonuçları elde etmek için de kullanılır.

3.2.1.1. *Aruco işareti*

Bir Aruco işaretleyici, geniş siyah bir kenarlıkla ve onun tanımlayıcısını (kimliği) belirleyen bir iç ikili (*ing.* binary) matristen oluşan yapay bir kare işaretçidir [31]. Siyah çerçeve görüntüde onun tespitini kolaylaştırır ve ikili kodlama işaretçinin tanımlanması ve hata algılama ve düzeltme tekniği uygulama sağlar. İşaretçi boyutu, iç matrisin boyutunu belirler. Örneğin 4x4'lük bir işaretçini boyutu 16 bitten oluşur.



Şekil 3.3. *Bazı Aruco İşaretçi Örnekleri [31]*

Bir işaretleyicinin çevrede döndürülebildiğine dikkat edilmelidir, ancak işaretçinin her köşesisin kesin olarak tanımlanması için, algılama işlemi orijinal rotasyonunu belirleyebilmelidir. Bu aynı zamanda ikili kodlamaya dayalı olarak yapılır.

Belirleyicilerden oluşan bir sözlük, belirli bir uygulamada ele alınmış bir dizi işaretçi grubudur. Bu basitçe, her bir işaretleyicinin ikili kodlamaları listesidir.

Bir sözlüğün ana özellikleri sözlük boyutu ve işaretçi boyutudur.

- Sözlük boyutu, sözlüğü oluşturan işaretçilerin sayısıdır.
- İşaretçi boyutu, bu işaretlerin boyutudur (bit sayısı).

Aruco modülü, farklı sözlük boyutlarını ve işaretçi boyutlarını kapsayan bazı önceden tanımlanmış sözlükler içerir.

Bir işaretleyici kimliği, ait olduğu sözlüğün içindeki işaretçi dizinidir. Örneğin, bir sözlük içindeki ilk 5 işaretçinin kimliği: 0, 1, 2, 3 ve 4'tür.

İşaretleyici Tespiti:

Bazı Aruco işaretçilerinin görülebildiği bir görüntü göz önüne alındığında, algılama işlemi tespit edilen işaretlerin bir listesini çıkarmalıdır. Tespit edilen her işaretleyici şunları içerir:

- Dört köşesinin görüntüdeki konumu,
- İşaretçinin kimliği.

İşaretleyici tespit süreci iki ana aşamadan oluşmaktadır.

1. İşaretleyici adayları tespit edilir. Bu adımda işaretleyici olmaya aday olan kare şekiller bulmak için görüntü analiz edilir. İşaretleyicileri sınıflandırmak için bir eşikleme (*ing.* thresholding) başlar, daha sonra eşiklenmiş görüntüden konturlar çıkarılır ve dışbükey olmayan ve kare şekline yaklaşmayan adaylar atılır. Bazı ekstra filtreleme uygulanmaktadır (çok küçük ya da çok büyük konturları çıkarma, konturları birbirine çok yakın olanları kaldırma vb.).
2. Adayın saptanmasından sonra, iç kodlamalarını analiz ederek bunların aslında işaretleyici olup olmadığını belirlemek gerekir. Bu adım, her işaretleyicinin işaretçi bitlerini çıkararak başlar. Bunu yapmak için, işaretçiyi kanonik biçimde elde etmek için perspektif dönüşümü uygulanır. Ardından, kanonik görüntü, beyaz ve siyah bitleri ayırmak için Otsu metodu kullanılarak eşik oluşturulur. Görüntü, işaretleyici boyutuna ve sınır boyutuna göre farklı hücrelere bölünür ve her hücredeki siyah veya beyaz piksellerin miktarı, beyaz veya siyah bit olup olmadığını belirlemek için sayılır. Son olarak, bitler, işaretçinin belirli bir sözlükte yer alıp almadığını belirlemek için analiz edilir ve gerektiğinde hata düzeltme teknikleri kullanılır.

Aruco modülünde, algılama işlemi detectMarkers() fonksiyonu ile gerçekleştirilir. Bu fonksiyon, modüldeki en önemli özelliktir, çünkü diğer tüm işlevler, detectMarkers() tarafından döndürülen daha önce algılanmış işaretleyicilere dayalıdır.

İşaretleyici tespiti örneği:

```
using namespace cv;
using namespace cv::aruco;
detectMarkers(inputImage, dictionary, markerCorners, markerIds,
parameters, rejectedCandidates);
```

Fonksiyonun parametreleri aşağıdaki gibidir:

- İlk parametre, işaretleyicinin algılanacağı görüntüdür.

```
Mat inputImage;
```

- İkinci parametre sözlük nesnesi, bu durumda önceden tanımlanmış sözlüklerden (DICT_6X6_250) biridir.

```
Dictionary dictionary= getPredefinedDictionary(DICT_6X6_250);
```

- Algılanan işaretler markerCorners ve markerIds yapılarında saklanır:

- MarkerCorners, belirlenen işaretlerin köşelerinin listesidir. Her işaretçi için, dört köşesi orijinal sırasıyla döndürülür (saat yönünde sol üstten başlar). Böylece, ilk köşe sol üst köşedir, bunu sağ üst, sağ alt ve alt sol takip eder.

```
vector< vector<Point2f> > markerCorners;
```

- MarkerIds, markerCorners'daki algılanan işaretlerin her birinin kimlik listesidir. Döndürülen markerCorners ve markerIds vektörlerinin aynı boyutlara sahiptir.

```
vector< int > markerIds;
```

- Dördüncü parametre DetectionParameters türündeki nesnesidir. Bu nesne, algılama işlemi sırasında özelleştirilebilen tüm parametreleri içerir.

```
DetectorParameters parameters;
```

- Son parametre olan rejectedCandidates, işaretçi adaylarının iade edilmiş olanların bir listesidir, yani bulunmuş ancak geçerli bir kodlamayı sunmayan karelerdir.

```
vector< vector<Point2f> > rejectedCandidates;
```

Pozisyon Tahmini:

İşaretleyiciler tespit edildikten sonra yapılacak işlem kamera pozisyonunun işaretleyicilerden tespit edilmesidir.

Kamera pozisyonunun tespit edilmesi için kullanılan kameranın kalibrasyon parametrelerinin bilinmesi gerekmektedir. Bu parametreler kamera matrisi ve bozulma katsayılarıdır. Kalibrasyon aşamaları ayrıntılı olarak Bölüm 2'de anlatılmıştır.

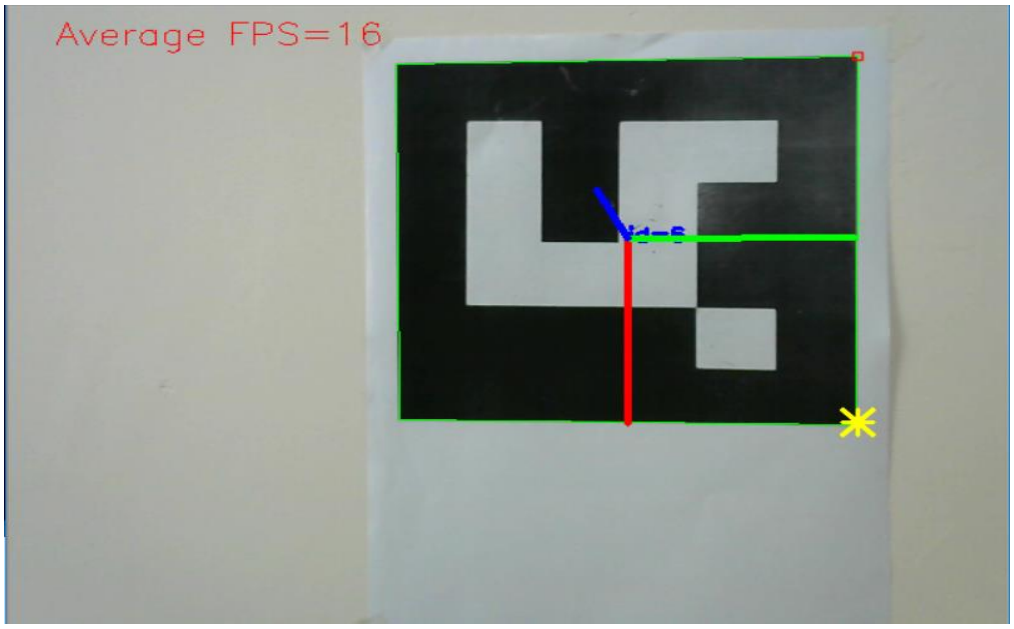
Kameranın bir işaretleyiciye göre pozisyonu, işaretleyici koordinat sisteminden kamera koordinat sistemine yapılan 3D dönüşümdür. Bu dönüşüm bir döndürme ve çeviri vektörü tarafından belirlenir. Bunun içinde daha öncede anlatılan solvePnP metodu kullanılmaktadır.

Aruco modülü, algılanan tüm işaretleyicilerin pozisyonlarını hesaplamak için bir fonksiyon sunmaktadır:

```
c++ Mat cameraMatrix, distCoeffs;
vector< Vec3d > rvecs, tvecs;
cv::aruco::estimatePoseSingleMarkers(corners, 0.05, cameraMatrix,
distCoeffs, rvecs, tvecs);
```

- Corners parametresi, detectMarkers() fonksiyonu tarafından döndürülen işaretçi köşelerinin vektörüdür.
- İkinci parametre, işaretleyicinin kenarının metre cinsinden veya herhangi başka birimden boyutudur.
- CameraMatrix ve distCoeffs a önceden bilinmesi gereken kamera kalibrasyon parametreleridir.
- rvecs ve tvecs köşelerdeki işaretçilerin sırasıyla rotasyon ve çeviri vektörleridir.

Bu fonksiyon tarafından üstlenilen işaretçi koordinat sistemi, Görsel 3.6.'da olduğu gibi Z ekseni işaret eden işaretçinin ortasına yerleştirilir. Eksen renkleri X: kırmızı, Y: yeşil, Z: mavi'dir.



Görsel 3.6. *İşaretleyici Kimliği 6 Olan 6x6 Bitlik İşaretleyici Tespiti Örneği*

3.2.1.2. *Satranç tahtası işareti*

Pozisyon çıkarımı işleminde temel problem yatay, dikey koordinatları ve derinlik bilgisinin tespit edilmesidir. Bu işlem ise belirli referans noktasına göre yapılmaktadır. Referans noktası herhangi bir işaretçi olabilir. Satranç tahtası resmiyle de bu problem çözülebilmektedir. Satranç tahtasında bulunan noktalar referans noktalar olarak kullanılabilir. Bu referans noktasının 3 boyutlu düzlemdeki konumu orijin yani (0,0,0) olarak kabul edilerek tespit edilecek nesnenin bu noktaya göre konumu nesnenin pozisyonunu vermektedir.

Satranç tahtası tespiti için kalibrasyon parametrelerine ihtiyaç duyulmaktadır. Kalibrasyon sonucu elde edilen parametreler kullanılmaktadır.

Kamera parametreleri XML/YAML dosyasından

```
FileStorage fs;
fs.open(filename, FileStorage::READ);
// read camera matrix and distortion coefficients from file
Mat intrinsics, distortion;
fs["Camera_Matrix"] >> intrinsics;
fs["Distortion_Coefficients"] >> distortion;
// close the input file
fs.release();
```

komutları ile alınır ve kullanılır. Kameradan alınan görüntünün gri kopyası alınır. Bu işlem ise aşağıdaki komut ile gerçekleştirilir.

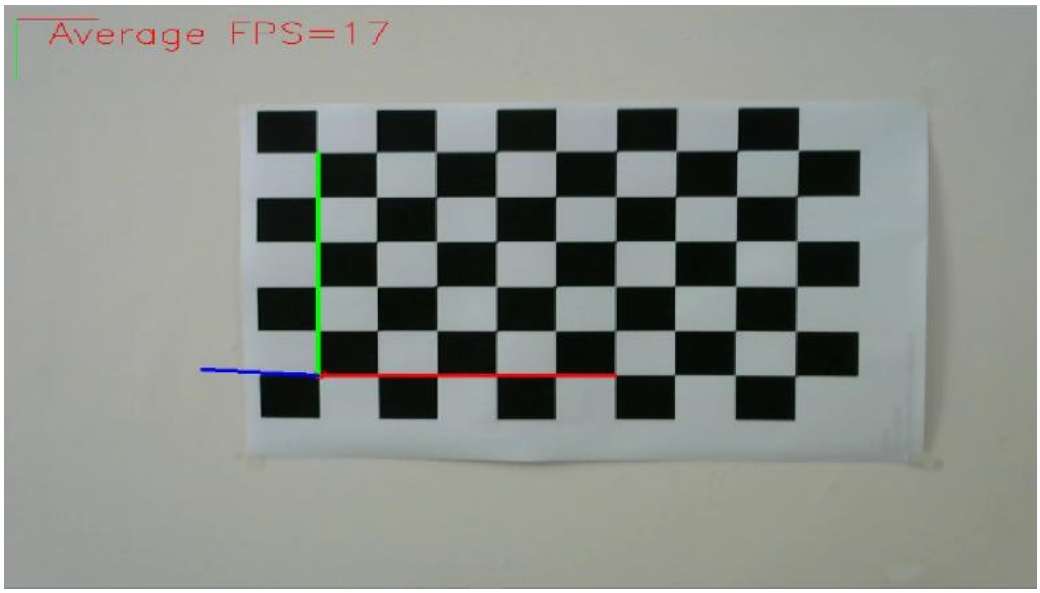
```
cvtColor(webcamImage, gray, COLOR_BGR2GRAY);
```

Aşağıdaki komut kullanılarak satranç tahtasının piksel koordinatları bulunmaktadır. Her bir noktanın derinlik bilgisi yani z değeri sıfır olarak kabul edilerek pozisyonları boardPoints vektörüne eklenmektedir.

```
bool found = findChessboardCorners(gray, cbSize, imagePoints, CALIB_CB_FAST_CHECK);
```

Köşe noktalar bulunduktan sonra ise solvePnP komutu kullanılarak kamera dönüşüm (*ing.* rotation) ve öteleme (*ing.* translation) parametreleri hesaplanmaktadır.

```
solvePnP(Mat(boardPoints), Mat(imagePoints), intrinsics, distortion, rvec, tvec, false);
```

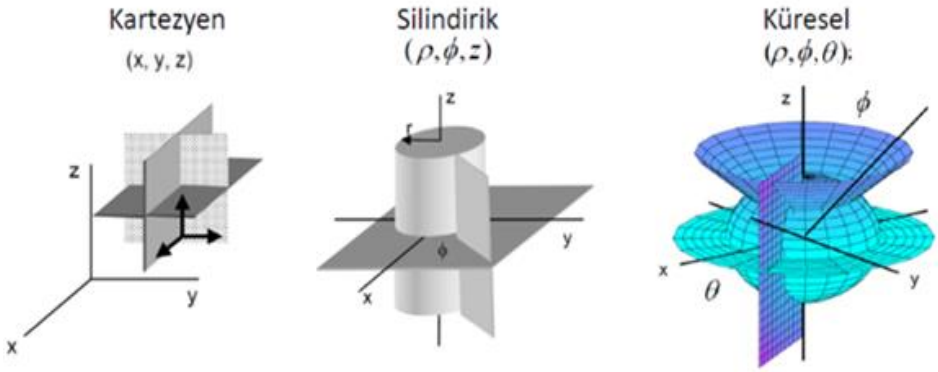


Görsel 3.7. *Satranç Tahtası Tespiti Örneği*

Görsel 3.7.'da 9x6'lık satranç tahtasının tespiti verilmiştir.

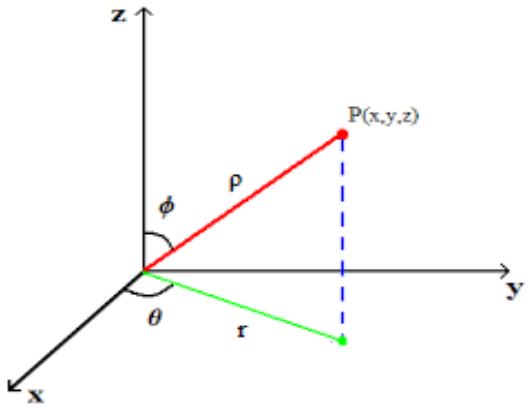
3.2.2. Küresel koordinatlar

Uzayda bir noktayı göstermek ve vektörleri görselleştirerek daha kolay anlaşılmasını sağlamak için koordinat sisteminden faydalanılır. Verilen bir vektör matematiksel olarak seçilen koordinat sistemi üzerinde bileşenlerine ayrılarak ifade edilir. Uzayda çok sayıda dikgen (*ing.* orthogonal) koordinat sistemi mevcuttur. Burada dikgen terimi koordinat sistemi içinde her bir noktanın birbirlerine dik üç yüzeyin kesişimi ile tanımlanabileceğini anlatmaktadır. Elektromanyetik teoride alanları ve dalgaları ifade etmek için kartezyen (*ing.* cartesian), silindirik (*ing.* cylindrical) ve küresel (*ing.* spherical) koordinat sistemlerinden faydalanılır. Verilen bir vektör ifadesi için koordinat sistemleri arasında dönüşüm yapmak mümkündür.



Şekil 3.4. *Koordinat Sistemleri*

Küresel koordinat sistemi, üç boyutlu uzayda nokta belirlemenin bir yoludur. Küre üzerindeki bir nokta bu sistemde üç tane bileşenle ifade edilir, bunlar ρ, ϕ *ve* θ 'dir. Koordinatların tanımlı oldukları aralıklar ve tanımları şu şekilde verilir [32].



Şekil 3.5. *Küresel Koordinat Sistemi [32]*

ρ : Yarıçap, P ve (0,0,0) arasındaki uzaklıktır. Tanım aralığı $0 \leq \rho < \infty$ olarak verilir.

ϕ : Enlem, z-ekseni ve çap arasındaki açıdır. $0 \leq \phi < 180°$ aralığında tanımlanır. Polar açı olarak da adlandırılır.

θ : Boylam, x-ekseni ile çapın xy-düzlemine izdüşümü arasındaki açıdır. $0 \leq \theta < 360°$ aralığında tanımlanır.

Bu sistem, dünya üzerinde coğrafi konum belirlerken kullanılan sistemdir. Dünya'nın yüzeyi üzerinde her noktada yarıçap aynı olduğundan, sadece enlem ve boylam ile bir yer belirlenebilir. Ayrıca fizikte küresel yapıya sahip sistemler, (dünya, güneş, yüklü bilye vs.) ele alınırken yine küresel koordinatlara geçiş yapılır. Küresel koordinatlarla Kartezyen koordinatlar arasındaki bağıntılar şu şekildedir.

$$x = \rho \sin\phi \cos\theta \tag{3.4}$$

$$y = \rho \sin\phi \sin\theta \tag{3.5}$$

$$z = \rho \cos\phi \tag{3.6}$$

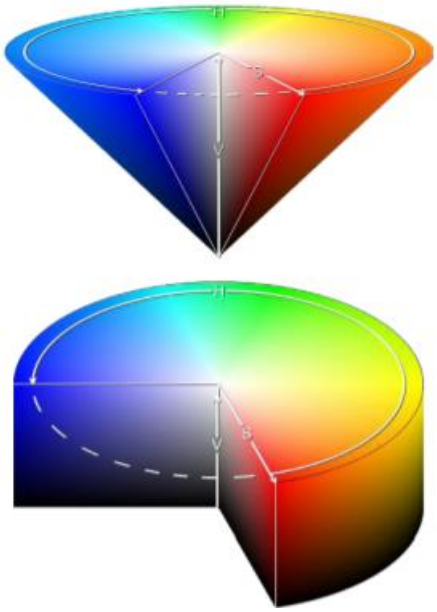
3.2.3. Görüntü işleme

Video kamera ve tarayıcı gibi görüntü yakalayıcı cihazlarla alınarak sayısallaştırılan görüntülerin geliştirilen yöntemlerle işlenip analiz edilerek görüntülerden bilgi elde edilmesi ve bu bilgilerin yorumlanması, görüntü işleme alanının temelini oluşturur [33]. Görüntü işleme uygulamalarına örnek olarak fabrika otomasyon uygulamalarından endüstriyel kalite/üretim kontrolleri, tıbbi uygulamalardan kan hücrelerinin sayımı, bilgi işlem uygulamalarından karakter tanıma, güvenlik amaçlı

uygulamalardan yüz tanıma, askeri uygulamalardan akıllı silahlar üretimi gibi örnekler verilebilir. Bu çalışmada, görüntü işleme algoritmalarından yararlanılarak renkli nesne algılama teknikleri kullanılmıştır. Görüntü işleme kısmını İntel®'in açık kaynaklı görüntü işleme kütüphanesi olan Open Source Computer Vision (OpenCV) kullanılmıştır. Bu çalışmada OpenCV kütüphanesini kullanılmasındaki asıl amaç gerçek zamanlı görüntü işleme uygulamalarına izin vermesidir. OpenCV'nin sağlamış olduğu renk algılama ve algılanan renk bileşenlerinin aralarında benzerlik gösteren belirli bölgelere ayırma işlemi yapılmış ve nesnemizin konumu koordinat olarak belirlenerek nesnelerin takibini gerçekleştirecek algoritma yazılmıştır. Sonuç olarak görüntü işleme yazılımı sayesinde renkli nesnenin konumu belirlenmiştir.

3.2.4. OpenCV ile hareketli nesne takibi

OpenCV görüntü işlemek üzere 1999'da İntel® tarafından açık kaynaklı olarak C yazılım dilinde geliştirilmiş olan görüntü işleme kütüphanesidir [33]. Bu çalışma OpenCV Kütüphanesi ile C++ yazılım dilinde gerçekleştirilmiştir. Renkli nesne algılamak ve algılanan nesnenin pozisyonunun takibi için HSV renk uzay modelinden yararlanılmıştır. HSV (Renk, Koyuluk, Parlaklık) veya bazı kaynaklarda HSB (Renk, Koyuluk, Parlaklık) renk uzayı, renkleri sırasıyla renk özü, koyuluk ve parlaklık olarak tanımlar. HSV renk modunun kullanılma amacı RGB (Kırmızı, Yeşil, Mavi) uzayına göre insan görü düzeneğine daha yakın bir yapı oluşturmaktadır. HSV, RGB renk uzayından doğrusal olmayan bir dönüşüm ile elde edilir. HSV, aygıt bağımlıdır. Yani bu uzayda tanımlı bir renk, rengi üreten aygıt cihazına göre değişim gösterebilir.



Şekil 3.6. *Konik ve Silindirik Biçimli HSV Renk Uzayı [33]*

3.2.5. Yazılımın gerçekleştirilmesi

Bu bölümde tek kamera kullanarak OpenCV'de tanımlı fonksiyonlar ile birlikte nesne pozisyon tespiti aşamaları açıklanacaktır.

Renkli nesne algılama:

Çalışmanın bu kısmında OpenCV'de görüntü işleme yöntemlerine yer verilmiştir. OpenCV'de görüntü işleme için öncelikle renkli bir görüntü algılayıcısına ihtiyacımız vardır. Burada işimizi bir web kamerası görebilir. Görsel 3.8.'de görüntü alınan kamera verilmiştir.

Web kamerasının görüntü almak için OpenCV' de bir görüntü işaretçisi tanımlanır ve o işaretçinin ayırdığı alana görüntü kare kare kayıt edilir.

```
VideoCapture capture;
capture.open(0);
```

Komutu ile kamera görüntüsü alınır. Kameradan alınan görüntü renkli ya da siyah/beyaz olabilir.

Görsel 3.8. *Görüntü Alınan Web Kamerası*

Kullanılan kamera, renkli kameradır ve 1280x720 piksel boyutunda çözünürlüğe sahiptir. Kameranın renkli olması önemlidir. Çünkü renk modellerinden birbirlerine dönüşüm sadece renkli görüntülerde olmaktadır. Kamera görüntüleri 25 FPS (saniyedeki kare) saniyede 25 kare olarak kayıt etmektedir. Kamera görüntüsünü işlemek için bir resim (*ing.* frame) alınmakta ve o resim üzerinden işlem yapılmaktadır. Video görüntüsünden resim almak için OpenCV'de aşağıdaki komut kullanılmaktadır.

```
capture >> img;
```

Bu komut görüntü aldığımız kamera nesnesinin içerisinden bir resimlik görüntü alır. Alınan bu görüntü RGB formatından olduğundan bu formatın asıl kullanacağımız renk uzayı olan HSV uzayına dönüştürmemiz gerekir. HSV uzayında dönüştürülen resimde görüntü işleme algoritmasını geliştirilmesi ve renkli hedef takibinin yazılım kısmının tamamlanması amaçlanmaktadır. OpenCV'nin renk dönüşümü için hazır komutlar vardır.

```
cv::CvtColor( const CvArr* src, CvArr* dst, int code );
```

Burada;

- `src`, 8-bit (8u), 16-bit (16u) ya da floating point 32-bit (32f) kaynak dosyası,
- `dst`, hedef dosyası,
- `code`, dönüşüm yöntemi olarak ifade edilmektedir.

Aşağıda bazı `code` dönüşümleri gösterilmiştir.

RGB formatından HSV formatına dönüştürmek için `CV_BGR2HSV` ve `CV_RGB2HSV`; HSV formatından RGB formatına dönüştürmek için ise `CV_HSV2BGR` ve `CV_HSV2RGB` dönüşümleri kullanılmaktadır.

RGB formatından HLS formatına dönüştürmek için `CV_BGR2HLS` ve `CV_RGB2HLS`; HLS formatından RGB formatına dönüştürmek için ise `CV_HLS2BGR` ve `CV_HLS2RGB` dönüşümleri kullanılmaktadır.

Filtreleme resmin üzerinde bir filtre varmış gibi düşünüp her piksel değerinin yeniden hesaplanmasıdır. Filtreleme sayesinde görüntü üzerinde netleştirme, belirli ayrıntıları ortaya çıkarma, görüntüyü yumuşatma, kenar keskinleştirme veya kenar bulma gibi işlemler gerçekleştirilir. Filtreler genelde 3x3 lük matrislerdir. Fakat boyutları 5x5, 7x7, 9x9, 11x11 şeklinde olabilir. Gaussian filtreleme aynı zamanda bir fourier dönüşümüdür. Gauss filtre ile sonsuz bir transfer fonksiyonuna karşılık mekânsal alanda sonlu bir pencerede filtreleme yapılabilmektedir. Bu da filtrelemenin temel problemini daha kolay çözülebilir hale getirir. OpenCV'nin gauss filtresi için hazır komutu vardır. Aşağıda gauss fonksiyonu gösterilmiştir.

```
cv::GaussianBlur(img, img, cv::Size(3, 3), 0);
```

Sarı renk için HSV değerleri;

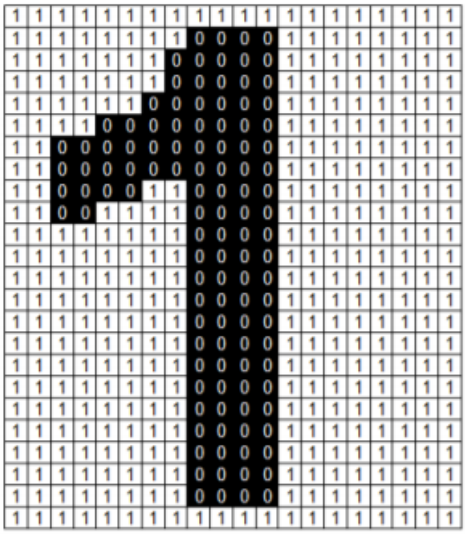
```
int lowH = 21;     // Set Hue
int highH = 30;

int lowS = 224;     // Set Saturation
int highS = 255;

int lowV = 26;      // Set Value
int highV = 225;
```

Hue değeri sarı renk için 21-30 arasındadır. Saturasyon değeri rengin koyuluk açıklığı için kullanılır. Value değeri rengin parlaklık ayarı için kullanılır.

```
cv::inRange( const CvArr* A src, CvScalar SL, CvScalar SU, CvArr*
D dst );
```

Yukarıdaki komutda ilk değişken kaynak resim işaretçisidir. Kaynağın üzerinde algılanacak olan rengin belirlenmesi için ikinci argüman olarak `CvScalar` SL komutu renk bilgisinin alt değerini alır. `CvScalar` SU komutu ise renk bilgisinin üst değerini alır. HSV uzayında renk değerleri alt ve üst değerler olarak girildikten sonra `CvInRangeS` komutu girilen renk değerini resim üzerinden arar ve bulduğu renkteki piksellerin piksel değerini 0 yapar. Diğer geri kalan pikseller 1 değerini alır. Şekil 3.7.'de bu durum kısaca gösterilmiştir. Bir sonraki bölümde algılanan renkli hedefin piksellerinin aldığı değerlere göre hedef ayrışımı ve hedefin takibine yönelik

algoritmalar geliştirilmiştir. Piksellerin aldığı değerlerin bir algoritmaya göre aranması ve bulunmasıyla hedef takibi yapılacaktır.



Şekil 3.7. *Algılanan Renkli Nesnenin Piksel Bit Durumu [33]*

```
cv::dilate(InputArray src, OutputArray dst, InputArray kernel);
```

Dilate fonksiyonu belli bir yapılandırma öğesi kullanarak bir görüntüyü genişletir. Fonksiyon, azami alınacak piksel komşularının şeklini belirleyen belirtilen yapılandırma öğesini kullanarak kaynak görüntüyü genişletir.

```
cv::erode(InputArray src, OutputArray dst, InputArray kernel);
```

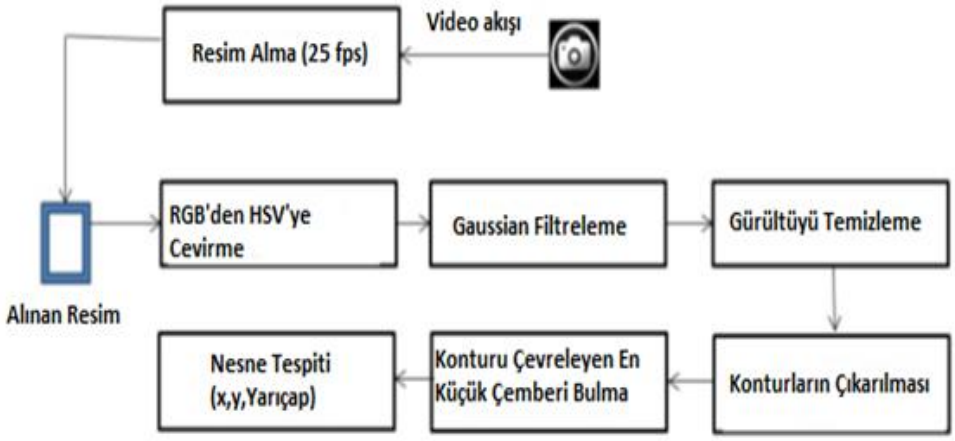
Erode fonksiyonu belli bir yapılandırma unsuru kullanarak bir görüntüyü yumuşatır. Fonksiyon asgari alınacak piksel komşularının şeklini belirleyen yapılanma öğesi kullanılarak kaynak görüntüyü aşındırır.

```
cv::findContours(InputArray image, OutputArrayOfArrays contours,
int mode, int method, Point offset=Point());
```

Yukarıdaki fonksiyon, belirli algoritmaları kullanarak ikili (*ing.* binary) görüntüden konturları alır. Konturlar, şekil analizi ve nesne algılama ve tanıma için yararlı bir araçtır.

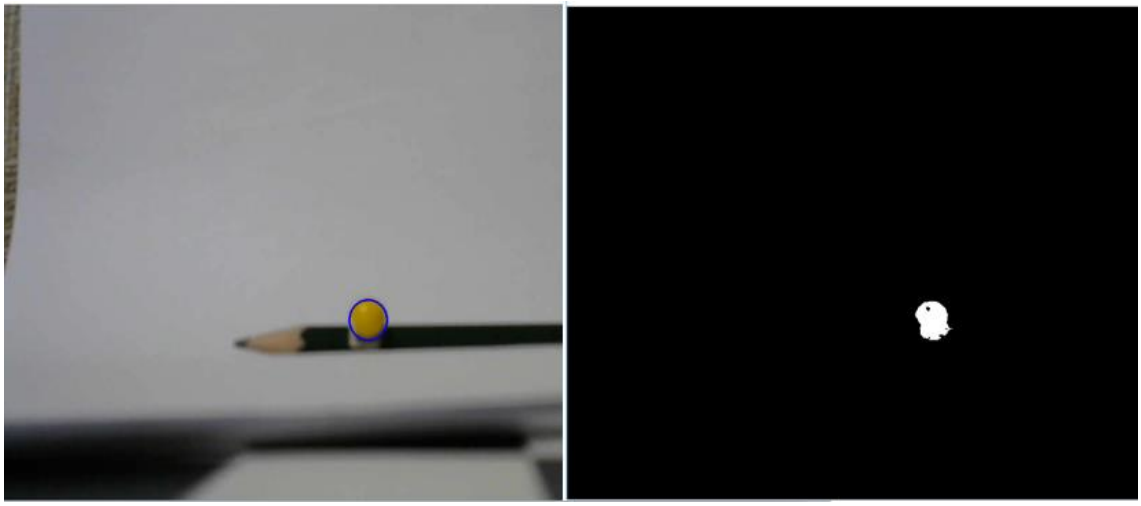
```
cv::minEnclosingCircle(InputArray points, Point2f& center,
float& radius);
```

Yukarıdaki fonksiyon, yinelemeli bir algoritma kullanarak 2 boyutlu nokta kümesindeki en küçük çevreleyen çemberi bulur.



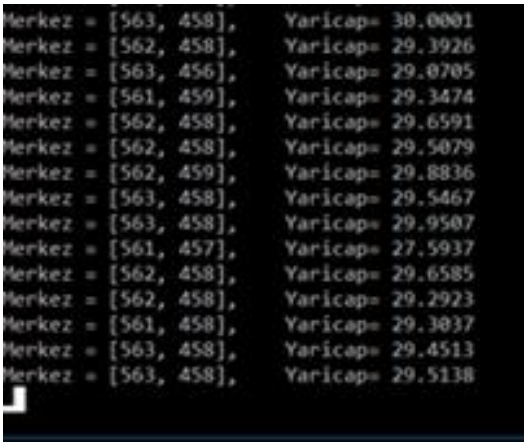
Şekil 3.8. *Nesne Tespiti Akış Diyagramı*

Renkli nesnenin genel akışı Şekil 3.8.'da gösterilmiştir. Nesne tespiti sonucunda nesnenin x, y ve yarıçap değerleri elde edilmektedir. Yarıçap bilgisi nesnenin derinlik bilgisini tespit ederken kullanılmıştır.



Görsel 3.9. *Kameradan Alınan Görüntüde Sarı Nesnenin Tespiti ve Kontur Uygulaması*

Belirlenen cisimlerin takip edilmesi için HSV renk uzayından renk kodlarını bularak cisimlerin renginin algılanıp takip edilmesini sağlanmıştır. Görsel 3.9.'da görüldüğü gibi görüntüdeki sarı nesnenin kontur uygulanarak tespiti gerçekleştirilmiştir. Bulunan beyaz pikseller nesneyi temsil eden noktalardır. Bu pikselleri içine alan en küçük çemberi bulmak için OpenCV'nin sağladığı `minEnclosingCircle()` fonksiyonu kullanılmıştır. Bu fonksiyon nesnenin piksel koordinatları ve çemberin yarıçapını vermektedir. Çalışmada sadece sarı rengin tespiti yapılmıştır.



Görsel 3.10. *Sarı Nesnenin Piksel Koordinatları ve Yarıçapının Bulunması*

Bulunan piksel koordinatları ve yarıçap bilgisi konum bilgisini elde etmede kullanılacaktır.

Derinlik Tespiti:

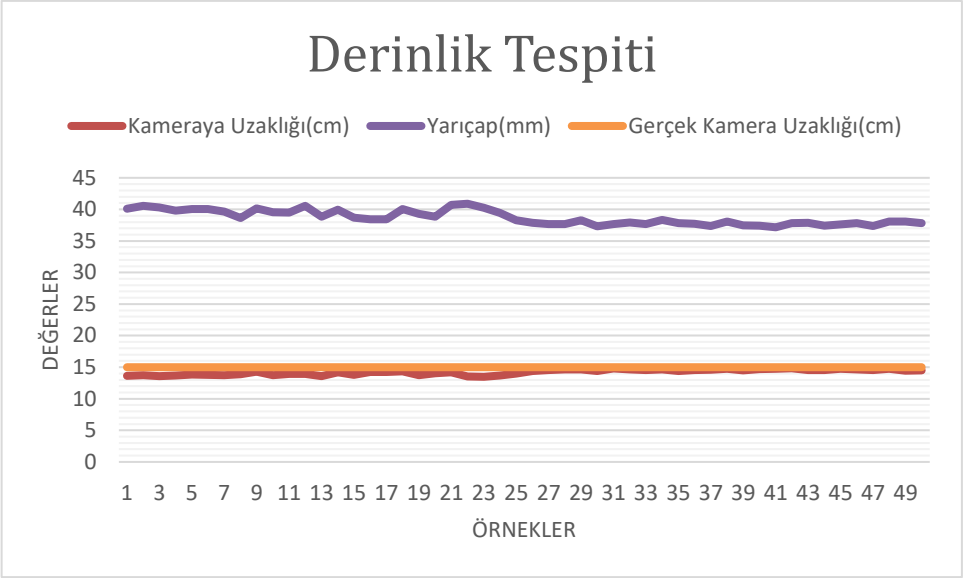
Kameradan bilinen bir işaretçiye veya nesneye olan mesafeyi belirlemek için üçgen benzerliği kullanıldı. Üçgen benzerliği şu şekildedir:

Genişliği W olduğu bilinen bir işaretleyici veya cisim olduğunu varsayarak, bu işaretçiyi kameraya D uzaklıkta bir noktaya yerleştirilir. Kamera ile görüntü alınarak nesnenin görünen genişliği piksel P olarak ölçülür. Bu, kameranın algılanan odak uzaklığı F'yi türetmeyi sağlar.

$$F = \frac{PD}{W} \tag{3.7}$$

Nesneyi kameraya yakına ve uzağa hareket ettirmeye devam ettikçe, nesnenin kameraya olan uzaklığını belirlemek için üçgen benzerliği uygulanabilir:

$$D' = \frac{WF}{P} \tag{3.8}$$



Şekil 3.9. *Yarıçap Verisine Göre Kameraya Olan Uzaklığın Değişimi*

Yapılan derinlik tespiti çalışmasını test etmek için nesne kameraya sabit bir uzaklığa konumlandırıldı. Bu sabit konumdan alınan örneklerde nesnenin yarıçapı ve kameraya olan uzaklığı Şekil 3.9.'da görülmektedir. Yarıçapta oluşan sapmalar nesnenin kameraya olan uzaklığında küçük farklara neden olmaktadır.

$$Hata\ Payı = \left|\frac{\sum_1^A B - \sum_1^A C}{A}\right| \tag{3.9}$$

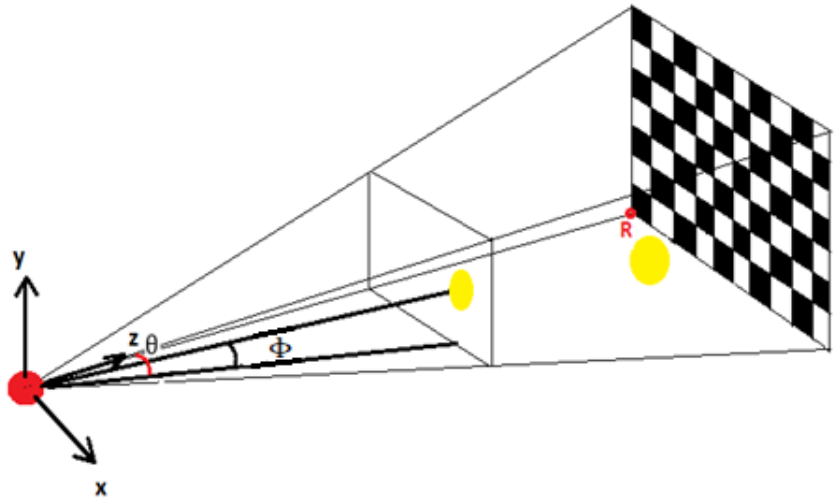
Burada;

- A: Alınan örnek sayısı
- B: Kameraya olan uzaklık
- C: Kameraya olan sabit uzaklık olarak ifade edilmiştir.

Kameraya olan uzaklık alınan örneklerde %5.07 ve ortalama 0.76 cm hata payı ile tespit edilmektedir.

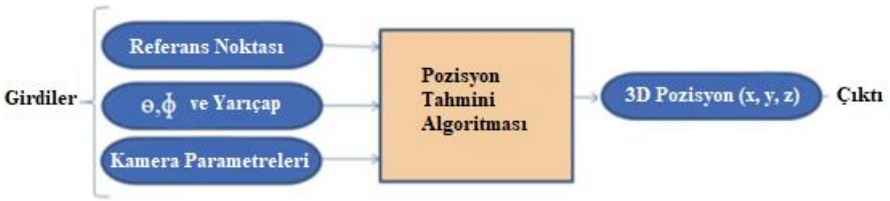
Küresel koordinat sistemi tasarımı Şekil 3.10.'da gösterilmiştir. Bu tasarıma göre kamera kullanılan 9x6 boyutunda satranç tahtasının referans noktasına dik olarak konumlandırılmaktadır. Referans noktası Şekil 3.10.'da R ile gösterilmiştir.

Küresel koordinat sistemi ile pozisyon tahmini:



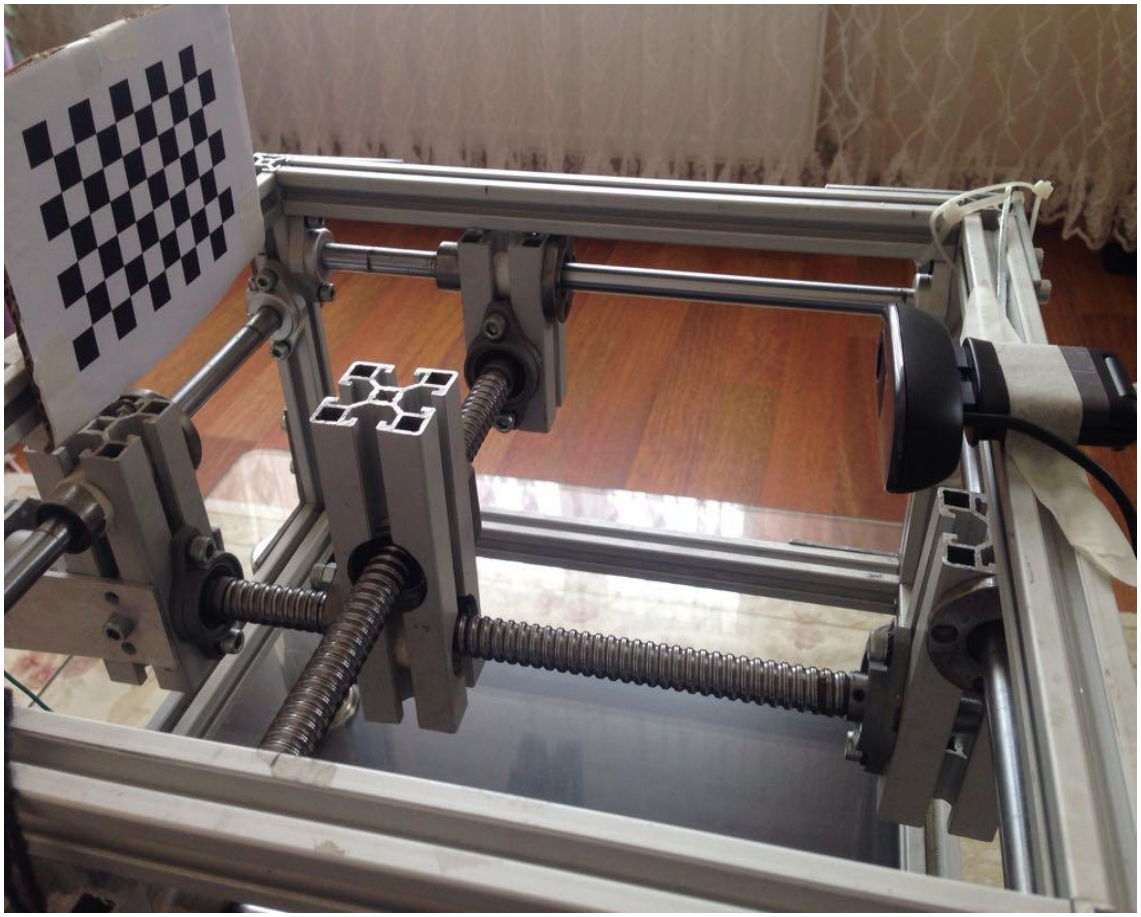
Şekil 3.10. *Küresel Koordinat Sistemi Tasarımı*

Kullanılan sarı renkteki 3D nesnenin projeksiyon ekranındaki izdüşüm değerlerine göre pozisyon tahmini yapılmaktadır. Nesnenin ekrandaki izdüşümünün referans noktasına yatayda yaptığı açı θ, dikeyde yaptığı açı ϕ'dir. Küresel sistemin yarıçapı ise nesnenin kameraya olan uzaklığı ile bulunmaktadır. θ, ϕ ve yarıçap bilgileri elde edildikten sonra bu veriler kullanılarak nesnenin küresel koordinat sistemindeki konumu belirlenmektedir. Konumu belirlemek için Eşitlik 3.4, 3.5 ve 3.6 kullanılmıştır.



Şekil 3.11. *Küresel Koordinatlar Pozisyon Çıkarımı Algoritma Yapısı*

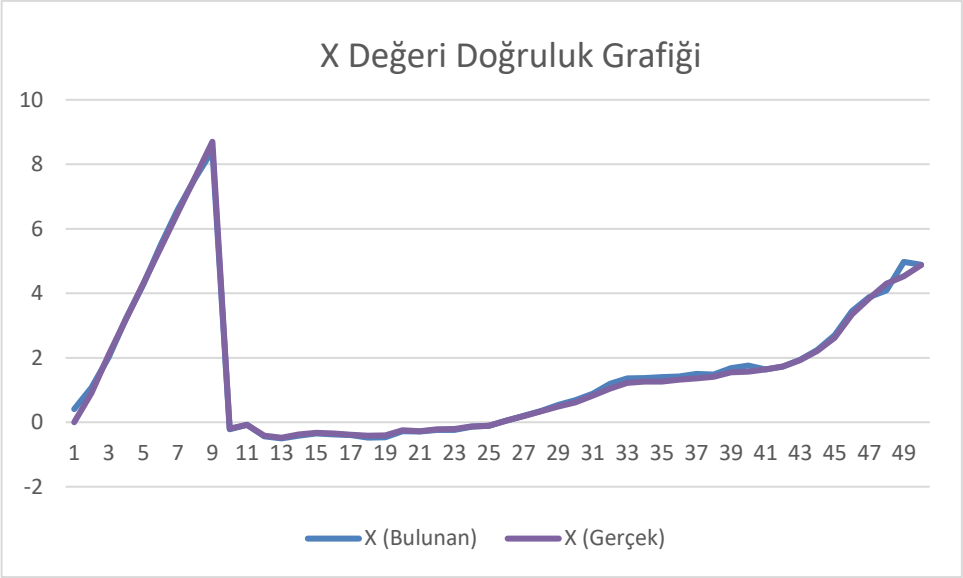
Şekil 3.11.'de pozisyon tahmini algoritmasının yapısı verilmiştir. Algoritma girdi olarak referans noktası, nesnenin piksel koordinatları, nesnenin yarıçapı ve kamera içsel parametrelerini almaktadır. Pozisyon tahmini algoritmasına verilen bu değerler ile nesnenin 3D koordinatları bulunmaktadır.



Görsel 3.11. *Küresel Koordinatlar Sistem Ekipmanlarının Görüntüsü*

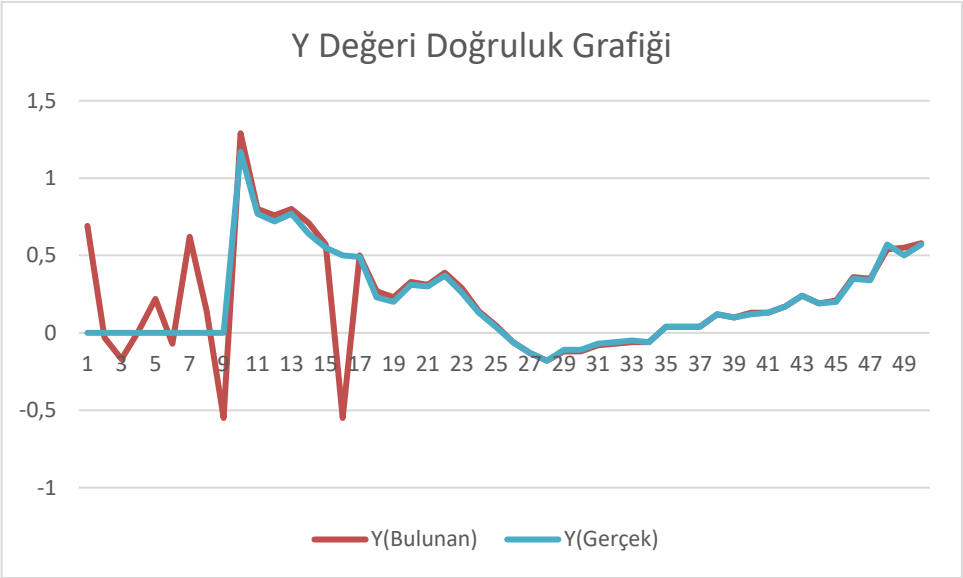
Görsel 3.11.'de gerçekleştirilen sistem verilmiştir. Referans noktasına dik konumlandırılan kamera ile 15x15 cm'lik bir alanda simülasyon işlemi gerçekleştirilmiştir.

Bu çalışmayı test etmek için 3D pozisyonları bilinen 50 örnek alındı ve bu pozisyonlara karşılık gelen değerler bulundu. Bulunan bu değerler ile gerçek değerler karşılaştırıldı. Sonuç olarak X, Y ve Z değeri doğruluk grafikleri Şekil 3.12., Şekil 3.13., Şekil 3.14.'te gösterilmiştir.



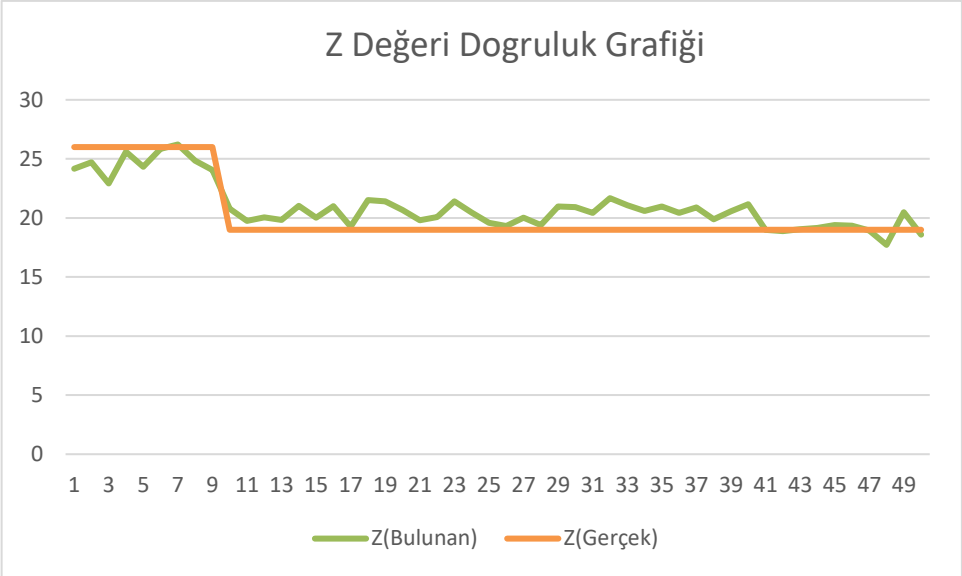
Şekil 3.12. *X Değeri Doğruluk Grafiği*

Bulunan X değerleri gerçek değerlere çok yakın değerlerde bulunmuştur. Alınan örneklerde X değeri %2,35 hata oranlarında tespit edilmiştir.



Şekil 3.13. *Y Değeri Doğruluk Grafiği*

Alınan örneklerde Y değeri %3,74 hata oranlarında tespit edilmiştir. Şekil 3.13.'te görülen sapmalar mm'lik değerlerdir.



Şekil 3.14. *Z Değeri Doğruluk Grafiği*

Alınan örneklerde Z değeri %3,36 hata oranlarında tespit edilmiştir. Tüm koordinatlar ortalama %3,15 hata oranlarında bulunmuştur.

Tablo 3.1. *Test İşlemleri Sonucunda Elde Edilen Hata Oranları (X1=Bulunan değer, X2=Gerçek değer, Y1=Bulunan değer, Y2=Gerçek değer, Z1=Bulunan değer, Z2=Gerçek değer)*

X_1	X_2	Y_1	Y_2	Z_1	Z_2	Hata(%)
84,3	87,0	-5,5	0,0	240,7	260,0	3,50
-2,9	-2,8	3,1	3,0	198	190	3,70
-3,5	-3,3	5,7	5,5	200,1	190	4,95
54,9	54,0	-0,7	0,0	258,6	260	0,73
13,7	12,6	-0,6	-0,6	205,9	190	5,69
14,8	14,1	1,2	1,2	198,8	190	3,19
40,9	43,0	5,4	5,7	177,2	190	5,62
66,0	65,0	6,2	0,0	262,2	260	0,79
11,9	10,4	-0,7	-0,6	216,7	190	9,49

Tablo 3.1.'de mm cinsinden alınan bazı örneklerin bulunan değerleri ve gerçekte olan değerleri sunulmuştur. Çok küçük hatalarla tespitler gerçekleştirilmiştir. Örneğin alınan birinci örnekte X değerinin sapması 2,7 mm kadardır.

3.3. Sonuç ve Öneriler

Bu çalışmada pozisyon çıkarımı algoritmaları ve görüntü işleme yöntemleri kullanılarak ortamdaki nesnenin tespiti yapılmıştır. Gerçekleştirilen sistemde tek iğne deliği kamera kullanılmış, bu kameranın kalibrasyonu gerçekleştirilmiş ve görüntülerin düzeltilmesi ile kamera lenslerindeki bozulmalar düzeltilmiştir.

Pozisyon çıkarımı için farklı yöntemler kullanılmıştır. İlk yöntemde doku kaplı nesnenin pozisyon çıkarımı yapılmış, ortalama 8-9 FPS ile eksenlerin net elde edilememesi ile sonuçlanmıştır. Bu işlem simülasyon için yeterli bulunmamıştır. Diğer bir yöntem olarak küresel koordinat sistemi kullanılmıştır. Bu sistemde renkli bir nesnenin pozisyon çıkarımı yapılmıştır. Bu yöntemde işaretleyiciler kullanılmıştır. İşaretleyiciler ile referans noktaları elde edildikten sonra küresel koordinatlarda pozisyon tahmini yapılmıştır. Aruco modülünün kullanım açısından satranç tahtası işaretleyicisinden daha kompleks olduğundan kullanılmamıştır. Kullanılacak olan işaretleyici satranç tahtası olarak seçilmiştir.

Tablo 3.1.'de verildiği gibi ölçüm sonuçları bulunmuş ve belirtilen hata oranlarıyla sistem gerçekleştirilmiştir. Geliştirilen sistemin avantajları olarak yalnızca görüntü bilgisinin değerlendirilmesiyle nesne ve koordinat tespiti yapabilmesi, gerçek zamanlı bir uygulamada kullanılabilecek düzeyde hızlı olması ve modüler yapısı sayesinde başka uygulamalara da entegre edilebilir olması sayılabilir.

Geliştirilen sistemin en büyük dezavantajı ise derinlik bilgisi elde edilmesi aşamasında kullanılan algoritmanın ortam şartlarına bağlı olarak çok çabuk bozulmaya uğramasıdır. Ortamın ışık kaynağının ışık şiddeti, arka plan, ortamdaki nesnenin hareket hızı ve bunlara bağlı olarak derinlik bilgisinin yanlış tespit edilmesi ölçülen değerlerle ilgili farklı sonuçların çıkmasına sebep olmaktadır. Bu sorun ortam şartlarının sabit olması durumunda çözülebilir. Bu sayede daha doğru sonuçlar elde edilebilir.

Küresel koordinat sistemi uygulaması ortalama 15-16 FPS ile çalışmaktadır. Uygulamanın gerçekliğini arttırmak için FPS değerinin gerçeğe yakın olması amaçlanmıştır. Bu nedenle 15-16 FPS yeterli bulunmamıştır.

4. İKİ KAMERA İLE POZİSYON TAHMİNİ

Bu bölümde, iki kamera kullanılarak nesnenin pozisyonun tespiti için yapılan işlemler açıklanmaktadır. Pozisyon çıkarımı için gereken ilk işlem kameraların kalibrasyonudur. Bu işlemden sonra içsel ve dışsal olarak iki parametre elde edilmektedir. Bu parametreler sunulacak olan pozisyon çıkarımı yöntemlerinde kullanılacaktır. Yazılım kısmı için Windows© 10 işletim sistemli, İntel® Core™ i7-4700HQ CPU @2.40GHz işlemciye sahip, 8,00 GB RAM ve 64 bit sisteme sahip bilgisayar altında çalışılmış olup, OpenCV 3.1 kütüphanesinden yararlanılmıştır.

Bölüm 4.1.'de stereo görme prensibi kullanılarak pozisyon tahminin uygulaması anlatılmıştır. Stereo görme prensibi ile ilgili genel tanımlamalar Bölüm 2'de verilmiştir. Bu bölümde aynı zamanda HSV renk uzayında belirlenen nesnenin derinlik bilgisinin stereo görme prensibi ile elde edilmesi için yapılan çalışmalar anlatılmıştır. Yazılımın gerçekleştirilmesi ve alt başlıkları de bu bölümde anlatılmıştır.

Bölüm 4.2.'de kartezyen koordinat sistemi kullanılarak renkli bir nesnenin pozisyon tahmini için yapılan çalışma anlatılmıştır. Çalışma kapsamında, renk tabanlı algılama yaklaşımı kullanarak nesnenin yatay ve dikey düzlemdeki konumunun ve bu nesnenin derinlik bilgisinin gerçek zamanlı tespiti ve takibini yapacak yazılım çalışması yapılmıştır. Nesne takibinde kullanılan OpenCV kütüphanesi ve HSV renk uzayı hakkında Bölüm 3.'de açıklama yapılmıştır. Yazılımın gerçekleştirilmesi ve alt başlıkları ise bu bölümde anlatılmıştır. Bu bölümün sonunda, bu yaklaşımın uygulanması ve deney sonuçları tartışılacaktır.

4.1. Stereo Görme Prensibi Kullanılarak Pozisyon Tahmini

Stereo görüş, bilgisayarla görme araştırma alanındaki önemli dallardan biridir. Stereo görme için verilebilecek en güzel örnek insan gözüdür. İnsan, iki gözü ile elde ettiği görüntüyü işleyerek, kendi etrafındaki cisimler hakkında bilgi sahibi olmaktadır. Stereo görmede de insan gözüne benzer şekilde iki kamera aracılığıyla, yazılımsal çalışmalar yapılarak dış dünya hakkında bilgi elde edilmektedir. Stereo görme prensibi ile ilgili detaylı bilgi Bölüm 2.'de verilmiştir.

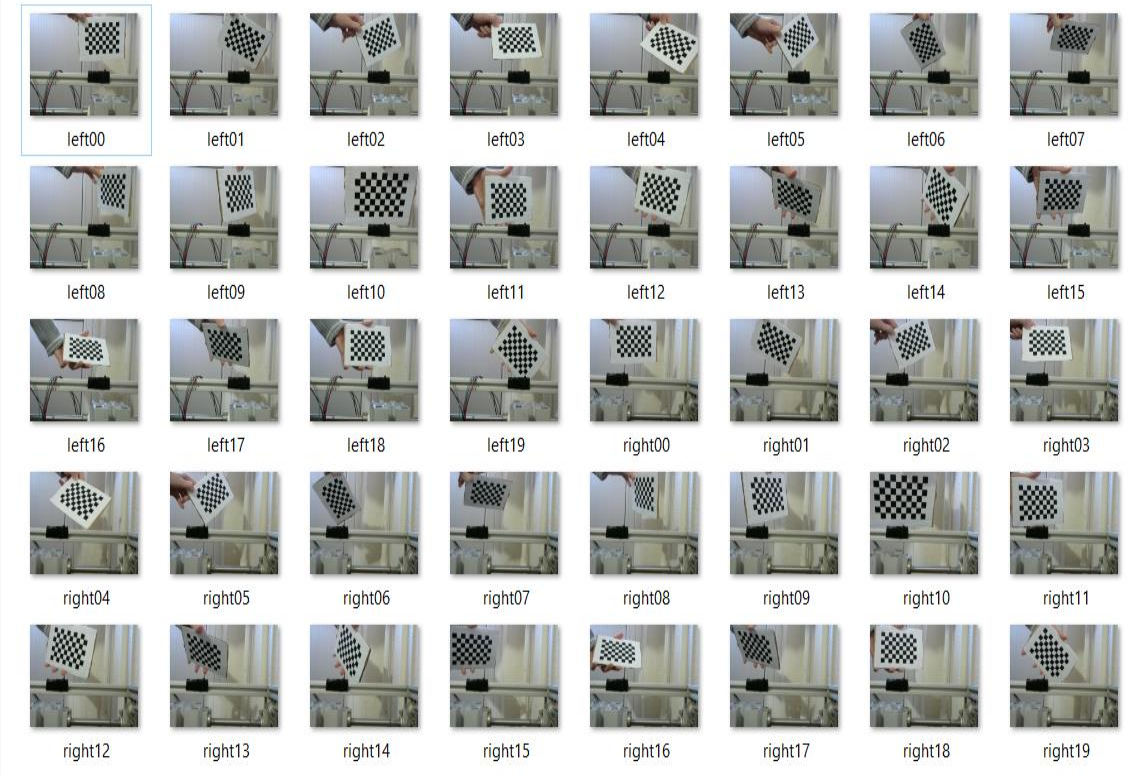
Bu bölümde, nesnenin 3D koordinat bilgisinin elde edilmesi, bir bilgisayarla görme metodu olan stereo görme kullanılarak görüntülerin işlenmesi ile gerçekleştirilmektedir.

İlk olarak web kameralardan alınan görüntüler ile kameraların kalibrasyonu yapılmıştır. Kalibrasyon sonucunda kameralara ait iç ve dış parametreler elde edilmiştir. Bu parametrelerin yardımıyla iki kameradan eş zamanlı alınan görüntüler düzeltme (*ing.* rectification) işlemi yapıldıktan sonra işlenerek, ortamın eşitsizlik (*ing.* disparity) haritası çıkarılmıştır. Bu haritadaki nesnenin kameralara bağlı konumu üçgen benzerliği kullanılarak tespit edilmiştir. Daha sonrasında nesneye olan mesafesi ile alakalı yazılımsal olarak elde edilen sonuçlar, gerçek dünyadaki mesafe bilgisi ile karşılaştırılarak hata analizi yapılmıştır.

Bölüm 4.1.4.'de geliştirilen sistemin uygulama senaryoları ile değerlendirilmesi yapılmış, sonucunda ise elde edilen sonuçlar yorumlanmıştır.

4.1.1. Stereo kamera kalibrasyon

Sistemin kalibrasyonunun yapılabilmesi için, iki adet Webcam ile satranç tahtası olarak adlandırılan deseni içeren 20'er çift fotoğraf, kameralardan aynı anda farklı açı ve uzaklıklarda çekilmiştir. İçsel ve dışsal parametrelerin hesaplanabilmesi için modeldeki tüm köşelerin ön işlemci fonksiyonu ile tespit edilmesi gerekmektedir.

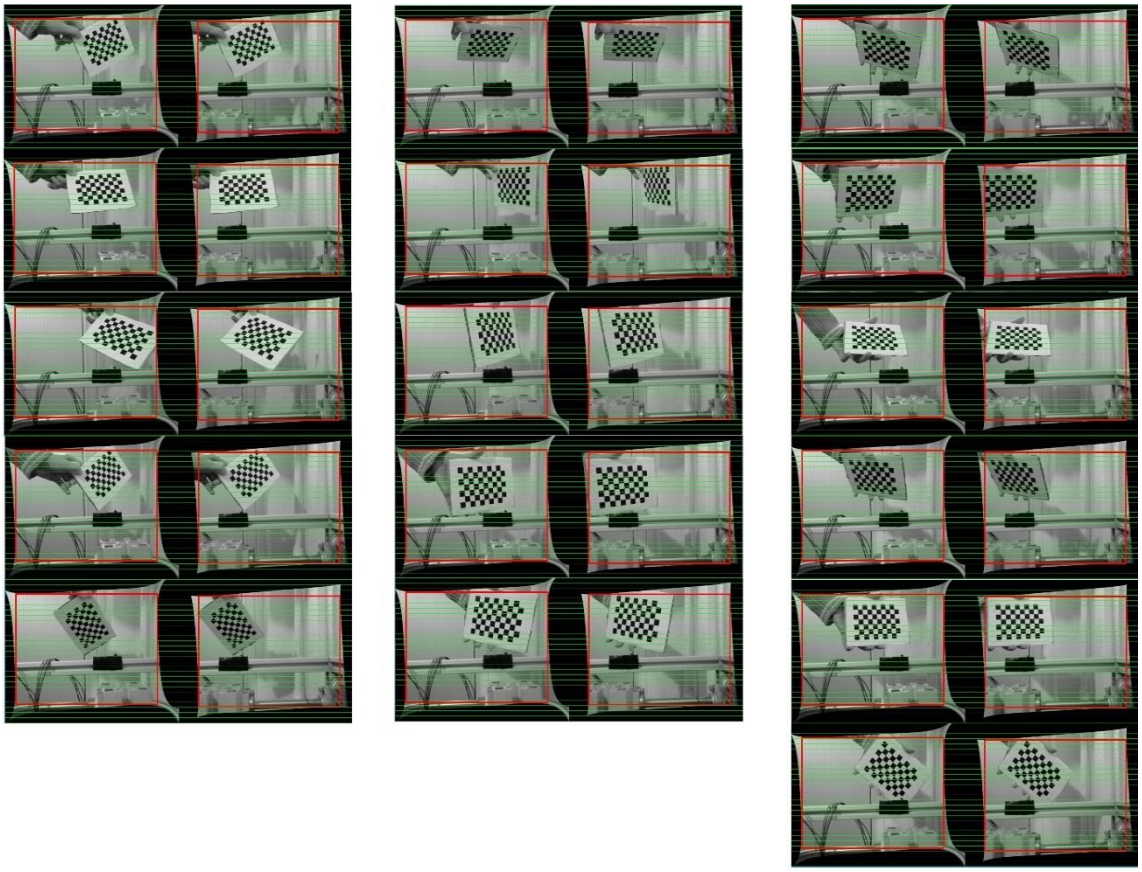


Görsel 4.1. *Sağ ve Sol Kameralardan Alınan Görüntüler*

Gerçekte hiçbir lens mükemmel değildir ve görüntüde üretim sürecinden kaynaklanan bozulmalar meydana gelir. Stereo görüş tekniklerinin doğrulukla uygulanabilmesi için kameraların kalibre edilmesi gerekmektedir. Kameralar ölçüleri bilinen bir satranç tahtası modelinin farklı açılardan ve uzaklıklardan eş zamanlı alınmış görüntüleri kullanılarak kalibre edilir. Kalibrasyon sonrasında elde edilen iç parametreler lens bozukluklarının düzeltilmesinde kullanılır. Dış parametreler ise kameralar arasındaki birbirlerine göreceli geometrik ilişkiyi göstermektedir. Stereo düzeltme işleminin ve derinlik kestiriminin gerçekleştirilebilmesi için kameralar arasındaki geometrik ilişkinin bilinmesi gereklidir.

4.1.2. Görüntülerin düzeltilmesi

Kalibrasyon işleminin ardından elde edilen kamera parametreleri ile görüntülerin rektifikasyonu yapılmaktadır. Rektifikasyon, epipolar doğruların görüntü satırlarında eşleşmesi amacıyla yapılan stereo görüntü çiftinin yeniden örneklenmesi işlemidir. Rektifikasyon matrisi oluşturabilmek için sistemin dışsal parametrelerine ihtiyaç duyulmaktadır.



Görsel 4.2. *Düzeltilmiş Kamera Görüntüleri*

Alınan 20 görüntüden 16'sı başarılı bir şekilde tespit edilmiştir.

Dışsal parametreler sonucunda elde edilen rektifikasyon matrisi kullanılarak kameralardan alınan görüntüler düzeltilir ve elde edilen yeni görüntülerinden aynı y ve z koordinatlarına sahip olması sağlanır. Bu işlem sonrasında düzeltilmiş görüntüler Görsel 4.2.'de gösterildiği gibi olmaktadır.

4.1.3. Stereo eşleme

Stereo eşleme problemi, 3 boyutlu bir noktanın farklı görüntü düzlemlerinde karşılık geldiği noktaları bulmaya çalışır. Eşleşen noktaların konumları arasındaki fark, stereo uzaklık (*ing.* disparity) değerini verir. Bu yöntem kullanılarak nesnenin derinlik bilgisi elde edilmiştir.

4.1.4. Stereo görme ile nesne tespiti

Geliştirilen sistem iki temel aşamadan oluşmaktadır. İlk aşamada öncelikle, optik eksenlerde dönme olmayacak şekilde, paralel olarak konumlandırılmış iki kameradan eş zamanlı olarak görüntü alımı yapılır. Ardından stereo eşleme yöntemi ile sağ ve sol

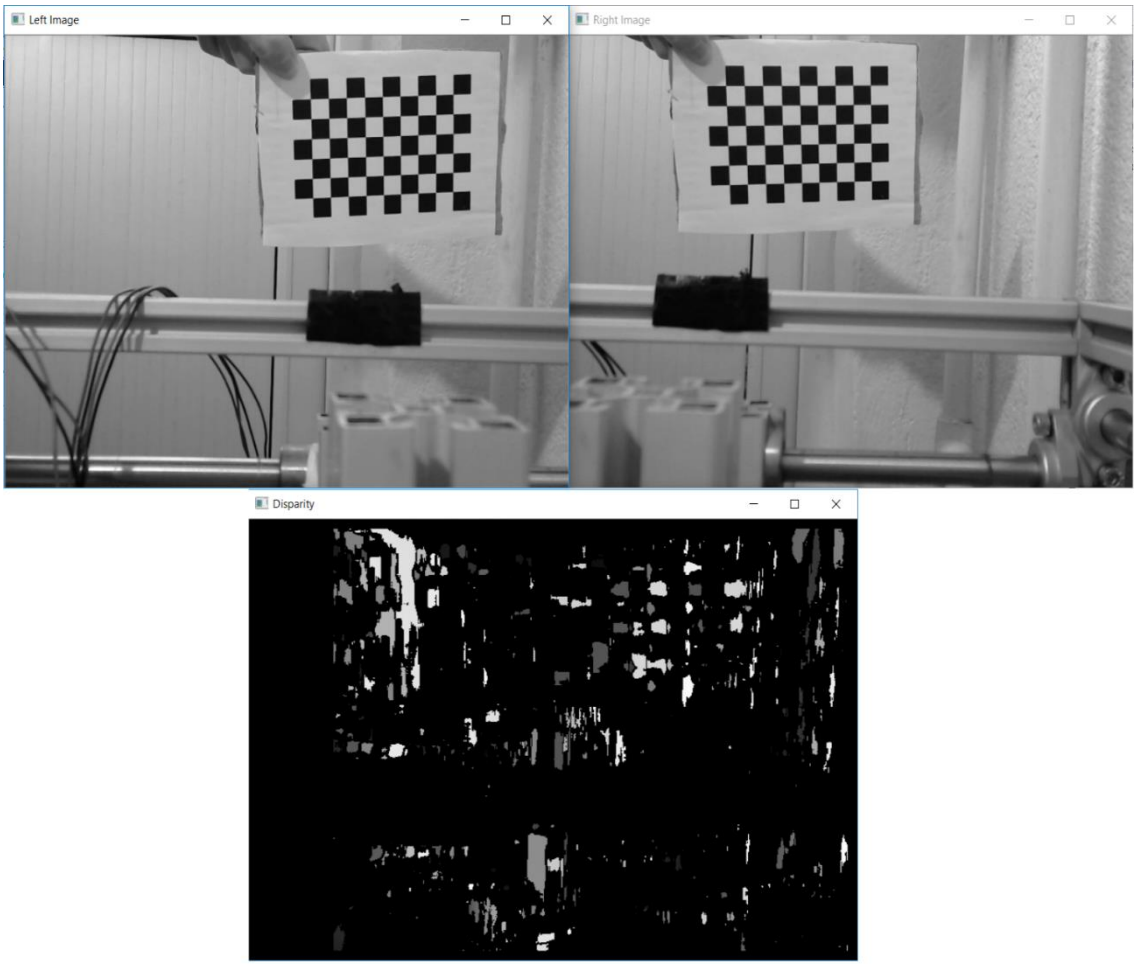
kameradan elde edilen eş zamanlı görüntüler kullanılarak stereo uzaklık haritası çıkarılır. İkinci aşamada ise kontur uygulayarak bulunan nesnenin konumu ve kameraya olan uzaklığı tespit edilir.

4.1.4.1. *Görüntü alımı*

Görüntü alımı, aralarındaki mesafe 8 cm olan iki adet 90 derece dönme açısına sahip, renkli video kamera ile 1280x720 çözünürlüğünde yapılmıştır. Kameralardan görüntü alımı yaparken kameraların birbirine paralel olacak şekilde konumlandırılması, stereo eşleme adımında arama uzayını tek boyuta indirir. Bu da çalışma zamanı konusunda önemli bir avantaj sağlar.

4.1.4.2. *Stereo uzaklık haritasının oluşturulması*

Bu çalışmada stereo eşleme problemi için OpenCV'nin eşleştirme kaynağı kullanılmıştır. Elde edilen uzaklık haritası Görsel 4.3.'deki gibidir.

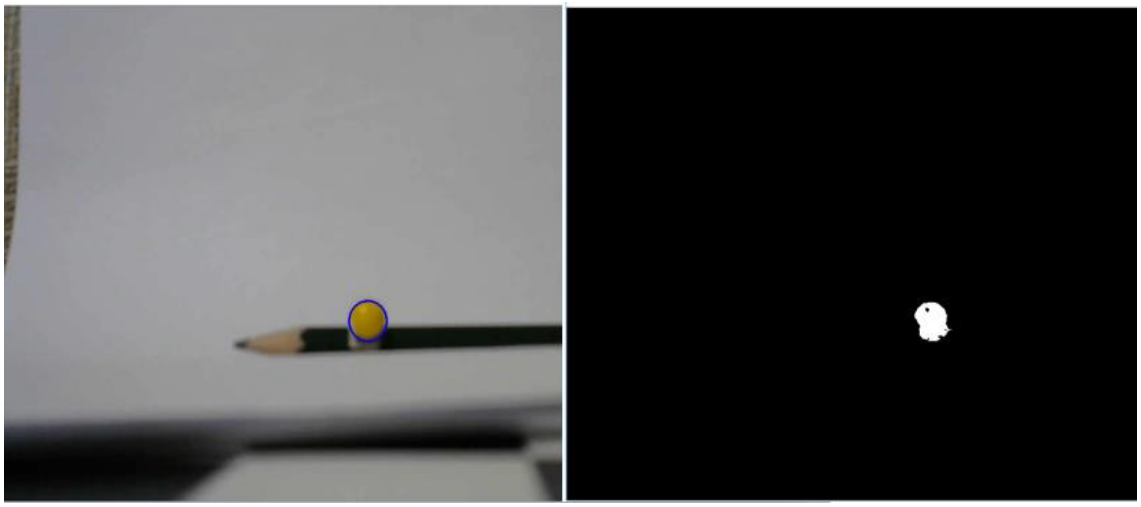


Görsel 4.3. *Stereo Uzaklık Haritasının Çıkarılması*

4.1.4.3. *Nesne tespiti*

Belirli bir renkteki nesnenin tespit edilmesi için renge göre resim segmentasyonu yapılmıştır. Kullanılan rengin HSV renk uzayındaki değerleri tespit edildi ve bu renge sahip olan nesne belirlendi. Alınan görüntülere filtrelemeler uygulanarak sağlam tespit yapılması sağlandı. Bu işlemler Bölüm 3.'de renkli nesne algılama kısmında detaylı anlatılmıştır.

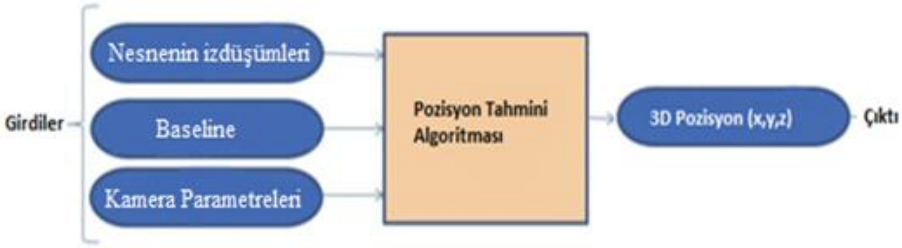
Renkli nesne tespiti her iki kameradan alınan resim içinde uygulanır. Tespit edilen nesnenin kontur bölgesini çevreleyen en küçük çember bulunur. Bulunan bu çemberin piksel koordinatları ve yarıçap bilgisi elde edilmektedir. Elde edilen veriler kullanılarak stereo görme prensibi ile derinlik bilgisi de elde edilmektedir.



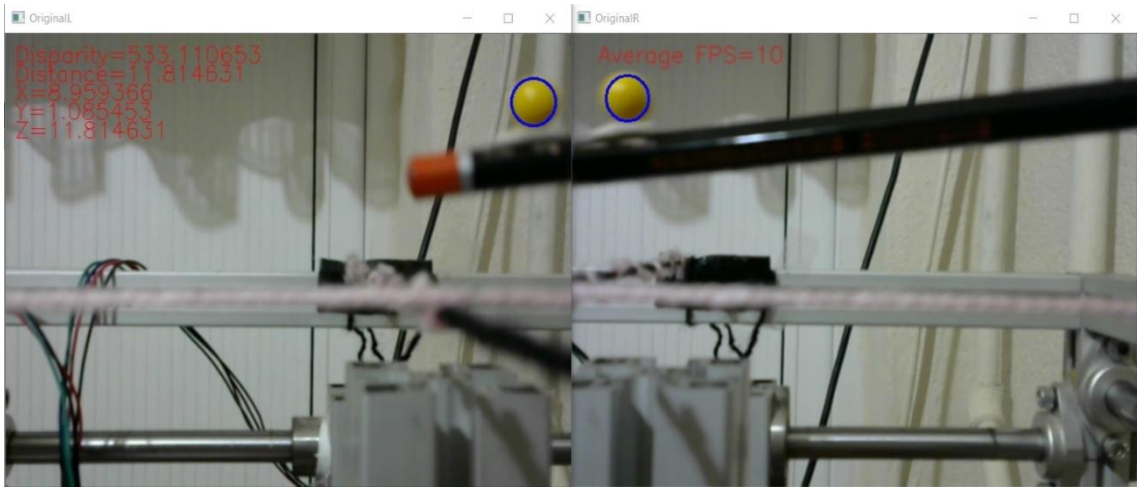
Görsel 4.4. *Kameradan Alınan Görüntüde Sarı Nesnenin Tespiti ve Kontur Uygulaması*

4.1.4.4. *Nesnenin konum tespiti*

Literatürde, stereo görme ile nesne tespiti yapan uygulamalar, nesnenin kameraya olan uzaklığını kameraların arasındaki mesafeyi, odak noktasını ve stereo uzaklık değerini kullanarak hesaplamaktadır (Eşitlik 2.49). Eşitlik 2.49'da kameralar arasındaki mesafe ve kameranın odak uzaklığı daha önceden belirlenmiş sabit değerlerdir. Tek değişken, tespit edilen nesnenin stereo uzaklık değeridir. Stereo uzaklık değeri ile nesnenin kameraya olan uzaklığı ters orantılı olarak değişmektedir. Algoritma yapısı Şekil 4.1'de gösterilmiştir.



Şekil 4.1. *Stereo Görme Pozisyon Çıkarımı Algoritma Yapısı*



Görsel 4.5. *Stereo Görme Uygulaması*

Uygulamanın çıktısı Görsel 4.5.'de gösterilmiştir. Sol ve sağ kameradan eş zamanlı alınan görüntüler ile sarı nesnenin 3D koordinatları tespit edilmiştir.

Nesnenin 3D koordinatlarını elde etmek için literatürde yaygın kullanılan eşitlikler kullanılmıştır (Eşitlik 2.47, Eşitlik 2.48, Eşitlik 2.49).

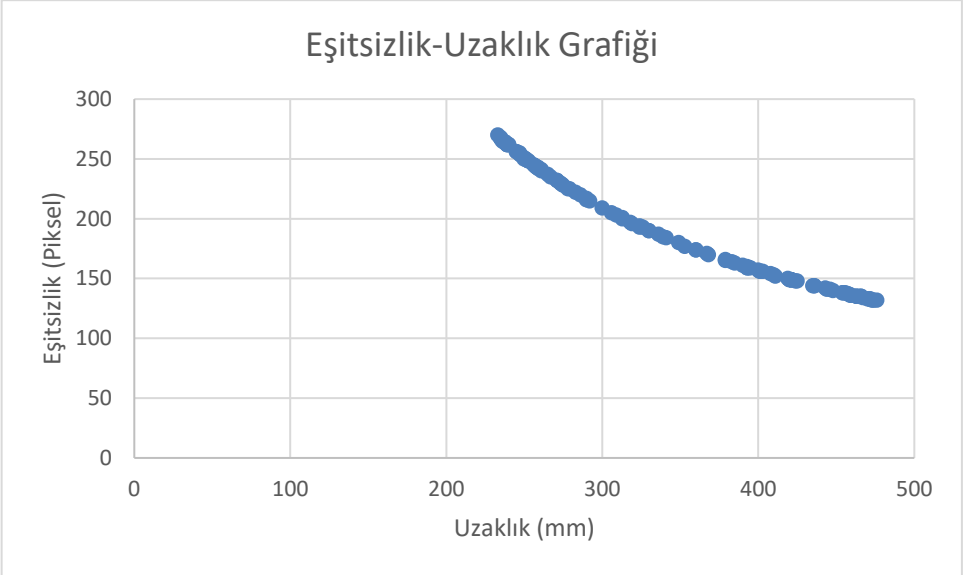
Stereo görme prensibine göre oluşturulan sistemde kameralar birbirine paralel olacak şekilde konumlandırılmıştır. Öncelikle eş zamanlı alınan görüntülerde kamera parametreleri kullanılarak resim düzeltme işlemi yapılmıştır. Bunun için aşağıdaki komut kullanılmıştır.

```
    initUndistortRectifyMap(M1, D1, R1, P1, Size(1280, 720), CV_16SC2,
cam1map1, cam1map2);
    initUndistortRectifyMap(M2, D2, R2, P2, Size(1280, 720), CV_16SC2,
cam2map1, cam2map2);
    Mat leftStereoUndistorted, rightStereoUndistorted;
    remap(imageL,     leftStereoUndistorted,     cam1map1,     cam1map2,
INTER_LINEAR);
    remap(imageR,     rightStereoUndistorted,     cam2map1,     cam2map2,
INTER_LINEAR);
```

Düzeltilen resimlerde sarı renkli nesne kontur uygulaması ile bulundu. Bulunan nesnenin piksel koordinatları elde edildi. Elde edilen bu piksel değerleri ile eşitsizlik değeri hesaplanmaktadır. Örneğin; Nesnenin sol resimdeki piksel değeri 300, sağ resimde 200 ise, bu nesnenin eşitsizlik değeri 300-200=100 olarak bulunmaktadır.

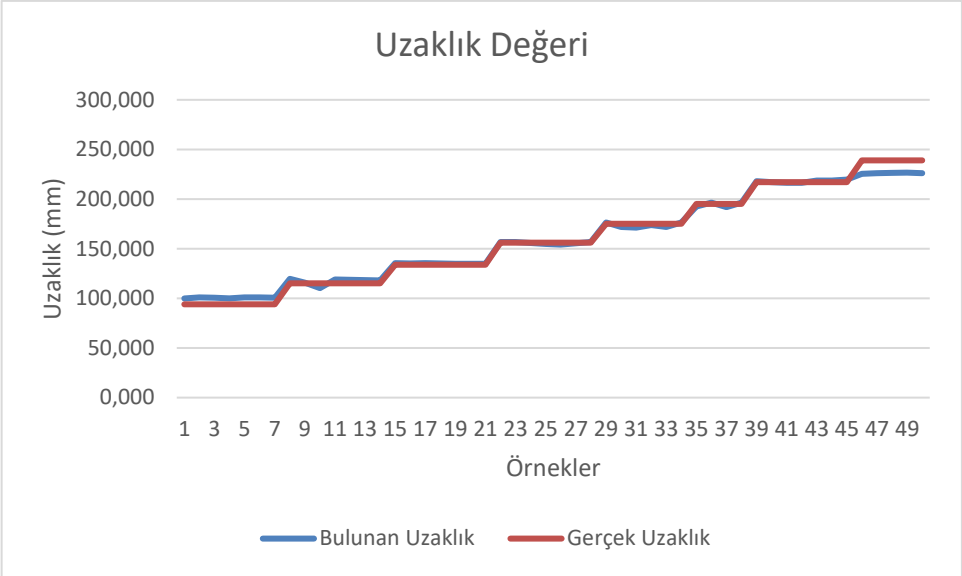
Kameraların odak noktaları arasındaki mesafe 8 cm olarak belirlenmiştir. Odak uzaklık değeri ise kamera kalibrasyon parametrelerinden elde edilmiştir. Bu sayede bulunan bu parametreler Eşitlik 2.49'da yerine koyularak nesnenin kameraya olan uzaklığı bulunmaktadır.

Stereo görme prensibinde sol kamera görüntüsü referans alınmaktadır. Bu nedenle x ve y konum bilgisi sol kameradan alınan x-y piksel değerleri kullanılarak hesaplanmaktadır. Nesnenin x-y koordinatları Eşitlik 2.47 ve Eşitlik 2.48'de kullanılarak bulunmaktadır.



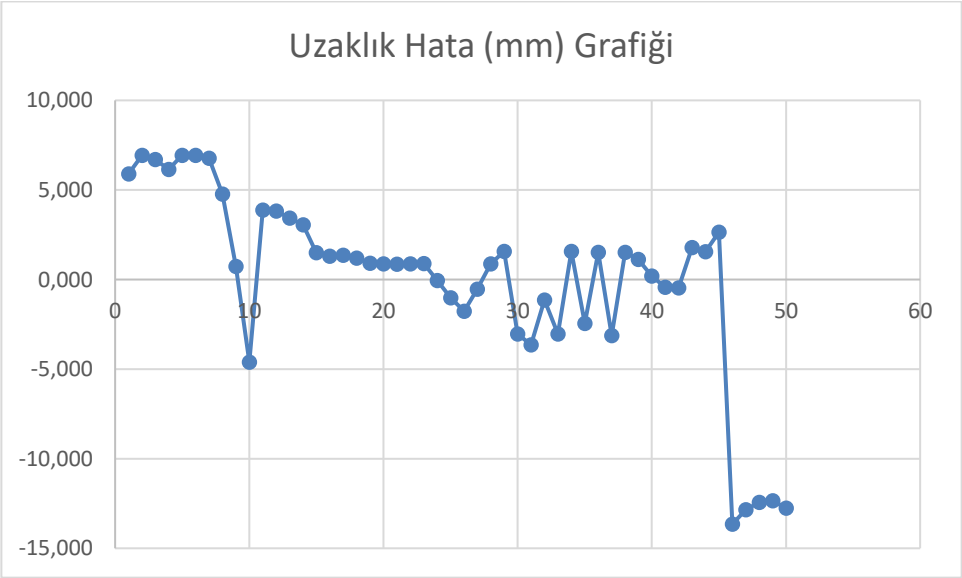
Şekil 4.2. *Eşitsizlik-Uzaklık Grafiği*

Eşitsizlik (*ing.* disparity) değeri ile nesnenin kameraya olan uzaklığı ters orantılı olarak değişmektedir. Şekil 4.2.'de eşitsizlik değeri azaldıkça nesnenin kameraya olan uzaklığı artmaktadır.



Şekil 4.3. *Uzaklık Değeri Grafiği*

Bu çalışmayı test etmek için kameraya olan uzaklıkları bilinen 50 örnek alındı ve bu pozisyonlara karşılık gelen değerler bulundu. Bulunan bu değerler ile gerçek değerler karşılaştırıldı. Sonuç olarak bulunan uzaklık değerinin gerçekte olması gereken değerle olan karşılaştırması Şekil 4.3.'de gösterilmiştir. Nesnenin kameraya olan uzaklığı ortalama %2,465 hata ile bulunmuştur.



Şekil 4.4. *Uzaklık Hata (mm) Grafiği*

Şekil 4.4.'da uzaklık değerinin hataları mm olarak ifade edilmiştir. Alınan örneklerde uzaklık değeri ortalama 3,3 mm'lik hata ile tespit edilmiştir.

Tablo 4.1. *Stereo Görme Sistemi Test İşlemleri Sonucunda Elde Edilen Hata Oranları*

Eşitsizlik (Piksel)	Bulunan Uzaklık (mm)	Gerçek Uzaklık (mm)	Hata (%)
628.5	100.215	94.0	6.611
544.186	115.742	115.0	0.645
467.39	134.759	134.0	0.566
403.912	155.937	156.0	0.040
356.714	176.570	175.0	0.897
320.503	196.519	195.0	0.778
292.885	215.051	217.0	0.898
278.386	226.251	239.0	5.334

Tablo 4.1.'de mm cinsinden alınan bazı örneklerin bulunan değerleri ve gerçekte olan değerleri sunulmuştur. Bulunan değerlerin hata oranları tespit edilmiştir. Örneğin alınan birinci örnekte uzaklık değerinin sapması 6,215 mm kadardır. Uygulama ortalama 11 FPS ile çalışmaktadır.

4.2. Kartezyen Koordinat Sistemi Kullanılarak Pozisyon Tahmini

Bu bölümde kartezyen koordinat sistemi kullanılarak renkli bir nesnenin pozisyon tahmini için yapılan çalışma anlatılmıştır. Çalışma kapsamında, renk tabanlı algılama yaklaşımı kullanarak nesnenin yatay ve dikey düzlemdeki konumunun gerçek zamanlı tespiti ve takibini yapacak yazılım çalışması yapılmıştır.

İlk olarak web kameralardan alınan görüntüler ile kameraların kalibrasyonu yapılmıştır. Kalibrasyon sonucunda kameralara ait iç ve dış parametreler elde edilmiştir. Bu parametrelerin yardımıyla iki kameradan eş zamanlı alınan görüntüler düzeltme (*ing.* rectification) işlemi yapılmıştır.

Bu bölümde, nesnenin 3D koordinat bilgisinin elde edilmesi, kartezyen koordinat sistemi kullanılarak görüntülerin işlenmesi ile gerçekleştirilmektedir.

Öncelikle bölüm 4.2.1.'de kartezyen koordinat sistemi hakkında bilgi verilmiştir. Bu bilgiler dahilinde 2D koordinat sistemine z düzlemindeki değerin eklenmesi ile pozisyon tahmini gerçekleştirildi.

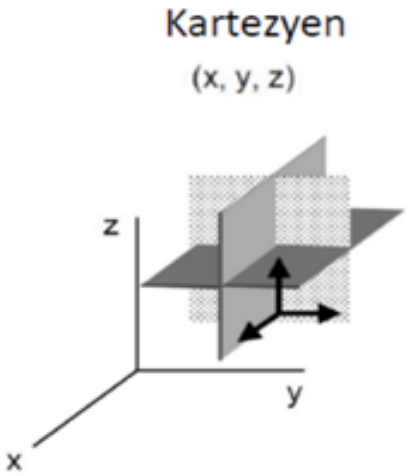
3D koordinat bilgisinin elde edilmesi için sarı renkli bir nesne kullanılmıştır. OpenCV görüntü işleme yöntemleri ile bu nesne tespit edilerek, bu nesnenin yatay ve dikey koordinatları tespit edildikten sonra bu bilgiler ile pozisyon çıkarımı yapılmıştır.

Bölüm 4.2.2.'de geliştirilen sistemin uygulama senaryoları ile değerlendirilmesi yapılmış, devamında ise elde edilen sonuçlar yorumlanmıştır.

Daha sonrasında nesnenin yazılımsal olarak elde edilen 3D koordinatları, gerçek dünyadaki koordinatları ile karşılaştırılarak hata analizi yapılmıştır. Kullanılan yöntemin çalışma zamanı değerlendirilmiş. Bu değerlendirmeler grafikler ile gösterilmiştir.

4.2.1. Kartezyen koordinat sistemi

Uzayda bir noktayı göstermek ve vektörleri görselleştirerek daha kolay anlaşılmasını sağlamak için koordinat sisteminden faydalanılır. Verilen bir vektör matematiksel olarak seçilen koordinat sistemi üzerinde bileşenlerine ayrılarak ifade edilir. Uzayda çok sayıda dikgen (*ing.* orthogonal) koordinat sistemi mevcuttur. Burada dikgen terimi koordinat sistemi içinde her bir noktanın birbirlerine dik üç yüzeyin kesişimi ile tanımlanabileceğini anlatmaktadır. Verilen bir vektör ifadesi için koordinat sistemleri arasında dönüşüm yapmak mümkündür.



Şekil 4.5. *Kartezyen Koordinat Sistemleri*

Kartezyen koordinat sistemi, üç boyutlu uzayda nokta belirlemenin bir yoludur. Şekil 4.5.'da kartezyen koordinat sistemi gösterilmektedir.

4.2.2. Kartezyen koordinat sistemi ile nesne tespiti

4.2.2.1. *Görüntü alımı*

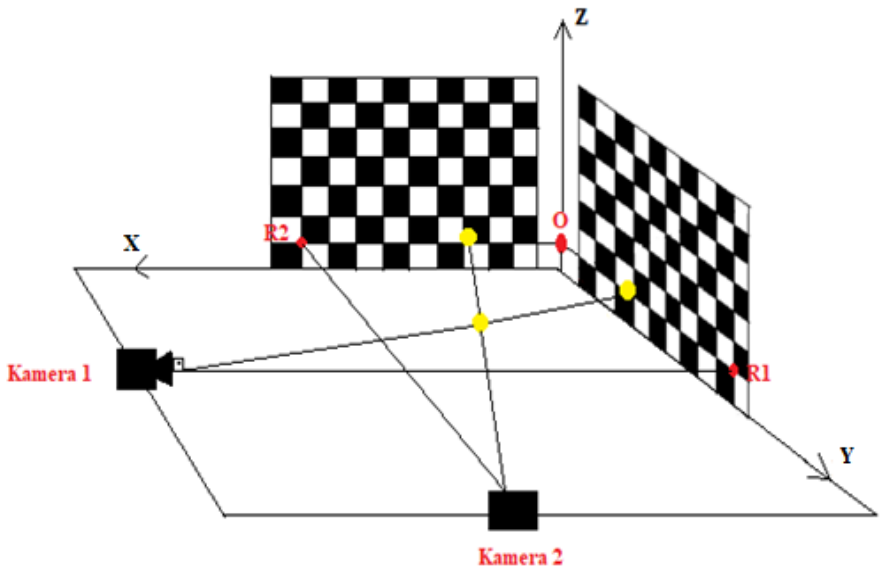
Görüntü alımı, iki adet 90 derece dönme açısına sahip, renkli video kamera ile 640x480 çözünürlüğünde yapılmıştır. Kameralardan görüntü alımı yaparken kameraların birbirine 90 derece açı olacak şekilde konumlandırılmıştır.

4.2.2.2. *Nesne tespiti*

Belirli bir renkteki nesnenin tespit edilmesi için renge göre resim segmentasyonu yapılmıştır. Kullanılan rengin HSV renk uzayındaki değerleri tespit edildi ve bu renge sahip olan nesne belirlendi. Alınan görüntülere filtrelemeler uygulanarak sağlam tespit yapılması sağlandı. Bu işlemler Bölüm 3.'de renkli nesne algılama kısmında detaylı anlatılmıştır.

4.2.2.3. *Nesnenin konum tespiti*

Kartezyen koordinat sistemi tasarımı Şekil 4.6.'da gösterilmiştir. Bu tasarıma göre kameralar kullanılan 9x6 boyutunda iki satranç tahtası modelinin referans noktalarına dik olarak konumlandırılmaktadır. Referans noktaları Şekil 4.6.'da R1 ve R2 ile gösterilmiştir. O noktası orijin (0,0,0) noktasını ifade etmektedir.

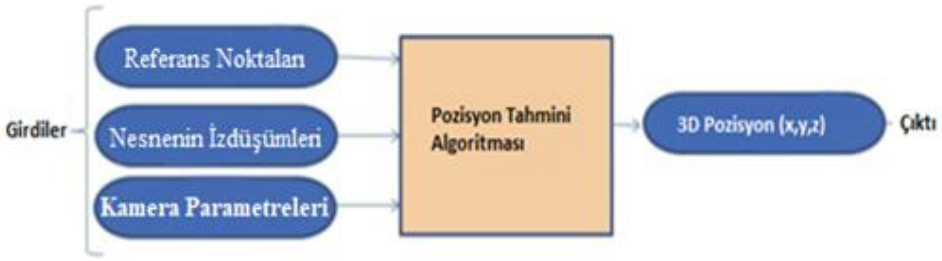


Şekil 4.6. *Kartezyen Koordinat Sistemi Tasarımı*

Kullanılan sarı renkteki 3D nesnenin projeksiyon ekranlarındaki izdüşüm değerlerine göre pozisyon tahmini yapılmaktadır. Nesnenin kamera 1 ekranındaki izdüşümünün R1 noktasına yataydaki uzaklığı ile kamera 2 ekranındaki izdüşümünün R2 noktasına yataydaki uzaklığı elde edilmiştir. Elde edilen bu değerler kullanılarak iterasyon yöntemi ile gerçek x, y değerleri elde edilmiştir. Nesnenin bu referans noktalarına olan dikey uzaklıklarının ortalaması z değerini vermektedir.

Kamera 1'den alınan görüntüde nesnenin O noktasına olan uzaklığı nesnenin x değerini elde ederken kullanılmıştır. Aynı şekilde Kamera 2'den alınan görüntüde nesnenin O noktasına olan uzaklığı nesnenin y değerini elde ederken kullanılmıştır. Her iterasyonda orijine olan uzaklık değerleri değiştiğinden yeni x ve y değerleri elde edilmektedir. Elde edilen yeni x değeri ile bir önceki arasındaki değişim 0.1 mm'nin altına düştüğünde iterasyona son verilmektedir.

Her iterasyonda bir değerin elde edilmesinde üçgen benzerliği kullanılmaktadır. Örneğin Kamera 1'den alınan görüntüde nesnenin izdüşüm değeri, R1 ve Kamera 1 bir üçgen varsayılmaktadır. Kamera 2'den alınan orijine uzaklık değeri ile Kamera 1'in R1 noktasına olan uzaklığı oranı, y test değeri olarak bilinmeyen değer ile nesnenin Kamera 1'deki izdüşümün R1'e olan uzaklığı oranı eşitlenerek y test değeri elde edilmektedir. Bu yeni elde edilen değer ile bir önceki değer arasındaki fark 0.1'den küçük olana kadar devam etmektedir. Elde edilen son değer nesne izdüşümü O-R1 noktası arasında ise R1'e çıkarılır aksi durumda eklenerek bulunmaktadır.



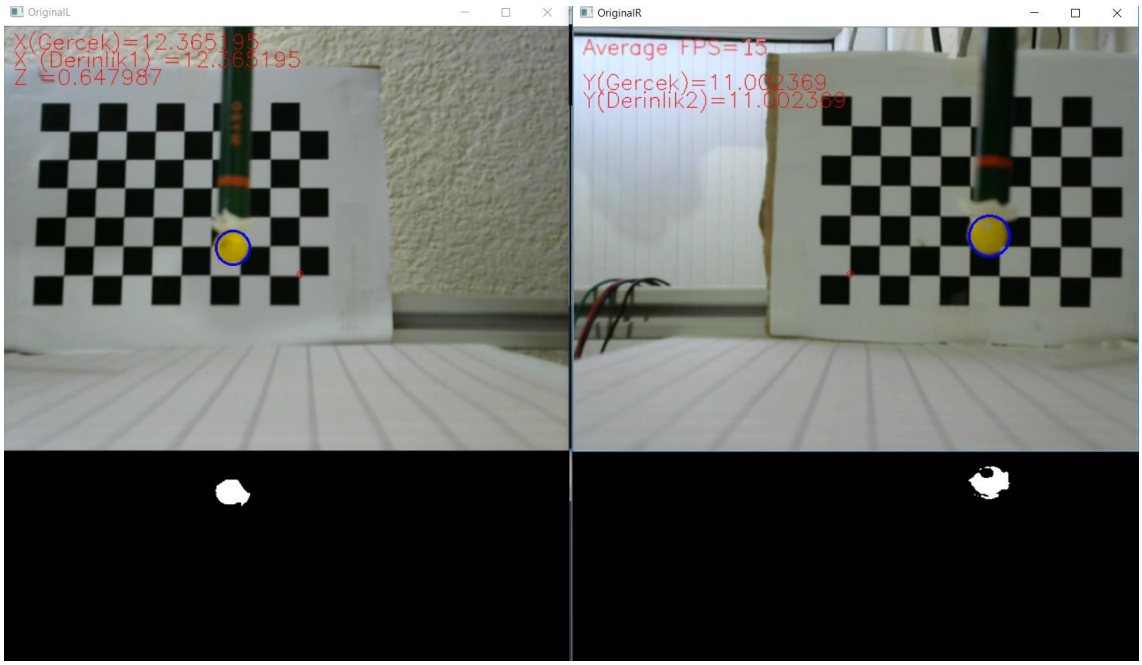
Şekil 4.7. *Kartezyen Koordinat Sistemi Pozisyon Çıkarımı Algoritma Yapısı*

Şekil 4.7.'de pozisyon tahmini algoritmasının yapısı verilmiştir. Algoritma girdi olarak referans noktaları (R1, R2), nesnenin iki kamerada oluşan izdüşüm değerleri ve kamera parametrelerini almaktadır. Pozisyon tahmini algoritmasına verilen bu değerler ile nesnenin 3D koordinatları bulunmaktadır.

Referans noktası orijin (0,0,0) olarak varsayılır. Kamera 1 ve Kamera 2'den alınan eş zamanlı görüntülerde nesnenin konumu referans noktasına göre belirlenmektedir. Nesnenin x eksenindeki konumu Kamera 2'den alınan görüntüde nesnenin referans noktasına olan uzaklığı ile ifade edilmektedir. Nesnenin y eksenindeki konumu Kamera 1'den alınan görüntüde nesnenin referans noktasına olan uzaklığı ile ifade edilir. Aynı şekilde nesnenin z eksenindeki konumu iki kameradan alınan görüntülerde nesnenin referans noktasına dikey olan uzaklıklarının ortalaması ile ifade edilmektedir.

Görsel 4.6. *Kartezyen Koordinat Sistem Ekipmanlarının Görüntüsü*

Görsel 4.6.'da gerçekleştirilen sistem verilmiştir. Referans noktasına dik konumlandırılan kamera ile 15x15 cm'lik bir alanda simülasyon işlemi gerçekleştirilmiştir.

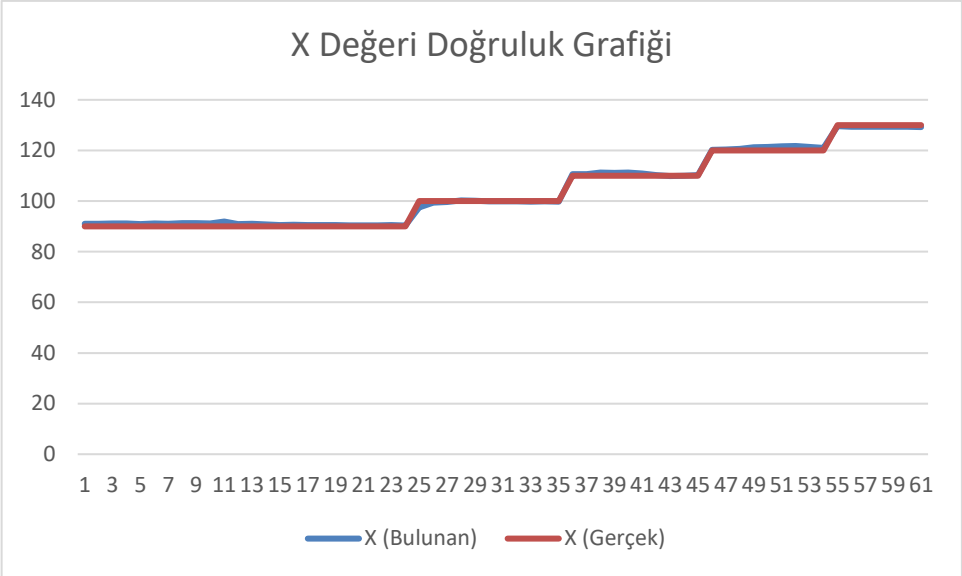


Görsel 4.7. *Kartezyen Koordinatlar Pozisyon Çıkarımı Uygulaması*

Uygulamanın çıktısı Görsel 4.7.'de gösterilmiştir. Sol ve sağ kameradan eş zamanlı alınan görüntüler ile sarı nesnenin 3D koordinatları tespit edilmiştir.

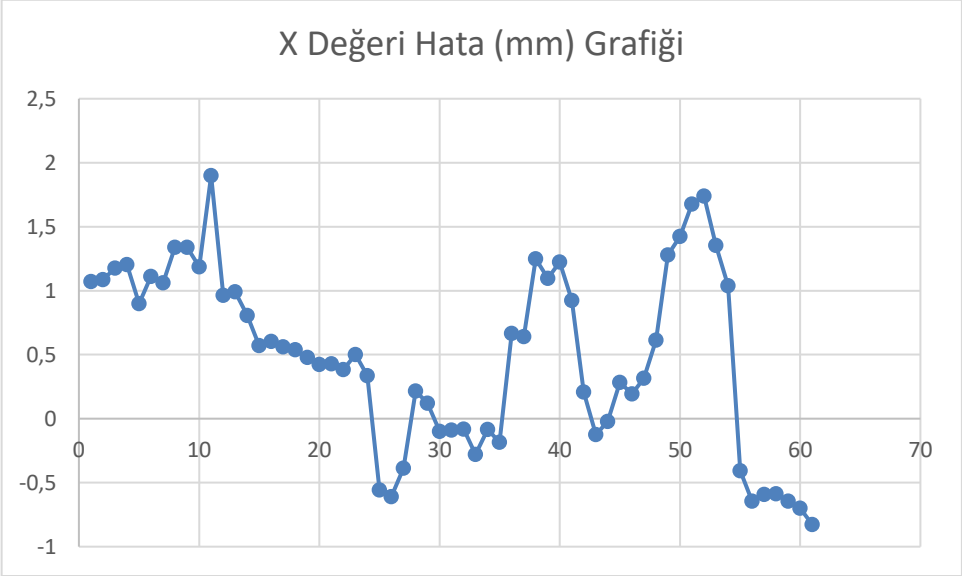
Kartezyen koordinat sistemine göre oluşturulan sistemde kameralar birbirine dik olacak şekilde konumlandırılmıştır. Öncelikle eş zamanlı alınan görüntülerde kamera parametreleri kullanılarak resim düzeltme işlemi yapılmıştır.

Bu çalışmayı test etmek için 3D pozisyonları bilinen örnekler alındı ve bu pozisyonlara karşılık gelen değerler bulundu. Bulunan bu değerler ile gerçek değerler karşılaştırıldı. Sonuç olarak X, Y ve Z değeri doğruluk grafikleri Şekil 4.8., Şekil 4.10. ve Şekil 4.12.'te gösterilmiştir. X, Y ve Z hata grafikleri Şekil 4.9., Şekil 4.11. ve Şekil 4.13.'te gösterilmiştir.



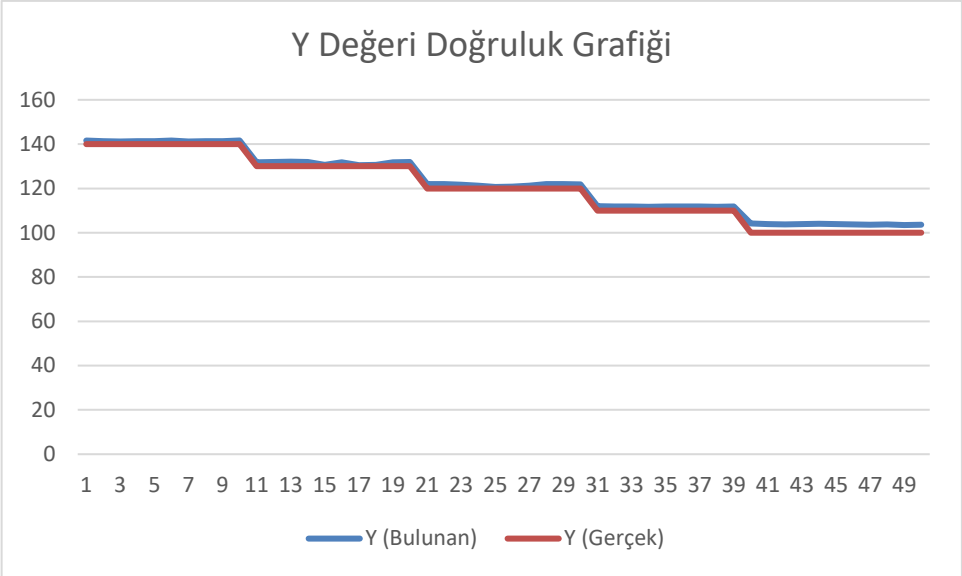
Şekil 4.8. *X Değeri Doğruluk Grafiği*

Bulunan X değerleri gerçek değerlere çok yakın değerlerde bulunmuştur. Alınan örneklerde X değeri %0,709 hata oranlarında tespit edilmiştir.



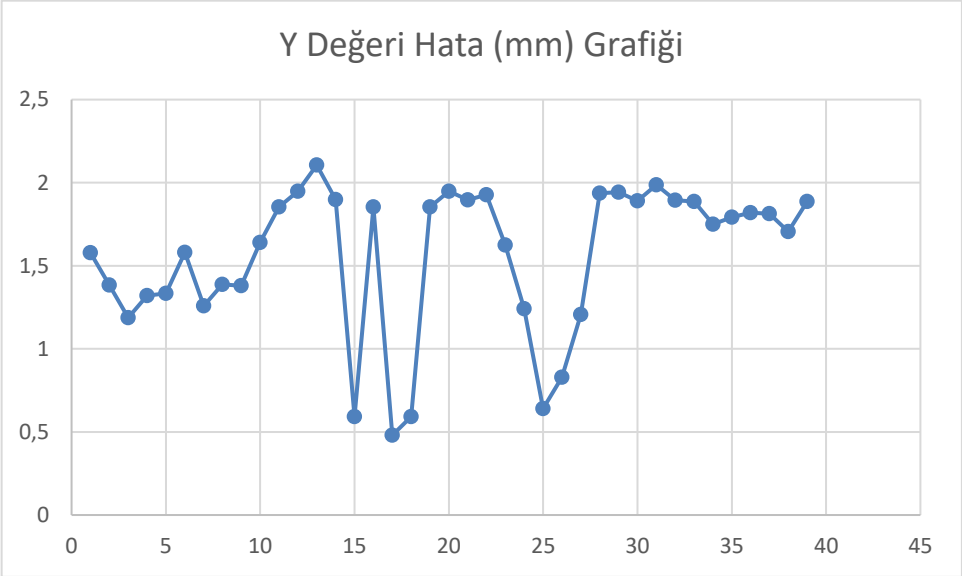
Şekil 4.9. *X Değeri Hata (mm) Grafiği*

Şekil 4.9.'da X değerinin hataları mm olarak ifade edilmiştir. Alınan örneklerde X değeri ortalama 0,7236 mm'lik hata ile tespit edilmiştir.



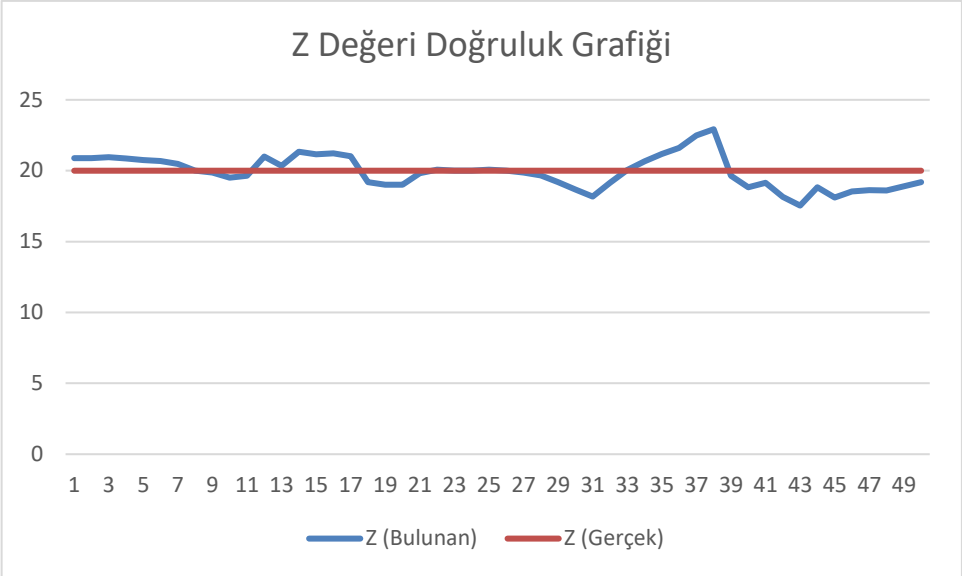
Şekil 4.10. *Y Değeri Doğruluk Grafiği*

Alınan örneklerde Y değeri %1,264 hata oranlarında tespit edilmiştir. Şekil 4.10.'te görülen sapmalar mm'lik değerlerdir.



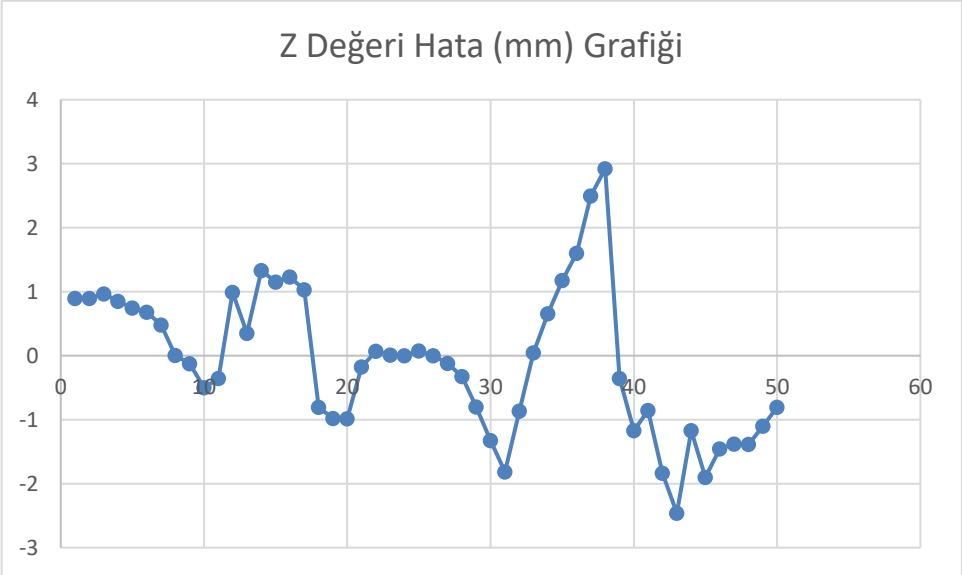
Şekil 4.11. *Y Değeri Hata (mm) Grafiği*

Şekil 4.11.'de Y değerinin hataları mm olarak ifade edilmiştir. Alınan örneklerde Y değeri ortalama 1,56 mm'lik hata ile tespit edilmiştir.



Şekil 4.12. *Z Değeri Doğruluk Grafiği*

Alınan örneklerde Z değeri %4,57 hata oranlarında tespit edilmiştir. Tüm koordinatlar ortalama %2,17 hata oranlarında bulunmuştur.



Şekil 4.13. *Z Değeri Hata (mm) Grafiği*

Şekil 4.13.'de Z değerinin hataları mm olarak ifade edilmiştir. Alınan örneklerde Z değeri ortalama 0,9542 mm'lik hata ile tespit edilmiştir.

Tablo 4.2. *Kartezyen Koordinat Sistemi Test İşlemleri Sonucunda Elde Edilen Değerler*

X(Bulunan)	X(Gerçek)	Y(Bulunan)	Y(Gerçek)	Z(Bulunan)	Z(Gerçek)
91,0702	90,0	103,670	100,0	20,8937	20,0
91,0875	90,0	103,458	100,0	20,8948	20,0
91,1762	90,0	103,545	100,0	20,9641	20,0
90,8985	90,0	103,645	100,0	20,7452	20,0
99,6126	100,0	111,886	110,0	20,8527	20,0
99,3896	100,0	111,750	110,0	20,6822	20,0
100,216	100,0	111,706	110,0	19,5006	20,0
100,122	100,0	111,813	110,0	20,4815	20,0
110,208	110,0	121,890	120,0	20,9895	20,0
111,248	110,0	121,943	120,0	19,1470	20,0
111,097	110,0	120,641	120,0	19,6429	20,0
109,979	110,0	120,829	120,0	20,0056	20,0
120,315	120,0	130,593	130,0	21,6003	20,0
120,613	120,0	130,480	130,0	21,1749	20,0
121,279	120,0	131,898	130,0	20,6544	20,0
121,677	120,0	131,854	130,0	19,8739	20,0
129,174	130,0	141,579	140,0	20,0476	20,0
129,301	130,0	141,188	140,0	19,1366	20,0
129,409	130,0	141,640	140,0	20,0736	20,0
129,414	130,0	141,384	140,0	20,3495	20,0

Tablo 4.2.'de mm cinsinden alınan bazı örneklerin bulunan değerleri ve gerçekte olan değerleri sunulmuştur. Çok küçük hatalarla tespitler gerçekleştirilmiştir. Örneğin alınan birinci örnekte X değerinin sapması 1,07 mm kadardır. Bu sapma uygulamanın doğruluğunu büyük oranda etkilememektedir.

Alınan örneklerde X değeri ortalama 0,7236 mm, Y değeri ortalama 1,56 mm ve Z değeri ortalama 0,9542 mm hata ile tespit edilmiştir.

Uygulama ortalama 15-16 FPS ile çalışmaktadır. Uygulamanın gerçekliğini arttırmak için pozisyon tahmininin yanı sıra FPS değerinin gerçeğe yakın olması amaçlanmaktadır. Bu nedenle uygulama yeterli bulunmamıştır.

4.3. Sonuç ve Öneriler

Bu çalışmada pozisyon çıkarımı algoritmaları ve görüntü işleme yöntemleri kullanılarak ortamdaki nesnenin tespiti yapılmıştır. Gerçekleştirilen sistemde iki kamera kullanılmış, bu kameraların kalibrasyonu gerçekleştirilmiştir. Elde edilen kamera parametreleri ile görüntüler ve kamera lenslerindeki bozulmalar düzeltilmektedir.

Pozisyon çıkarımı için farklı yöntemler kullanılmıştır. İlk olarak stereo görme prensibi ele alınmıştır. Bu yöntemde web kameralardan alınan görüntüler ile kameraların kalibrasyonu yapılmıştır. Kalibrasyon sonucunda kameralara ait iç ve dış

parametreler elde edilmiştir. Bu parametrelerin yardımıyla iki kameradan eş zamanlı alınan görüntüler düzeltme (*ing.* rectification) işlemi yapıldıktan sonra işlenerek, ortamın eşitsilik(*ing.* disparity) haritası çıkarılmıştır. Bu haritadaki nesnenin kameralara bağlı konumu üçgen benzerliği kullanılarak tespit edilmiştir. Bu algoritma kamera parametreleri, nesnenin iki kameradaki izdüşümleri ve odak noktaları arasındaki uzaklığı (*ing.* baseline) girdi olarak almaktadır. Daha sonrasında nesneye olan mesafesi ile alakalı yazılımsal olarak elde edilen sonuçlar, gerçek dünyadaki mesafe bilgisi ile karşılaştırılarak hata analizi yapılmıştır.

Bu çalışmayı test etmek için kameraya olan uzaklıkları bilinen örnekler alındı ve bu pozisyonlara karşılık gelen değerler bulundu. Bulunan bu değerler ile gerçek değerler karşılaştırıldı. Şekil 4.3.'de elde edilen sonuçlar verilmiştir. Bu değerlere göre nesnenin kameraya olan uzaklığı ortalama %2,465 hata ile bulunmuştur.

Uygulama ortalama 11 FPS ile çalışmaktadır. Uygulamanın FPS oranın düşük olması çalışmanın gerçeklik oranını düşürmektedir. Bu nedenle FPS değerinin arttırılması gerekmektedir.

Tablo 4.1.'de verildiği gibi ölçüm sonuçları bulunmuş ve belirtilen hata oranlarıyla sistem gerçekleştirilmiştir. Geliştirilen sistemin avantajları olarak yalnızca görüntü bilgisinin değerlendirilmesiyle nesne ve koordinat tespiti yapabilmesi, gerçek zamanlı bir uygulamada kullanılabilecek düzeyde hızlı olması ve modüler yapısı sayesinde başka uygulamalara da entegre edilebilir olması sayılabilir. Bu çalışmanın dezavantajı nesne tespiti aşamasında kullanılan algoritmanın ortam şartlarına bağlı olarak çabuk bozulmaya uğramasıdır.

Bu bölümde iki kamera kullanılarak geliştirilen diğer yöntem ise kartezyen koordinat sistemidir. Bu sistemde kullanılan algoritma girdi olarak referans noktaları, nesnenin iki kamerada oluşan izdüşüm değerleri ve kamera parametrelerini almaktadır. Pozisyon tahmini algoritmasına verilen bu değerler ile nesnenin 3D koordinatları bulunmaktadır. Kullanılan sarı renkteki 3D nesnenin projeksiyon ekranlarındaki izdüşüm değerlerine göre pozisyon tahmini yapılmaktadır. Nesnenin birinci kamera ekranındaki izdüşümünün birinci referans noktasına yataydaki uzaklığı ile ikinci kamera ekranındaki izdüşümünün ikinci referans noktasına yataydaki uzaklığı elde edilmiştir. Elde edilen bu değerler kullanılarak iterasyon yöntemi ile gerçek x, y değerleri elde edilmiştir. Nesnenin bu referans noktalarına olan dikey uzaklıklarının ortalaması z değerini vermektedir.

Bu değerleri test etmek için alınan örnekler üzerinde değerlendirmeler yapılmıştır. Bu değerlendirmelere göre X değeri %0,709 hata oranlarında ve ortalama 0,7236 mm'lik hata ile tespit edilmiştir. Y değeri %1,264 hata oranlarında ve ortalama 1,56 mm'lik hata ile tespit edilmiştir. Son olarak Z değeri %4,57 hata oranlarında ve ortalama 0,9542 mm'lik hata ile tespit edilmiştir.

Tablo 4.2.'de mm cinsinden alınan bazı örneklerin bulunan değerleri ve gerçekte olan değerleri sunulmuştur. Çok küçük hatalarla tespitler gerçekleştirilmiştir. Bu hatalar uygulamanın doğruluğunu büyük oranda etkilememektedir.

Uygulama ortalama 15-16 FPS ile çalışmaktadır. Uygulamanın gerçekliğini arttırmak için pozisyon tahmininin yanı sıra FPS değerinin gerçeğe yakın olması amaçlanmaktadır. Bu nedenle FPS değeri yeterli bulunmamıştır.

5. RASPBERRY PI KAMERA MODÜLÜ İLE POZİSYON TAHMİNİ

Bu bölümde, Raspberry Pi kamera modülü kullanılarak nesnenin pozisyonun tespiti için yapılan işlemler açıklanmaktadır.

Bölüm 5.1.'de kullanılacak olan Raspberry Pi modülü hakkında bilgi verilmiştir. Raspberry Pi için işletim sistemi kurulumu, kullanılacak olan programların ve OpenCV kütüphanesinin kurulumu açıklanmıştır.

Bölüm 5.2.'de kartezyen koordinat sistemi kullanılarak aktif işaretçinin pozisyon tahmini için yapılan çalışma anlatılmıştır. Çalışma kapsamında, renk tabanlı algılama yaklaşımı kullanarak nesnenin yatay ve dikey düzlemdeki konumunun ve bu nesnenin derinlik bilgisinin gerçek zamanlı tespiti ve takibini yapacak yazılım çalışması yapılmıştır. Yazılım kısmı için Windows© 10 işletim sistemli, İntel® Core™ i7-4700HQ CPU @2.40GHz işlemciye sahip, 8,00 GB RAM ve 64 bit sisteme sahip bilgisayar ve Raspbian işletim sistemli Raspberry Pi kamera modülleri altında çalışılmış olup, OpenCV 3.1 kütüphanesinden yararlanılmıştır.

Raspberry kamera modülü ile elde edilen veriler Windows işletim sistemine sahip bilgisayardaki programda kullanılarak 3D konum bilgisi elde edilmiştir. Raspberry modül ile bilgisayar arasındaki bağlantı TCP (ing. Transmission Control Protokol) protokolü ile sağlanmıştır. Nesne takibinde kullanılan OpenCV kütüphanesi ve HSV renk uzayı hakkında Bölüm 3.'de açıklama yapılmıştır. Yazılımın gerçekleştirilmesi ve alt başlıkları ise bu bölümde anlatılmıştır. Bu bölümün sonunda, bu yaklaşımın uygulanması ve deney sonuçları tartışılacaktır.

5.1. Raspberry Pi

Raspberry Pi Birleşik Krallık'ta Raspberry Pi Vakfı tarafından okullarda bilgisayar bilimini öğretmek amacıyla geliştirilmiş kredi kartı büyüklüğünde tek kartlı bir bilgisayardır [34]. Raspberry Pi 2 modeli¸ Şubat 2015'te çıkmıştır. Sonraki model Raspberry Pi Zero, Kasım 2015'te çıkmıştır. Son model olan Raspberry Pi 3 ise¸ Şubat 2016'da çıkmıştır.

Raspberry Pi, ilk modellerinde ARM1176JZF-S 700 MHz merkezi işlem birimini içeren Broadcom BCM2835 mikroçipi üzerine kurulmuştur. Daha sonra piyasaya çıkan Raspberry Pi 2 modelinde Broadcom BCM2836 kullanmıştır. VideoCore IV GPU grafik işlem birimine sahiptir. Booting ve veri depolaması için SD kart kullanır.

Üzerinde USB 2.0 portları, HDMI video çıkışı, ses çıkışı, MIPI kamera girişi, GPIO arayüzü ve 5V Micro-USB güç girişi bulunmaktadır.

Görsel 5.1. *Raspberry Pi*

Vakfın web sitesinden Raspbian (Debian Wheezy tabanlı), Pidora (Fedora tabanlı), Snappy Ubuntu Core veya desteklenen diğer işletim sistemleri indirilebilir. Sitesi dışındaki Pardus ARM, Arch Linux ARM ve Windows 10 IoT Core işletim sistemlerini de destekler. Python programlama dili ile programlanabildiği gibi BBC Basic, C ve Perl programlama dilleri de kullanılabilmektedir.

Görsel 5.2. *Raspberry Pi 3*

Raspberry Pi ile üzerine microSD karta Linux ya da Windows 10 kurularak normal bir bilgisayar ile yapılabilecek pek çok şey yapılabilirmektedir. Sunucu olarak

kullanmak, yazılım geliştirmek, akademik çalışmalarda kullanmak, eğitim amaçlı öğrenci bilgisayarları yerine kullanmak gibi çeşitli amaçlar için kullanılabilmektedir.

Raspberry Pi 3 Teknik Özellikleri

- 64-bit quad-core ARMV8 işlemci
- 1.2GHz
- 1GB RAM
- Dahili WiFi - BCM43143
- Bluetooth 4.1 (Bluetooth Low Energy - BLE)
- 40 Adet GPIO
- 4 Adet USB 2
- 4 uçlu Stereo çıkışı ve Composite video çıkışı
- Full HDMI
- Raspberry Pi Kamera bağlanstını için CSI kamera portu
- Raspberry Pi 7" dokunmatik ekran için DSI ekran portu
- Micro SD soketi
- Güncellenmiş güç katı (2,5A'e kadar destekliyor.)
- Güç ve aksiyon ledi.

Raspberry Pi, kredi kartı büyüklüğünde monitor ve klavye bağlayabileceğiniz mini bir bilgisayar kartıdır. ARM7 tabanlı bu mini bilgisayar, temel ofis uygulamaları ve oyunlar gibi normal bir bilgisayarda yapabileceğiniz çoğu işlemi yapmanıza olanak sağlamaktadır. Bununla birlikte yüksek çözünürlüklü(HD) video oynatabilme yeteneğine sahiptir.

5.1.1. Raspberry Pi 3 kurulumu

5.1.1.1. *Gereken donanımlar*

SD kart: Raspberry Pi, düzgün biçimlendirilmiş ve işletim sistemi içeren bir SD kart olmadan başlamamaktadır. Raspberry Pi ile birlikte kullanılabilecek farklı Linux sürümleri ve Microsoft Windows 10 sürümü mevcuttur. Raspberry Pi A ve B için normal SD kart, Raspberry Pi B+ ve Raspberry Pi 2 Model B (ikinci nesil) için en küçük, MicroSD kart gerekmektedir. Bu çalışmada Raspberry Pi 3 modeli için 8 GB SD kart kullanılmıştır.

Ekran ve bağlantı kabloları: HDMI/DVI monitör ya da ekran yerine kullanılabilecek bir televizyon, en iyi sonuç için kart üzerinde mevcut HDMI girişi ile elde edilmektedir.

Klavye ve fare: Uyumlu bir USB bağlantıya sahip klavye ve fare kullanılmaktadır.

Güç kaynağı: Standart bir kullanım için en az 5 Volt ya da daha yüksek akım sağlayan bir USB güç kaynağı kullanılmaktadır.

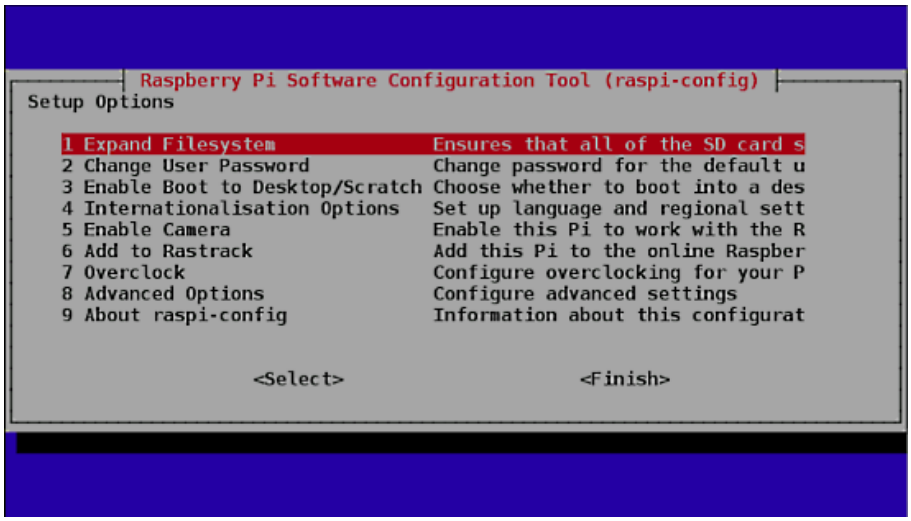
5.1.1.2. *İşletim sistemi kurulumu*

Raspberry Pi için düzenlenmiş NOOBS (New Out Of Box Software), Raspbian (Debian Wheezy tabanlı), Ubuntu Mate, Openelec, Pidora (Fedora tabanlı), Risck OS ve Windows 10 IOT gibi işletim sistemleri kullanılabilir. Öncelikle Raspberry Pi ile kullanılacak ilgili işletim sistemi https://www.raspberrypi.org/downloads/ sitesinden indirilmelidir. Bu çalışmada Raspbian işletim sistemi kullanılmıştır.

Raspbian kurulumu

Öncelikle verilen siteden Raspbian işletim sistemi indirilir. İndirilen dosya zip dosyasıdır. Bu dosyadan img uzantılı imaj dosyası çıkmaktadır. Bu dosyanın SD karta yazılması için Windows ortamında win32diskimager programı ya da konuyla ilgili mevcut başka bir program kullanılabilir.

Yükleme işlemi tamamlandıktan sonra hazırlanan SD kart Raspberry Pi'ye takılır ve açıldığında (Görsel 5.3.) yapılandırma menüsü (Raspi-config) ekrana gelmektedir. Kullanıcı adı pi ve raspberry şifresi ile giriş yapıldıktan sonra startx komutu ile grafiksel ortama geçilir.



Görsel 5.3. *Raspberry Pi Yapılandırma Menüsü*

Expand Filesystem: Raspbian kurulumu NOOBS kullanılarak yapıldıysa, hafıza kartının tümü otomatik olarak genişletilmektedir, kurulum bir işletim sisteminin img dosyasından yapıldıysa kartın tamamının kullanılması için izin verilmesi gerekmektedir.

Change User Password: İlk girişte var olan parolayı (raspberry) değiştirmek için kullanılır.

Enable Boot To Desktop or Scratch: Raspberry Pi'nin açılışında grafiksel ya da konsol karşılama ekran seçimi yapılır.

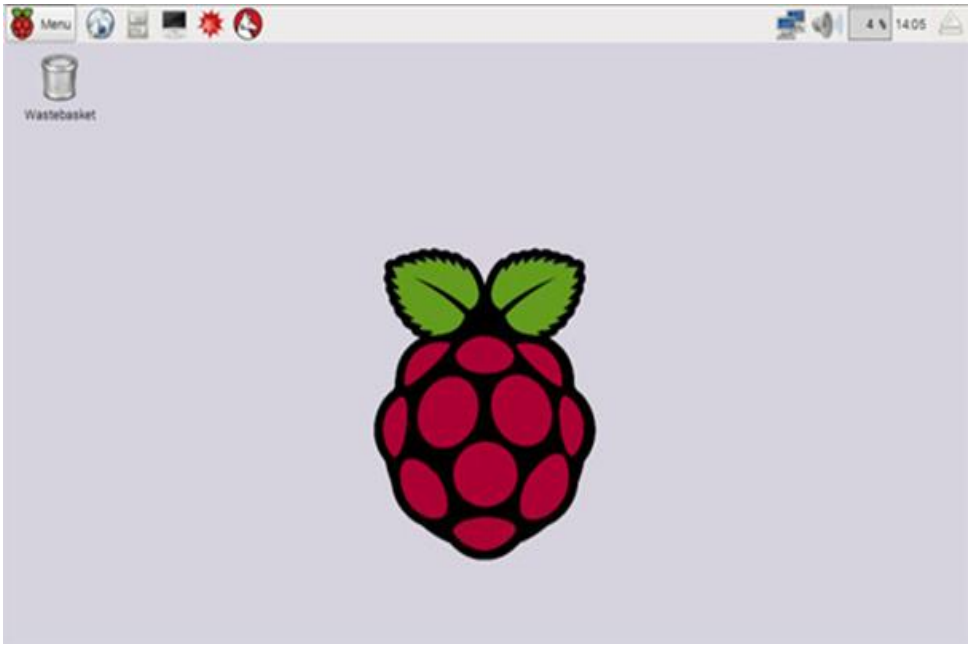
International Options: Dil, klavye, saat gibi yerel ayarlar yapılır.

Enable Camera: Eğer kamera kullanılacaksa etkinleştirilir. Bu durumda en az 128MB hafıza GPU atanır.

Add to Rastrack: 2012 yılında kurulan ve dünyada Raspberry Pi kullanıcılarının gösterildiği http://rastrack.co.uk/ haritasında yer almak için Rastrack'e kullanıcılar katılabilmektedir.

Overclock: Bu seçenekle overclock ayarlanarak işlemci normalden daha yüksek hızlarda çalıştırılabilmektedir.

Advanced Options: Bu seçenekle özellikle eski tip televizyonlardaki görüntü kayıplarını önlemek ayarlamalar, SSH bağlantısına izin verme, sisteme hostname ismi vermek, ses çıkışını HDMI ya da 3.5 mm jack'e yönlendirmek ve güncelleme gibi işlemler yapılır.



Görsel 5.4. *Raspberry Pi Masaüstü Görüntüsü*

Tüm işlemler bittikten sonra Raspberry Pi masaüstü açılmaktadır (Görsel 5.4.).

5.1.1.3. *OpenCV ve gerekli yazılımların yüklenmesi*

Raspberry Pi terminal açılarak gerekli yazılımlar yüklenmektedir. Yükleme işlemi için aşağıdaki adımlar takip edildi.

- Paketleri güncelleyip yükselttikten sonra, Raspberry Pi'yi güncellendi.

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo rpi-update
```

- Gerekli geliştirici araçları ve paketleri yüklendi.

```
$ sudo apt-get install build-essential cmake pkg-config
```

- Gerekli görüntü G/Ç paketleri yüklendi. Bu paketler, JPEG, PNG, TIFF, vb. gibi çeşitli resim dosyası formatlarını yüklemeye izin vermektedir.

```
$ sudo apt-get install libjpeg8-dev libtiff4-dev libjasper-dev
libpng12-dev
```

- GTK geliştirme kitaptığı yüklenmiştir. Bu kütüphane, Grafik kullanıcı arayüzlerini (GUI'ler) oluşturmak için kullanılır ve OpenCV'nin highgui kütüphanesi için ekranında görüntüleri görmeye sağlamaktadır.

```
$ sudo apt-get install libgtk2.0-dev
```

- Gerekli video G/Ç paketleri kuruldu. Bu paketler OpenCV kullanarak video dosyalarını yüklemek için kullanılmaktadır.

```
$ sudo apt-get install libavcodec-dev libavformat-dev
libswscale-dev libv4l-dev
```

- OpenCV içinde çeşitli işlemleri optimize etmek için kullanılan kitaplıkları yüklendi.

```
$ sudo apt-get install libatlas-base-dev gfortran
```

- C++ kodlarını derlemek için Qt Creator editörü indirildi.

```
$ sudo qt4-dev-tools libqt4-dev libqt4-core libqt4-gui
$ sudo apt-get install v4l-utils
$ sudo apt-get install gcc
$ sudo apt-get install xterm
$ sudo apt-get install git-core
$ sudo apt-get install subversion
$ sudo apt-get install qtcreator
```

- OpenCV indirildi ve paket açıldı.

```
$ sudo wget
http://sourceforge.net/projects/opencvlibrary/files/opencv-
```

```
unix/3.1.0/opencv-3.1.0.zip/download
$ sudo unzip opencv-3.1.0.zip
$ cd opencv-3.1.0
```

- OpenCV kurulumu ayarlandı.

```
$ sudo mkdir build
$ cd build
$ sudo cmake -D CMAKE_BUILD_TYPE=RELEASE -D
INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D
BUILD_EXAMPLES=ON -D WITH_QT=ON  -D
CMAKE_INSTALL_PREFIX=/usr/local -D WITH_OPENGL=ON -D
WITH_V4L=ON -D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_TBB=ON ..
```

- OpenCV derlendi.

```
$ sudo make
```

- Son olarak, OpenCV kuruldu.

```
$ sudo make install
$ sudo nano /etc/ld.so.conf.d/opencv.conf
/usr/local/lib (satırı eklendi ve kaydedildi. )
$ sudo ldconfig
$ sudo nano /etc/bash.bashrc
PKG_CONFIG PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig
export PKG_CONFIG_PATH
```

Yukarıdaki satırlar eklendi ve kaydedildi.

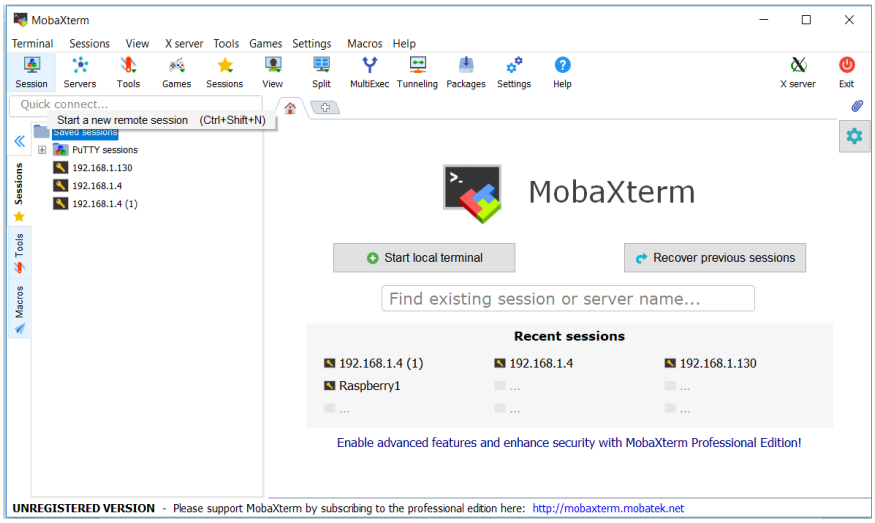
- Raspberry Pi kamera modülünü kullanmak için raspicam yüklendi.

```
$ git clone https://github.com/cedricve/raspicam
$ cd raspicam
$ mkdir build
$ cd build
$ cmake ..
$ make
$ sudo make install
$ sudo Idconfig
```

Yukarıda verilen adımlar gerçekleştirildikten sonra programlama ortamı sağlanmıştır.

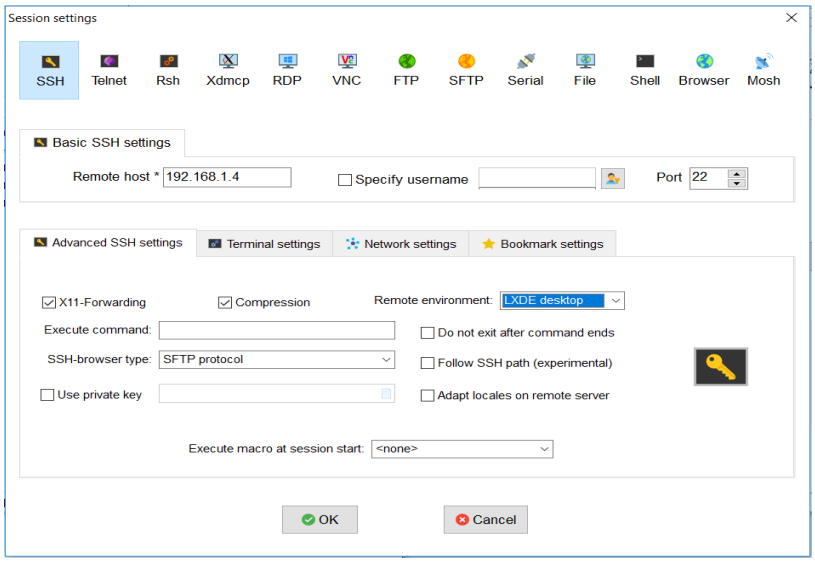
MobaXterm Kurulumu

MobaXterm ile uzak masaüstü bağlantısı yapılmaktadır. Windows ortamında Linux komutlarının çalıştırılmasına ortam sağlar [35]. MobaXterm yazılımı bilgisayar indirilip kuruldu. Görsel 5.5.'de MobaXterm programı görülmektedir.



Görsel 5.5. *MobaXterm Programı*

Session seçeneği ile yeni bir oturum açılır ve açılan pencerede yeni oturum için gerekli bilgiler yazılır. Bağlanılacak olan Raspberry Pi'nin IP adresi ve port numarası yazılır. Masaüstüne görsel olarak da bağlanmak isteniyorsa Remote Environment seçeneği LXDE Desktop olarak seçilir. İşlem tamamlandıktan sonra MobaXterm Raspberry Pi'nin kullanıcı adı ve şifre bilgilerini istemektedir. Bu bilgiler girildikten sonra uzak masaüstü bağlantısı tamamlanmaktadır.



Görsel 5.6. *MobaXterm Yeni Oturum Açma*

5.2. Kartezyen Koordinat Sistemi Kullanılarak Pozisyon Tahmini

Bu bölümde kartezyen koordinat sistemi kullanılarak aktif işaretçinin pozisyon tahmini için yapılan çalışma anlatılmıştır. Çalışma kapsamında, renk tabanlı algılama yaklaşımı kullanarak nesnenin yatay ve dikey düzlemdeki konumunun gerçek zamanlı tespiti ve takibini yapacak yazılım çalışması yapılmıştır.

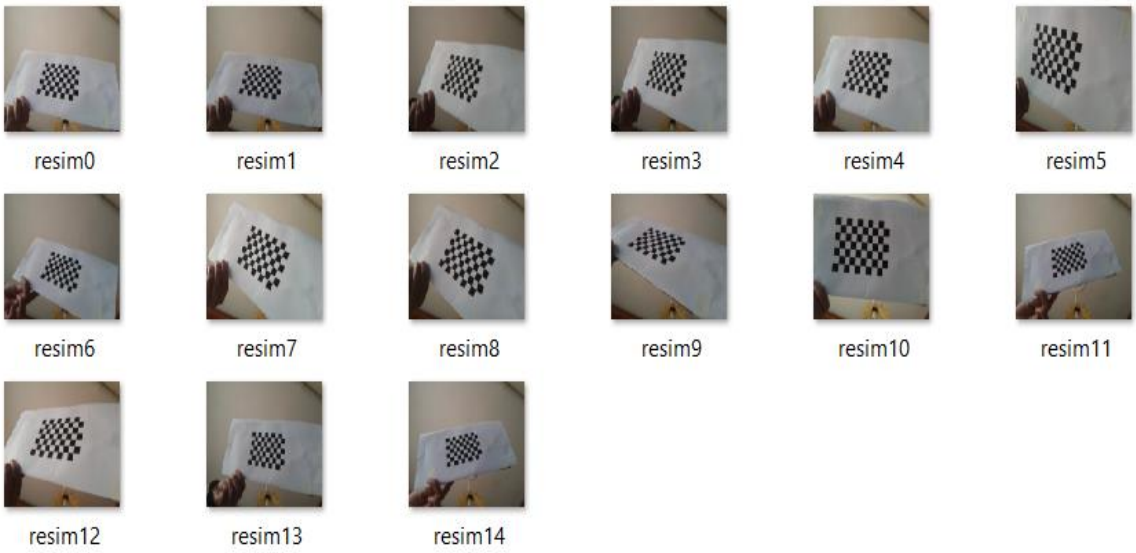
İlk olarak Raspberry Pi kameralardan alınan görüntüler ile kameraların kalibrasyonu yapılmıştır. Kalibrasyon sonucunda kameralara ait kamera parametreleri elde edilmiştir. Bu parametreler yardımıyla kameradan alınan görüntülerde düzeltme (*ing.* rectification) işlemi yapılmıştır.

Görsel 5.7. *Raspberry Pi Kamera Modülü*

Görsel 5.7.'de Raspberry Pi kameranın porta bağlanmış hali görülmektedir. Raspberry Pi kamera 5MP çözünürlüğe sahiptir ve üzerinde sabit odaklı bir lens bulunmaktadır. Yüksek çözünürlüklü bu video kamera Raspberry Pi üzerindeki CSI (Camera Serial Interface) portuna bağlanmaktadır. Kameranın üzerine monte edildiği kartın boyutları 25x20 mm, kamera dahil derinliği ise 9 mm'dir. Raspberry Pi kamera modülü kayıtlarında, 1080p30, 720p60 ve 640x480p (60/90 fps) çözünürlüğü desteklemektedir. Fotoğraflarda ise çözünürlük 2592x1944 olarak ayarlanabilmektedir. Bu kameranın en büyük avantajı FPS değerini dışarıdan alabiliyor olmasıdır. Piyasada bulunan web kameralar üzerinde FPS değişimi yapılamamaktadır.

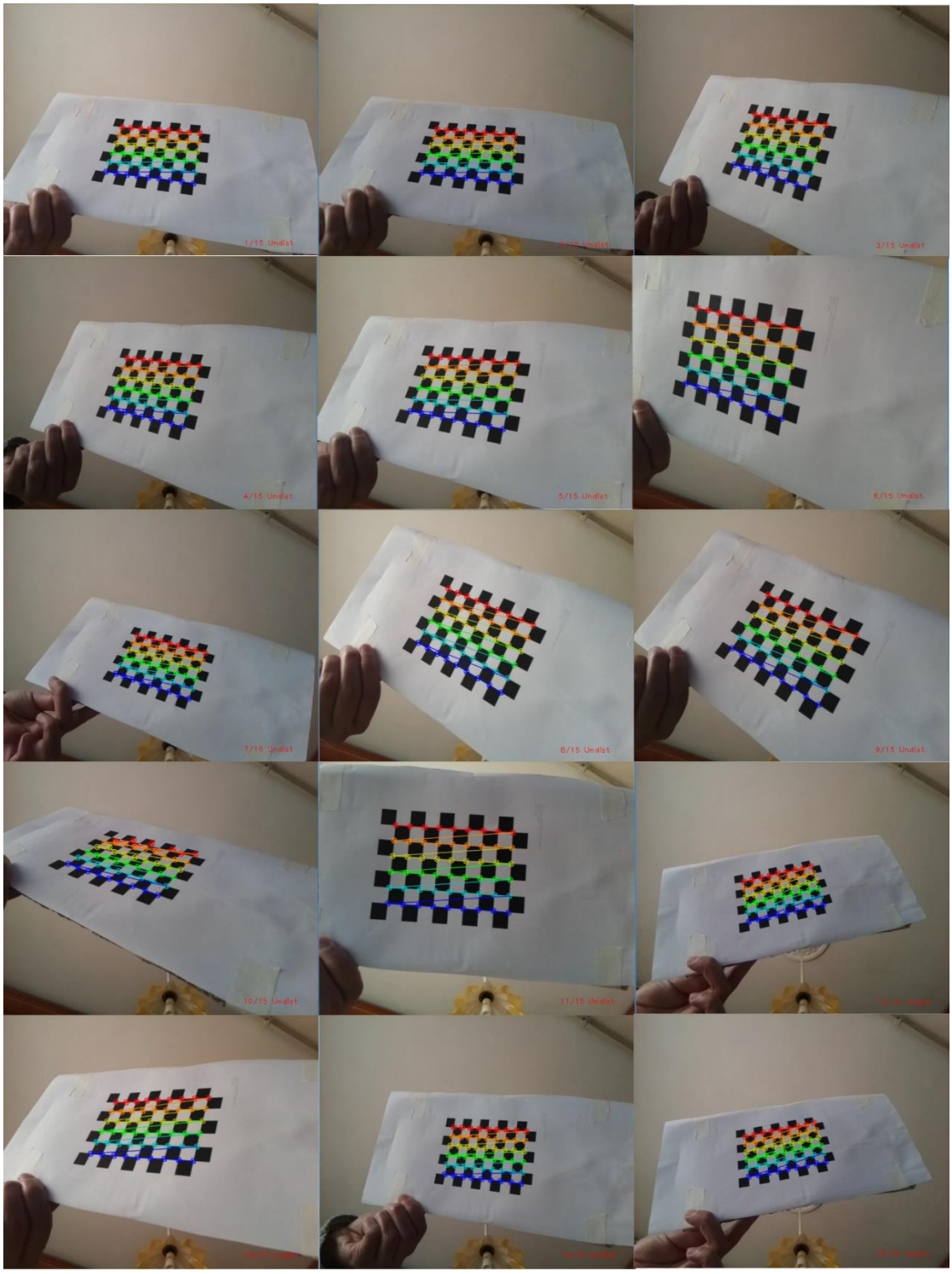
5.2.1. Raspberry Pi kamera kalibrasyonu

Sistemin kalibrasyonunun yapılabilmesi için, iki adet Raspberry Pi kamera modülü kullanılmıştır. Satranç tahtası olarak adlandırılan deseni içeren 15 fotoğraf kameradan farklı açı ve uzaklıklarda çekilmiştir. Bu işlem iki kamera için ayrı yapılmıştır. Kameraların bireysel olarak kalibrasyonları yapılmıştır. Kalibrasyon sonucunda kamera parametreleri ve bozulma katsayıları elde edilmiştir.



Görsel 5.8. *Raspberry Pi Kameradan Alınan Görüntüler*

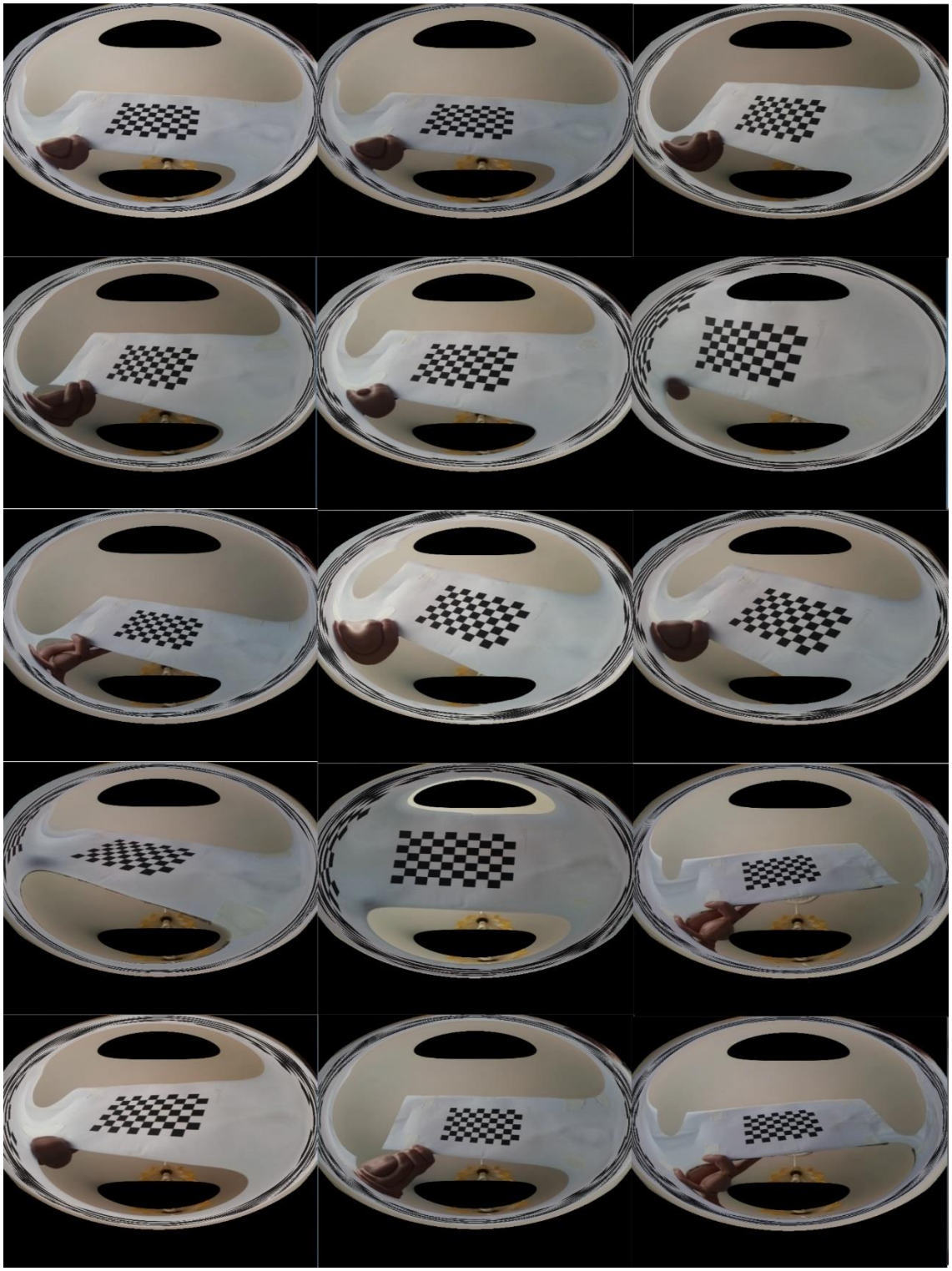
Gerçekte hiçbir lens mükemmel değildir ve görüntüde üretim sürecinden kaynaklanan bozulmalar meydana gelir. Pozisyon tahmini algoritmasının doğrulukla uygulanabilmesi için kameraların kalibre edilmesi gerekmektedir.



Görsel 5.9. *Raspberry Pi Kamera Kalibrasyonu*

5.2.2. Görüntülerin düzeltilmesi

Kalibrasyon işleminin ardından elde edilen kamera parametreleri ile görüntülerin rektifikasyonu yapılmaktadır.



Görsel 5.10. *Düzeltilmiş Raspberry Pi Kamera Görüntüleri*

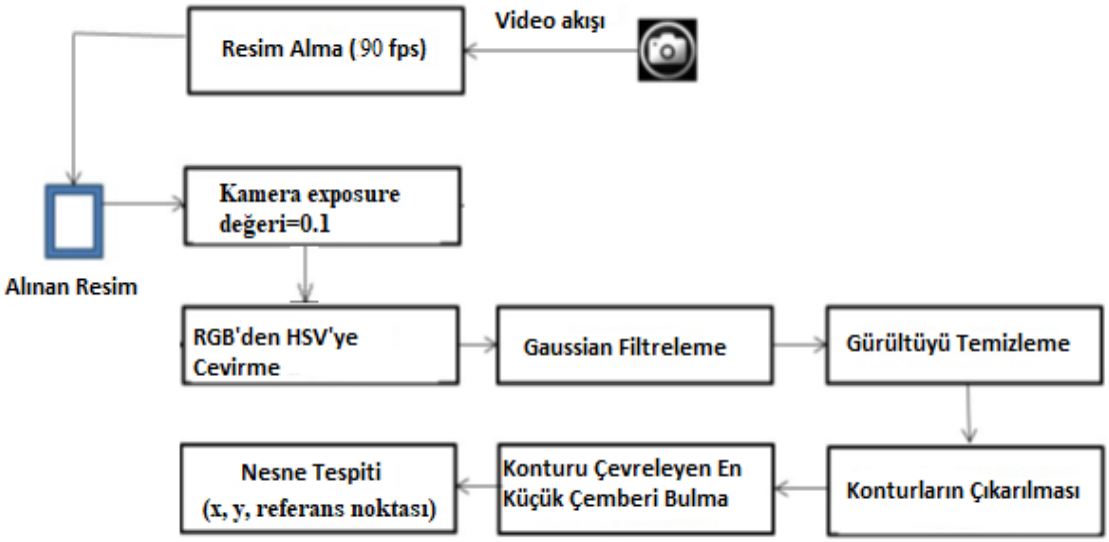
Alınan 15 görüntüden 15'sı başarılı bir şekilde tespit edilmiştir. Kalibrasyon sonucunda elde edilen parametreler kullanılarak kameralardan alınan görüntüler düzeltilir. Bu işlem sonrasında düzeltilmiş görüntüler Görsel 5.10.'de gösterildiği gibi olmaktadır.

5.2.3. Görüntü alımı

Görüntü alımı, iki adet Raspberry Pi kamera modülü ile 640x480 çözünürlüğünde yapılmıştır. Kameralardan görüntü alımı yaparken kameraların birbirine 90 derece açı olacak şekilde konumlandırılmıştır.

5.2.4. Nesne tespiti

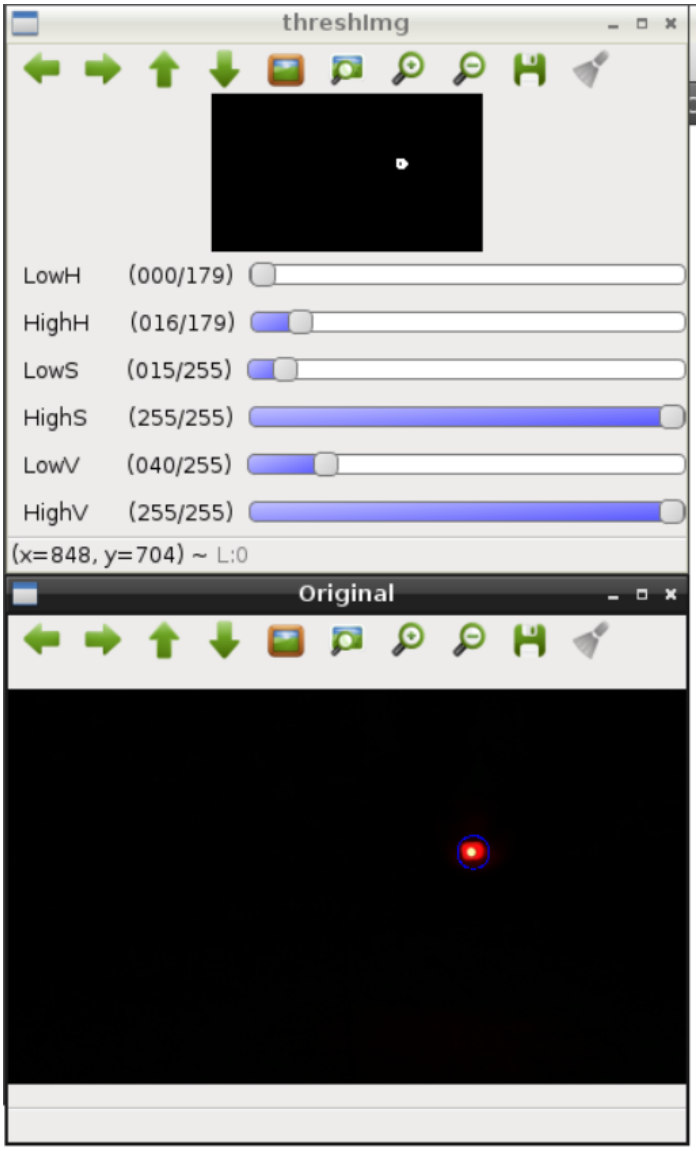
Belirli bir renkteki aktif işaretçinin tespit edilmesi için renge göre resim segmentasyonu yapılmıştır. Aktif işaretçinin rengi kırmızı olduğundan gerekli HSV renk uzayı tespit edildi ve bu renge sahip olan nesne belirlendi. Karanlık ortamda ucunda led olan bir işaretçi kullanılmıştır.



Şekil 5.1. *Raspberry Pi Kamera ile Nesne Tespiti Akış Şeması*

Şekil 5.1.'de alınan görüntü üzerinde yapılan işlemler görülmektedir. Raspberry Pi kameradan alınan görüntüler exposure değeri 0,1 olarak ayarlanarak ortam karartılmıştır. Karartılan resimler HSV renk uzayına dönüştürülür, gaussian filtresi ile görüntü netleştirilir, gürültüler temizlenir, HSV renk uzayında tanımlanan aktif marker kontur ile belirlenir, konturu çevreleyen en küçük çember bulunur ve sonuç olarak işaretçinin piksel koordinatları ve yarıçap bilgisi elde edilir.

Raspberry Pi modülleri üzerinde çalıştırılan programlarda nesnenin HSV renk uzayındaki değerleri ile nesnenin resimdeki piksel koordinatları elde edilmiştir. Görsel 5.11.'de HSV renk uzayı ve nesnenin tespit edilmesine örnek verilmiştir.



Görsel 5.11. *Raspberry Pi Kamera ile Nesne Tespiti*

Alınan görüntülere filtrelemeler uygulanarak sağlam tespit yapılması sağlandı. Ortam karartıldığından işaretçi tespiti ortam şartlarından etkilenmemektedir. Bu nedenle daha doğru sonuçlar ile bulunmaktadır.

Önceki bölümlerde pozisyon çıkarımı için sarı renkli bir nesne kullanılmıştı. Geliştirilen sistemlerde en büyük dezavantaj pozisyon çıkarımı aşamasında kullanılan algoritmanın ortam şartlarına bağlı olarak çok çabuk bozulmaya uğramasıydı. Ortamın ışık kaynağının ışık şiddeti, arka plan, ortamdaki nesnenin hareket hızı ve bunlara bağlı olarak nesnenin yanlış tespit edilmesi ölçülen değerlerle ilgili farklı sonuçların çıkmasına sebep olmaktaydı. Bu sorun Raspberry Pi kamera ile çözülmüştür. Kameralar

ilk açıldığında normal şartlar ile satranç tahtası üzerindeki referans noktalarını tespit ettikten sonra, görüntü karartıldı. Exposure değerini değiştirerek ortamı karatma işlemi aşağıdaki kod ile yapılmıştır. Bu sayede aktif işaretçi tespit edilirken ortam şartlarından etkilenmesi olumsuzluğu ortadan kaldırıldı.

```
Camera.set(CV_CAP_PROP_EXPOSURE, 0.1);
```

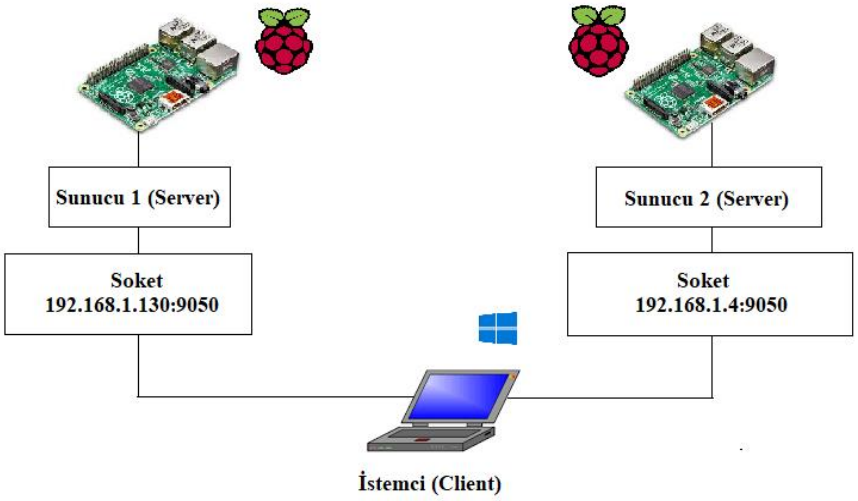
Raspberry Pi kamera modülünün avantajlarından biri de kameranın FPS değerinin ayarlanabiliyor olmasıdır. FPS değeri arttıkça sistemin gerçekliği de artmaktadır. Bu nedenle simülasyonlarda büyük önem taşımaktadır. Bu işlem aşağıdaki kod ile yapılmıştır.

```
Camera.Set(CV_CAP_PROP_FPS, 90);
```

Kameralardan alınan referans noktaları ve aktif işaretçinin piksel koordinatları istemciye ağ üzerinden iletilmektedir.

5.2.5. Raspberry Pi ile bilgisayar bağlantısı

Sistemde iki Raspberry Pi modülü sunucu, bilgisayar ise istemci olarak tanımlanmaktadır. Raspberry modüller ile elde edilen veriler soket ile bağlanılmış olan istemciye gönderilmektedir. Sunucular istemciyi port üzerinden dinler, istemci ise sunuculara port ve IP bilgisi ile bağlanmaktadır. İstemci gerçek zamanlı ulaştığı verileri kullanarak pozisyon çıkarımı yapmaktadır. Sunucular üzerinde Raspbian işletim sistemi kurulu olduğundan, ona uyumlu olan WringPi kütüphanesi ile soket programlama yapılmıştır. İstemci bölümünde ise Windows işletim sistemi kullanıldığından Windows soket kütüphanesi kullanılmıştır. Şekil 5.2.'de istemci sunucu yapısı gösterilmiştir.



Şekil 5.2. *İstemci-Sunucu Yapısı*

İstemci-Sunucu Mimarisi

İstemci-sunucu mimarisi, bilgisayar ağı ile birbirine bağlı iki yazılımın biri birinden hizmet almasını amaçlamaktadır. Bunlardan sunucu yazılımı hizmet verirken, istemci yazılım ise hizmet alan taraftır. Sunucunun verdiği hizmet, dosya aktarımı, e-posta göndermek gibi servisler olabilir. İstemcinin bu hizmeti alabilmesi için önce sunucu ile bağlantı kurması ve daha sonra bağlantıyı kurarak servise erişmesi gerekir. Soketler ise bu bağlantının kurulması ve servise ulaşılması için gerekli olan yazılım alt yapısını sunmaktadır. Soket, işletim sistemlerince desteklenen, farklı programlama dillerinden uzaktan ulaşılabilen bir yazılım erişim noktasıdır. Soket programlama modeli istemci-sunucu mimarisi ile uyumludur.

Sunucu tarafında soket haberleşmenin adımları:

1. Servisi duyur,
2. Bağlantı için bekle,
3. Bağlanan istemci için servis yap,
4. İstemci ile bağlantıyı kopar,
5. İkinci adıma geri dön.

İstemci tarafta soket haberleşmenin adımları:

1. Servisin sunucusunu bul,
2. Sunucu ile bağlantı kur,
3. Servis için istek gönder,
4. Sunucu ile olan bağlantıyı kopar.

İstemciler ve sunucular iletim protokolleri üzerinden (*örn.* TCP veya UDP) mesajları karşılıklı değişirler. İstemci ve sunucunun ikisi de aynı protokol yığınına sahip olmalıdır ve ikisi de iletim katmanıyla etkileşmelidir.

TCP/IP

Kullanım olarak iki katmanlı bir haberleşme protokolüdür. Üst katman TCP (Transfer Control Protokol) verinin iletimden önce paketlere ayrılmasını ve alıcıda bu paketlerin yeniden düzgün bir şekilde birleştirilmesini sağlar. Alt katman IP (Internet Protokol) ise, iletilen paketlerin istenilen ağ adresine yönlendirilmesini kontrol eder. TCP/IP protokol yapısı:

Uygulama Katmanı (ing. Application Layer) : Farklı sunucular üzerindeki süreç ve uygulamalar arasında iletişimi sağlar.

Taşıma Katmanı (*ing.* Transport Layer) : Noktadan noktaya veri akışını sağlar.

İnternet Katmanı: Routerlar ile birbirine bağlanmış ağlar boyunca verinin kaynakta hedefe yönlendirilmesini sağlar.

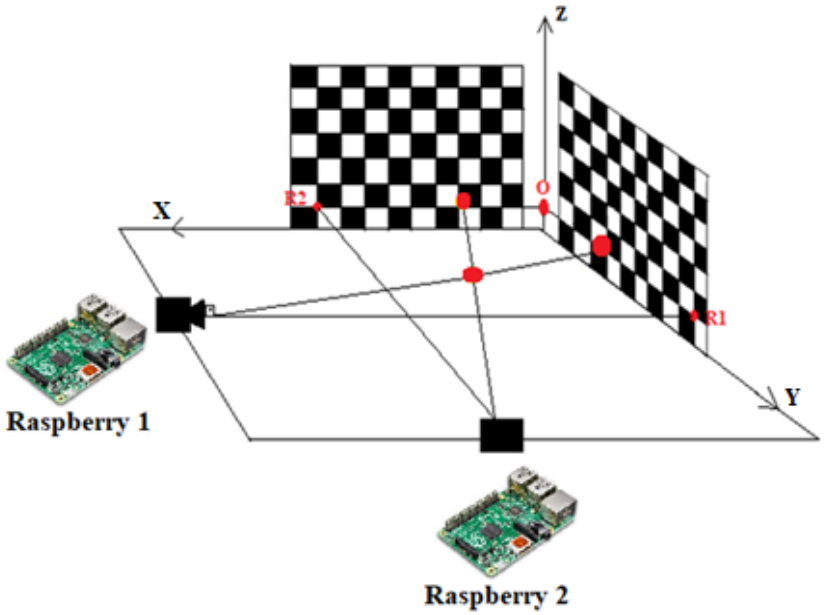
Ağ Erişim Katmanı: Uç sistem ile alt ağ arasındaki lojik arabirimine ilişkin katmandır.

Fiziksel Katman: İletişim ortamının karakteristik özelliklerini, sinyalleşme hızını ve kodlama şemasını belirler.

5.2.6. Raspberry Pi modül ile pozisyon çıkarımı

Kartezyen koordinat sistemi tasarımı Şekil 5.3.'da gösterilmiştir. Bu tasarıma göre Raspberry Pi kameralar kullanılan 9x6 boyutunda iki satranç tahtası modelinin referans noktalarına dik olarak konumlandırılmaktadır. Referans noktaları Şekil 5.3.'da R1 ve R2 ile gösterilmiştir. O noktası orijin (0,0,0) noktasını ifade etmektedir.

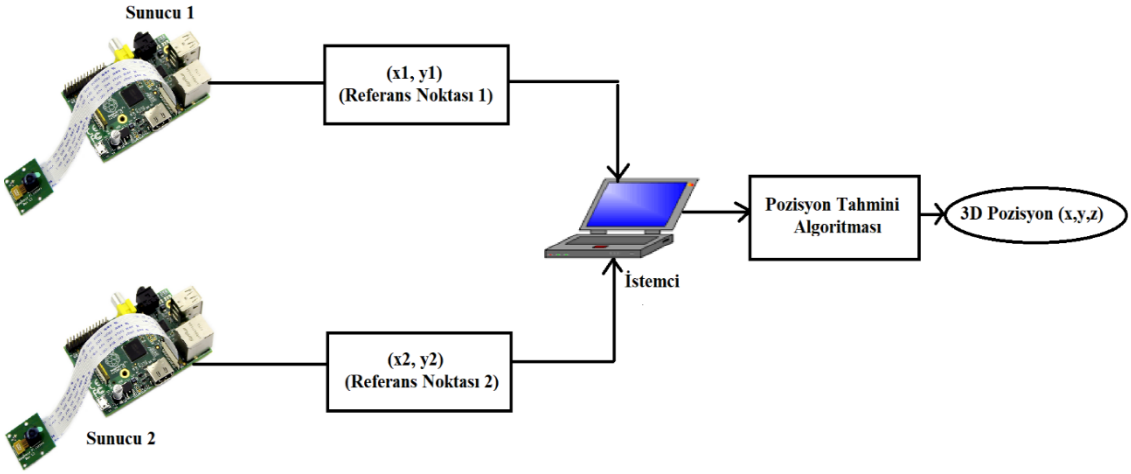
Kullanılan aktif işaretçinin projeksiyon ekranlarındaki izdüşüm değerlerine göre pozisyon tahmini yapılmaktadır. İşaretçinin kamera 1 ekranındaki izdüşümünün R1 noktasına yataydaki uzaklığı ile kamera 2 ekranındaki izdüşümünün R2 noktasına yataydaki uzaklığı elde edilmiştir. Elde edilen bu değerler kullanılarak iterasyon yöntemi ile gerçek x, y değerleri elde edilmiştir. Nesnenin bu referans noktalarına olan dikey uzaklıklarının ortalaması z değerini vermektedir.



Şekil 5.3. *Raspberry Pi ile Kartezyen Koordinat Sistemi Tasarımı*

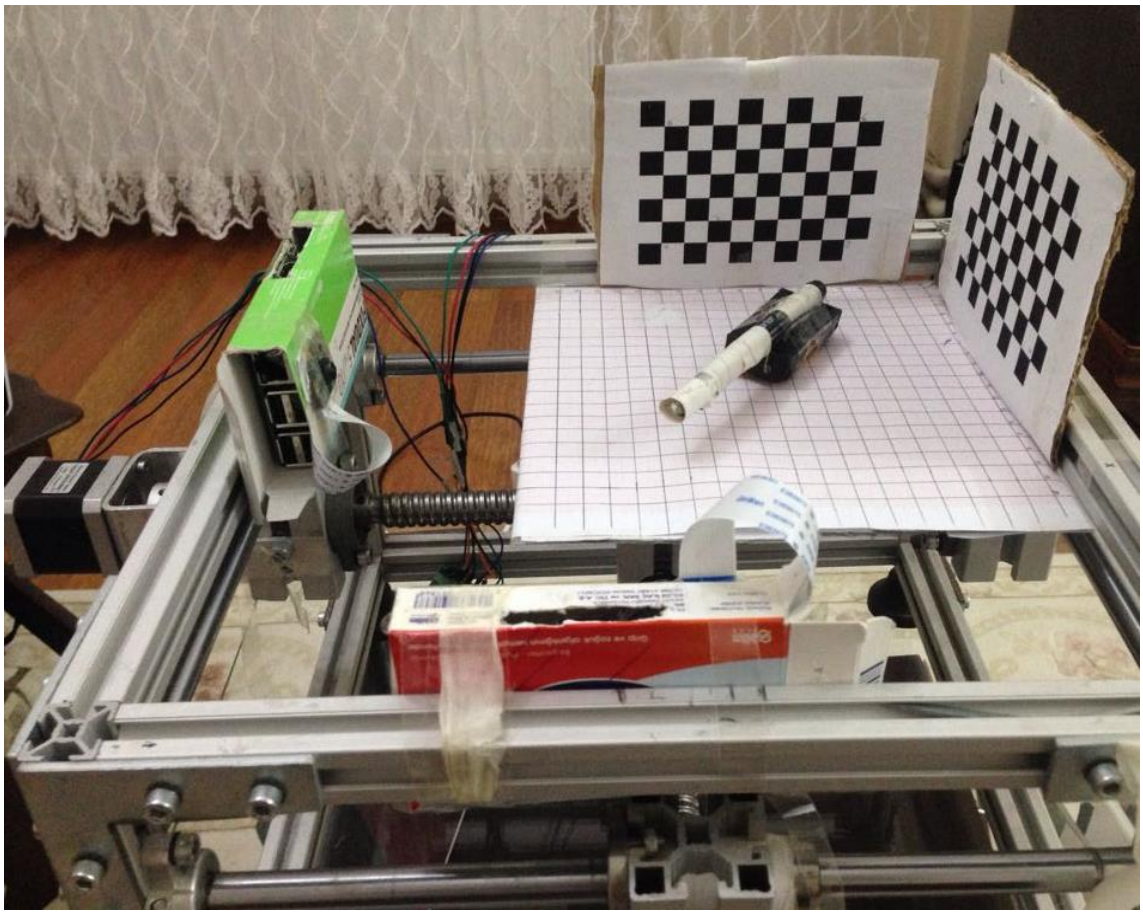
Birinci Pi kameradan alınan görüntüde nesnenin O noktasına olan uzaklığı nesnenin y değerini elde ederken kullanılmıştır. Aynı şekilde ikinci Pi kameradan alınan görüntüde nesnenin O noktasına olan uzaklığı nesnenin x değerini elde ederken kullanılmıştır. Her iterasyonda orijine olan uzaklık değerleri değiştiğinden yeni x ve y değerleri elde edilmektedir. Elde edilen yeni x değeri ile bir önceki arasındaki değişim 0.1 mm'nin altına düştüğünde iterasyona son verilmektedir.

Her iterasyonda bir değerin elde edilmesinde üçgen benzerliği kullanılmaktadır. Örneğin Pi Kamera 1'den alınan görüntüde nesnenin izdüşüm değeri, R1 ve Pi Kamera 1 bir üçgen varsayılmaktadır. Pi Kamera 2'den alınan orijine uzaklık değeri ile Pi Kamera 1'in R1 noktasına olan uzaklığı oranı, y test değeri olarak bilinmeyen değer ile nesnenin Pi Kamera 1'deki izdüşümün R1'e olan uzaklığı oranı eşitlenerek y test değeri elde edilmektedir. Bu yeni elde edilen değer ile bir önceki değer arasındaki fark 0.1'den küçük olana kadar devam etmektedir. Elde edilen son değer nesne izdüşümü O-R1 noktası arasında ise R1'e çıkarılır aksi durumda eklenerek bulunmaktadır.



Şekil 5.4. *Raspberry Pi ile Pozisyon Tahmini Algoritması*

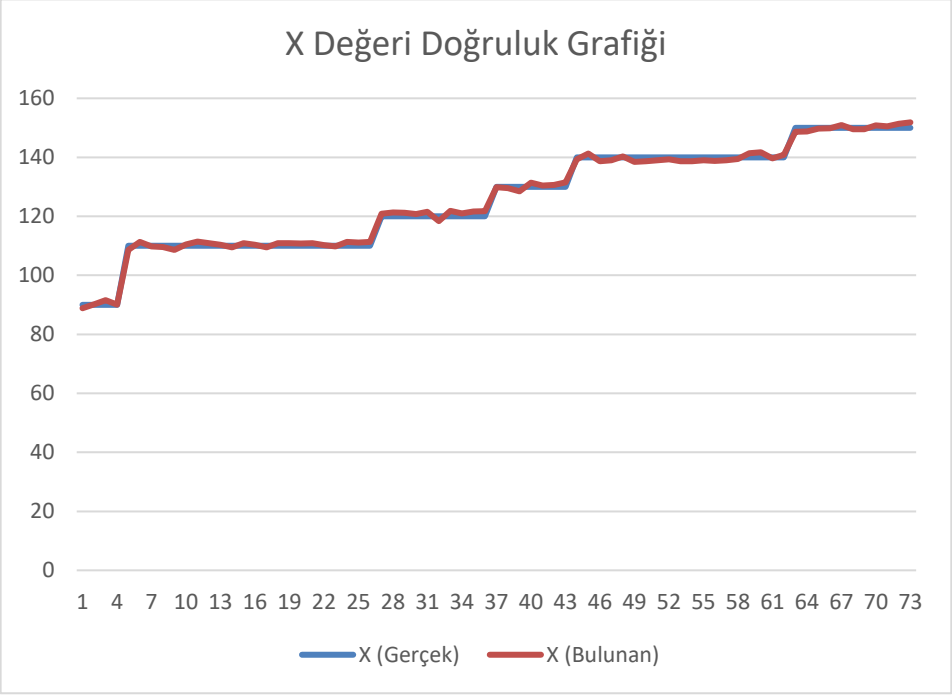
Şekil 5.4.'de pozisyon tahmini algoritmasının yapısı verilmiştir. Algoritma girdi olarak referans noktaları, nesnenin iki kamerada oluşan izdüşüm değerleri ve kamera parametrelerini almaktadır. Pozisyon tahmini algoritmasına verilen bu değerler ile nesnenin 3D koordinatları bulunmaktadır.



Görsel 5.12. *Raspberry Pi ile Kartezyen Koordinat Sistemi*

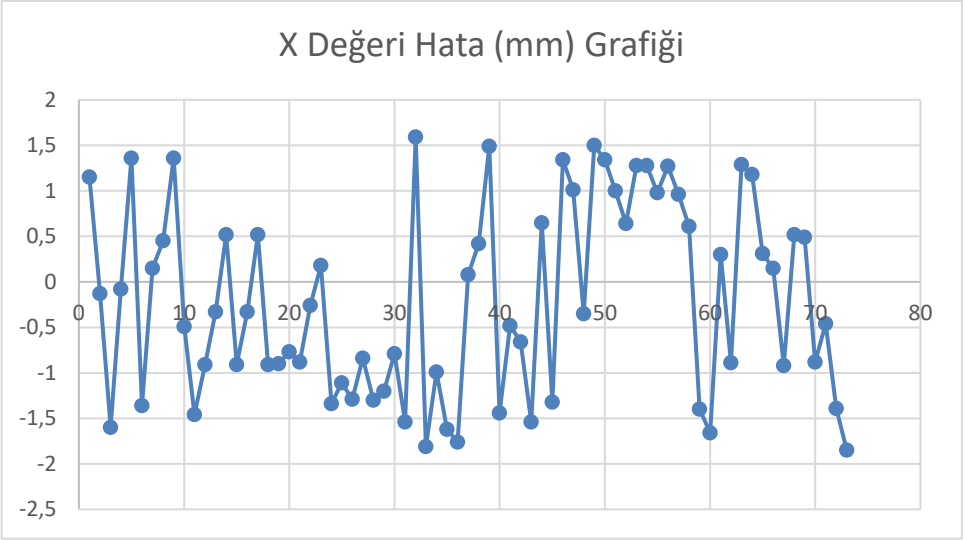
Referans noktası orijin (0,0,0) olarak varsayılır. Pi Kamera 1 ve Pi Kamera 2'den alınan eş zamanlı görüntülerde nesnenin konumu referans noktasına göre belirlenmektedir. Nesnenin x eksenindeki konumu Pi Kamera 2'den alınan görüntüde nesnenin referans noktasına olan uzaklığı ile ifade edilmektedir. Nesnenin y eksenindeki konumu Pi Kamera 1'den alınan görüntüde nesnenin referans noktasına olan uzaklığı ile ifade edilir. Aynı şekilde nesnenin z eksenindeki konumu iki kameradan alınan görüntülerde nesnenin referans noktasına dikey olan uzaklıklarının ortalaması ile ifade edilmektedir.

Bu çalışmayı test etmek için 3D pozisyonları bilinen örnekler alındı ve bu pozisyonlara karşılık gelen değerler bulundu. Bulunan bu değerler ile gerçek değerler karşılaştırıldı. Sonuç olarak X, Y ve Z değeri doğruluk grafikleri Şekil 5.5, Şekil 5.7. ve Şekil 5.9.'te gösterilmiştir. X, Y ve Z hata grafikleri Şekil 5.6., Şekil 5.8. ve Şekil 5.10.'te gösterilmiştir.



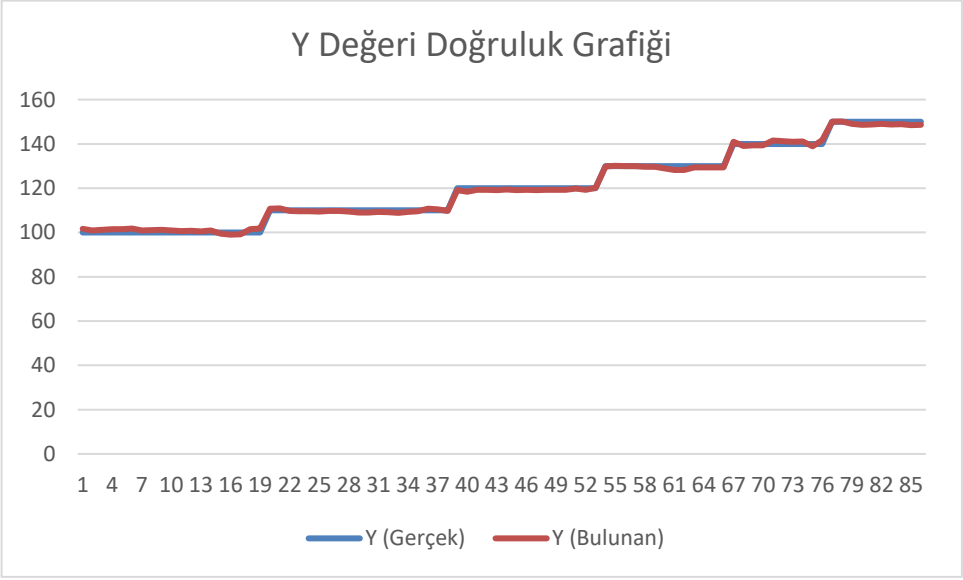
Şekil 5.5. *X Değeri Doğruluk Grafiği*

Bulunan X değerleri gerçek değerlere çok yakın değerlerde bulunmuştur. Alınan örneklerde X değeri %0,76 hata oranlarında tespit edilmiştir.



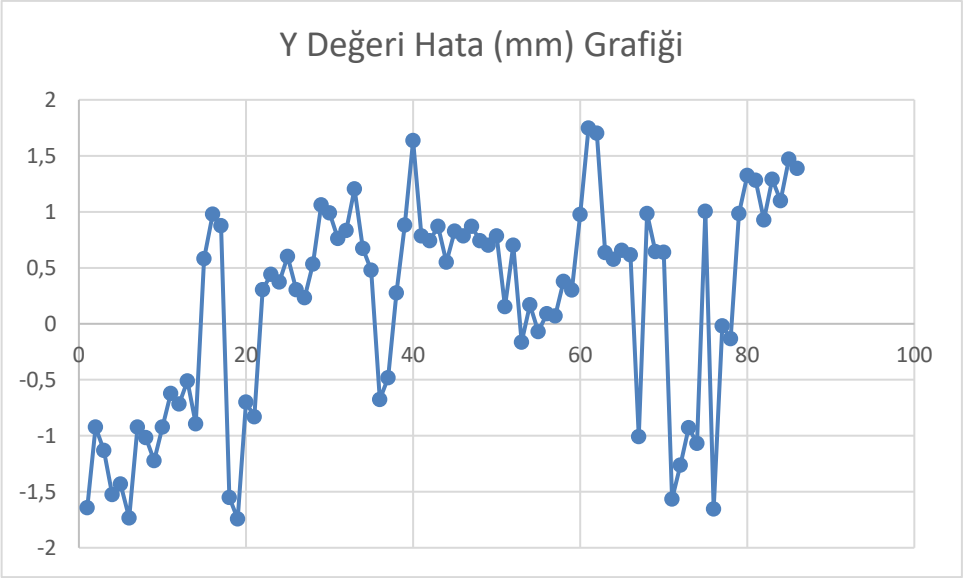
Şekil 5.6. *X Değeri Hata (mm) Grafiği*

Şekil 5.6.'de X değerinin hataları mm olarak ifade edilmiştir. Alınan örneklerde X değeri ortalama 0,95 mm'lik hata ile tespit edilmiştir.



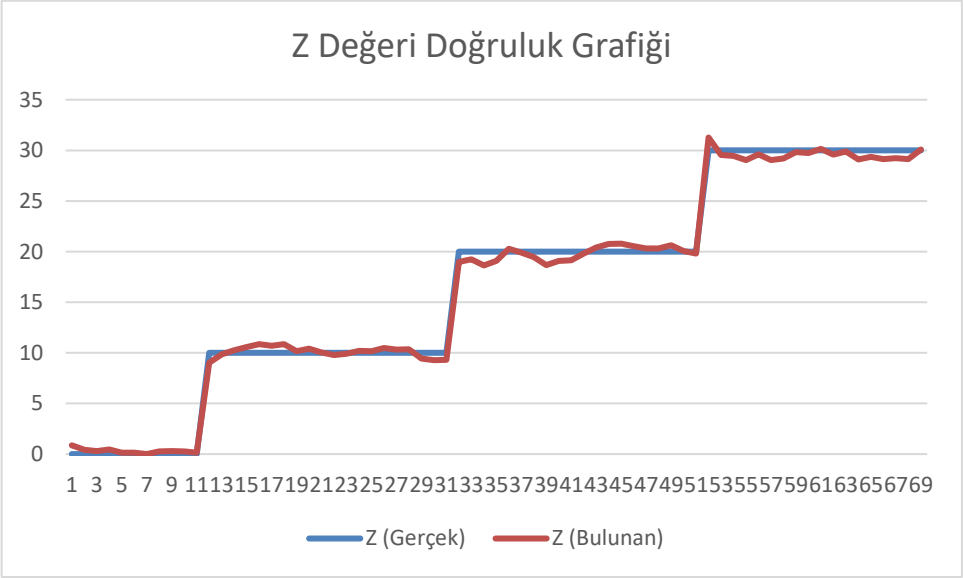
Şekil 5.7. *Y Değeri Doğruluk Grafiği*

Alınan örneklerde Y değeri %0,71 hata oranlarında tespit edilmiştir. Şekil 5.7.'te görülen sapmalar mm'lik değerlerdir.



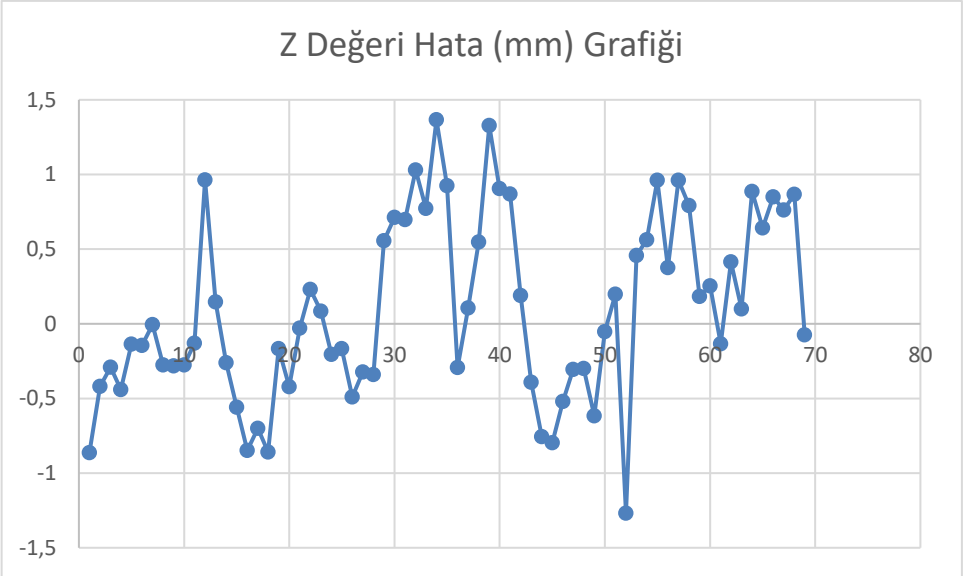
Şekil 5.8. *Y Değeri Hata (mm) Grafiği*

Şekil 5.8.'de Y değerinin hataları mm olarak ifade edilmiştir. Alınan örneklerde Y değeri ortalama 0,843 mm'lik hata ile tespit edilmiştir.



Şekil 5.9. *Z Değeri Doğruluk Grafiği*

Alınan örneklerde Z değeri %2,65 hata oranlarında tespit edilmiştir. Tüm koordinatlar ortalama %1,37 hata oranlarında bulunmuştur.



Şekil 5.10. *Z Değeri Hata (mm) Grafiği*

Şekil 5.10.'de Z değerinin hataları mm olarak ifade edilmiştir. Alınan örneklerde Z değeri ortalama 0,5 mm'lik hata ile tespit edilmiştir.

Tablo 5.1. *Kartezyen Koordinat Sistemi Test İşlemleri Sonucunda Elde Edilen Değerler*

X(Gerçek)	X(Bulunan)	Y(Gerçek)	Y(Bulunan)	Z(Gerçek)	Z(Bulunan)
90,0	90,13	100,0	100,92	0	0,86
110,0	109,85	100,0	101,13	0	0,42
110,0	109,55	100,0	101,52	0	0,44
110,0	110,49	100,0	101,73	0	0,27
110,0	110,91	110,0	110,83	0	0,29
110,0	111,11	110,0	109,69	10,0	9,03
120,0	120,84	110,0	109,55	10,0	9,85
120,0	121,20	110,0	108,93	10,0	10,26
120,0	120,79	120,0	119,25	10,0	10,55
120,0	120,99	120,0	119,13	10,0	10,84
130,0	129,92	120,0	119,29	20,0	18,97
130,0	130,48	120,0	120,16	20,0	19,22
130,0	128,51	130,0	129,91	20,0	18,63
130,0	129,58	130,0	129,62	20,0	19,81
140,0	139,35	130,0	129,02	20,0	20,75
140,0	138,99	140,0	141,26	20,0	20,79
140,0	140,35	140,0	140,92	30,0	29,81
140,0	139,00	140,0	138,99	30,0	30,13
150,0	149,69	150,0	150,13	30,0	29,11
150,0	149,85	150,0	149,01	30,0	29,35
150,0	150,92	150,0	148,67	30,0	30,07
150,0	149,51	150,0	149,07	30,0	29,13

Tablo 5.1.'de mm cinsinden alınan bazı örneklerin bulunan değerleri ve gerçekte olan değerleri sunulmuştur. Çok küçük hatalarla tespitler gerçekleştirilmiştir.

Alınan örneklerde X değeri ortalama 0,95 mm, Y değeri ortalama 0,84 mm ve Z değeri ortalama 0,50 mm hata ile tespit edilmiştir. Uygulama ortalama saniyede 34 ms ile çalışmaktadır.

6. SONUÇ VE ÖNERİLER

Bu çalışmada pozisyon çıkarımı algoritmaları ve görüntü işleme yöntemleri kullanılarak ortamdaki nesnenin tespiti yapılmıştır. Gerçekleştirilen sistemde tek iğne deliği kamera, çift kamera ve Raspberry pi kamera modülleri kullanılmış, bu kameranın kalibrasyonu gerçekleştirilmiş ve görüntülerin düzeltilmesi ile kamera lenslerindeki bozulmalar düzeltilmiştir. Bu tezde, yazılım kısmı için Windows© 10 işletim sistemli, İntel® Core™ i7-4700HQ CPU @2.40GHz işlemciye sahip, 8,00 GB RAM ve 64 bit sisteme sahip bilgisayar altında çalışılmış olup, OpenCV 3.1. kütüphanesinden yararlanılmıştır.

Pozisyon çıkarımı için farklı yöntemler kullanılmıştır. Bölüm 3.'de tek kamera kullanarak farklı yöntemler ile pozisyon tespiti yapılmıştır. İlk yöntemde tek kamera kullanarak doku kaplı nesnenin pozisyon çıkarımı yapılmış, ortalama 8-9 FPS ile eksenlerin net elde edilememesi ile sonuçlanmıştır. Bu işlem simülasyon için yeterli bulunmamıştır. Diğer bir yöntem olarak küresel koordinat sistemi kullanılmıştır. Bu sistemde tek kamera kullanılarak renkli bir nesnenin pozisyon çıkarımı yapılmıştır. Bu yöntemde işaretleyiciler kullanılmıştır. İşaretleyiciler ile referans noktaları elde edildikten sonra küresel koordinatlarda pozisyon tahmini yapılmıştır. Aruco modülünün kullanım açısından satranç tahtası işaretleyicisinden daha kompleks olduğundan kullanılmamıştır. Kullanılacak olan işaretleyici satranç tahtası olarak seçilmiştir. Tablo 3.1.'de verildiği gibi ölçüm sonuçları bulunmuş ve belirtilen hata oranlarıyla sistem gerçekleştirilmiştir. Uygulama ortalama 15-16 FPS ile çalışmaktadır. Uygulamanın gerçekliğini arttırmak için FPS değerinin gerçeğe yakın olması amaçlanmıştır. Bu nedenle 15-16 FPS yeterli bulunmamıştır. Geliştirilen sistemin en büyük dezavantajı derinlik bilgisi elde edilmesi aşamasında kullanılan algoritmanın ortam şartlarına bağlı olarak çok çabuk bozulmaya uğramasıdır. Ortamın ışık kaynağının ışık şiddeti, arka plan, ortamdaki nesnenin hareket hızı ve bunlara bağlı olarak derinlik bilgisinin yanlış tespit edilmesi ölçülen değerlerle ilgili farklı sonuçların çıkmasına sebep olmaktadır.

Bölüm 4.'de pozisyon çıkarımı algoritmaları ve görüntü işleme yöntemleri kullanılarak ortamdaki nesnenin tespiti yapılmıştır. Gerçekleştirilen sistemde iki kamera kullanılmış, bu kameraların kalibrasyonu gerçekleştirilmiştir. Pozisyon çıkarımı için farklı yöntemler kullanılmıştır. İlk olarak stereo görme prensibi ele alınmıştır. Bu yöntemde web kameralardan alınan görüntüler ile kameraların kalibrasyonu yapılmıştır.

Kalibrasyon sonucunda kameralara ait iç ve dış parametreler elde edilmiştir. Bu parametrelerin yardımıyla iki kameradan eş zamanlı alınan görüntüler düzeltme (*ing.* rectification) işlemi yapıldıktan sonra işlenerek, ortamın eşitsilik(*ing.* disparity) haritası çıkarılmıştır. Bu haritadaki nesnenin kameralara bağlı konumu üçgen benzerliği kullanılarak tespit edilmiştir. Bu algoritma kamera parametreleri, nesnenin iki kameradaki izdüşümleri ve odak noktaları arasındaki uzaklığı (*ing.* baseline) girdi olarak almaktadır. Daha sonrasında nesneye olan mesafesi ile alakalı yazılımsal olarak elde edilen sonuçlar, gerçek dünyadaki mesafe bilgisi ile karşılaştırılarak hata analizi yapılmıştır. Şekil 4.3.'de elde edilen sonuçlar verilmiştir. Bu değerlere göre nesnenin kameraya olan uzaklığı ortalama %2,465 hata ile bulunmuştur. Tablo 4.1.'de verildiği gibi ölçüm sonuçları bulunmuş ve belirtilen hata oranlarıyla sistem gerçekleştirilmiştir. Uygulama ortalama 11-12 FPS ile çalışmaktadır. Uygulamanın gerçekliğini arttırmak için FPS değerinin gerçeğe yakın olması amaçlanmıştır. Bu nedenle 11-12 FPS yeterli bulunmamıştır. Bu çalışmanın dezavantajı nesne tespiti aşamasında kullanılan algoritmanın ortam şartlarına bağlı olarak çabuk bozulmaya uğramasıdır.

Bölüm 4.'de iki kamera kullanılarak geliştirilen diğer yöntem ise kartezyen koordinat sistemidir. Bu sistemde kullanılan algoritma girdi olarak referans noktaları, nesnenin iki kamerada oluşan izdüşüm değerleri ve kamera parametrelerini almaktadır. Pozisyon tahmini algoritmasına verilen bu değerler ile nesnenin 3D koordinatları bulunmaktadır. Kullanılan sarı renkteki 3D nesnenin projeksiyon ekranlarındaki izdüşüm değerlerine göre pozisyon tahmini yapılmaktadır. Nesnenin birinci kamera ekranındaki izdüşümünün birinci referans noktasına yataydaki uzaklığı ile ikinci kamera ekranındaki izdüşümünün ikinci referans noktasına yataydaki uzaklığı elde edilmiştir. Elde edilen bu değerler kullanılarak iterasyon yöntemi ile gerçek x, y değerleri elde edilmiştir. Nesnenin bu referans noktalarına olan dikey uzaklıklarının ortalaması z değerini vermektedir. Bu değerleri test etmek için alınan örnekler üzerinde değerlendirmeler yapılmıştır. Bu değerlendirmelere göre X değeri %0,709 hata oranlarında ve ortalama 0,72 mm'lik hata ile tespit edilmiştir. Y değeri %1,264 hata oranlarında ve ortalama 1,56 mm'lik hata ile tespit edilmiştir. Son olarak Z değeri %4,57 hata oranlarında ve ortalama 0,95 mm'lik hata ile tespit edilmiştir. Tablo 4.2.'de mm cinsinden alınan bazı örneklerin bulunan değerleri ve gerçekte olan değerleri sunulmuştur. Çok küçük hatalarla tespitler gerçekleştirilmiştir. Bu hatalar uygulamanın doğruluğunu büyük oranda etkilememektedir. Uygulama ortalama 15-16 FPS ile

çalışmaktadır. Uygulamanın gerçekliğini arttırmak için pozisyon tahmininin yanı sıra FPS değerinin gerçeğe yakın olması amaçlanmıştır. Bu nedenle 15-16 FPS yeterli bulunmamıştır.

Bölüm 5.'de, pozisyon çıkarımı algoritmaları ve görüntü işleme yöntemleri kullanılarak ortamdaki nesnenin tespiti yapılmıştır. Gerçekleştirilen sistemde iki Raspberry Pi kamera kullanılmış, bu kameraların kalibrasyonu gerçekleştirilmiş ve görüntülerin düzeltilmesi ile kamera lenslerindeki bozulmalar düzeltilmiştir. Çalışma kapsamında, renk tabanlı algılama yaklaşımı kullanarak nesnenin 3D konum bilgisinin gerçek zamanlı tespiti ve takibini yapacak yazılım çalışması yapılmıştır. Yazılım kısmı için Windows ve Raspbian işletim sistemi altında çalışılmış olup, OpenCV kütüphanesinden yararlanılmıştır. Raspberry kamera modülü ile elde edilen veriler Windows işletim sistemine sahip bilgisayardaki programda kullanılarak 3D konum bilgisi elde edilmiştir. Raspberry modül ile bilgisayar arasındaki bağlantı TCP (ing. Transmission Control Protokol) protokolü ile sağlanmıştır. Raspberry Pi kamera modülleri kullanılarak Kartezyen koordinat sisteminde pozisyon tespiti yapılmıştır. Kartezyen koordinat sisteminde kullanılan algoritma girdi olarak referans noktaları, nesnenin iki kamerada oluşan izdüşüm değerleri ve kamera parametrelerini almaktadır. Pozisyon tahmini algoritmasına verilen bu değerler ile nesnenin 3D koordinatları bulunmaktadır. Kullanılan aktif işaretçi kırmızı renklidir. Bu renkteki renkteki 3D nesnenin projeksiyon ekranlarındaki izdüşüm değerlerine göre pozisyon tahmini yapılmaktadır. Nesnenin birinci Pi kamera ekranındaki izdüşümünün birinci referans noktasına yataydaki uzaklığı ile ikinci Pi kamera ekranındaki izdüşümünün ikinci referans noktasına yataydaki uzaklığı elde edilmiştir. Elde edilen bu değerler kullanılarak iterasyon yöntemi ile gerçek x, y değerleri elde edilmiştir. Nesnenin bu referans noktalarına olan dikey uzaklıklarının ortalaması z değerini vermektedir.

Sistemde kameralar ilk açıldığında normal şartlar ile satranç tahtası üzerindeki referans noktalarını tespit ettikten sonra, görüntü karartıldı. Bu sayede aktif işaretçi tespit edilirken ortam şartlarından etkilenmesi olumsuzluğu ortadan kaldırıldı. Alınan görüntülere filtrelemeler uygulanarak sağlam tespit yapılması sağlandı. Ortam karartıldığından işaretçi tespiti ortam şartlarından etkilenmemektedir. Bu nedenle aktif işaretçinin konum bilgisi daha doğru sonuçlar ile bulunmaktadır.

Bu çalışmayı test etmek için 3D konum bilgileri bilinen örnekler alındı ve bu pozisyonlara karşılık gelen değerler bulundu. Bulunan bu değerler ile gerçek değerler

karşılaştırıldı. Tablo 5.1.'de verildiği gibi ölçüm sonuçları bulunmuş ve belirtilen hata oranlarıyla sistem gerçekleştirilmiştir. Alınan örneklerde X değeri ortalama 0,95 mm, Y değeri ortalama 0,84 mm ve Z değeri ortalama 0,50 mm hata ile tespit edilmiştir. Tüm koordinatlar ortalama %1,37 hata oranlarında bulunmuştur. Uygulama ortalama 34 ms ile çalışmaktadır.

Önceki bölümlerde pozisyon çıkarımı için sarı renkli bir nesne kullanılmıştı. Geliştirilen sistemlerde en büyük dezavantaj pozisyon çıkarımı aşamasında kullanılan algoritmanın ortam şartlarına bağlı olarak çok çabuk bozulmaya uğramasıydı. Ortamın ışık kaynağının ışık şiddeti, arka plan, ortamdaki nesnenin hareket hızı ve bunlara bağlı olarak nesnenin yanlış tespit edilmesi ölçülen değerlerle ilgili farklı sonuçların çıkmasına sebep olmakta idi. Bu sorun Raspberry Pi kamera modülü ile çözülmüştür.

Bu tez çalışmasında geliştirilen sistemlerin avantajları olarak yalnızca görüntü bilgisinin değerlendirilmesiyle nesne ve koordinat tespiti yapabilmesi, gerçek zamanlı bir uygulamada kullanılabilecek düzeyde hızlı olması ve modüler yapısı sayesinde başka uygulamalara da entegre edilebilir olması sayılabilir.

Bu tez çalışmasında kartezyen koordinat sistemi ile konum bilgisi yüksek doğruluk elde edilmiştir. Konum bilgisinin yanı sıra saniyede 30 görüntü için gerekli olan hız sağlanmıştır. Bu uygulamada koordinat bilgisi ortalama 34 ms ile elde edilmiştir.

KAYNAKÇA

[1] Elektrikport, Teknik Kütüphane Sayfası <http://www.elektrikport.com/> E.Tarihi: 20.04.2017.

[2] J.BORG, “DETECTING AND TRACKING PLAYERS IN FOOTBALL USING STEREO VISION”, Department of Electrical Engineering Linköping University, 2007.

[3] Edgar R., Dr. Francesc Moreno, N.Implementation of a 3D pose estimation algorithm, June 2015.

[4] OpenCV (Open Source Computer Vision) <http://opencv.org/> E.Tarihi: 25.04.2017.

[5] Erhan AKKAYA, Muhammed Fatih TALU, Nesne Görüntülerinden 3D Model İnşası İçin Etkin F-matrisi Hesaplanma Yöntemleri, Ekim 2014.

[6] R. Hartley and A. Zisserman, Multiple view geometry in computer vision. Cambridge, UK; New York: Cambridge University Press, 2003..

[7] V. Lepetit, F. Moreno-Noguer, and P. Fua. EPnP: An accurate O(n) solution to the PnP problem. 81(2):151{166, 2008..

[8] Hüseyin K., Engin A., Ali İhsan Ç., "Evaluation of cost functions for stereo matching", IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), Minneapolis, Haziran 2007.

[9] Hirschmüller, H., Scharstein, D., "Evaluation of cost functions for stereo matching", IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), Minneapolis, Haziran 2007.

[10] U. Christian, “Contributions to Stereo Vision”, TECHNISCHE UNIVERSITAT MÜNCHEN, 2013.

[11] Compan, P., Satorre, R., Rizo, R., Molina, R., "Improving Depth Estimation using Colour Information in Stereo Vision", Visualization, Imaging, and Image Processing, Eylül 2005..

[12] Kack, P., "Robust stereo correspondence using graph cuts", Yüksek Lisans Tezi, School of Computer Science and Engineering, Royal Institute of Technology, Stockholm, 2004..

[13] Y. L. Abdel-Aziz & H. Karara, “Direct Linear Transformation from Comparator Coordinates into Object Space Coordinates in Close-Range Photogrammetry”, In Proc. ASP/UI Symp. Close-Range Photogrammetry, pp. 1-18, 1971..

[14] V. Lepetit, F. Moreno-Noguer & P. Fua, “EPnP: An Accurate O(n) Solution to the PnP Problem”, International Journal of Computer Vision, vol. 81, pp. 155–166, 2009..

[15] X.-S. Gao, X.-R. Hou, J. Tang, and H.-F. Cheng. Complete solution classification for the perspective-three-point problem. 25(8):930{943, 2003..

[16] M.A. Fischler and R.C. Bolles. Random sample consensus: a paradigm for model.

[17] A. Ansar & K. Daniilidis, “Linear Pose Estimation from Points or Lines”, Proc. European Conf. Computer Vision, vol. 4, pp. 282-296, May 2002..

[18] R. Kumar & A. R. Hanson, “Robust Methods for Estimating Pose and a Sensitivity Analysis”, Computer Vision and Image Analysis, vol. 60, pp. 313-342, 1994..

[19] P. David, D. DeMenthon, R. Duraiswami & H. Samet, “Simultaneous Pose and Correspondence Determination Using Line Features”, In Computer Vision and Pattern Recognition, Vol. 2, pp. 424-431, 2003..

[20] D.G Lowe, “Fitting Parametrized Three-Dimensional Models to Images”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 13, pp. 441-450, 1991..

[21] A. Lipnickas, A. Knys, “A Stereovision System for 3D Perception” , Electronics And Electrical Engineering, 2009.

[22] G. ÇETİNKAYA, “A COMPARATIVE STUDY ON POSE ESTIMATION ALGORITHMS USING VISUAL DATA”, 2012.

[23] C. Changhyun, B. Seung-Min, L. Sukhan, “Real-Time 3D Object Pose Estimation and Tracking for Natural Landmark Based Visual Servo”, IEEE.

[24] P. Edgar Riba, “Implementation of a 3D pose estimation algorithm”, Universitat Politecnica de Catalunya, 2015.

[25] T. Song, Z. Lyu, X. Ding, Y. Wan, “3D Suface Reconstruction Based on Kinect Sensor”, International Journal of Computer Theory and Engineering, Vol. 5, No. 3, June 2013.

[26] B. Christian, P. Mathias, G. Hartmut, H. Horst, “Evaluation of Ratio Based, Optical and Barometric Localization for Indoor Altitude Estimation in Medical Applications”, International Conference on Indoor Positioning and Indoor Navigation, October 2014.

[27] K. Hüseyin, A. Engin, Ç. Ali İhsan, “ OpenCV ve Kinect ile Stereo Görüntüden Derinlik Algılama”, Akademik Platform.

[28] Shu Zhang, Hui Yu, Junyu Dong, Ting Wang, Lin Qi, Honghai Liu, “Combining Kinect and PnP for Camera Pose Estimation”.

[29] Tomas Möller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. J. Graph. Tools, 2(1):21{28, October 1997. ISSN 1086-7651.

[30] Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 36, 2014..

[31] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez. 2014. "Automatic generation and detection of highly reliable fiducial markers under occlusion". Pattern Recogn. 47, 6 (June 2014), 2280-2292. DOI=10.1016/j.patcog.2014.01.0.

[32] Küresel Koordinat Sistemi <https://tr.wikipedia.org/wiki/K%C3%BCresel_koordinat_sistemi> E.Tarihi:01.04.2017.

[33] K.Yalçın, Ö.Sıtkı, K.Melih, “Hareket Eden Renkli Nesnelerin Takibinin PID ile Gerçekleştirilmesi“, Otomatik Kontrol Ulusal Toplantısı, TOK-2012, 11-13 Ekim 2012, Niğde.

[34] Raspberry Pi <https://www.raspberrypi.org/help/what-is-a-raspberry-pi/> E.Tarihi :17.04.2017.

[35] MobaxTerm <https://blogspot.tenettech.com/remotely-access-raspberry-pi-using-mobaxterm.html> E.Tarihi:21.04.2017.

EK-1 Doku Kaplı Nesnenin Pozisyon Tespiti

```
/**  Main program  **/
int main(int argc, char *argv[])
{
	help();
	const String keys =
		"{help h		|	| print this message				}"
		"{video v		|	| path to recorded video			}"
		"{model		|	| path to yml model				}"
		"{mesh		|	| path to ply mesh				}"
		"{keypoints k	|10000	 | number of keypoints to detect		 }"
		"{ratio r		|0.7	| threshold for ratio test			}"
		"{iterations it	|100	| RANSAC maximum iterations count		}"
		"{error e		|2.0	| RANSAC reprojection error			}"
		"{confidence c	|0.95	| RANSAC confidence				}"
		"{inliers in	|30	| minimum inliers for Kalman update		}"
		"{method  pnp	|0	| PnP method: (0) ITERATIVE - (1) EPNP - (2)
P3P - (3) DLS}"
		"{fast f		|true	| use of robust fast match			}"
		;
	CommandLineParser parser(argc, argv, keys);
	if (parser.has("help"))
	{
		parser.printMessage();
		return 0;
	}
	else
	{
		video_read_path = parser.get<string>("video").size() > 0 ?
parser.get<string>("video") : video_read_path;
		yml_read_path = parser.get<string>("model").size() > 0 ?
parser.get<string>("model") : yml_read_path;
		ply_read_path = parser.get<string>("mesh").size() > 0 ?
parser.get<string>("mesh") : ply_read_path;
		numKeyPoints = !parser.has("keypoints") ?
parser.get<int>("keypoints") : numKeyPoints;
		ratioTest = !parser.has("ratio") ? parser.get<float>("ratio") :
ratioTest;
		fast_match = !parser.has("fast") ? parser.get<bool>("fast") :
fast_match;
		iterationsCount = !parser.has("iterations") ?
parser.get<int>("iterations") : iterationsCount;
		reprojectionError = !parser.has("error") ?
parser.get<float>("error") : reprojectionError;
		confidence = !parser.has("confidence") ?
parser.get<float>("confidence") : confidence;
		minInliersKalman = !parser.has("inliers") ?
parser.get<int>("inliers") : minInliersKalman;
		pnpMethod = !parser.has("method") ? parser.get<int>("method") :
pnpMethod;
	}

	PnPProblem pnp_detection(params_WEBCAM);
	PnPProblem pnp_detection_est(params_WEBCAM);

	Model model;                // instantiate Model object
	model.load(yml_read_path); // load a 3D textured object model

	Mesh mesh;                  // instantiate Mesh object
	mesh.load(ply_read_path);  // load an object mesh
```

```
      RobustMatcher rmatcher;
      // instantiate RobustMatcher
       Ptr<FeatureDetector> orb = ORB::create();
       rmatcher.setFeatureDetector(orb);           // set feature detector
       rmatcher.setDescriptorExtractor(orb);       // set descriptor extractor
       Ptr<flann::IndexParams> indexParams = makePtr<flann::LshIndexParams>(6,
12, 1); // instantiate LSH index parameters
       Ptr<flann::SearchParams> searchParams = makePtr<flann::SearchParams>(50);
      // instantiate flann search parameters
       // instantiate FlannBased matcher
       Ptr<DescriptorMatcher> matcher = makePtr<FlannBasedMatcher>(indexParams,
searchParams);
      rmatcher.setDescriptorMatcher(matcher);
      // set matcher
       rmatcher.setRatio(ratioTest); // set ratio test parameter

       KalmanFilter KF;         // instantiate Kalman Filter
       int nStates = 18;            // the number of states
       int nMeasurements = 6;       // the number of measured states
       int nInputs = 0;             // the number of control actions
       double dt = 0.125;           // time between measurements (1/FPS)

       initKalmanFilter(KF, nStates, nMeasurements, nInputs, dt);    // init
function
       Mat measurements(nMeasurements, 1, CV_64F); measurements.setTo(Scalar(0));
       bool good_measurement = false;

       // Get the MODEL INFO
       vector<Point3f> list_points3d_model = model.get_points3d();  // list with
model 3D coordinates
       Mat descriptors_model = model.get_descriptors();
   // list with descriptors of each 3D coordinate
   // Create & Open Window
       namedWindow("REAL TIME DEMO", WINDOW_KEEPRATIO);
       VideoCapture cap;                           // instantiate VideoCapture
       cap.open(video_read_path);                      // open a recorded video
       if (!cap.isOpened())   // check if we succeeded
       {
              cout << "Could not open the camera device" << endl;
              return -1;
       }
       // start and end times
       time_t start, end;
       // fps calculated using number of frames / seconds
       // floating point seconds elapsed since start
       double fps, sec;
       // frame counter
       int counter = 0;
       // start the clock
       time(&start);
       Mat frame, frame_vis;
       while (cap.read(frame) && waitKey(30) != 27) // capture frame until ESC is
pressed
       {
              frame_vis = frame.clone();    // refresh visualisation frame
// -- Step 1: Robust matching between model descriptors and scene descriptors
              vector<DMatch> good_matches;       // to obtain the 3D points of
the model
              vector<KeyPoint> keypoints_scene;  // to obtain the 2D points of
the scene
```

```
		if (fast_match)
		{
			rmatcher.fastRobustMatch(frame, good_matches,
keypoints_scene, descriptors_model);
		}
		else
		{
			rmatcher.robustMatch(frame, good_matches, keypoints_scene,
descriptors_model);
		}
// -- Step 2: Find out the 2D/3D correspondences
		vector<Point3f> list_points3d_model_match; // container for the
model 3D coordinates found in the scene
		vector<Point2f> list_points2d_scene_match; // container for the
model 2D coordinates found in the scene
		for (unsigned int match_index = 0; match_index <
good_matches.size(); ++match_index)
		{
			Point3f point3d_model =
list_points3d_model[good_matches[match_index].trainIdx];  // 3D point from model
			Point2f point2d_scene =
keypoints_scene[good_matches[match_index].queryIdx].pt; // 2D point from the
scene
			list_points3d_model_match.push_back(point3d_model);
// add 3D point
			list_points2d_scene_match.push_back(point2d_scene);
// add 2D point
		}
		// Draw outliers
		draw2DPoints(frame_vis, list_points2d_scene_match, red);
		Mat inliers_idx;
		vector<Point2f> list_points2d_inliers;
		if (good_matches.size() > 0) // None matches, then RANSAC crashes
		{
			// -- Step 3: Estimate the pose using RANSAC approach
			pnp_detection.estimatePoseRANSAC(list_points3d_model_match,
list_points2d_scene_match,
				pnpMethod, inliers_idx,
				iterationsCount, reprojectionError, confidence);
			// -- Step 4: Catch the inliers keypoints to draw
			for (int inliers_index = 0; inliers_index < inliers_idx.rows;
++inliers_index)
			{
				int n = inliers_idx.at<int>(inliers_index);// i-inlier
				Point2f point2d = list_points2d_scene_match[n];
				// i-inlier point 2D
				list_points2d_inliers.push_back(point2d);
				 // add i-inlier to list
			}

			// Draw inliers points 2D
			draw2DPoints(frame_vis, list_points2d_inliers, blue);
			// -- Step 5: Kalman Filter
			good_measurement = false;
			// GOOD MEASUREMENT
			if (inliers_idx.rows >= minInliersKalman)
			{
				// Get the measured translation
				Mat translation_measured(3, 1, CV_64F);
				translation_measured = pnp_detection.get_t_matrix();
```

```
                        // Get the measured rotation
                        Mat rotation_measured(3, 3, CV_64F);
                        rotation_measured = pnp_detection.get_R_matrix();
                        // fill the measurements vector
                        fillMeasurements(measurements, translation_measured,
rotation_measured);
                        good_measurement = true;

                }

                // Instantiate estimated translation and rotation
                Mat translation_estimated(3, 1, CV_64F);
                Mat rotation_estimated(3, 3, CV_64F);

                // update the Kalman filter with good measurements
                updateKalmanFilter(KF, measurements,
                        translation_estimated, rotation_estimated);
                // -- Step 6: Set estimated projection matrix
                pnp_detection_est.set_P_matrix(rotation_estimated,
translation_estimated);
            }
            // -- Step X: Draw pose
            if (good_measurement)
            {
                    drawObjectMesh(frame_vis, &mesh, &pnp_detection, green);
                    // draw current pose
            }
            else
            {
                    drawObjectMesh(frame_vis, &mesh, &pnp_detection_est, yellow);
// draw estimated pose
            }

            float l = 5;
            vector<Point2f> pose_points2d;

      pose_points2d.push_back(pnp_detection_est.backproject3DPoint(Point3f(0, 0,
0)));  // axis center

      pose_points2d.push_back(pnp_detection_est.backproject3DPoint(Point3f(l, 0,
0)));  // axis x

      pose_points2d.push_back(pnp_detection_est.backproject3DPoint(Point3f(0, l,
0)));  // axis y

      pose_points2d.push_back(pnp_detection_est.backproject3DPoint(Point3f(0, 0,
l)));  // axis z
            draw3DCoordinateAxes(frame_vis, pose_points2d);   // draw axes
      // FRAME RATE
     // see how much time has elapsed
            time(&end);
            // calculate current FPS
            ++counter;
            sec = difftime(end, start);
            fps = counter / sec;
            drawFPS(frame_vis, fps, yellow); // frame ratio
            double detection_ratio = ((double)inliers_idx.rows /
(double)good_matches.size()) * 100;
            drawConfidence(frame_vis, detection_ratio, yellow);
```

```
            // -- Step X: Draw some debugging text
            // Draw some debug text
            int inliers_int = inliers_idx.rows;
            int outliers_int = (int)good_matches.size() - inliers_int;
            string inliers_str = IntToString(inliers_int);
            string outliers_str = IntToString(outliers_int);
            string n = IntToString((int)good_matches.size());
            string fpss = to_string(fps);
            string text = "Found: " + inliers_str + " of " + n + " matches";
            string text2 = "Inliers: " + inliers_str + " - Outliers: " +
outliers_str;
            drawText(frame_vis, text, green);
            drawText2(frame_vis, text2, red);
            imshow("REAL TIME DEMO", frame_vis);
      }
      // Close and Destroy Window
      destroyWindow("REAL TIME DEMO");

}
```

EK-2 Küresel Koordinatlarda Pozisyon Tespiti

```
void KureselKoordinat()
{
	//store image to matrix
	capture.read(webcamImage);

	frameCounter++;

	std::time_t timeNow = std::time(0) - timeBegin;

	if (timeNow - tick >= 1)
	{
		tick++;
		fps = frameCounter;
		frameCounter = 0;
	}
	cv::putText(webcamImage, cv::format("Average FPS=%d", fps), cv::Point(550,
30), cv::FONT_HERSHEY_SIMPLEX, 0.8, cv::Scalar(0, 0, 255));
	//make a gray copy of the webcam image
	cvtColor(webcamImage, gray, COLOR_BGR2GRAY);
	cv::GaussianBlur(webcamImage, webcamImage, cv::Size(3, 3), 0);
	cv::cvtColor(webcamImage, hsvImg, CV_BGR2HSV);
	// Convert Original Image to HSV Thresh Image
	cv::inRange(hsvImg, cv::Scalar(lowH, lowS, lowV), cv::Scalar(highH, highS,
highV), threshImg);
	cv::dilate(threshImg, threshImg, 0);
	cv::erode(threshImg, threshImg, 0);

	vector<vector<cv::Point> > contours;
	vector<cv::Vec4i> heirarchy;
	vector<cv::Point2i> center;
	vector<int> radius;

	cv::findContours(threshImg.clone(), contours, heirarchy, CV_RETR_TREE,
CV_CHAIN_APPROX_NONE);
	size_t count = contours.size();

	//sarı nesnenin kameraya uzaklığı
	int max = 0;
	int maxi = 0;

	if (count > 1){
		for (int i = 0; i < count; i++)
		{
			if (contours[i].size() > max){
				max = contours[i].size();
				maxi = i;
			}
		}
	}

	float minTargetRadius = 10;
	if (count > 0){
		cv::Point2f c;
		float r;
		cv::minEnclosingCircle(contours[maxi], c, r);
		if (r >= minTargetRadius)
		{
			circle(img, Point(c.x, c.y), r, Scalar(255, 0, 0), 2);
			center.push_back(c);
			x_izdusum = c.x;
```

```
                    y_izdusum = c.y;
                    //sarı nesnenin kameraya uzaklığı
                    double W_ball = 1 / 2.54; //inç genişiliği
                    double P_ball = 2 * 45.93;  //algılanan piksel genişiliği
width of the paper piksel
                    double D_ball = 12 / 2.54;   // kameraya konulan uzaklığın
inç karşılığı inches in front of my camera and take a photo
                    double focal_length_ball = (P_ball*D_ball) / W_ball;
                    double P_ball2 = 2 * r;
                    distance_camera = ((W_ball * focal_length_ball) /
P_ball2)*2.54;
                    r2 = 27;
                    mesafe = 8.7;
                           if (sayac>0){
                                  alfa_aci = (mesafe * 360) / (2 * PI * r2);

                                  x_cm = (mesafe*(x_izdusum - imagePoints[0].x) /
(imagePoints[48].x - imagePoints[0].x));

                                  teta_aci = (x_cm*alfa_aci) / mesafe;

                                  y_cm = (mesafe*(imagePoints[0].y - y_izdusum) /
(imagePoints[48].x - imagePoints[0].x));

                                  fi_aci = (y_cm*alfa_aci) / mesafe;

                                  double z = distance_board - distance_camera;
                                  r2 = distance_camera;

                                  x_gercek = r2*sin((teta_aci)*PI /
180)*cos((fi_aci)*PI / 180);
                                  y_gercek = r2*sin(fi_aci*PI / 180);
                                  z_gercek = r2*cos(fi_aci*PI /
180)*cos(teta_aci*PI / 180);


                                  X_Point = 27 - r2;
                                  Y_Point = 14 - x_gercek;
                                  Z_Point = y_gercek;
                           }
                    }
             }

}
```

EK-3 Stereo Görme Pozisyon Tahmini

```
void StereoGorme()
{
	while (1){
		captureL.read(imageL);
		captureR.read(imageR);
		frameCounter++;
		std::time_t timeNow = std::time(0) - timeBegin;
		if (timeNow - tick >= 1)
		{
			tick++;
			fps = frameCounter;
			frameCounter = 0;
		}
		cv::putText(imageR, cv::format("Average FPS=%d", fps),
cv::Point(30, 30), cv::FONT_HERSHEY_SIMPLEX, 0.8, cv::Scalar(0, 0, 255));
		//Create transformation and rectification maps
		Mat cam1map1, cam1map2;
		Mat cam2map1, cam2map2;

		initUndistortRectifyMap(M1, D1, R1, P1, Size(1280, 720), CV_16SC2,
cam1map1, cam1map2);
		initUndistortRectifyMap(M2, D2, R2, P2, Size(1280, 720), CV_16SC2,
cam2map1, cam2map2);

		Mat leftStereoUndistorted, rightStereoUndistorted; //Create
matrices for storing rectified images

		//Rectify and undistort images
		remap(imageL, leftStereoUndistorted, cam1map1, cam1map2,
INTER_LINEAR);
		remap(imageR, rightStereoUndistorted, cam2map1, cam2map2,
INTER_LINEAR);
		cv::GaussianBlur(imageL, imageL, cv::Size(3, 3), 0);
	//Blur Effect
		cv::cvtColor(imageL, hsvImg0, CV_BGR2HSV);
		// Convert Original Image to HSV Thresh Image
		cv::inRange(hsvImg0, cv::Scalar(lowH, lowS, lowV),
cv::Scalar(highH, highS, highV), threshImg0);
		cv::dilate(threshImg0, threshImg0, 0);
		cv::erode(threshImg0, threshImg0, 0);
		cvtColor(imageR, gray, COLOR_BGR2GRAY);
		cv::GaussianBlur(imageR, imageR, cv::Size(3, 3), 0);
		cv::cvtColor(imageR, hsvImg1, CV_BGR2HSV);
		// Convert Original Image to HSV Thresh Image
		cv::inRange(hsvImg1, cv::Scalar(lowH, lowS, lowV),
cv::Scalar(highH, highS, highV), threshImg1);
		cv::dilate(threshImg1, threshImg1, 0);
		cv::erode(threshImg1, threshImg1, 0);

		vector<vector<cv::Point> > contours0, contours1;
		vector<cv::Vec4i> heirarchy0, heirarchy1;
		vector<cv::Point2i> center0, center1;
		vector<int> radius0, radius1;

		cv::findContours(threshImg0.clone(), contours0, heirarchy0,
CV_RETR_TREE, CV_CHAIN_APPROX_NONE);
		cv::findContours(threshImg1.clone(), contours1, heirarchy1,
CV_RETR_TREE, CV_CHAIN_APPROX_NONE);
```

```
		size_t count0 = contours0.size();
		size_t count1 = contours1.size();

		float minTargetRadius = 10;
		cv::Point2f c0, c1;
		float r0, r1;
		//sarı nesnenin kameraya uzaklığı
		int max = 0;
		int maxi = 0;

		if (count0 > 1){
			for (int i = 0; i < count0; i++)
			{
				if (contours0[i].size() > max){
					max = contours0[i].size();
					maxi = i;
				}
			}
		}

		if (count0 > 0){
			cv::minEnclosingCircle(contours0[maxi], c0, r0);
			if (r0 >= minTargetRadius)
			{
				circle(imageL, Point(c0.x, c0.y), r0, Scalar(255, 0,
0), 2);
				center0.push_back(c0);
				ttx0 = c0.x;
				tty0 = c0.y;
			}
		}

		int maxy = 0;
		int maxyi = 0;

		if (count1 > 1){
			for (int i = 0; i < count1; i++)
			{
				if (contours1[i].size() > maxy){
					maxy = contours1[i].size();
					maxyi = i;
				}
			}
		}
		if (count1 > 0){
			cv::minEnclosingCircle(contours1[maxyi], c1, r1);
			if (r1 >= minTargetRadius)
			{
				circle(imageR, Point(c1.x, c1.y), r1, Scalar(255, 0,
0), 2);
				center1.push_back(c1);
				ttx1 = c1.x;
				tty1 = c1.y;
			}
		}

		//caculating disparity and distance.Then, display them on images
		if ((ttx0>0) && (ttx1>0))
		{
			double focal_length = 7.8731320620831173e+02;
			double baseline = 8;
```

```
                    disparity = ttx0 - ttx1;
                    double dispY = tty0 - tty1;
                    dist = (focal_length * baseline) / disparity;
                    _itoa(dist, chardist, 10);
                    _itoa(disparity, chardisparity, 10);

                    X = ttx0*baseline / disparity;
                    Y = tty0*baseline / dispY;
                    Z = dist;
        }
}
```

EK-4 Satranç Tahtası İşaretçisi Tespiti

```
bool detectChessboard(cv::Mat gray){

	string filename = "out_camera_data.yml";
	FileStorage fs;
	fs.open(filename, FileStorage::READ);
	if (fs.isOpened())
	{
		cout << "File is opened\n";
	}
	// read camera matrix and distortion coefficients from file
	//Mat intrinsics, distortion;
	fs["Camera_Matrix"] >> intrinsics;
	fs["Distortion_Coefficients"] >> distortion;
	// close the input file
	fs.release();

	//generate vectors for the points on the chessboard
	for (int i = 0; i<boardWidth; i++)
	{
		for (int j = 0; j<boardHeight; j++)
		{
			boardPoints.push_back(Point3d(double(i), double(j), 0.0));
		}
	}
	//generate points in the reference frame
	framePoints.push_back(Point3d(0.0, 0.0, 0.0));
	framePoints.push_back(Point3d(5.0, 0.0, 0.0));
	framePoints.push_back(Point3d(0.0, 5.0, 0.0));
	framePoints.push_back(Point3d(0.0, 0.0, 5.0));
	//detect chessboard corners
	bool found = findChessboardCorners(gray, cbSize, imagePoints,
CALIB_CB_FAST_CHECK);
	//find camera orientation if the chessboard corners have been found
	if (found)
	{
		//find the camera extrinsic parameters
		solvePnP(Mat(boardPoints), Mat(imagePoints), intrinsics,
distortion, rvec, tvec, false);

		//project the reference frame onto the image
		projectPoints(framePoints, rvec, tvec, intrinsics, distortion,
imageFramePoints);


	}
	if (imagePoints.size() == 0){
		return false;
	}
	else{
		return true;
	}
}
```

EK-5 Aruco İşaretçisi Tespiti

```
const float markerLength = 2.0;
void detectAruco()
{
	printf("This program detects ArUco markers.\n");
	printf("Hit the ESC key to quit.\n");
	// Camera intrinsic matrix (fill in your actual values here).
	double K_[3][3] =
	{ { 1.3979459334509404e+03, 0, 6.3950000000000000e+02 },
	{ 0, 1.3979459334509404e+03, 3.5950000000000000e+02 },
	{ 0, 0, 1 } };
	cv::Mat K = cv::Mat(3, 3, CV_64F, K_).clone();
	// Distortion coeffs (fill in your actual values here).
	double dist_[] = { 6.5667538045390203e-02, -1.1495887577977015e+00, 0, 0,
4.4420217064201886e+00 };
	cv::Mat distCoeffs = cv::Mat(5, 1, CV_64F, dist_).clone();
	cv::VideoCapture cap(0);// open the camera
	//cv::VideoCapture cap("video.avi"); // or open the video file
	if (!cap.isOpened()) { // check if we succeeded
		printf("error - can't open the camera or video; hit any key to
quit\n");
		system("PAUSE");
		return EXIT_FAILURE;
	}
	// Let's just see what the image size is from this camera or file.
	double WIDTH = cap.get(CV_CAP_PROP_FRAME_WIDTH);
	double HEIGHT = cap.get(CV_CAP_PROP_FRAME_HEIGHT);
	printf("Image width=%f, height=%f\n", WIDTH, HEIGHT);

	// Allocate image.
	cv::Mat image;
	cv::Ptr<cv::aruco::Dictionary> dictionary =
cv::aruco::getPredefinedDictionary(cv::aruco::DICT_4X4_100);

	int frameCounter = 0;
	int tick = 0;
	int fps;
	std::time_t timeBegin = std::time(0);
	cv::Ptr<cv::aruco::DetectorParameters> detectorParams =
cv::aruco::DetectorParameters::create();
	// Run an infinite loop until user hits the ESC key.
	while (1) {
		cap >> image; // get image from camera
		if (image.empty()) break;
		frameCounter++;
		std::time_t timeNow = std::time(0) - timeBegin;
		if (timeNow - tick >= 1)
		{
			tick++;
			fps = frameCounter;
			frameCounter = 0;
		}

		cv::putText(image, cv::format("Average FPS=%d", fps), cv::Point(30,
30), cv::FONT_HERSHEY_SIMPLEX, 0.8, cv::Scalar(0, 0, 255));

		std::vector< int > markerIds;
		std::vector< std::vector<cv::Point2f> > markerCorners,
rejectedCandidates;
```

```
		cv::aruco::detectMarkers(
			image, // input image
			dictionary, // type of markers that will be searched for
			markerCorners, // output vector of marker corners
			markerIds, // detected marker IDs
			detectorParams, // algorithm parameters
			rejectedCandidates);
		if (markerIds.size() > 0) {
			// Draw all detected markers.
			cv::aruco::drawDetectedMarkers(image, markerCorners,
markerIds);
			std::vector< cv::Vec3d > rvecs, tvecs;

			cv::aruco::estimatePoseSingleMarkers(
				markerCorners, // vector of already detected markers
corners
				markerLength, // length of the marker's side
				K, // input 3x3 floating-point instrinsic camera
matrix K
				distCoeffs, // vector of distortion coefficients of 4,
5, 8 or 12 elements
				rvecs, // array of output rotation vectors
				tvecs); // array of output translation vectors

			// Display pose for the detected marker with id=0.
			for (unsigned int i = 0; i < markerIds.size(); i++) {
				if (markerIds[i] == 6) {
					cv::Vec3d r = rvecs[i];
					cv::Vec3d t = tvecs[i];
					// Draw coordinate axes.
					cv::aruco::drawAxis(image,
						K, distCoeffs, // camera parameters
						r, t, // marker pose
						0.5*markerLength); // length of the axes
to be drawn
				std::vector<cv::Point3d> pointsInterest;

	pointsInterest.push_back(cv::Point3d(markerLength / 2, markerLength / 2,
0));
					std::vector<cv::Point2d> p;
					cv::projectPoints(pointsInterest, rvecs[i],
tvecs[i], K, distCoeffs, p);
					cv::drawMarker(image,
						p[0], // image point
						cv::Scalar(0, 255, 255), // color
						cv::MARKER_STAR, // type of marker to
draw
						20, // marker size
						2); // thickness

				}
			}
		}
		cv::imshow("Image", image); // show image
	}
}
```

EK-6 Kartezyen Koordinat Sistemi ile Pozisyon Tahmini

```
void kartezyenKoordinat()
{
	VideoCapture captureL(0);
	captureL.set(CV_CAP_PROP_FRAME_WIDTH, FRAME_WIDTH);
	captureL.set(CV_CAP_PROP_FRAME_HEIGHT, FRAME_HEIGHT);
	captureL.set(CV_CAP_PROP_CONVERT_RGB, 1);
	if (!captureL.isOpened()) cout << "L doesn't work" << endl;
	VideoCapture captureR(1);
	captureR.set(CV_CAP_PROP_FRAME_WIDTH, FRAME_WIDTH);
	captureR.set(CV_CAP_PROP_FRAME_HEIGHT, FRAME_HEIGHT);
	captureR.set(CV_CAP_PROP_CONVERT_RGB, 2);
	if (!captureR.isOpened()) cout << "R doesn't work" << endl;

	Mat imageL, imageR;

	//fps calculate
	int frameCounter = 0;
	int tick = 0;
	int fps;
	std::time_t timeBegin = std::time(0);

	while (1){

		captureL.read(imageL);
		captureR.read(imageR);
		frameCounter++;
		std::time_t timeNow = std::time(0) - timeBegin;
		if (timeNow - tick >= 1)
		{
			tick++;
			fps = frameCounter;
			frameCounter = 0;
		}
		cv::putText(imageR, cv::format("Average FPS=%d", fps),
cv::Point(10, 30), cv::FONT_HERSHEY_SIMPLEX, 0.8, cv::Scalar(0, 0, 255));

		waitKey(1);
		clock_t t1, t2;
		t1 = clock();

		cvtColor(imageL, grayl, COLOR_BGR2GRAY);

		cv::GaussianBlur(imageL, imageL, cv::Size(3, 3), 0);
	//Blur Effect
		cv::cvtColor(imageL, hsvImg0, CV_BGR2HSV);
		// Convert Original Image to HSV Thresh Image
		cv::inRange(hsvImg0, cv::Scalar(lowH, lowS, lowV),
cv::Scalar(highH, highS, highV), threshImg0);
		cv::dilate(threshImg0, threshImg0, 0);
		cv::erode(threshImg0, threshImg0, 0);
		cvtColor(imageR, grayr, COLOR_BGR2GRAY);
		cv::GaussianBlur(imageR, imageR, cv::Size(3, 3), 0);
		cv::cvtColor(imageR, hsvImg1, CV_BGR2HSV);
		// Convert Original Image to HSV Thresh Image
		cv::inRange(hsvImg1, cv::Scalar(lowH, lowS, lowV),
cv::Scalar(highH, highS, highV), threshImg1);
		cv::dilate(threshImg1, threshImg1, 0);
		cv::erode(threshImg1, threshImg1, 0);

		vector<vector<cv::Point> > contours0, contours1;
```

```
		vector<cv::Vec4i> heirarchy0, heirarchy1;
		vector<cv::Point2i> center0, center1;
		vector<int> radius0, radius1;

		cv::findContours(threshImg0.clone(), contours0, heirarchy0,
CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
		cv::findContours(threshImg1.clone(), contours1, heirarchy1,
CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE, Point(0, 0));

		size_t count0 = contours0.size();
		size_t count1 = contours1.size();

		float minTargetRadius = 10;
		cv::Point2f c0, c1;
		float r0, r1;
		double mesafe = 8.7; //mm
		double orijineuzaklık = 13;
		//sarı nesnenin kameraya uzaklığı
		int max = 0;
		int maxi = 0;
		if (count0 > 1){
			for (int i = 0; i < count0; i++)
			{
				if (contours0[i].size() > max){
					max = contours0[i].size();
					maxi = i;
				}
			}
		}

		if (count0 > 0){
			cv::minEnclosingCircle(contours0[maxi], c0, r0);
			if (r0 >= minTargetRadius)
			{
				circle(imageL, Point(c0.x, c0.y), r0, Scalar(255, 0,
0), 2);
				center0.push_back(c0);
				ttx0 = c0.x;
				tty0 = c0.y;

			}
		}

		int maxy = 0;
		int maxyi = 0;
		if (count1 > 1){
			for (int i = 0; i < count1; i++)
			{
				if (contours1[i].size() > maxy){
					maxy = contours1[i].size();
					maxyi = i;
				}
			}
		}
		if (count1 > 0){
			cv::minEnclosingCircle(contours1[maxyi], c1, r1);
			if (r1 >= minTargetRadius)
			{
				circle(imageR, Point(c1.x, c1.y), r1, Scalar(255, 0,
0), 2);
				center1.push_back(c1);
```

```
                        ttx1 = c1.x;
                        tty1 = c1.y;
                }
        }
        //calculating disparity and distance.Then, display them on images
        if ((ttx0>0) && (ttx1>0))
        {
                //sol kamera için 48.nokta referans olarak alındı
                if (ttx0 <R1){
                        taban1 = (mesafe*(imagePointsL[48].x - ttx0) /
(imagePointsL[48].x - imagePointsL[0].x));
                        derinlik1 = sol_distance - taban1;
                }
                else
                {
                        taban1 = (mesafe*(ttx0 - imagePointsL[48].x) /
(imagePointsL[48].x - imagePointsL[0].x));
                        derinlik1 = sol_distance + taban1;
                }
                //sağ kamera için referans noktası 0 olarak alındı
                if (ttx1 <R2){
                        taban2 = (mesafe*(imagePointsR[0].x - ttx1) /
(imagePointsR[48].x - imagePointsR[0].x));
                        derinlik2 = sag_distance + taban2;
                }
                else
                {
                        taban2 = (mesafe*(ttx1 - imagePointsR[0].x) /
(imagePointsR[48].x - imagePointsR[0].x));
                        derinlik2 = sag_distance - taban2;
                }

                c = taban1;
                bool dongu = true;

                while (dongu){
                        //sol kamera (X) için
                        if (ttx0 <R1){
                                b = distance_camera - derinlik2;
                                x_test = (b*taban1) / distance_camera;
                                hataPayi = c - x_test;
                                x_gercek = sol_distance - x_test;
                                derinlik1 = sol_distance - x_test;
                                c = x_test;

                        }
                        else
                        {
                                b = distance_camera - derinlik2;
                                x_test = (b*taban1) / distance_camera;
                                hataPayi = c - x_test;
                                x_gercek = sol_distance + x_test;
                                derinlik1 = sol_distance + x_test;
                                c = x_test;
                        }

                        //sağ kamera (Y) için
                        if (ttx1 <R2){
                                a = distance_camera - derinlik1;
                                y_test = (a*taban2) / distance_camera;
                                y_gercek = sag_distance + y_test;
```

```
                        derinlik2 = sag_distance + y_test;
                }
                else
                {
                        a = distance_camera - derinlik1;
                        y_test = (a*taban2) / distance_camera;
                        y_gercek = sag_distance - y_test;
                        derinlik2 = sag_distance - y_test;
                }

                if (hataPayi < 0.1)
                {
                        t2 = clock();
                        float diff = (((float)t2 - (float)t1) /
1000000.0F) * 1000;

                        std::cout << "Milisaniye= " << diff <<"\n";
                        dongu = false;
                }
        }
        //z değeri için
        if (tty0 <R1){
                tabanz1 = (mesafe*(imagePointsL[48].y - tty0) /
(imagePointsL[48].x - imagePointsL[0].x));
        }
        else
        {
                tabanz1 = (mesafe*(tty0 - imagePointsL[48].x) /
(imagePointsL[48].x - imagePointsL[0].x));
        }
        tabanz1 = (mesafe*(imagePointsL[48].y - tty0) /
(imagePointsL[48].x - imagePointsL[0].x));
        tabanz2 = (mesafe*(imagePointsR[0].y - tty1) /
(imagePointsR[48].x - imagePointsR[0].x));

        z1 = (b*tabanz1) / distance_camera;
        z2 = (a*tabanz2) / distance_camera;
        z_gercek = (z1 + z2) / 2;
}
```

EK-7 Raspberry Pi Sunucu Kodu

```
//////////////client server için//////////////////7
#include <wiringPi.h>
#include <cstring>      // Needed for memset
#include <sys/socket.h> // Needed for the socket functions
#include <netdb.h>      // Needed for the socket functions
#include <pthread.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <unistd.h>
#include <errno.h>
////////////////////////////////////////////////
void StartServer()
{
    memset(&host_info, 0, sizeof host_info);
    std::cout << "Setting up the structs..."  << std::endl;
    host_info.ai_family = AF_UNSPEC;     // IP version not specified. Can be
both.
    host_info.ai_socktype = SOCK_STREAM; // Use SOCK_STREAM for TCP or SOCK_DGRAM
for UDP.
    host_info.ai_flags = AI_PASSIVE;     // IP Wildcard
    status = getaddrinfo(NULL, "9050", &host_info, &host_info_list);
    if (status != 0)
        std::cout << "getaddrinfo error" << gai_strerror(status) ;
    std::cout << "Creating a socket..."  << std::endl;
    socketfd = socket(host_info_list->ai_family, host_info_list->ai_socktype,
host_info_list->ai_protocol);
    if (socketfd == -1)
        std::cout << "socket error " ;

    std::cout << "Binding socket..."  << std::endl;
    status = setsockopt(socketfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int));
    status = bind(socketfd, host_info_list->ai_addr, host_info_list->ai_addrlen);
    if (status == -1)
        std::cout << "bind error" << std::endl ;
    std::cout << "Listen()ing for connections..."  << std::endl;

    status =  listen(socketfd, 5);
    if (status == -1)
        std::cout << "listen error" << std::endl ;
    socklen_t addr_size = sizeof(their_addr);
    new_sd = accept(socketfd, (struct sockaddr *)&their_addr, &addr_size);
    if (new_sd == -1)
    {
        std::cout << "listen error" << std::endl ;
    }
    else
    {
        std::cout << "Connection accepted. Using new socketfd : "  <<  new_sd <<
std::endl;
    }

    std::cout << "Waiting to receive data..."  << std::endl;
}
```

EK-8 Windows İstemci Kodu

```
#define WIN32_LEAN_AND_MEAN
#include <winsock2.h>
#include <windows.h>
#include <ws2tcpip.h>
#pragma comment(lib,"ws2_32.lib")
#define DEFAULT_BUFLEN 512
#define DEFAULT_PORT "9050"
void StartClient(){

	printf("\nInitialising Winsock...");
	if (WSAStartup(MAKEWORD(2, 2), &wsa1) != 0)
	{
		printf("Failed. Error Code : %d", WSAGetLastError());
		//	return 1;
	}

	printf("Initialised.\n");

	//Create a socket
	if ((s1 = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
	{
		printf("Could not create socket : %d", WSAGetLastError());
	}

	printf("Socket created.\n");


	server1.sin_addr.s_addr = inet_addr("192.168.43.186");
	server1.sin_family = AF_INET;
	server1.sin_port = htons(9050);

	//Connect to remote server
	if (connect(s1, (struct sockaddr *)&server1, sizeof(server1)) < 0)
	{
		puts("connect error");
		//return 1;
	}

	puts("Connected");

}
```