İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

YÜRÜYEN ROBOTLAR İÇİN BİR ÇİFT KAMERA İLE ELDE EDİLEN GÖRÜNTÜLERDEN CİSİMLERİN KONUM BİLGİLERİNİN ELDE EDİLMESİ

YÜKSEK LİSANS TEZİ

Cafer Sinan KATI

Makina Mühendisliği Anabilim Dalı

Konstrüksiyon Yüksek Lisans Programı

MAYIS 2015

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

YÜRÜYEN ROBOTLAR İÇİN BİR ÇİFT KAMERA İLE ELDE EDİLEN GÖRÜNTÜLERDEN CİSİMLERİN KONUM BİLGİLERİNİN ELDE EDİLMESİ

YÜKSEK LİSANS TEZİ

Cafer Sinan KATI
(503101204)

Makina Mühendisliği Anabilim Dalı

Konstrüksiyon Yüksek Lisans Programı

Tez Danışmanı: Prof. Dr. Hikmet KOCABAŞ

MAYIS 2015

İTÜ, Fen Bilimleri Enstitüsü'nün **503101204** numaralı Yüksek Lisans Öğrencisi **Cafer Sinan KATI**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"YÜRÜYEN ROBOTLAR İÇİN BİR ÇİFT KAMERA İLE ELDE EDİLEN GÖRÜNTÜLERDEN CİSİMLERİN KONUM BİLGİLERİNİN ELDE EDİLMESİ"** başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Prof. Dr. Hikmet KOCABAŞ** İstanbul Teknik Üniversitesi	**……..**
Jüri Üyeleri :	**Yrd. Doç. Dr. Erkan GÜNPINAR** İstanbul Teknik Üniversitesi	**……..**
	Yrd.Doç.Dr. Utku BÜYÜKŞAHİN Yıldız Teknik Üniversitesi	**……..**

Teslim Tarihi : **04 Mayıs 2015**
Savunma Tarihi : **29 Mayıs 2015**

ÖNSÖZ

Desteğini hiçbir zaman esirgemeyen danışmanım Prof. Dr. Hikmet Kocabaş'a bu tez çalışması kapsamındaki yardımları ve rehberliği için teşekkürlerimi sunarım.

Mayıs 2015

Cafer Sinan Katı
(Makine Mühendisi)

İÇİNDEKİLER

Sayfa

KISALTMALAR

2B **:** İki Boyutlu
3B **:** Üç Boyutlu
OpenCV **:** Open Source Computer Vision
MATLAB **:** Matrix Labratory
MUT **:** Mutlak Uzaklıklar Toplamı
MRF **:** Markov Random Field

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

YÜRÜYEN ROBOTLAR İÇİN STEREO KAMERA İLE ELDE EDİLEN GÖRÜNTÜLERDEN CİSİMLERİN KONUM BİLGİLERİNİN ELDE EDİLMESİ

ÖZET

Otomatik çalışan ve hareket kabiliyeti olan bir sistemde etraftaki cisimlerin algılanması kaçınılmaz bir gereksinimdir. Mesafe sensörleriyle bu bir nebze yapılabilse de robotlar gibi karar mekanizmalarına sahip olması gereken sistemler için yeterli olmamaktadır. Bu tarz sistemlerin bir lineer doğrultuda bulunan cisimlerin tespitinden çok bulunulan çevreyi görebilmeleri, bu çevredeki cisimleri algılayıp konumlarını tespit edebilmeleri gerekmektedir ki insan kontrolü olmadan hareket edebilme yeteneğine sahip olsun. Bu tarz bir gereksinim için de dijital kameralar oldukça uygundur. Ancak dijital kameralar ile alınan iki boyutlu görüntüler kendi başlarına cisimlerin konumlarının algılanmasında yeterli olmamaktadır. Konum tespitinde önemli olan derinlik bilgisinin elde edilebilmesidir. İki boyutlu görüntülerden derinlik bilgisinin elde edilebilmesi için günümüz teknolojisinde birçok çalışma yapılmış ve çeşitli yöntemler geliştirilmiştir. Genel prensip öncelikle cisimlerin kenarlarının tespit edilmesidir. Kenar tespiti görüntülerin yapısal özelliklerini bozmadan işlenilen data miktarını büyük ölçüde azaltmakta ve kullanılmayan bilgileri filtrelemektedir. Bu sebeplerden ötürü cisimlerin tespiti ve derinlik bilgisinin elde edilmesi için kenar tespiti en önemli işlemlerden birisidir. Bu önemli proses için çeşitli algoritmalar oluşturulmuştur. Sobel, Prewitt, Roberts, Canny bu algoritmalardan bazılarıdır.

Kenar belirleme algoritmaları gradyan ve laplasyan olmak üzere iki temel kategoride incelenebilir. Gradyan metodunda görüntüden gelen sinyal fonksiyonlarının birinci türevindeki minimum ve maksimum noktalarına bakarak kenar belirlemesi yapılır. Laplasyan metodunda ise fonksiyonun ikinci türevinde sıfırdan geçişe bakılarak kenar tespiti yapılır.

İnsan görme sisteminde sağ göz ve sol göz manzarayı farklı açılardan görmektedir ve bu görüntüler beyinde birleştirilerek derinlik bilgisi elde edilir. Bu tekniklerden yola çıkarak, iki kamera belirli bir aralıkta yerleştirilerek anlık olarak bir manzaranın iki farklı açıdan görüntüsü elde edilir. Oluşan görüntüler üst üste konulduğunda manzaranın aynı noktaları aynı kare içerisinde farklı konumlarda olacaktır. Bunun sebebi de iki kameranın manzarayı farklı açılardan görüntülemesidir. İşte aynı iki noktanın birbirlerine olan mesafesi tespit edilerek derinlik bilgisi hesap edilebilir.

Görüntü işlemede kullanmak amacıyla C++ programlama dilinde oluşturulmuş açık kaynak kodlu kütüphaneler mevcuttur. Bu kütüphanelerin başında da Intel Rusya tarafından geliştirilen OpenCV gelmektedir. OpenCV kütüphanesi görüntü işleme üzerine çok geniş araştırmalar sonucu optimize edilmiş kodlar içermektedir ve bunları yeni yazılımların geliştirilmesi amacıyla ücretsiz olarak insanların kullanımına

açmaktadır. OpenCV kütüphanesinin stereo görüntüleme araştırmalarına ağırlıklı olarak kullanılan temel bileşenleri CV ve HighGUI bileşenleridir. CV bileşeni stereo görüntü işlemede kullanılan tüm bilgisayarlı görüntüleme algoritmalarına sahiptir. HighGUI ise okuma, yazma, görüntüleme amacıyla ve parametre ayarlamak amacıyla kullanılan ayar çubuklarını oluşturmak üzere kullanılmaktadır. Uzaklık tespiti için C++'ta geliştirilen yazılımda OpenCV kütüphanesinden faydalanılmıştır.

Kameralardan gelen görüntüleri doğru şekilde işleyebilmek için kameraların kalibre edilmesi gerekmektedir. Pratikte, kamera referansına göre üç boyutlu koordinatları bilinen her obje kalibrasyon targeti olarak kullanılabilir. Ancak bu çalışmada, yaygın olarak kullanılan dama tahtası tercih edilmiştir. Kamera kalibrasyon metodu olarak ise Heikkila ve Silven'in geliştirdiği metot kullanılmıştır. Bu metot Bougeuet tarafından bir MATLAB yazılımı haline getirilmiştir.

Stereo görüntü çiftlerin rektifikasyonu, bu görüntülerin biçimsel bozulmasını gidererek görüntülerdeki eş piksellerin aynı çizgi üzerinde yerleşmesi sağlanır. Daha teknik ifade etmek gerekirse aşağıda görülen epipolar çizgilerin eş doğrusal olması sağlanır.

Korelasyon işleminde sağ ve sol kameradan alınan düzeltilmiş görüntüler kullanarak, bu görüntülerdeki aynı noktalar eşleştirilir. Kameradan olan uzaklığı tespit etmek için ise, görüntülerdeki aynı noktanın lokasyonunun kayması anlamına gelen "disparity" bilgisinin elde edilmesi gerekmektedir. Blok eşleştirme tekniği, korelasyonu hesaplamak ve disparity bilgisini elde etmek için kullanılan algoritma türlerinden birisidir. Bu algoritmada mutlak uzaklıkların toplamı (MUT) pencereleri kullanılarak uyumluluk bulunur. MUT pencereleri, görüntüdeki her bir piksele çevresindeki komşularını da göz önünde bulundurarak bir skorlama metodu kullanır. OpenCV'de blok eşleştirme tekniği için; ön filtreleme, uyumluluk ve son filtreleme olmak üzere üç adım uygulanır. Bu işlemler sırasında sol ve sağ görüntüler normalize edilerek aynı ışık seviyelerinin elde edilmesi sağlanır. Sağ görüntüde aranan kısımdaki en düşük skora sahip piksel en iyi eşleşme olarak öngörülür. Bu piksel ile sol görüntüdeki orijinal piksel arasındaki fark disparity değeri olarak alınır ve bu değer ile derinlik bilgisi hesaplanır

Tez kapsamında uzaklık tespiti için kurulan sistem donanımları bir çift kamera ve dizüstü bilgisayardır. Kameralar bilgisayara "Evrensel Seri Veriyolu" ile bağlanmıştır. Kameraların aynı anda çalışması ve işletilmesiyle alakalı tüm yazılım C++ dilinde Visual Studio programı yardımıyla derlenmiştir. Bu sistem ile kenar tespiti başarı ile yapılmış ve geliştirilen yazılım ile uzaklık tespiti gerçekleştirilmiştir. Ancak daha doğru ölçümler için kameraların kalibrasyonun yapılması gerekmektedir ve bu çalışma ilerleyen safhalarda yapılacaktır.

THE POSITION DETERMINATION OF THE OBJECTS FROM IMAGES TAKEN BY A PAIR OF CAMERA FOR WALKING ROBOTS

SUMMARY

In the mobile machines such as robots or vehicles, it is unavoidable that to have a detection system that give the ability of detection of objects around and prevent the crash of the machine. Especially the ones which have no human control there must also be a desicion making mechanism additional to detection systems. In these kind of systems where there is no human control the importance of the detection system increases. They must see the objects not only in a linear direction but also a wide angle so that they can move without supervision of a human. In this requirement, cameras are good solutions because they have enough angle of vision which is close to a human eye. However, cameras have a 2D vision and it is not possible to get the depth information directly. To get the depth information from a 2D vision lots of studies conducted in recent years.

The main principal in getting depth information from 2D views is firstly deceting the edges. By detecting edges it is possible to process very less data and reduce unnecessary information without deforming the structural properties of the image. Therefore it is one of the most important steps to detect objects and get depth information. There are many ways to perform edge detection, however, the majority of the different methods may be grouped by “gradient based edge detection” and “laplacian based edge detection”. In the gradient method, the edge detection is done by looking for the minimum and maximum points of the first derivative of the signal coming from the image. In the laplacian method, it is looked for the zero crossing in the second derivative of the function. In the human vision system, the left and right eye see the view from different angles, and these images are somehow processed in the brain which lets us to get the depth information. By using two cameras which are fixed on the same line and have a constant distance between them, it is possible to simulate human vision. The problem is to simulate what the brain does. This thesis aim to solve this problem by developing a source code which can process the images and get the depth information. To achieve this, firstly the images are converted to mathematical model, than the images are put in to the same frame. The distance between the same pixels on different images is detected. Than, by using the stereo vision geometry, the deth information is calculated.

In this study, C++ programming language is used to process the images. Also there are some libraries which are specially developed for image processing, in C++. OpenCV is one of the most popular open source library which is developed by Intel Russia. OpenCV is a library which is develop by comprehensive studies and includes optimized codes for image processing and these codes are free and open for everyone to develop new software for image processing. CV and HighGUI are the main

components of OpenCV which are widely used. CV has all computer vision algortithms which are used in image processing. HighGUI is used for reading, writing and screening and also to create slide bars to set parameters.

To process the images correctly, the cameras have to be calibrated. Practically, any object can be used as a calibration target as long as the coordinates of the target are known in reference to the camera. However in this study chessboard is used which is one of the most popular one.

After getting the images from calibrated cameras, rectification process is done. The rectification of the image pairs is putting the corresponding pixels in the same line. More formally, epipolar lines becomes collinear afte rectification. Rectification is commonly performed to improve the speed of the search for stereo correspondences. Given two cameras where the intrinsic parameters are known, the images undistorted, and the image planes have been rectified, the cameras can be calibrated so that the rotation and translation of one camera is known with respect to the other. These relations are found in the rotation matrix and the translation vector. The rotation matrix contains parameters that rotate the left camera so that its image plane is mathematically coplanar with the right camera. The translation vector relates the positioning, or offset, of the left camera with respect to the right camera.

The correlation process, in which the depth information is calculated, follows the rectification. In this process, rectified images are used to match the points from one image to the other. In order to determine the distance from the camera, the disparity needs to be found which is the change in location of points in the left image to the right. It follows that there must be an overlap in the two images so that a point in the left image also exist in the right image and a correspondence can be found. An algorithm that can be used to compute correlation and find disparities is a block matching technique where sum of absolute distances (SAD) windows are used to find correspondences. SAD windows are used as a scoring method for each pixel in the image based on its surrounding neighbors. There are three steps to the block matching technique that OpenCV uses; prefiltering, correspondence search, and post filtering. In prefiltering step, the left and right images are normalized by a filter. In the correspondence search step, the matcing points in the image pairs are found. The point in the right image within the search area with the lowest score is considered the best match for the point in the left image. The offset of this point from the original point in the left image is taken as the disparity for that correspondence and from that information the depth is computed. Points with large disparity values represent points that are closer to the camera and smaller disparities represent points that are farther away from the camera. Post filtering is done to remove correspondences that are considered false matches. For this, OpenCV uses a uniqueness ratio as well as a texture threshold. The uniqueness ratio is used to make sure that the value that was calculated for the matched point is not just the closest score, but is an outlier score where it is surrounded by scores that are far from being a match. The texture threshold is set so that noise can be reduced during the matching process, not score that is below the texture threshold is considered.

In this study, in order to determine the distance, a laptop and two cameras are used as hardwares. Cameras are connected to laptop by USB ports. The operation of two cameras at the same time and all image processing codes are created in Visual Studio and C++ programming language. The code accepts two images and a calibration data file as arguments. The calibration file was created using the Matlab toolkit during the calibration phase. The application is set up to process the images automatically and then display the distance. A slide bar window was created so that various parameters dealing with the correlation process can be adjusted. With the code developed distance determination is done however the calibration of the camera will be improved to get better result.

1. GİRİŞ

İki boyutlu görüntülerden derinlik bilgisi elde edilmesi 1838 yılında Sir Charles Wheatstone'un stereoskopu bulmasıyla başladı [1]. Streoskop bir cismin veya manzaranın, yatayda kaydırılan bir kamera ile elde edilen iki adet görüntüsünden üç boyutlu bir görünüş çıkartan optik alettir. Wheatstone'un aynalı stereoskopundan sonra aynı temelde fakat farklı tekniklerde daha basit stereoskoplar icat edilmiştir. Şekil 1.1 Holmes'un icat ettiği stereoskop gözükmektedir. Stereoskopik iki boyutlu görüntülerden, çıplak gözle dahi, gözler çaprazlaştırılarak üç boyutlu görüntü elde etmek mümkündür. Bu şekilde baktığınız stereoskopik görüntülerin ortasında üçüncü bir görüntü belirir ve her iki göz de farklı görüntü görmeye başlar böylece derinlik bilgisi oluşur.

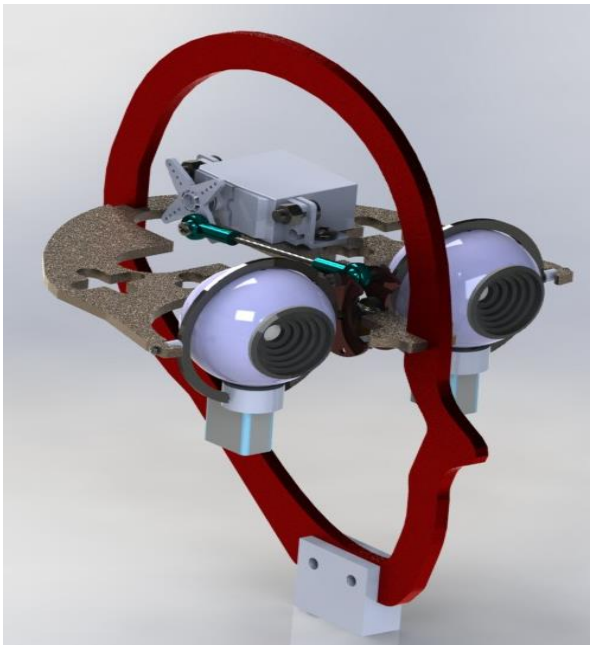
Şekil 1.1 Holmes'un stereoskobu

1.1 Tezin Amacı

Bu çalışmada, yukarıda bahsedilen basit temele dayanarak, bir çift dijital kamera kullanılarak yürüyen robotların, cisimlerin konumlarını tespit etmeleri sağlayacak, C++ dilinde bir yazılım geliştirilmesi amaçlanmaktadır.

1.2 Literatür Araştırması

Otomatik çalışan ve hareket kabiliyeti olan bir sistemde etraftaki cisimlerin algılanması kaçınılmaz bir gereksinimdir. Mesafe sensörleriyle bu bir nebze yapılabilse de robotlar gibi karar mekanizmalarına sahip olması gereken sistemler için yeterli olmamaktadır. Bu tarz sistemlerin bir lineer doğrultuda bulunan cisimlerin tespitinden çok bulunulan çevreyi görebilmeleri, bu çevredeki cisimleri algılayıp konumlarını tespit edebilmeleri gerekmektedir ki insan kontrolü olmadan hareket edebilme yeteneğine sahip olsun. Bu tarz bir gereksinim için de dijital kameralar oldukça uygundur. Şekil 1.2 de stereo kamera kullanılarak yapılmış bir robot kafasının konstrüksiyonu görülmektedir.



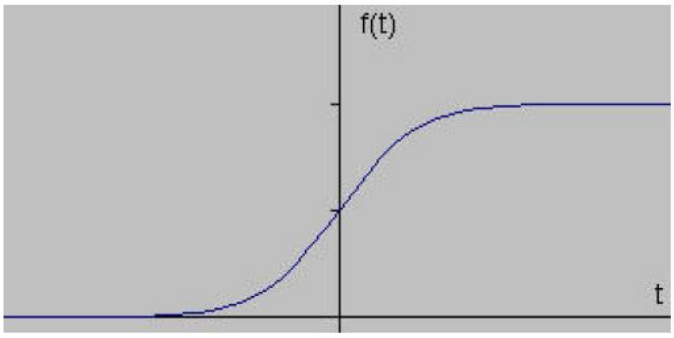
Şekil 1.2 Kamera yerleştirilmiş robot kafası konstrüksiyonu

Ancak dijital kameralar ile alınan iki boyutlu görüntüler kendi başlarına cisimlerin konumlarının algılanmasında yeterli olmamaktadır. Konum tespitinde önemli olan derinlik bilgisinin elde edilebilmesidir. İki boyutlu görüntülerden derinlik bilgisinin elde edilebilmesi için yukarıda bahsedilen stereoskopun dışında günümüz teknolojisinde ilerleyen kısımlarda bir kısmından bahsedilen birçok çalışma yapılmış ve çeşitli yöntemler geliştirilmiştir. Genel prensip öncelikle cisimlerin kenarlarının tespit edilmesidir. Kenar tespiti görüntülerin yapısal özelliklerini bozmadan işlenilen data miktarını büyük ölçüde azaltmakta ve kullanılmayan bilgileri filtrelemektedir. Bu sebeplerden ötürü cisimlerin tespiti ve derinlik bilgisinin elde edilmesi için kenar

tespiti en önemli işlemlerden birisidir. Bu önemli proses için çeşitli algoritmalar oluşturulmuştur. Sobel, Prewitt, Roberts, Canny bu algoritmalardan bazılarıdır.

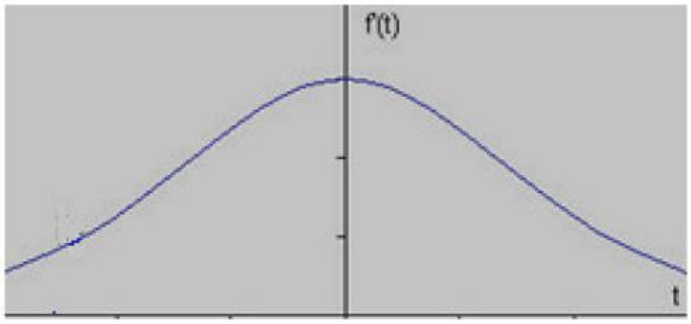
Kenar belirleme algoritmaları gradyan ve Laplasyan olmak üzere iki temel kategoride incelenebilir kategoride incelenebilir. Gradyan metodunda görüntüden gelen sinyal fonksiyonlarının birinci türevindeki minimum ve maksimum noktalarına bakarak kenar belirlemesi yapılır. Laplasyan metodunda ise fonksiyonun ikinci türevinde sıfırdan geçişe bakılarak kenar tespiti yapılır.

Yoğunluktaki artışla gösterilen bir kenar ile Şekil 1.3 ’de gösterilen sinyale sahip olduğumuzu varsayalım. Bu sinyalin gradyanını alırsak (tek boyutta fonksiyonun ”t”ye göre birinci türevi) Şekil 1.4 ’de gösterilen fonksiyonu elde ederiz.



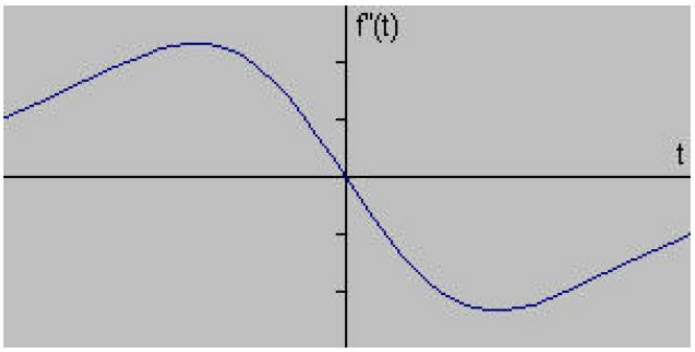
Şekil 1.3 Yoğunluktaki artışı gösteren bir kenar sinyali[2]

Fonksiyonun türevi orjinal sinyalde kenarın merkezinde açık şekilde bir maksimum noktası göstermektedir. Bu şekilde tespit edilen gradyan değeri belirli bir eşik değerini aştığı zaman, pikselin bulunduğu konum kenar noktası olarak tanımlanır. Dolayısıyla bir eşik değeri belirlendiğinde gradyan değerini eşik değerine kıyaslayarak kenar tespiti yapılabilir.



Şekil 1.4 Şekil 1.3 ’de beliritlen sinyalin gradyanı[2]

Ayrıca birinci türev bir maksimum belirttiğinde, ikinci türev de sıfırdır. Sonuç olarak, ikinci türevde sıfırları tespit etmek de Laplasyan metodu denilen diğer bir yöntemdir. Yukarıda bahsedilen foknsiyonun ikinci türevi Şekil 1.5 ’te gösterilmiştir.[2]



Şekil 1.5 Şekil 1.3 ’de belirtilen sinyalin ikinci türevi[2]

”Sobel Kenar Belirleme Algoritması” gradyan kategorisindeki en etkili kenar belirleme algoritmalarından birisidir. Sobel operatörü Bu algoritmada Şekil 1.6 ’de görülen, bir çift 3x3 lük evrişim çekirdeği (convolution kernel) kullanılmaktadır.[2]

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

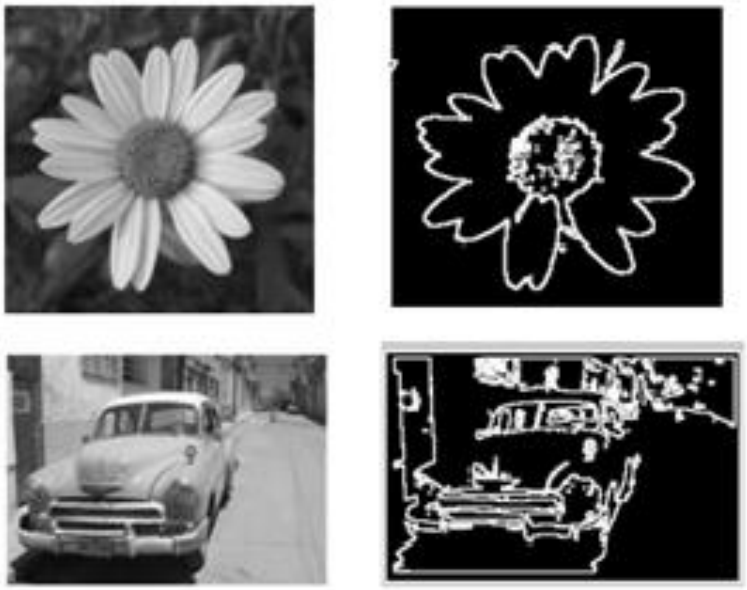
Şekil 1.6 Sobel algoritmasında kullanılan operatörler

Bu operatörler aydınlıktan karanlığa geçişteki mümkün olan en büyük artışın yönünü vererek ve de değişimdeki oranı belirleyerek, görüntünün her bir noktasındaki ışık yoğunluğunun gradyanını hesaplar. Geniş operatör kullanmak lokal ortalama almalar sebebiyle gürültüden kaynaklı hataları azaltır.

Sobel kenar belirleme algoritması için psödö kodlar aşağıdaki gibidir:

Girdi: Bir adet görüntü
Çıktı: Tespit edilen kenarlar
Adım 1: Görüntü girdisini kabul et
Adım 2: Görüntü girdisine Gx,Gy maskelerini uygula
Adım 3: Sobek kenar belirleme algoritmasını ve gradyanı uygula
Adım 4: Görüntü girdisinde ayrı ayrı Gx,Gy maske manipülasyonlarını yap
Adım 5: Mutlak gradyan büyüklüğünü bulmak için sonuçlar birleştirilir
Adım 6: Mutlak büyüklük kenar çıktısıdır [3]

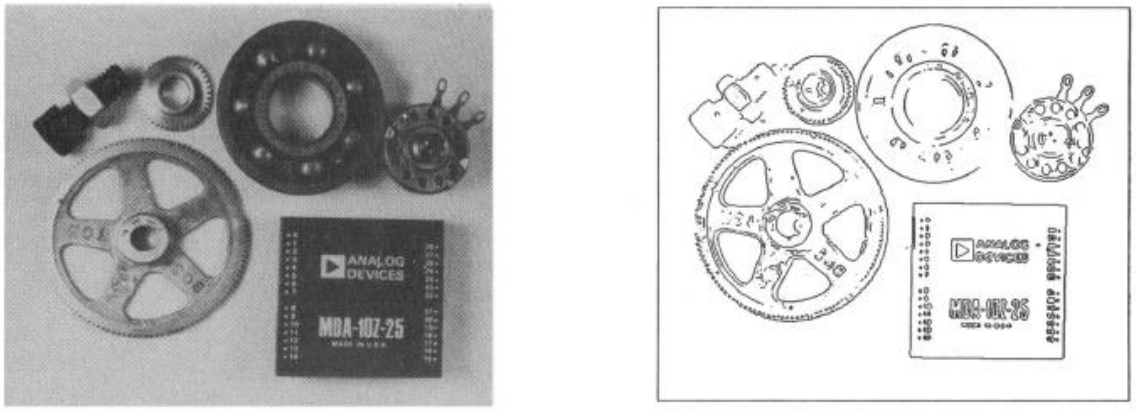
Şekil 1.7 ’de Sobel kenar belirleme algoritması kullanılarak işlenmiş fotoğraflar gözükmektedir. [4]



Şekil 1.7 Model yapıları[4]

Kenar belirleme için kullanılan diğer bir yaygın algoritma da “Canny Kenar Belirleme Algoritması”dır. Bu algoritma farklı şartlardaki kenar tespit kabiliyetinden dolayı birçok kişiye göre en iyi kenar belirleme algoritmasıdır. [2] John Canny tarafından geliştirilen bu algoritmada, kenarların belirlenme ve lokalize edilmesi kirterleri tanımlanmış ve bu kriterler için matematik modeller oluşturulmuştur. Ayrıca belirleyicinin bir kenardan yalnızca bir adet yanıt geldiğinden emin olmak için üçüncü bir kriter eklenmiştir. [5] Canny algoritmasını uygulamak için bazı adımları takip etmek gerekiyor. Birinci adımda, görüntüdeki tüm gürültünün filtrelenmesi gerekmektedir. İkinci adımda ise görüntünün gradyanını alarak kenar şiddetini belirlemek gerekmektedir. Sobel operatörü Şekil 1.6 ‘de gösterilen matrisleri kullanarak görüntü üzerinde iki boyutlu gradyan ölçümü gerçekleştirilebilmektedir. Böylelikle her noktadaki yaklaşık gradyan büyüklüğü (kenar şiddeti) bulunabilir. Üçüncü adımda x ve y yönlerindeki gradyanlar kullanılarak kenarın yönü hesaplanmaktadır. X yönündeki gradyan sıfıra eşit olduğunda y yönndeki gradyana bağlı olarak, kenar yönü 90 dereceye veya 0 dereceye eşit olmalıdır. Eğer GY 0 değerine sahipse kenar yönü 0 derecedir, aksi halde ise kenar yönü 90 derecedir. Son olarak histerisis uygulanarak kenar hatlarındaki eşik değeri etrafındaki çıktılarda yaşanan dalgalanmalardan ötürü oluşan kırılmaların giderilmesi sağlanır.[6] Canny kenar belirleme algoritması kullanarak, webcamden elde edilen bir video

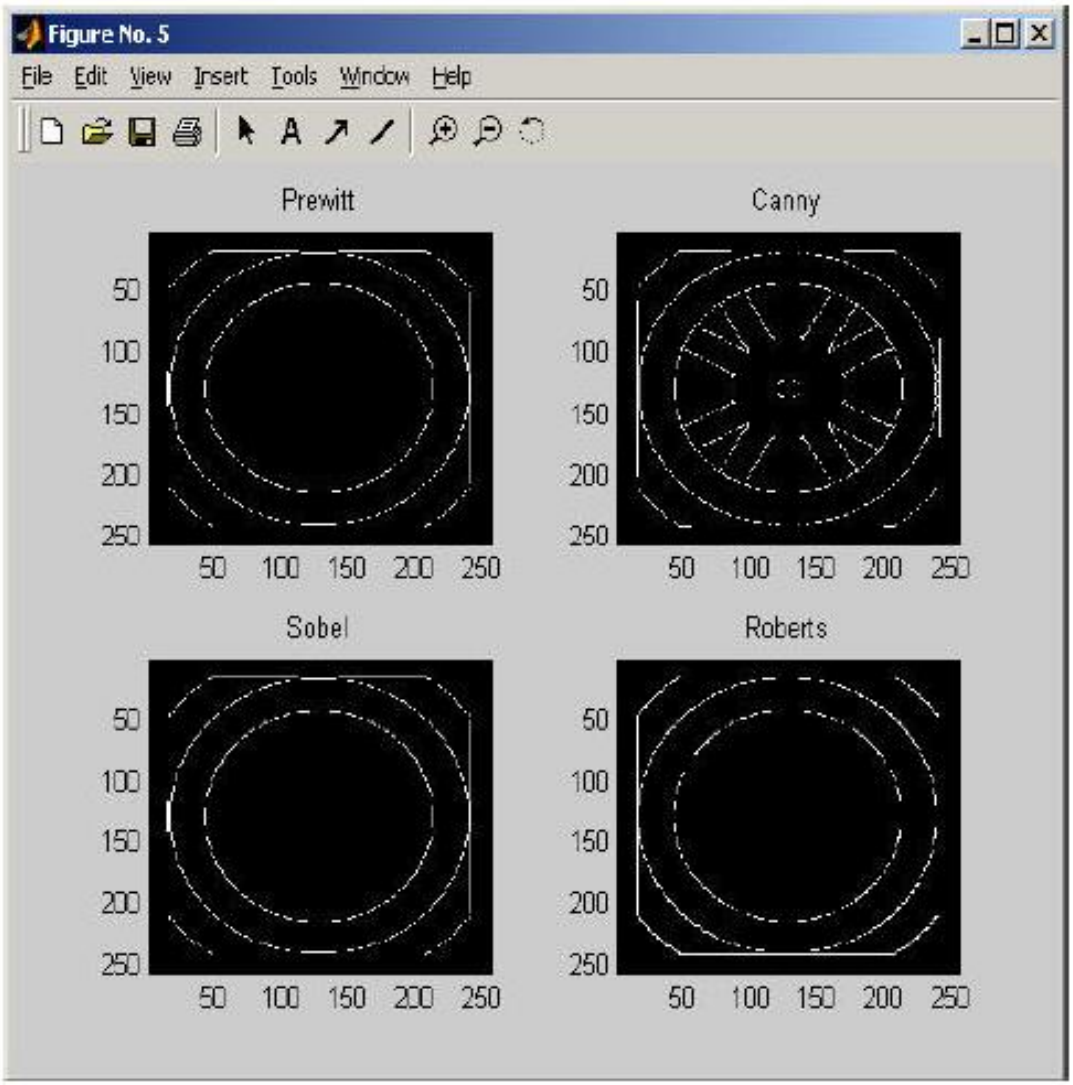
görüntüsünden anlık olarak kenar belirleme yapmak için C++'da yazılmış bir kod EK A'da verilmiştir. Canny algoritması ile işlenmiş bir görüntü Şekil 1.8 'de verilmiştir [4]



Şekil 1.8 Canny kenar belirleme algoritması ile işlenmiş bir görüntü[4]

Şekil 1.9 'de gösterilen görüntünün dört farklı kenar belirleme algoritması uygulanarak elde edilen işlenmiş görüntüler Şekil 1.10 'da gösterilmiştir.[2]

Şekil 1.9 Orijinal görüntü[2]



Şekil 1.10 Şekil 1.9 'deki görüntüye uygulanan 4 farklı kenar belirleme algoritmasının sonuçları[2]

Kenar belirleme algoritmalarının dışında stereo kamera ile derinlik tespiti amacıyla yapılmış çalışmalar mevcuttur. Nathaniel Short tarafından yapılan bir çalışmada stereo kamera ile 3B nokta bulutu oluşturmak hedeflenmiştir. Şekil 1.11 de gösterildiği üzere helikoptere bağlanan stereo kamera ile yeryüzü görüntüleri alınmış ve 3B nokta bulutu oluşturulmuştur. Bu çalışmanın temel prensibine baktığımız zaman, stereo kamera ile bir çift görüntü elde edilmekte ve bu görüntüler işlenerek disparity değeri hesaplanmakadır. Stereo kamera sisteminin özellileri ve disparity değeri bilindiği için derinlik bilgisi geometrik hesaplamalarla elde edilmiştir. [7]

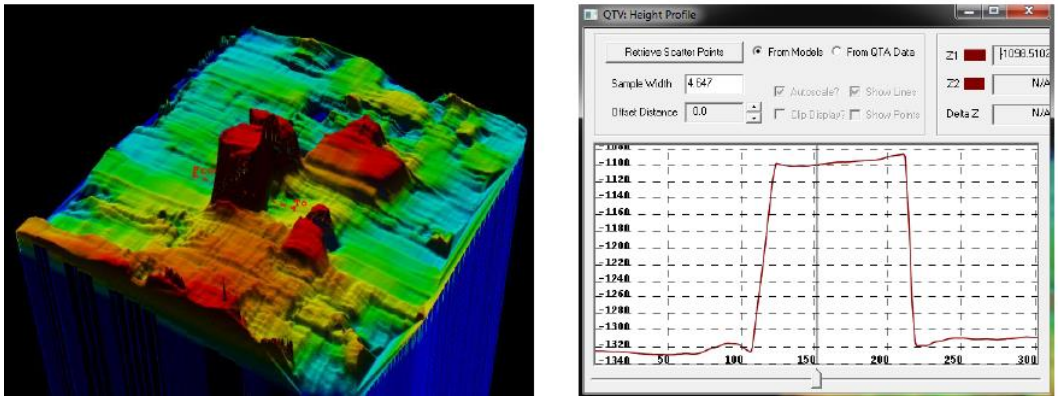
Şekil 1.11 Stereo kamera ile 3B nokta bulutu tespiti[7]

Bu çalışmada kullanılan kamera sistemi de Şekil 1.12 de gösterilmiştir.

Şekil 1.12 2Stereo kamera sistemi[7]

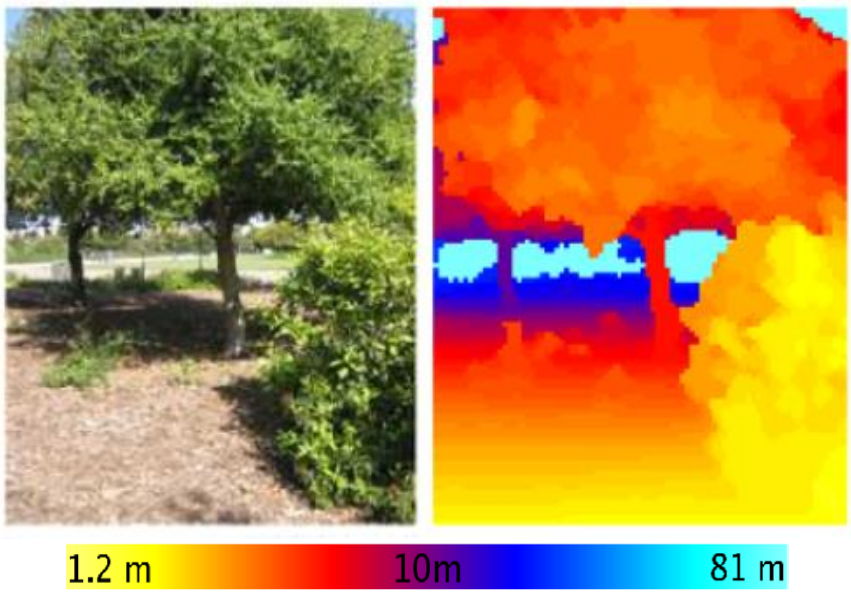
Şekil 1.13 de, çalışma kapsamında oluşturulan stereo sistem ile elde edilmiş nokta bulutundan oluşturulmuş 3B model yer almaktadır. Yükseklik profilinde soldaki değerler kamera sisteminden olan uzaklığı ifade etmektedir. Kadrajdaki en yüksek

objeden alınan yükseklik profili incelendiğinde, tespit edilen uzaklığın gerçek uzaklığa %13,22'lik bir hata payı ile yaklaştığı görülmektedir.



Şekil 1.13 Görüntünün 3B modeli ve yükseklik profili[7]

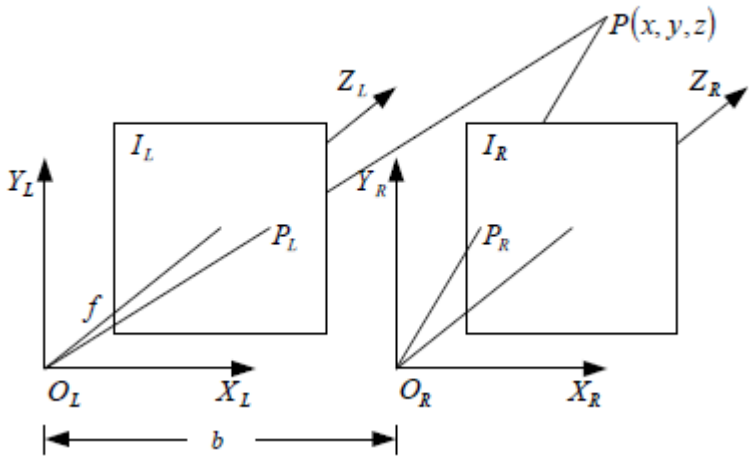
Stereo kamera ile derinlik tespiti dışında tek bir kamera ile derinlik tespiti de Saxena, Chung ve Andrew'ın yaptığı bir çalışma ile hedeflenmiştir. Bu çalışmada derinlik hesabı için stereo görüntülemedeki disparity hesabının aksine görüntüdeki global doku değişimleri, odaksızlık ve renk gibi bilgileri gibi global özellikler kullanılmaktadır. Ayrıca stereo sistemlerde kullanılan blok eşleştirme yöntemi ve MUT pencerelerinden farklı olarak görüntüdeki lokal ve global özellikleri birleştiren hiyerarşik, çok sklalalı "Markov Random Field" (MRF) modeli kullanılmıştır. Çalışmada tek bir görüntü ile kabul edilebilir düzeyde doğruluk olanına sahip derinlik bilgisinin elde edilebildiği gösterilmiştir. Çalışma sonucunda elde edilmiş bir derinlik haritası Şekil 1.14 'de gösterilmiştir. [8]



Şekil 1.14 Tek bir görüntüden elde edilen derinlik haritası

1.3 Hipotez

Yukarıda bahsedilen stereoskopik görüntü oluşturmada kullanılan teknik, yatayda kaydırılan bir kamera kullanılarak bir objenin iki farklı açıdan fotoğraflarını çekip stereoskop ile bu foroğrafları görüntüleyerek derinlik elde edilmişti. İnsan görme sisteminde sağ göz ve sol göz manzarayı farklı açılardan görmektedir ve bu görüntüler beyinde birleştirilerek derinlik bilgisi elde edilir. Bu tekniklerden yola çıkarak, kamera kaydırmak yerine Şekil 1.15 ‘da gösterildiği gibi iki kamera belirli bir aralıkta yerleştirilerek anlık olarak bir manzarının iki farklı açıdan görüntüsü elde edilir. Bu sisteme baktığımızda “f” odak uzaklığındaki iki kamera birbirine “b” uzaklığında yerleştirilmiştir. Oluşan görüntüler üst üste konulduğunda manzaranın aynı noktaları aynı kare içerisinde farklı konumlarda olacaktır. Bunun sebebi de iki kameranın manzarayı farklı açılaradan görüntülemesidir. İşte aynı iki noktanın biribirlerine olan mesafesini “d” (disparity) tespit edebilirsek elimizdeki “f” ve “b” bilgileriyle derinlik bilgisi hesap edilebilir.



Şekil 1.15 Basit stereo görüntüleme sistemi

Şekil 1.15 ‘da belirtilen P noktasının koordinat bilgileri benzer üçgenler dikkate alındığında aşağıdaki gibi ifade edilebilir:

$$x = \frac{bx_L}{d},\ y = \frac{by_L}{d},\ z = \frac{bf}{d} \qquad (1.1)$$

$$Derinlik = \frac{Kameralar\ arası\ uzaklık\ x\ Odak\ Uzaklığı}{Disparity} \qquad (1.2)$$

2. GÖRÜNTÜ İŞLEME

Kameralar vasıtasıyla elde edilen sinyaller matematiksel verilere dönüştürülür. Çağımızın bilgisayar teknolojisini kullanarak da bu matematiksel verileri çeşitli işlemlerden geçirerek görüntüler üzerinde değişiklik yapılabilir.

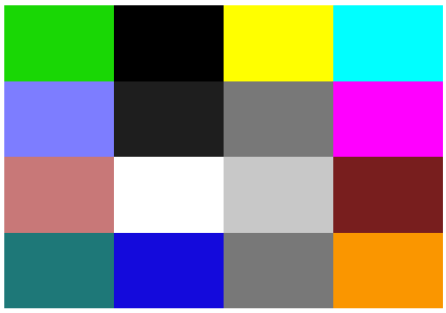
2.1 Dijital Görüntü

Dijital görüntüler kameralar vasıtasıyla elde edilen sinyallerin matematiksel verilere dönüştürülmesi ile oluşturulur. Şekil 2.1 ‘ de bir görüntünün dijital olarak nasıl ifade ediliği gösterilmektedir. [9] Görüntüdeki her bir piksel için bir sayısal değer mevcuttur ve bu değer pikselin şiddetini ifade eder.

Şekil 2.1 Dijital görüntü[9]

Dijital görüntüler de renk kırmızı (R), yeşil (G), mavi (B)’nin birleşimi olarak belirlenir. Şekil 2.2 ‘de Mathcad’te oluşturulmuş bir dijital görüntü görülmektedir. Görüldüğü gibi her bir renk için 4x4’lük bir matris oluşturulmuştur ve her piksel için bir değer girilmiştir. Örneğin RGB(1,1) pikselinde her renk için 255 değeri olduğu için görüntü beyaz RGB(1,2) pikseli için ise her renk değeri için 0 girildiği için siyah renk elde edilmiştir.

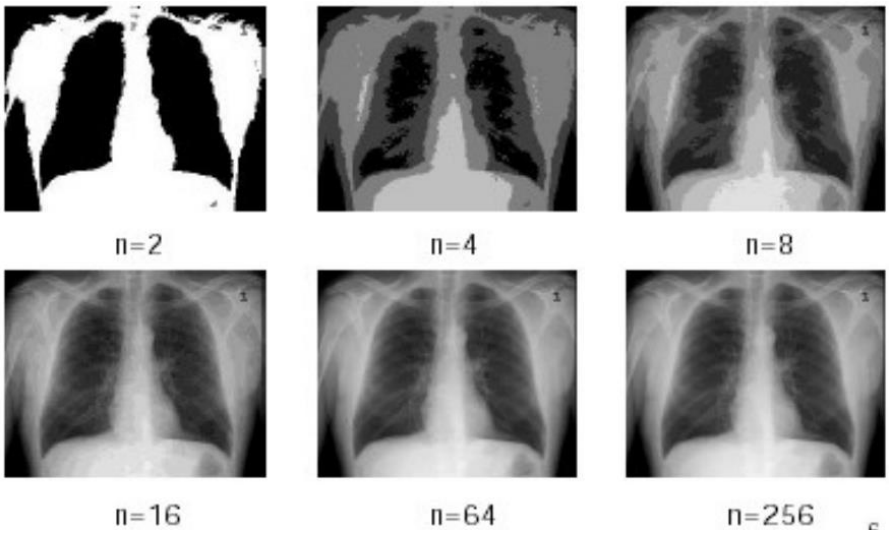
$$R := \begin{pmatrix} 25 & 0 & 255 & 0 \\ 125 & 30 & 120 & 255 \\ 200 & 255 & 200 & 120 \\ 30 & 20 & 120 & 250 \end{pmatrix} \quad G := \begin{pmatrix} 215 & 0 & 255 & 255 \\ 125 & 30 & 120 & 0 \\ 120 & 255 & 200 & 30 \\ 120 & 10 & 120 & 150 \end{pmatrix} \quad B := \begin{pmatrix} 5 & 0 & 0 & 255 \\ 255 & 30 & 120 & 255 \\ 120 & 255 & 200 & 30 \\ 120 & 220 & 120 & 0 \end{pmatrix}$$



Şekil 2.2 Mathcad'te oluşturulmuş 8 bit değerinde 4x4 çözünürlüğünde bir görüntü

2.1.1 Bit değeri

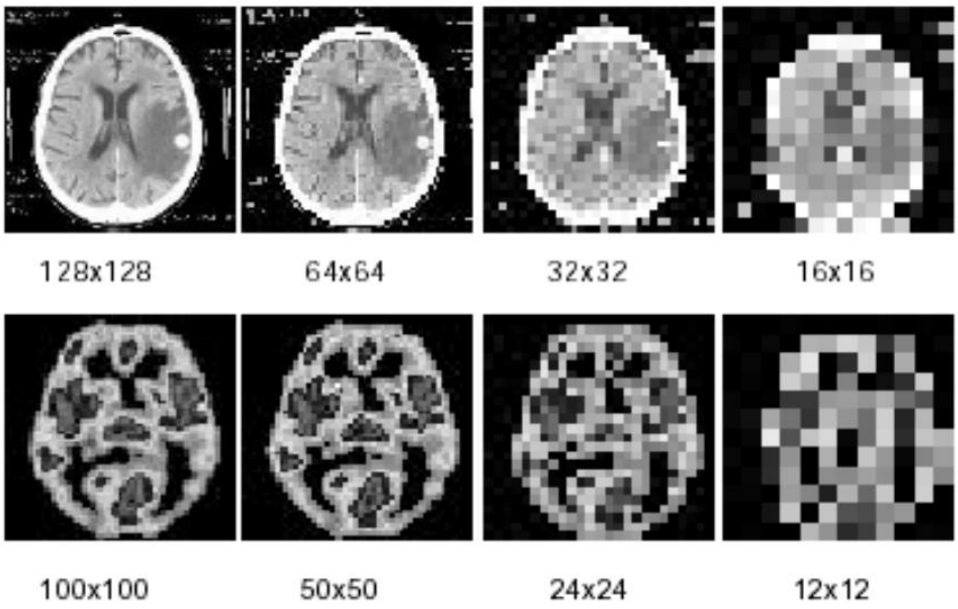
Piksellerin içerdiği şiddet değerleri görüntünün niteliğini belirler. Dijital bir görüntüde piksellerin değer aralıkları "bit" olarak ifade edilir ve $n=2^b$ formülü ile belirlenir. Örneğin 8 bitlik bir görüntüde (b=8) piksel değer aralığı 256'dır. Piksel değer aralığı görüntünün niteliğini belirler. Bir piksel kırmızı, yeşil ve mavi renkler için aynı değere sahipse değerin büyüklüğüne bağlı olarak grinin tonlarını verir. Örneğin üç renk için de piksel 255 değerine sahipse beyaz, 0 değerine sahipse siyah renge sahiptir diyebiliriz. Şekil 2.3 bit değerindeki değişime bağlı olarak farklı piksel değer aralıklarında görüntüdeki değişim ifade edilmiştir.



Şekil 2.3 Aynı görüntünün farklı bit değerlerindeki gösterimi[10]

2.1.2 Çözünürlük

Çözünürlük dijital bir görüntüdeki piksel sayısını ifade eder ve yataydaki ve düşeydeki piksel sayılarının çarpılması ile ifade edilir. Şekil 2.4 iki farklı görüntünün çözünürlük değerine bağlı olarak değişimi gösterilmiştir.



Şekil 2.4 Aynı görüntünün farklı çözünürlük değerlerindeki gösterimi[10]

2.2 Görüntü İşleme Teknikleri

Stereo görüntülemede kullanılan birçok görüntü işleme tekniği mevcuttur. Bu tekniklerden bazıları aşağıda ele alınmıştır.

2.2.1 Gri skala transformasyonu

Gri skaladaki bir dijital görüntüde pikseller yalnızca yoğunluk ve konum bilgisi içerir. Bu görüntüler aynı zamanda siyah-beyaz olarak da anılır ve grinin tonlarını içerir. En düşük piksel değeri siyahı en yüksek piksel değeri ise beyazı ifade eder. [11] Renkli görüntüler basit bir tansformasyonla gri skalaya dönüştürülebilir. Renkli bir görüntünün 8 bitlik bir gri skalaya dönüştürülmesi ile görüntü sadeleştirilmiş olur ve yapılacak diğer işlemlerin daha hızlı gerçekleşmesi sağlanır. Bu sebeple görüntü işlemedeki ilk basamaklardan birisidir. [12]

2.2.2 Düzleştirme filtreleri

Düzleştirme filtrelerinin amacı görüntülerdeki gürültüyü azaltmaktır ve genellikle görüntü işlemedeki ilk adımlardan birisidir. Bu filtereler ortalama alma filtreleri olarak

da adlandırılır çünkü bu filtrelerin çıktısı etki ettiği alandaki piksellerin ortalamasıdır. Düzleştirme filtrelerinin temel mantığı, filtrenin etki ettiği alandaki tüm piksellerin değerlerinin, ortalamaları ile yer değiştirmesidir. [13] Filtreni matris boyutu kadarlık bir alana etki eden filtre görüntü boyunca kayar ve etki ettiği alandaki piksel değerlerini yine bu alandaki ortalama değer ile değiştirir. [14] Şekil 2.5 örnek düzleştirme filtreleri verilmiştir.

$$\frac{1}{9}\times\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} \qquad \frac{1}{16}\times\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$$

Şekil 2.5 3x3'lük düzleştirme filtreleri

2.2.3 Eşik değeri belirleme

Eşik değeri belirleme görüntü işlemede en çok kullanılan tekniklerden birisidir. Bu operasyon ile belirli piksel değer aralığında bulunan veya belirli bir değere sahip olan pikseller seçilebilir. Böylelikle arka plan ile şekiller ayrılır.[15] Şekil 2.6 'da eşik değerinin kenar belirleme algoritması uygulanmış bir görüntü üzerindeki etkileri incelenmiştir.

Şekil 2.6 Eşik değerinin görüntü üzerindeki etkisi

Eşik değeri belirleme metodları kulladıkları yönteme göre 6 kategoride incelenebilir:

- Histogram şekli temelli metodlar; düzleştirilmiş histogramların tepe noktaların ve eğriliklerinin analiz edildiği metodlardır

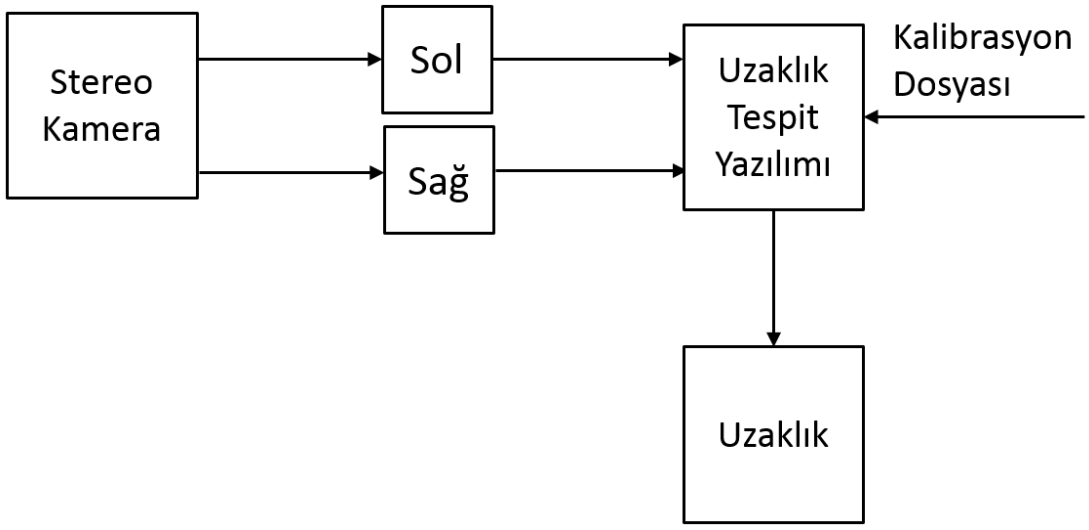
- Gruplaşma temelli metodlarda gri skaladaki görüntüler arkaplan ve önplan olmak üzere iki kategoride gruplandırılır veya alternatif olarak iki Gausyanın karışımı olarak modellenebilir
- Entropi temelli metodlar arkaplan ve önplan hatlarını, orjinal ve binarize edilmiş görüntüler arasındaki entropi geçişlerini kullanan algoirtmalara dayanır
- Obje niteliklerine dayalı metodlar gri skaladaki ve binarize edilmiş görüntüler arasındaki bulanık şekiller, kenar kesişimleri gibi özelliklerin benzerlik değerlerini araştırır
- Uzaysal metodlar yüksek dereceli olasılık dağılımlarını ve/veya pikseller arasındaki uyumu kullanır
- Lokal metodlar, eşik değerini lokal görüntü karakteristiklerine göre adapte eder. [16]

2.2.4 Kenar tespiti

Kenar tespiti stereo görüntülemede derinlik bilgisi elde edilmesi için kaçınılmaz bir operasyondur. Kenar tespiti görüntülerin yapısal özelliklerini bozmadan işlenilen data miktarını büyük ölçüde azaltmakta ve kullanılmayan bilgileri filtrelemektedir. 1. Bölüm'de literatür araştırması altında konu ile ilgili detaylı bilgi verilmiş ve EK A'da Canny kenar belirleme algoritması verilmiştir. Şekil 2.6 'da ise bu algoritma ile işlenmiş bir görüntü verilmiştir.

3. UZAKLIK TESPİTİ

Tez konusu kapsamında stereo kameradan elde edilen görüntüleri işlemek için C++ programlama dili kullanılmıştır. Görüntü işlemede kullanmak amacıyla C++ programlama dilinde oluşturulmuş açık kaynak kodlu C++ kütüphaneleri mevcuttur. Bu kütüphanelerin başında da Intel Rusya tarafından geliştirilen OpenCV gelmektedir. OpenCV kütüphanesi görüntü işleme üzerine çok geniş araştırmalar sonucu optimize edilmiş kodlar içermektedir ve bunları yeni yazılımların geliştirilmesi amacıyla ücretsiz olarak insanların kullanımına açmaktadır. OpenCV kütüphanesinin stereo görüntüleme araştırmalarına ağırlıklı olarak kullanılan temel bileşenleri CV ve HighGUI bileşenleridir. CV bileşeni stereo görüntü işlemede kullanılan tüm bilgisayarlı görüntüleme algoritmalarına sahiptir. HighGUI ise okuma, yazma, görüntüleme amacıyla ve parametre ayarlamak amacıyla kullanılan ayar çubuklarını oluşturmak üzere kullanılmaktadır. Uzaklık tespiti için C++'ta geliştirilen yazılımda OpenCV kütüphanesinden faydalanılmıştır. Yazılım iki adet webcamden gelen sol ve sağ görüntüyü almakta ve geliştirilen algoritmayı uygulayarak uzaklık tespiti yapmaktadır. Yazılımın temel olarak iş akışı Şekil 3.1 'de verilmiştir.

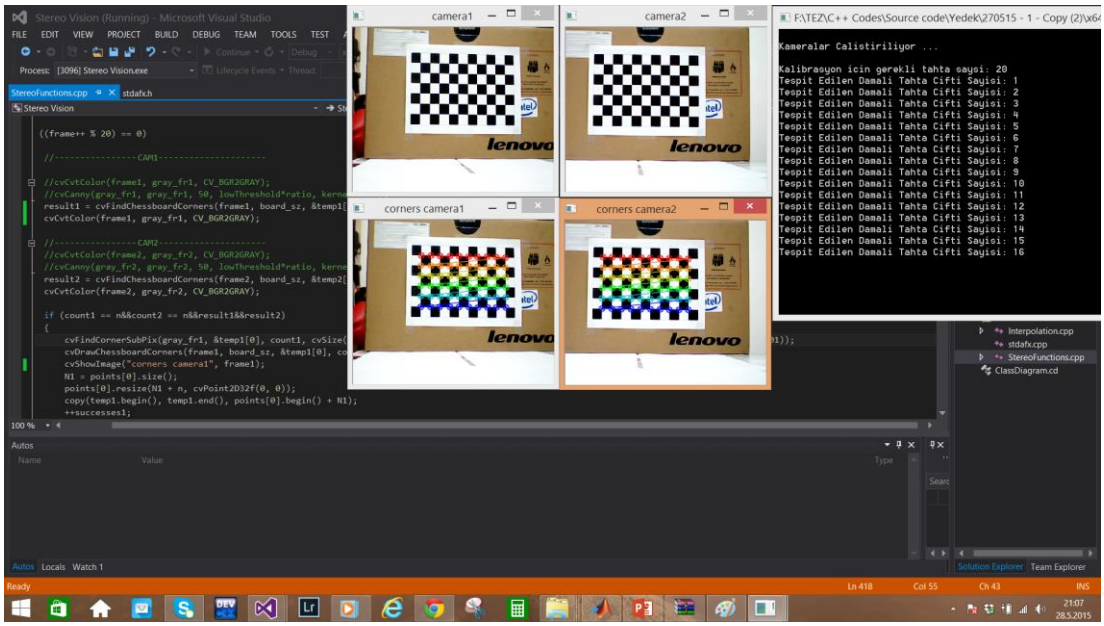


Şekil 3.1 Uzaklık tespiti işleyiş şematiği

3.1 Uzaklık Tespiti Algoritması

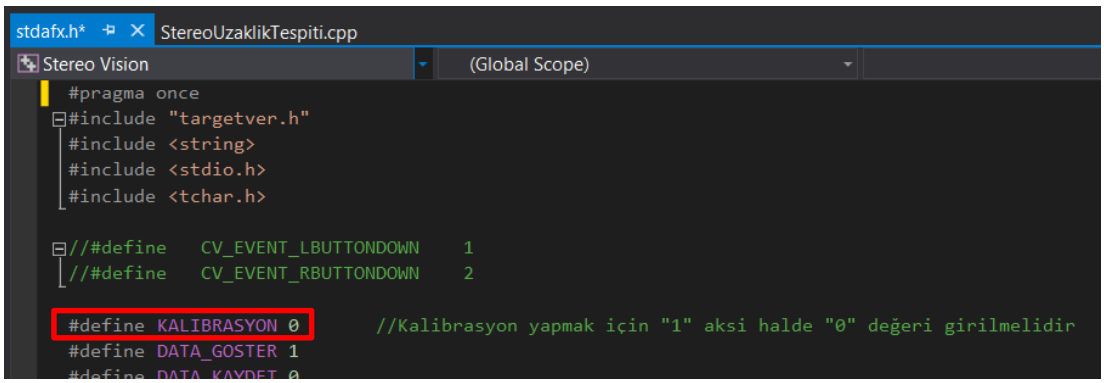
3.1.1 Kalibrasyon

Kameralardan gelen görüntüleri doğru şekilde işleyebilmek için kameraların kalibre edilmesi gerekmektedir. Pratikte, kamera referansına göre üç oyutlu koordinatları bilinen her obje kalibrasyon targeti olarak kullanılabilir. Geçmişte 3 boyutlu ve tek boyutlu kalibrasyon yöntemleri geliştirilmiş ancak bu yöntemlerde perspektiften kaynaklı problemler çıkmıştır. Bu metodlara göre daha doğru sonuçlar veren ve basit şekilde uygulanabilen yöntem olan dama tahtası ile kalibrasyon günümüzde daha çok tercih edilmektedir. [17] Heikkila ve Silven'in geliştirdiği bu method [18] Bougeuet tarafından bir MATLAB yazılımı haline getirilmiştir[19]. OpenCV kütüphanesinde de benzer şekilde geliştirilmiş bir yazılım mevcuttur. Metod sağ ve sol kameradan elde edilmiş bir dizi Şekil 3.2 görülen dama tahtası görüntüsünden paterni çıkartır ve köşelerin koordinatları tespit edilir. Hesaplama iki boyutlu patern ve onun görüntü üzerindeki projeksiyonu arasındaki eşyazımlılığa dayanır. OpenCV'deki algoritma kullanıcı tarafından girilen kare kenar boyutları bilgisini kullanır. [20] Bu yazılım kullanılarak farklı görüntüleri alınmış bir dama tahtasındaki karelerin köşeleri tespit edildi ve tespit edilen bu köşelerin koordinat bilgileri kullanılarak döndürme ve taşıma vektörleri elde edilmiştir.



Şekil 3.2 Kalibrasyon için kullanılan dama tahtası

Geliştirilen kodda kalibrasyon yapabilmek için Şekil 3.3 de gösterildiği gibi *stdafx.h* başlık dosyasındaki *KALIBRASYON* değeri "1" e getirilirmesi gerekmektedir.



Şekil 3.3 Kalibrasyonun aktif/inaktif hale getirilmesi

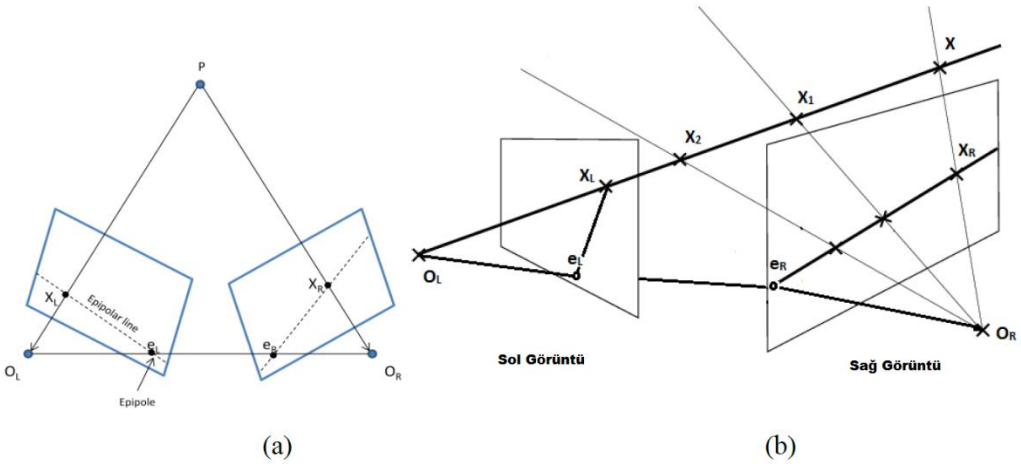
KALIBRASYON değerini "1" girdikten sonra ana yazılım dosyası olan *StereoUzaklikTespiti.cpp* içerisindeki Şekil 3.4 de başlangıcı gösterilen kalibrasyon yazılımı aktif hale gelir. Yazılımın devamını EK-A'da görülebilir.

```
//Kalibrasyon:

void StereoFunctions::stereoCalibration(StereoGrabber* grabb)
{

    int  nx=10, ny=6, frame = 0, n_boards =5, N;
    int count1 = 0,count2 = 0, result1=0, result2=0;
    int  successes1 = 0,successes2 = 0 ;
    const int maxScale = 1;
    const float squareSize = 24;        //Damalı tahtadaki kare boyu
    CvSize imageSize = {0,0};
    CvSize board_sz = cvSize( nx,ny );

    int i, j, n = nx*ny, N1 = 0, N2 = 0;
    vector<CvPoint2D32f> points[2];
    vector<int> npoints;
    vector<CvPoint3D32f> objectPoints;
    vector<CvPoint2D32f> temp1(n);
    vector<CvPoint2D32f> temp2(n);

    double M1[3][3], M2[3][3], D1[5], D2[5];
    double R[3][3], T[3], E[3][3], F[3][3];
    double Q[4][4];
    CvMat _Qcalib  = cvMat(4,4, CV_64F, Q);
    CvMat _M1calib = cvMat(3, 3, CV_64F, M1 );
    CvMat  M2calib = cvMat(3, 3, CV_64F, M2 );
```

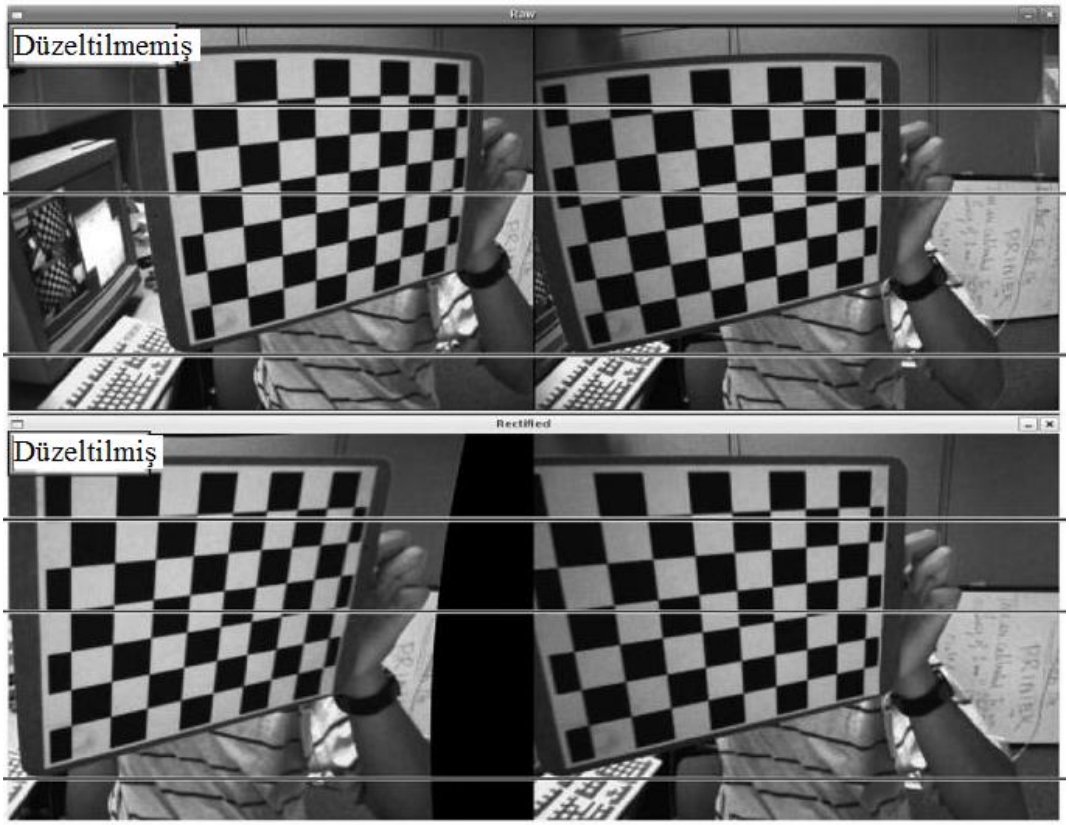
Şekil 3.4 Kalibrasyon yazılımının başlangıcı

3.1.1.1 Rektifikasyon

Stereo görüntü çiftlerin rektifikasyonu, bu görüntülerin biçimsel bozulmasını gidererek görüntülerdeki eş piksellerin aynı çizgi üzerinde yerleşmesi sağlanır. Daha teknik ifade etmek gerekirse aşağıda görülen epipolar çizgilerin eş doğrusal olması sağlanır. Stereo görüntüleme geometrisi Şekil 3.5 ‘de gösterilmiştir.



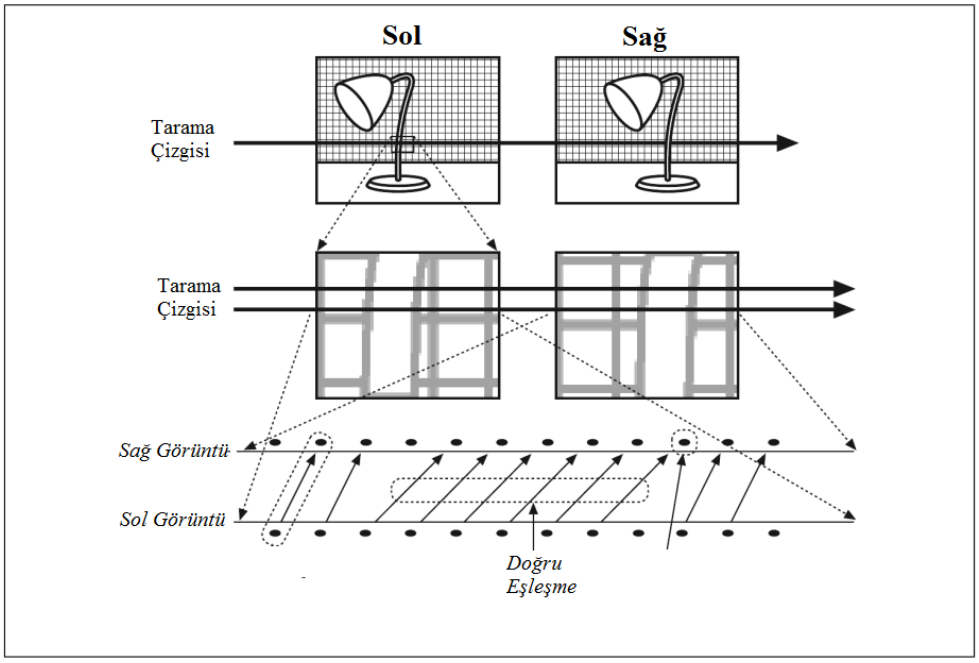
Şekil 3.5 Stereo görüntüleme geometrisi a) Tek noktalı epipolar düzlem b) Çok noktalı epipolar üzlem



Şekil 3.6 Düzeltilmiş ve düzeltilmemiş görüntüler arasındaki farklar

3.1.2 Korelasyon

Korelasyon işleminde sağ ve sol kameradan alınan düzeltilmiş görüntüler kullanarak, bu görüntülerdeki aynı noktalar eşleştirilir. Kameradan olan uzaklığı tespit etmek için ise, görüntülerdeki aynı noktanın lokasyonun kayması anlamına gelen "disparity" bilgisinin elde edilmesi gerekmektedir. Şekil 3.7 'te pikselerin eşleştirilmesi şematik olarak gösterilmiştir.



Şekil 3.7 Sol ve sağ görüntülerdeki piksellerin eşleştirilmesi[21]

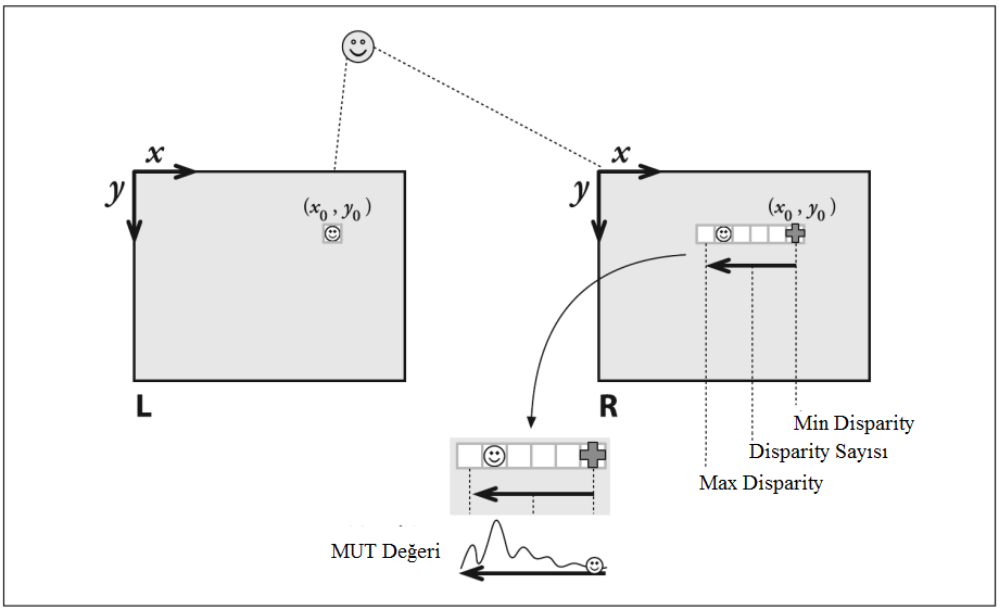
Blok eşleştirme tekniği, korelasyonu hesaplamak ve disparity bilgisini elde etmek için kullanılan algoritma türlerinden birisidir. Bu algoritmada mutlak uzaklıkların toplamı (MUT) pencereleri kullanılarak uyumluluk bulunur. MUT pencereleri, görüntüdeki her bir piksele çevresindeki komşularını da göz önünde bulundurarak bir skorlama methodu kullanır. OpenCv hızlı ve efektif bir blok eşleştirme algoritması kullanır. Bu algoritma *cvFindStereoCorrespondenceBM()* komutu ile çağrılır ve MUT pencereleri kullarak çalışır. Bu algoritma iki görüntüdeki güçlü eşleşmeleri tespit eder. Örneğin yüksek dokulu bir ormandaki her bir piksel için derinlik hesaplanabilirken daha az dokulu iç mekandaki bir koridorda bir kaç pikselin derinliği hesaplanabilir. Eşleştirme sırasında gürültülerin üstesinden gelmek için yeterli seviyede doku olduğundan emin olmak için *textureTreshold* komutu kulllanılır. Bu sadece denklem **(3.2)** ile hesaplanan MUT değerine limit koymak için kullanılır. Böylelikle bu limitin altındaki değerlerde eşleşme olmadığı varsayılır. OpenCV'de blok eşleştirme tekniği için; ön

filtreleme, uyumluluk ve son filtreleme olmak üzere üç adım uygulanır. Bu işlemler sırasında sol ve sağ görüntüler normalize edilerek aynı ışık seviyelerinin elde edilmesi sağlanır. Daha önce bahsedilen düzleştirme filtrelerine benzer şekilde boyutu ayarlanabilen bir pencere her bir piksele uygulanır ve her piksel değeri aşağıdaki denklemdeki şekilde değiştirilir:

$$\min[max(I_c - \bar{I}, -I_{cap}), I_{cap}] \quad \textbf{(3.1)}$$

Bu denkemde $\bar{I}$ değeri pecere içerisindeki ortalama yoğunluğu, I_{cap} değeri üst limiti, I_c değeri ise pencerenin uygulandığı merkezdeki piksel değerini temsil etmetedir.

Artık sol ve sağ görüntüdeki piksellerin eşleştirilmesi gerekmektedir. Rektifikasyondan sonra iki görüntüdeki her bir satır eşleştirilir böylece eş noktalar teoride aynı satırda yer almaktadır. MUT pencereleri sol görüntüdeki bir piksele ayarlanır ve bir skor hesaplanır. Sonrasında algoritma sağ görüntüde aynı koordinattaki pikseli bulur ve sonra maksimum disparity değerine ulaşana kadar x ekseni doğrultusunda sola hareket ederek tüm piksellerin skorunu hesaplar [21]. Disparity değeri ilk piksel ile uyumlu olan piksel arasındaki farkı ifade eder. Geliştirilen kodda eşleştirmeyi kontrol eden aynı zamanda eşleşmeyi başlatan ilk parametre *minDisparity* komutudur. Varsayılan *minDisparity* değeri "0" dır. Sonrasında uyumluluk arayışı *numberOfDisparities* komutu ile devam eder. Minimum disparity ve aranacak disparity değerlerinin ayarlanması ile stereo algoritmasının tarama sınırlarını oluşturan 3B hacim anlamına gelen "horopter" oluşturulur.



Şekil 3.8 Görüntülerdeki eşpiksellerin aranışı[21]

MUT değeri ise aşağıdaki denklem ile hesaplanır:

$$MUT(r,c) = \sum_{y=-w}^{y=w} \sum_{x=-w}^{x=w} |Sağ(y+r, x+c+d) - Sol(y+r, x+c)| \quad \textbf{(3.2)}$$

Bu denklemde (r,c) sağ görüntüde soldaki piksele uyumlu olan pikseli ifade eder, d değeri ise sağ görüntüdeki noktanın sol görüntüdeki orijinal nokta arasındaki disparity değerini ifade eder. Bu denklem ile, skorların pikselin etrafındaki komşu piksellerin yoğunluk değerine dayanarak hesaplandığı ortaya koyuluyor. Sağ görüntüde aranan kısımdaki en düşük skora sahip piksel en iyi eşleşme olarak öngörülüyor. Bu piksel ile sol görüntüdeki orjinal piksel arasındaki fark disparity değeri olarak alınır ve bu değer ile daha önce de bahsedilen aşağıdaki denklemler ile derinlik bilgisi hesaplanır. [21]

$$Z = \frac{Kameralar\ arası\ uzaklık\ x\ Odak\ Uzaklığı}{Disparity} \quad \textbf{(3.3)}$$

```
//Korelasyon:

void StereoFunctions::stereoCorrelation(StereoGrabber* grabb)
{
    //BMState data güncellemesi
    CvStereoBMState *BMState = cvCreateStereoBMState();
    assert(BMState != 0);
    BMState->preFilterSize = stereoPreFilterSize;
    BMState->preFilterCap = stereoPreFilterCap;
    BMState->SADWindowSize = stereoDispWindowSize;
    BMState->minDisparity = 0;
    BMState->numberOfDisparities = stereoNumDisparities;
    BMState->textureThreshold = stereoDispTextureThreshold;
    BMState->uniquenessRatio = stereoDispUniquenessRatio;
    BMState->speckleWindowSize = 200;
    BMState->speckleRange = 32;
    BMState->disp12MaxDiff = 2;

    cvSplit(grabb->imageLeft, r_detect, g_detect, b_detect, NULL);
    cvRemap(r_detect, r_detect_r, mx1, my1); // Undistort image
    cvRemap(g_detect, g_detect_r, mx1, my1); // Undistort image
    cvRemap(b_detect, b_detect_r, mx1, my1); // Undistort image
    cvMerge(r_detect_r, g_detect_r, b_detect_r, NULL, img_detect);
```

Şekil 3.9 Korelasyon yazılımının başlangıcı

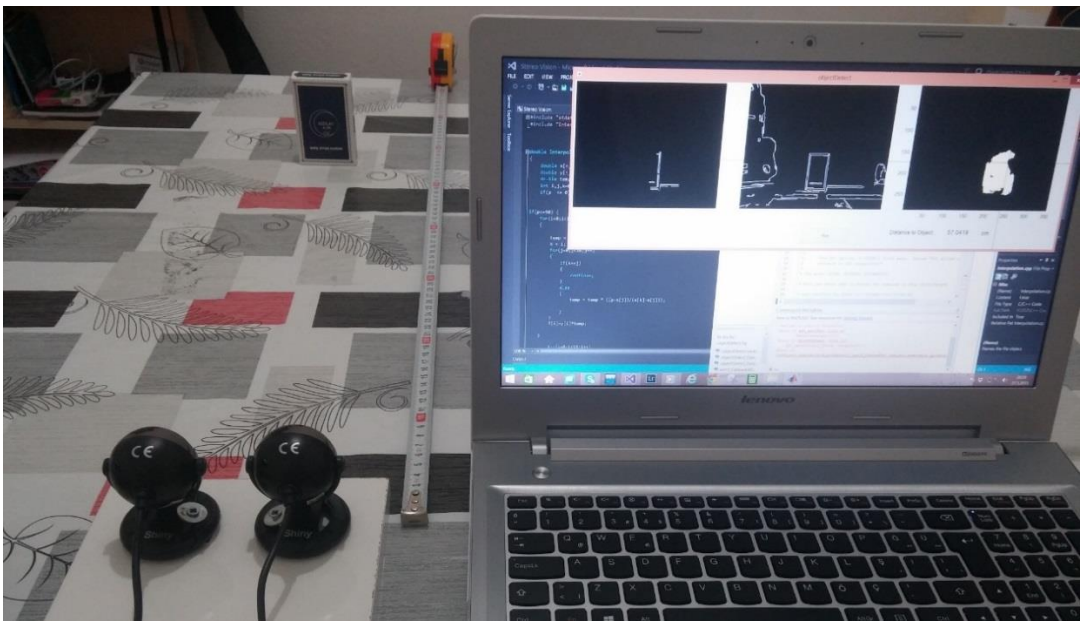
Yürüyen robotlar için geliştirilen bir stereo görüntüleme sisteminin hızlı çalışması kaçınılmaz bir gereksinimdir. Bu sebeple stereo eşleştirme işlemleri hızlı çalışacak şekilde geliştirilmiştir. Blok eşleştirme parametreleri *CvStereoBMstate* olarak adlandırılan data yapısında saklanır.

4. SONUÇ

Tez kapsamında uzaklık tespiti için kurulan sistem donanımları bir çift kamera ve dizüstü bilgisayardır. Şekil 4.1 kullanılan kameralar gözükmektedir.

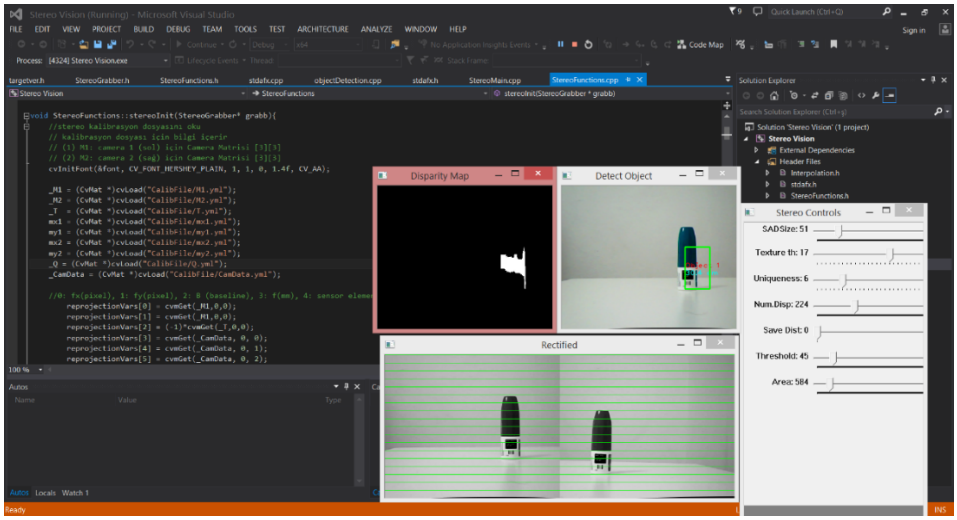
Şekil 4.1 Stereo kamera sistemi

Kurulan sistemin fotoğrafı Şekil 4.2 yer almaktadır. Kameralar bilgisayara "Evrensel Seri Veriyolu" ile bağlanmıştır. Kameraların aynı anda çalışması ve işletilmesiyle alakalı tüm yazılım C++ dilinde Visual Studio programı yardımıyla derlenmiştir



Şekil 4.2 Uzaklık tespiti için kurulan stereo sistem

Bu sistem ile Şekil 4.3 görüleceği üzere kenar tespiti başarı ile yapılmış ve geliştirilen yazılım ile uzaklık tespiti gerçekleştirilmiştir.



Şekil 4.3 Geliştirilen kod ile uzaklık tespiti

Çizelge 4.1'de görüldüğü üzere geliştirilen sistem ile uzaklık tespiti belirli bir hata payıyla gerçekleştirilmiştir.

Çizelge 4.1 Ölçülen uzaklık ve hata oranı

Gerçek Uzaklık	Ölçülen Uzaklık	Hata
80	76,7	4%
78	75,8	3%
76	74	3%
74	72,9	1%
72	71,2	1%
70	69,5	1%
68	67,7	0%
66	65,3	1%
64	63,8	0%
62	62,6	1%
60	61,1	2%
58	59,9	3%
56	59,5	6%
54	55,6	3%
52	53,8	3%
50	52,5	5%
48	51,3	7%
46	49,8	8%
44	48,1	9%
42	45,9	9%
40	44,3	11%

Hata payı 64-68mm aralığında oldukça düşük iken bu mesafelerden kameraya yaklaştıkça veya uzaklaştıkça artmaktadır. Bunun kameralardaki yapısal hatalardan kaynaklandığı düşünülmektedir. Daha kaliteli kameralar ile hassas şekilde sistem kurulduğundan hata payı düşecektir.

KAYNAKLAR

[1] **Welling, W**. **1978**., Photography in America, *Crowell*, s. 23.

[2] **Maini, R. ve Dr. Aggarwal, H. 2013,** Study and Comparison of Various Image Edge Detection Techniques, *International Journal of Image Processing*, s.1-3

[3] **Vincent, O.R., Folorunso, O. 2009**, A Descriptive Algorithm for Sobel Image Edge Detection, *InSITE*, s.102

[4] **Gupta, S. ve Mazumdar S.G. 2013** Sobel Edge Detecton Algorithm, *International Journal of Computer Science and Management E-Research*, s.1578,1579

[5] **Canny, J. 1986**, A Computational Approach to Edge Detection, *IEEE*, s. 1

[6] **Basu, M., 2002,** Gaussian-Bsed Edge-Detection Methods – A Survey, *IEEE*, s.2

[7] **Short, N.J., 2009**, *"3-D Point Cloud Generation from Rigid and Flexible Stereo Vision Systems"*, Yüksek Lisans Tezi, Virginia Polytechnic Institute and State University, Blacksburg, s. 12-18

[8] **Saxena, A., Chung S.H. ve Ng A.Y. 2007** 3-D Depth Reconstruction from a Single Still Image, *Springer Science+Busines Media*, s.1,8

[9] **Url-1** <*http://www.whydomath.org/node/wavlets/imagebasics.html*>, alındığı tarih: 20.05.2015.

[10] **Yıldırım, S.**, **2003** "Görüntü İşleme", Bilgisayar Mühendisliği Bölümü, Ege Üniversitesi, s. 11-18

[11] **Johnson, S., 2006** Stephen Johnson on Digital Photography, O'Reilly, s.26

[12] **Solomon C. ve Breckon T., 2011** Fundamentals of Digital Image Processing, *Wiley-Blackwell*, s. 48

[13] **Liu, H., 2005** "Distance Determination From Pairs of Images from Low Cost Cameras", *Edinburgh University Report*, No: HSM 1805, s.7

[14] **Ekstrom M., 1984** Digital Image Processing Technique, Academic Press, s.26

[15] **Nixon, M. S. ve Aguado ,A. S., 2002**, Feature Extraction and Image Processing, Newnes, REPP Ltd., s. 10

[16] **Sezgin, M. ve Sankur B. 2004** Survey over image thresholding techniques and quantitive performance evaluation, *Journal of Electronic Imaging*, s.2

[17] **Zhang, Z. 2000** A flexible new technique for camera calibration, *IEEE Trans Pattern Anal. Mach. Intell, s1330-1334*

[18] **Heillila, J. Ve Silven, O., 2007**, A Four-step Camera Calibration Procedure with Implicit Image Correction, *Finland*, s.1-7

[19] **Url-2 <***http://www.vısıon.caltech.edu/bouguetj/calıb_doc/>,* alındığı tarih:

21.05.2015.

[20] **Escalera, A. ve Armingol J. 2010** Automatic Chessboard Detection for Instrinsic and Extrinsic Camera Parameter Calibration, *Molecular Diversity Preservation International,* s.1

[21] **Kaehler A. ve Bradski G. 2008**, Learning OpenCV Computer Vision With the Open CV Library, O'Reilly, s439-443

EKLER

EK A: Canny Metodu ile Geliştirilmiş Kenar Belirleme Algoritması
EK B: Stereo Sistem ile Uzaklık Tespit Kaynak Kodu

EK A

```
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <stdlib.h>
#include <stdio.h>

using namespace cv;

Mat src, src_gray;
Mat dst, detected_edges;

int edgeThresh = 1;
int lowThreshold;
int const max_lowThreshold = 100;
int ratio = 3;
int kernel_size = 3;
char* window_name = "Edge Map";

void CannyThreshold(int, void*)
{


// 3x3'lük bir kernel ile gürültü azaltımı
blur(src_gray, detected_edges, Size(3, 3));


// Canny Kenar Belirleme Algoritması
Canny(detected_edges, detected_edges, lowThreshold, lowThreshold*ratio,
kernel_size);


dst = Scalar::all(0);
src.copyTo(dst, detected_edges);
imshow(window_name, dst);
}

int main(int argc, char** argv)
{

//Kameradan görüntü elde edilmesi
VideoCapture cap;
        cap.open(0);

        while (1)
        {
                cap >> src;

                // src ile aynı özelliklerde bir matris oluşturulması
                dst.create(src.size(), src.type());
```

```
// Görüntünün gri skalaya çevrilmesi
cvtColor(src, src_gray, CV_BGR2GRAY);

// Pencere oluşturulması
namedWindow(window_name, CV_WINDOW_AUTOSIZE);

// Treshold değerinin değiştirilmesi için bir "Trackbar" oluşturulması
createTrackbar("Min Threshold:", window_name, &lowThreshold,
max_lowThreshold, CannyThreshold);

// Görüntüleme
CannyThreshold(0, 0);

// Bir tuşa basılana kadar programın açık kalması
waitKey(1);

}
}
```

EK B

Stereofunction.h:

```
#pragma once
#include "stdafx.h"

#include <iostream>
#include <sstream>
#include <string>
#include <fstream>
#include <vector>

#include "BlobResult.h"
#include "cv.h"
#include "highgui.h"
#include "cvaux.h"
#include "StereoGrabber.h"
#include "Interpolation.h"
#include "opencv2/core/types_c.h"
#include <tchar.h>
#include <stdlib.h>
#include <stdio.h>
using namespace std;

struct StereoFunctions{

	CvMat* _M1;
	CvMat* _M2;
	CvMat* _T;
	CvMat* mx1;
	CvMat* mx2;
	CvMat* my1;
	CvMat* my2;
	CvMat* _Q;			//Reprojeksiyon Matrisi
	CvMat* _CamData;

	CvSize imageSize;			//Görüntülerin boyutu
	IplImage		*img1,		//Sol Görüntü
				*img2,		//Sağ Görüntü
				*rgb,
				*thres_img,
				*blobs_img,
				*img_detect,
				*real_disparity;
	CvMat *img1r,			//Düzeltilmiş Sol Görüntü
		*img2r,		//Düzeltilmiş Sağ Görüntü
		*disp,		//Disparity Haritası
		*vdisp,		//Ölçeklendirilmiş Disparity Haritası
		*pair,
```

```
                    *depthM;


        void stereoInit(StereoGrabber*);
        void stereoCorrelation(StereoGrabber*);
        void CorrelationBMState();
        void stereoSavePointCloud();
        void stereoCalibration(StereoGrabber*);
        void stereoDetect();
        double ComputeDis(CvMat*,int row, int col);
        double reprojectionVars[6];


        int stereoPreFilterSize,
                stereoPreFilterCap,
                stereoDispWindowSize,
                stereoNumDisparities,
                stereoDispTextureThreshold,
                stereoDispUniquenessRatio,
                stereoSavePointCloudValue,
                stereoSaveOriginal;


        int fileNO, threshold, blobArea;



};
```

Stereograbber.h:

```
#pragma once
#include "stdafx.h"
#include <iostream>
#include "cv.h"
#include "highgui.h"
#include "cvaux.h"
#define WIDTH 352
#define HEIGHT 288
struct StereoGrabber{

        void stereoGrabberInitFrames();
        void stereoGrabberGrabFrames();
        void stereoGrabberStopCam();
        IplImage* imageLeft;
        IplImage* imageRight;
};
```

Stdafx.h:

```
#pragma once
#include "targetver.h"
#include <string>
#include <stdio.h>
#include <tchar.h>

#define KALIBRASYON 0		//Kalibrasyon yapmak için "1" aksi halde "0"
değeri girilmelidir
#define DATA_GOSTER 1
#define DATA_KAYDET 0
#define UZAKLIK_GOSTER 1
#define UZAKLIK_KAYDET 0	// interpolasyon için görüntü kaydetme
#define KENAR_TESPITI 1		// Programın Kenar Tespiti ile Çalıştırılması
#define AYAR_PENCERESI 1	// Ayar penceresini açar
#define PI	3.141592654
```

Stereofunction.cpp:

```
#include "stdafx.h"
#pragma warning( disable: 4996 )
#ifdef _MSC_VER
#define _CRT_SECURE_NO_WARNINGS
#endif
#include "StereoFunctions.h"
#include "math.h"
#include "opencv2/core/types_c.h"
#include <iomanip>
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <stdlib.h>
#include <stdio.h>
#include "StereoGrabber.h"
#include <vector>
#include <algorithm>
#include <fstream>
#include <sstream>
#include <iostream>
#include <ctype.h>
#include <stdlib.h>
#include <windows.h>
#include "cv.h"
#include "highgui.h"
#include "cvaux.h"
using namespace cv;


//Kamerlardan görüntü almak için gerekli algoritmalar:
```

```
CvCapture *capture1 = NULL, *capture2 = NULL; //capture1 <-> sol, capture2 <-> sağ

void StereoGrabber::stereoGrabberInitFrames()
{
	capture1 = cvCaptureFromCAM(2);
	assert(capture1 != NULL);
	cvWaitKey(100);
	capture2 = cvCaptureFromCAM(1);
	assert(capture2 != NULL);

	//Görüntülerin çözünürlüğü ayarlanıyor:

	cvSetCaptureProperty(capture1, CV_CAP_PROP_FRAME_WIDTH, WIDTH);
	cvSetCaptureProperty(capture1, CV_CAP_PROP_FRAME_HEIGHT, HEIGHT);
	cvSetCaptureProperty(capture2, CV_CAP_PROP_FRAME_WIDTH, WIDTH);
	cvSetCaptureProperty(capture2, CV_CAP_PROP_FRAME_HEIGHT, HEIGHT);

}

//Alınan görüntüler stereo işlemede kullanılmak üzere tanımlanıyor:

void StereoGrabber::stereoGrabberGrabFrames()
{
	imageLeft = cvQueryFrame(capture1); //cvQueryFrame komutu ile görüntü OpenCV özel buffer alanına alınır ver alınan her kare birbiri ardına eklenir.
	imageRight = cvQueryFrame(capture2);
}

void StereoGrabber::stereoGrabberStopCam()
{
	cvReleaseCapture(&capture1);
	cvReleaseCapture(&capture2);
}

//Görüntüler işlenmeye başlanıyor:

Interpolation* Interpol = new Interpolation();
int fileNO = 0;
IplImage *r_detect, *g_detect, *b_detect, *r_detect_r, *g_detect_r, *b_detect_r;
int threshold, blobArea;
CvFont font;

int edgeThresh = 1;
int lowThreshold;
```

```
int const max_lowThreshold = 150;
int ratio = 3;
int kernel_size = 3;

void StereoFunctions::stereoInit(StereoGrabber* grabb){

	cvInitFont(&font, CV_FONT_HERSHEY_PLAIN, 1, 1, 0, 1.4f, CV_AA);

	//stereo kalibrasyon dosyasını okunarak kalibrasyon biglileri alınır
	//M1: camera 1 (sol) için Camera Matrisi [3][3]
	//M2: camera 2 (sağ) için Camera Matrisi [3][3]

	_M1 = (CvMat *)cvLoad("CalibFile/M1.yml");
	_M2 = (CvMat *)cvLoad("CalibFile/M2.yml");
	_T = (CvMat *)cvLoad("CalibFile/T.yml");
	mx1 = (CvMat *)cvLoad("CalibFile/mx1.yml");
	my1 = (CvMat *)cvLoad("CalibFile/my1.yml");
	mx2 = (CvMat *)cvLoad("CalibFile/mx2.yml");
	my2 = (CvMat *)cvLoad("CalibFile/my2.yml");
	_Q = (CvMat *)cvLoad("CalibFile/Q.yml");
	_CamData = (CvMat *)cvLoad("CalibFile/CamData.yml");

	//0: fx(pixel), 1: fy(pixel), 2: B (baseline), 3: f(mm), 4: sensor element size, 5:
baseline in mm
	reprojectionVars[0] = cvmGet(_M1, 0, 0);
	reprojectionVars[1] = cvmGet(_M1, 0, 0);
	reprojectionVars[2] = (-1)*cvmGet(_T, 0, 0);
	reprojectionVars[3] = cvmGet(_CamData, 0, 0);
	reprojectionVars[4] = cvmGet(_CamData, 0, 1);
	reprojectionVars[5] = cvmGet(_CamData, 0, 2);

	//Görüntüleri yükleme
	img1 = cvCreateImage(cvSize(grabb->imageLeft->width, grabb->imageLeft-
>height), IPL_DEPTH_8U, 1);
	img2 = cvCreateImage(cvSize(grabb->imageRight->width, grabb-
>imageRight->height), IPL_DEPTH_8U, 1);

	imageSize = cvSize(img1->width, img1->height);
	img1r = cvCreateMat(imageSize.height, imageSize.width, CV_8U);

	img2r = cvCreateMat(imageSize.height, imageSize.width, CV_8U);

	disp = cvCreateMat(imageSize.height, imageSize.width, CV_16S);
	vdisp = cvCreateMat(imageSize.height, imageSize.width, CV_8U);
	depthM = cvCreateMat(imageSize.height, imageSize.width, CV_32F);


	thres_img = cvCreateImage(imageSize, img1->depth, 1);
	blobs_img = cvCreateImage(imageSize, img1->depth, 3);
```

```
	img_detect = cvCreateImage(imageSize, grabb->imageLeft->depth, 3);
	r_detect = cvCreateImage(imageSize, 8, 1);//subpixel
	r_detect_r = cvCreateImage(imageSize, 8, 1);
	g_detect = cvCreateImage(imageSize, 8, 1);//subpixel
	g_detect_r = cvCreateImage(imageSize, 8, 1);
	b_detect = cvCreateImage(imageSize, 8, 1);//subpixel
	b_detect_r = cvCreateImage(imageSize, 8, 1);

	pair = cvCreateMat(imageSize.height, imageSize.width * 2, CV_8UC3);

}

Mat src1, src1_gray;
Mat dst1, detected_edges1;
Mat src2, src2_gray;
Mat dst2, detected_edges2;

void StereoFunctions::stereoCorrelation(StereoGrabber* grabb)
{
	//BMState data güncellemesi
	CvStereoBMState *BMState = cvCreateStereoBMState();
	assert(BMState != 0);
	BMState->preFilterSize = stereoPreFilterSize;
	BMState->preFilterCap = stereoPreFilterCap;
	BMState->SADWindowSize = stereoDispWindowSize;
	BMState->minDisparity = 0;
	BMState->numberOfDisparities = stereoNumDisparities;
	BMState->textureThreshold = stereoDispTextureThreshold;
	BMState->uniquenessRatio = stereoDispUniquenessRatio;
	BMState->speckleWindowSize = 200;
	BMState->speckleRange = 32;
	BMState->disp12MaxDiff = 2;

	cvSplit(grabb->imageLeft, r_detect, g_detect, b_detect, NULL);
	cvRemap(r_detect, r_detect_r, mx1, my1); // Undistort image
	cvRemap(g_detect, g_detect_r, mx1, my1); // Undistort image
	cvRemap(b_detect, b_detect_r, mx1, my1); // Undistort image
	cvMerge(r_detect_r, g_detect_r, b_detect_r, NULL, img_detect);

	cvCvtColor(grabb->imageLeft, img1, CV_BGR2GRAY);
	cvCvtColor(grabb->imageRight, img2, CV_BGR2GRAY);

	// Canny Kenar Belirleme Algoritması
	if (KENAR_TESPITI)
	{

		cvCanny(img1, img1, 100, lowThreshold*ratio, kernel_size);
		cvCanny(img2, img2, 100, lowThreshold*ratio, kernel_size);

	}
```

```
	cvRemap(img1, img1r, mx1, my1);
	cvRemap(img2, img2r, mx2, my2);
	cvFindStereoCorrespondenceBM(img1r, img2r, disp, BMState); //blok
eşleştirme

	cvNormalize(disp, vdisp, 0, 256, CV_MINMAX);

	if (UZAKLIK_GOSTER)
	{
		stereoSavePointCloud();
		stereoDetect();
	}


	if (DATA_GOSTER)
	{
		cvNamedWindow("Rectified", 1);

		cvNamedWindow("Disparity Map", 1);
		CvMat part;
		cvGetCols(pair, &part, 0, imageSize.width);

		cvCvtColor(img1r, &part, CV_GRAY2BGR);
		cvGetCols(pair, &part, imageSize.width, imageSize.width * 2);
		cvCvtColor(img2r, &part, CV_GRAY2BGR);
		for (int j = 0; j < imageSize.height; j += 16)
			cvLine(pair, cvPoint(0, j), cvPoint(imageSize.width * 2, j),
CV_RGB(0, 255, 0));

		cvShowImage("Rectified", pair);
		cvShowImage("Disparity Map", vdisp);

		if (DATA_KAYDET)
		{
			string dispFile;
			stringstream str, str2, str3;
			string left, right;
			str << "Disparities/DisparityMap" << "-" << fileNO << ".jpg";
			dispFile = str.str();
			cvSaveImage(&dispFile[0], vdisp);
			str2 << "Rectified/Left" << "-" << fileNO << ".jpg";
			str3 << "Rectified/Right" << "-" << fileNO << ".jpg";
			left = str2.str();
			right = str3.str();
			cvSaveImage(&left[0], img1r);
			cvSaveImage(&right[0], img2r);

			fileNO++;
		}
```

```
	}
	cvReleaseStereoBMState(&BMState);

}


CvPoint		rect1, rect2, rect, rect_dist;
CBlobResult blobs;
CBlob		*currentBlob;
void StereoFunctions::stereoDetect()
{


	cvCmpS(vdisp, threshold, thres_img, CV_CMP_GT);
	blobs = CBlobResult(thres_img, NULL, 0);

	/* Bloblar belirlenen alandan büyük olmazsa kaldır */
	blobs.Filter(blobs, B_EXCLUDE, CBlobGetArea(), B_LESS, blobArea,
4000);

	/* Blobları renklendir */
	for (int i = 0; i < blobs.GetNumBlobs(); i++)
	{
		currentBlob = blobs.GetBlob(i);
		currentBlob->FillBlob(blobs_img, CV_RGB(0, 0, 255));

		rect1.x = (int)currentBlob->MinX();
		rect1.y = (int)currentBlob->MinY();

		rect2.x = (int)currentBlob->MaxX();
		rect2.y = (int)currentBlob->MaxY();

		rect.x = rect1.x;
		rect.y = (rect1.y + rect2.y) / 2;

		rect_dist.x = rect.x;
		rect_dist.y = rect.y + 15;

		stringstream blobstream;
		blobstream << "Object " << i + 1;
		string blobNumObj = blobstream.str();

		cvPutText(img_detect, &blobNumObj[0], rect, &font, cvScalar(0, 0,
255, 0));

		stringstream DistText;
		DistText << setprecision(3) <<
max(max(max((float)ComputeDis(disp, (rect1.y + rect2.y) / 2, (rect1.x + rect2.x) /
2),
```

```
                        (float)ComputeDis(disp, rect1.y + 2 * (rect2.y - rect1.y) / 3,
rect1.x + (rect2.x - rect1.x) / 3)),
                        (float)ComputeDis(disp, rect1.y + 2 * (rect2.y - rect1.y) / 3,
rect1.x + 2 * (rect2.x - rect1.x) / 3)),
                        max((float)ComputeDis(disp, rect1.y + (rect2.y - rect1.y) / 3,
rect1.x + (rect2.x - rect1.x) / 3),
                        (float)ComputeDis(disp, rect1.y + (rect2.y - rect1.y) / 3,
rect1.x + 2 * (rect2.x - rect1.x) / 3))) << " cm";

                string ShowDistText = DistText.str();
                cvPutText(img_detect, &ShowDistText[0], rect_dist, &font,
cvScalar(255, 255, 0, 0));

                cvRectangle(img_detect, rect1, rect2, CV_RGB(0, 255, 0), 2, 8, 0);
        }

        cvShowImage("Binary Map", thres_img);
        cvShowImage("Detect Object", img_detect);

}


double StereoFunctions::ComputeDis(CvMat* Mat, int row, int col)
{
        double focal = reprojectionVars[0];
        double baseline = 7.2;//reprojectionVars[2];
        double sensorSize = reprojectionVars[4];
        double          k1 = 8e-7,
                k2 = -0.001,
                k3 = 0.4,
                k4 = 17.44;
        double          depth = 0;
        //      if(cvGet2D(Mat,row,col).val[0]== 0) return 0;
        depth = (double)((baseline*focal) / ((double)(cvGet2D(Mat, row, col).val[0] /
16)));
        depth = (k1*depth*depth*depth + k2*depth*depth + k3*depth + k4);
        return depth;
}

void StereoFunctions::stereoSavePointCloud()
{
        //0: fx(pixel), 1: fy(pixel), 2: base line (pixel), 3: f(mm), 4: sensor element
size, 5: baseline (mm)
        double focal = reprojectionVars[0];
        double baseline = 7.2;//reprojectionVars[2];
        double sensorSize = reprojectionVars[4];
        double          k1 = 8e-7,
                k2 = -0.001,
                k3 = 0.4,
                k4 = 17.44;
```

```
	double		depth = 0;
	real_disparity = cvCreateImage(imageSize, IPL_DEPTH_32F, 1);
	cvConvertScale(disp, real_disparity, 1.0 / 16, 0);

	//Uzaklık ölçümü
	for (int y = 0; y<vdisp->rows; y++)
	{
		for (int x = 0; x<vdisp->cols; x++)
		{
			if (cvGet2D(vdisp, y, x).val[0]>0){
				depth = (double)((baseline*focal) /
((double)(cvGet2D(disp, y, x).val[0] / 16)));
				depth = (k1*depth*depth*depth + k2*depth*depth +
k3*depth + k4);

			}
			else
				depth = 0;
			cvmSet(depthM, y, x, depth);
		}
	}

	if (UZAKLIK_KAYDET){
		FILE *distanceFile;
		stringstream strDistance;
		strDistance << "Distance/Distance.dat";
		string Distance = strDistance.str();
		distanceFile = fopen(&Distance[0], "w");

		for (int y = 0; y < vdisp->rows; y++){
			for (int x = 0; x < vdisp->cols; x++){

				fprintf(distanceFile, "%d %d %f %f\n", y, x,
(float)cvmGet(depthM, y, x), (float)cvGet2D(real_disparity, y, x).val[0]);
			}
		}
		fclose(distanceFile);
	}

}

//Kalibrasyon:

void StereoFunctions::stereoCalibration(StereoGrabber* grabb)
{

	int  nx = 10, ny = 6, frame = 0, n_boards = 20, N;
	int count1 = 0, count2 = 0, result1 = 0, result2 = 0;
	int  successes1 = 0, successes2 = 0;
```

```
const int maxScale = 1;
const float squareSize = 0.024f;                    //Damalı tahtadaki kare boyutu
CvSize imageSize = { 0, 0 };
CvSize board_sz = cvSize(nx, ny);

int i, j, n = nx*ny, N1 = 0, N2 = 0;
vector<CvPoint2D32f> points[2];
vector<int> npoints;
vector<CvPoint3D32f> objectPoints;
vector<CvPoint2D32f> temp1(n);
vector<CvPoint2D32f> temp2(n);

double M1[3][3], M2[3][3], D1[5], D2[5];
double R[3][3], T[3], E[3][3], F[3][3];
double Q[4][4];
CvMat _Qcalib = cvMat(4, 4, CV_64F, Q);
CvMat _M1calib = cvMat(3, 3, CV_64F, M1);
CvMat _M2calib = cvMat(3, 3, CV_64F, M2);
CvMat _D1 = cvMat(1, 5, CV_64F, D1);
CvMat _D2 = cvMat(1, 5, CV_64F, D2);
CvMat _R = cvMat(3, 3, CV_64F, R);
CvMat _Tcalib = cvMat(3, 1, CV_64F, T);
CvMat _E = cvMat(3, 3, CV_64F, E);
CvMat _F = cvMat(3, 3, CV_64F, F);

//Kameraları çalıştırma
printf("\nKameralar Calistiriliyor ...\n");
grabb->stereoGrabberInitFrames();
grabb->stereoGrabberGrabFrames();
IplImage *frame1 = grabb->imageLeft;
IplImage* gray_fr1 = cvCreateImage(cvGetSize(frame1), 8, 1);
IplImage *frame2 = grabb->imageRight;
IplImage* gray_fr2 = cvCreateImage(cvGetSize(frame2), 8, 1);

imageSize = cvGetSize(frame1);

cvNamedWindow("camera2", 1);
cvNamedWindow("camera1", 1);
cvNamedWindow("corners camera1", 1);
cvNamedWindow("corners camera2", 1);
printf("\nKalibrasyon icin gerekli tahta saysi: %d", n_boards);
while ((successes1<n_boards) || (successes2<n_boards))
{

        //Damalı tahtayı bulma ve köşe tespiti

        if ((frame++ % 20) == 0)
        {
                //----------------CAM1----------------------
```

```
                result1 = cvFindChessboardCorners(frame1, board_sz,
&temp1[0], &count1, CV_CALIB_CB_ADAPTIVE_THRESH |
CV_CALIB_CB_FILTER_QUADS);
                cvCvtColor(frame1, gray_fr1, CV_BGR2GRAY);

                //----------------CAM2---------------------

                result2 = cvFindChessboardCorners(frame2, board_sz,
&temp2[0], &count2, CV_CALIB_CB_ADAPTIVE_THRESH |
CV_CALIB_CB_FILTER_QUADS);
                cvCvtColor(frame2, gray_fr2, CV_BGR2GRAY);

                if (count1 == n&&count2 == n&&result1&&result2)
                {
                        cvFindCornerSubPix(gray_fr1, &temp1[0], count1,
cvSize(11, 11), cvSize(-1, -1), cvTermCriteria(CV_TERMCRIT_ITER +
CV_TERMCRIT_EPS, 30, 0.01));
                        cvDrawChessboardCorners(frame1, board_sz,
&temp1[0], count1, result1);
                        cvShowImage("corners camera1", frame1);
                        N1 = points[0].size();
                        points[0].resize(N1 + n, cvPoint2D32f(0, 0));
                        copy(temp1.begin(), temp1.end(), points[0].begin() +
N1);
                        ++successes1;

                        cvFindCornerSubPix(gray_fr2, &temp2[0], count2,
cvSize(11, 11), cvSize(-1, -1), cvTermCriteria(CV_TERMCRIT_ITER +
CV_TERMCRIT_EPS, 30, 0.01));
                        cvDrawChessboardCorners(frame2, board_sz,
&temp2[0], count2, result2);
                        cvShowImage("corners camera2", frame2);
                        N2 = points[1].size();
                        points[1].resize(N2 + n, cvPoint2D32f(0, 0));
                        copy(temp2.begin(), temp2.end(), points[1].begin() +
N2);
                        ++successes2;

                        printf("\nTespit Edilen Damali Tahta Cifti Sayisi: %d",
successes2);
                }

                else
                {
                        cvShowImage("corners camera2", gray_fr2);
                        cvShowImage("corners camera1", gray_fr1);
                }
```

```
                        grabb->stereoGrabberGrabFrames();
                        frame1 = grabb->imageLeft;
                        cvShowImage("camera1", frame1);
                        frame2 = grabb->imageRight;
                        cvShowImage("camera2", frame2);

                        if (cvWaitKey(15) == 27) break;
                }
        }

        grabb->stereoGrabberStopCam();
        cvDestroyWindow("camera1");
        cvDestroyWindow("camera2");
        cvDestroyWindow("corners camera1");
        cvDestroyWindow("corners camera2");
        printf("\nDone Capture!");


        //--------------Kalibrasyon İçin Hesapalama-------------------
        N = n_boards*n;
        objectPoints.resize(N);
        for (i = 0; i < ny; i++)
                for (j = 0; j < nx; j++)   objectPoints[i*nx + j] =
cvPoint3D32f(i*squareSize, j*squareSize, 0);
        for (i = 1; i < n_boards; i++) copy(objectPoints.begin(), objectPoints.begin()
+ n, objectPoints.begin() + i*n);
        npoints.resize(n_boards, n);

        CvMat _objectPoints = cvMat(1, N, CV_32FC3, &objectPoints[0]);
        CvMat _imagePoints1 = cvMat(1, N, CV_32FC2, &points[0][0]);
        CvMat _imagePoints2 = cvMat(1, N, CV_32FC2, &points[1][0]);
        CvMat _npoints = cvMat(1, npoints.size(), CV_32S, &npoints[0]);
        cvSetIdentity(&_M1calib);
        cvSetIdentity(&_M2calib);
        cvZero(&_D1);
        cvZero(&_D2);

        printf("\nKalibrasyon Yapılıyor ...");
        fflush(stdout);
        cvStereoCalibrate(&_objectPoints, &_imagePoints1, &_imagePoints2,
&_npoints, &_M1calib, &_D1, &_M2calib, &_D2, imageSize, &_R, &_Tcalib,
&_E, &_F,
                cvTermCriteria(CV_TERMCRIT_ITER + CV_TERMCRIT_EPS,
100, 1e-5),
                CV_CALIB_FIX_ASPECT_RATIO +
CV_CALIB_ZERO_TANGENT_DIST + CV_CALIB_SAME_FOCAL_LENGTH);


        printf("\nKalibrasyon Tamamlandı");
```

```
	cvUndistortPoints(&_imagePoints1, &_imagePoints1, &_M1calib, &_D1, 0,
&_M1calib);
	cvUndistortPoints(&_imagePoints2, &_imagePoints2, &_M2calib, &_D2, 0,
&_M2calib);


	CvMat* mx1calib = cvCreateMat(imageSize.height, imageSize.width,
CV_32F);
	CvMat* my1calib = cvCreateMat(imageSize.height, imageSize.width,
CV_32F);
	CvMat* mx2calib = cvCreateMat(imageSize.height, imageSize.width,
CV_32F);
	CvMat* my2calib = cvCreateMat(imageSize.height, imageSize.width,
CV_32F);

	double R1[3][3], R2[3][3], P1[3][4], P2[3][4];
	CvMat _R1 = cvMat(3, 3, CV_64F, R1);
	CvMat _R2 = cvMat(3, 3, CV_64F, R2);

	CvMat _P1 = cvMat(3, 4, CV_64F, P1);
	CvMat _P2 = cvMat(3, 4, CV_64F, P2);
	cvStereoRectify(&_M1calib, &_M2calib, &_D1, &_D2, imageSize, &_R,
&_Tcalib, &_R1, &_R2, &_P1, &_P2, &_Qcalib,
0/*CV_CALIB_ZERO_DISPARITY*/);

	cvInitUndistortRectifyMap(&_M1calib, &_D1, &_R1, &_P1, mx1calib,
my1calib);
	cvInitUndistortRectifyMap(&_M2calib, &_D2, &_R2, &_P2, mx2calib,
my2calib);

	printf("\nSaving matries for later use ...\n");
	cvSave("CalibFile//M1.yml", &_M1calib);
	cvSave("CalibFile//M2.yml", &_M2calib);
	cvSave("CalibFile//Q.yml", &_Qcalib);
	cvSave("CalibFile//T.yml", &_Tcalib);
	cvSave("CalibFile//mx1.yml", mx1calib);
	cvSave("CalibFile//my1.yml", my1calib);
	cvSave("CalibFile//mx2.yml", mx2calib);
	cvSave("CalibFile//my2.yml", my2calib);

}



using std::string;
using std::vector;
using namespace std;

StereoGrabber* grab = new StereoGrabber();
StereoFunctions* stereoFunc = new StereoFunctions();
```

```
void mouseHandler(int event, int x, int y, int flags, void* param);  //mouse
hareketlerini ele alma

//stereo korelasyon fonksiyonlari
void onWindowBarSlide(int pos);
void onTextureBarSlide(int pos);
void onUniquenessBarSlide(int pos);
void onPreFilterSizeBarSlide(int pos);
void onPreFilterCapBarSlide(int pos);
void onSaveDataSlide(int pos);
void onSaveOriginalsSlide(int pos);
void stereoCreateCorrelationControls();
void loadCorrelationParams();
void loadCorrelationParams();

bool fexists(const char *filename);

int main()
{
	if (KALIBRASYON) stereoFunc->stereoCalibration(grab);

	loadCorrelationParams();
	grab->stereoGrabberInitFrames();
	grab->stereoGrabberGrabFrames();
	stereoFunc->stereoInit(grab);

	while (1)
	{
		if (AYAR_PENCERESI)	stereoCreateCorrelationControls();

		stereoFunc->stereoCorrelation(grab);
		cvSetMouseCallback("Detect Object", mouseHandler, NULL);

		//////stereoFunc->stereoSavePointCloud();

		grab->stereoGrabberGrabFrames();

		if (cvWaitKey(10) == 27) break;
	}

	grab->stereoGrabberStopCam();

	return 0;
}


void loadCorrelationParams()
{
	ifstream params;
```

```
	params.open("CorrelationParams.txt");
	if (!params) {
		cerr << "Correlation Params file! dosyası acilamiyor";
		exit(1);// islemi durdur
	}
	else{
		string temp;
		int value;
		while (params >> temp)
		{
			params >> value;
			stereoFunc->stereoDispWindowSize = value;
			params >> temp;
			params >> value;
			stereoFunc->stereoDispTextureThreshold = value;
			params >> temp;
			params >> value;
			stereoFunc->stereoDispUniquenessRatio = value;
			params >> temp;
			params >> value;
			stereoFunc->stereoNumDisparities = value;
			params >> temp;
			params >> value;
			stereoFunc->threshold = value;
			params >> temp;
			params >> value;
			stereoFunc->blobArea = value;
		}
		stereoFunc->stereoPreFilterSize = 63;
		stereoFunc->stereoPreFilterCap = 63;
		stereoFunc->stereoSavePointCloudValue = 0;

		params.close();
	}
}

void onWindowBarSlide(int pos)
{
	stereoFunc->stereoDispWindowSize = cvGetTrackbarPos("SADSize",
"Stereo Controls");
	if (stereoFunc->stereoDispWindowSize < 5)
	{
		stereoFunc->stereoDispWindowSize = 5;
		stereoFunc->stereoCorrelation(grab);
	}
	else if (stereoFunc->stereoDispWindowSize % 2 == 0)
	{
		stereoFunc->stereoDispWindowSize += 1;
		stereoFunc->stereoCorrelation(grab);
	}
```

```
	else stereoFunc->stereoCorrelation(grab);
}

void onTextureBarSlide(int pos)
{
	stereoFunc->stereoDispTextureThreshold = cvGetTrackbarPos("Texture th",
"Stereo Controls");
	if (stereoFunc->stereoDispTextureThreshold)
		stereoFunc->stereoCorrelation(grab);
}

void onUniquenessBarSlide(int pos)
{
	stereoFunc->stereoDispUniquenessRatio = cvGetTrackbarPos("Uniqueness",
"Stereo Controls");
	if (stereoFunc->stereoDispUniquenessRatio >= 0)
		stereoFunc->stereoCorrelation(grab);
}

void onNumDisparitiesSlide(int pos)
{
	stereoFunc->stereoNumDisparities = cvGetTrackbarPos("Num.Disp", "Stereo
Controls");
	while (stereoFunc->stereoNumDisparities % 16 != 0 || stereoFunc-
>stereoNumDisparities == 0)
		stereoFunc->stereoNumDisparities++;

	stereoFunc->stereoCorrelation(grab);
}
void onSaveDataSlide(int pos)
{
	cvSaveImage("Distance//Img1r.jpeg", stereoFunc->img1r);
	cvSaveImage("Distance//DisparityMap.jpeg", stereoFunc->vdisp);
	stereoFunc->stereoSavePointCloud();
}

void onSaveOriginalsSlide(int pos)
{

	cvSaveImage("Left.jpeg", stereoFunc->img1);
	cvSaveImage("Right.jpeg", stereoFunc->img2);

}

void stereoCreateCorrelationControls()
{
	cvNamedWindow("Stereo Controls", 0);
	cvResizeWindow("Stereo Controls", 350, 600);
	cvCreateTrackbar("SADSize", "Stereo Controls", &stereoFunc-
>stereoDispWindowSize, 255, onWindowBarSlide);
```

```
	cvCreateTrackbar("Texture th", "Stereo Controls", &stereoFunc-
>stereoDispTextureThreshold, 25, onTextureBarSlide);
	cvCreateTrackbar("Uniqueness", "Stereo Controls", &stereoFunc-
>stereoDispUniquenessRatio, 25, onUniquenessBarSlide);
	createTrackbar("Min Threshold:", "Rectified", &lowThreshold,
max_lowThreshold);
	cvCreateTrackbar("Num.Disp", "Stereo Controls", &stereoFunc-
>stereoNumDisparities, 640, onNumDisparitiesSlide);
	cvCreateTrackbar("Save Dist", "Stereo Controls", &stereoFunc-
>stereoSavePointCloudValue, 1, onSaveDataSlide);
	cvCreateTrackbar("Threshold", "Stereo Controls", &stereoFunc->threshold,
300, 0);
	cvCreateTrackbar("Area", "Stereo Controls", &stereoFunc->blobArea, 5000,
0);
}

void mouseHandler(int event, int x, int y, int flags, void *param){

	switch (event){
	case CV_EVENT_LBUTTONDOWN:
		//l = cvGet2D(stereoFunc->depthM, x, y);
		printf("Distance to this object is: %f cm \n",
(float)cvGet2D(stereoFunc->depthM, x, y).val[0]);
		break;
	}
}
```

ÖZGEÇMİŞ

Ad Soyad **:** Cafer Sinan KATI

Doğum Yeri ve Tarihi **:** Bursa - 1987

E-Posta **:** cafersinankati@gmail.com

ÖĞRENİM DURUMU:

- **Lisans** **:** 2010, İTÜ, Makine Fakültesi, Makine Mühendisliği

MESLEKİ DENEYİM:

03/2014 – Halen Aselsan A.Ş. - *Mekanik Tasarım Mühendisi*

02/2013 – 03/2014 3M Türkiye A.Ş. – *Proses Mühendisi*

08/2011 – 02/2013 Ford Otosan A.Ş. – *Ürün Geliştirme Mühendisi*