KOCAELİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

YÜKSEK LİSANS TEZİ

ÜÇ BOYUTLU DERİNLİK KAMERASI KULLANILARAK ÇOK MODLU DOĞAL ETKİLEŞİM

FATİH ERGÜNER

KOCAELİ 2014

KOCAELİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

YÜKSEK LİSANS TEZİ

ÜÇ BOYUTLU DERİNLİK KAMERASI KULLANILARAK ÇOK MODLU DOĞAL ETKİLEŞİM

FATİH ERGÜNER

Yrd.Doç.Dr. Pınar ONAY DURDU **Danışman, Kocaeli Üniv.**	.....
Doç.Dr. Mehmet GÖKTÜRK **Jüri Üyesi, GYTE**	.....
Yrd.Doç.Dr. Ahmet SAYAR **Jüri Üyesi, Kocaeli Üniv.**	.....

Tezin Savunulduğu Tarih: 01.07.2014

ÖNSÖZ VE TEŞEKKÜR

Geçen birkaç on yıl öncesinden bu günümüze gelinceye kadar insanlar bilgisayarlarla farklı şekillerde etkileşime girmişlerdir. Zaman geçtikçe ve teknolojik olanaklar ilerledikçe insanlar bilgisayarlarla daha doğal etkileşim yöntemleri geliştirmeye başlamıştır. Bu çalışmada sadece üç boyutlu derinlik kamerası kullanılarak ve dokunma gerektiren başka hiçbir girdi aygıtı kullanmadan üç boyutlu görüntülerin kontrolü sağlanmıştır. Ardından daha doğal olan bu etkileşim şeklinin fare ile mukayese edilerek bir kullanılabilirlik değerlendirmesi yapılmıştır. Sonuç olarak klasik girdi aygıtı olan klavye ve fare kullanımının zor olduğu durumlarda jest tabanlı kontrol cihazı olan Kinect kullanılarak üç boyutlu görüntüleri kontrol etmenin uygun bir seçim olduğu gözlenmiştir.

Çalışmam boyunca benden yardımını esirgemeyen ve sabırla bana destek olan ailem ve değerli hocam Yrd. Doç. Dr. Sayın Pınar ONAY DURDU Hanımefendi'ye, Yrd. Doç. Dr. Sayın Ahmet SAYAR'a, Doç. Dr. Sayın Mehmet GÖKTÜRK'e teşekkür ederim. Ayrıca pek değerli ve kalbimde müstesna yerleri olan Emre, Niyazi, Abdulkadir ve Şaban'a en derin muhabbet ve minnet duygularımı sunarım.

Haziran – 2014 Fatih ERGÜNER

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

SİMGELER DİZİNİ VE KISALTMALAR

bps : bits per second (saniyede bir bit)
sn : saniye

Kısaltmalar

3B : Üç Boyutlu
ANOVA : Analysis of Variance (Varyans Analizi)
CLI : Command Line Interface (Komut Satırı Arayüzü)
CMOS : Complementary Metal–Oxide–Semiconductor (Tamamlayıcı Metal-Oksit-Yarıiletken)
CT : Computed Tomography (Hesaplamalı Tomografi)
GUI : Graphical User Interface (Grafik Kullanıcı Arayüzü)
HCI : Human Computer Interaction (İnsan Bilgisayar Etkileşimi)
ISO : International Standard Organization (Uluslararası Standart Organizasyonu)
İBE : İnsan Bilgisayar Etkileşimi
MR : Magnetic Resonance (Manyetik Rezonans)
NUI : Natural User Interface (Doğal Kullanıcı Arayüzü)
RGB : Red Green Blue (Kırmızı Yeşil Mavi)
VTK : Visualization Toolkit Software Library (Görselleştirme Alet Takımı Yazılım Kütüphanesi)

ÜÇ BOYUTLU DERİNLİK KAMERASI KULLANILARAK ÇOK MODLU DOĞAL ETKİLEŞİM

ÖZET

Günümüzde kullanıcı arayüzü türleri; komut satırı arayüzlerinden (command line interface), grafiksel kullanıcı arayüzlerine (graphical user interface) doğru bir evrim geçirmiş ve son dönemde de doğal kullanıcı arayüzleri popülerlik kazanmıştır. Doğal kullanıcı arayüzleri insan-bilgisayar etkileşiminin, insan-insan etkileşiminde olduğu gibi doğal ve verimli şekilde gerçekleştirilmesi için geliştirilmeye devam etmektedir. Jest ve konuşma gibi birden fazla etkileşim yöntemi ile insan bilgisayar etkileşimi çok modlu doğal etkileşim şeklinde sağlanabilmektedir. Bu çalışma kapsamında çok modlu doğal etkileşim için üç boyutlu derinlik kamerası olan Kinect ile dokunma ya da başka hiçbir girdi aygıtı kullanmadan jest ve ses tabanlı üç boyutlu görüntü kontrolünü sağlayan bir uygulama gerçeklenmiştir. Bu amaçla gerçeklenen sistem klavye ve fare kullanmadan sadece ses ve jest komutları ile uygulamanın başlatılıp sonlandırılması ve üç boyutlu görüntülerle etkileşime olanak sağlamaktadır. Sistem üç farklı ses komutu ile sistemi başlatma, sistemi sonlandırma ve üç boyutlu görüntüyü ilk durumuna getirme işlevlerine sahiptir. Ayrıca jest komutları ile üç boyutlu görüntünün döndürülmesi (rotate), taşınması/sürüklenmesi (pan) ve yakınlaştırılıp uzaklaştırılması (zoom) işlemleri yapılabilmektedir.

Aynı zamanda çalışma kapsamında doğal etkileşim arayüzü kullanılarak gerçeklenen bu sistemin ne kadar kullanılabilir olduğunu belirlemek için kullanılabilirlik değerlendirilmesi yapılmıştır. Kullanılabilirlik değerlendirilmesinde kullanıcı testine dayalı deneysel yaklaşımla 12 katılımcıya kullanıcı testi uygulanarak sistemin verimlilik ve etkililiği tespit edilmeye çalışılmıştır. Katılımcılardan ses ve jestlerden oluşan 7 görevi hem Kinect ile hem de fare ile tamamlamaları istenmiş ve katılımcıların görevleri tamamlama süreleri ölçülmüştür. Bunun arkasından katılımcılardan ISO 9241-9'da tanımlanan memnuniyet anketini doldurmuştur. Kullanılabilirlik çalışması sonucunda fare deneyimi fazla olan katılımcıların fare performanslarının daha yüksek olması ile birlikte Kinect cihaz deneyimi olan katılımcının operasyonları her iki girdi cihazı ile gerçekleştirme süreleri birbirine yakın olarak ölçülmüştür. Gerçeklenen sistemin klasik girdi aygıtı olan klavye ve fare kullanımının zor olduğu koşullarda klavye ve fareye iyi bir alternatif olabileceği düşünülmektedir.

Anahtar Kelimeler: Çok Modlu Doğal Etkileşim, Doğal Kullanıcı Arayüzü, İnsan Bilgisayar Etkileşimi, Kinect, Kullanıcı Testi.

MULTIMODAL NATURAL INTERACTION USING THREE DIMENSIONAL DEPTH CAMERA

ABSTRACT

User interfaces has been evolved from command line interfaces to graphical user interfaces. Currently, natural user interfaces gained more attention. Natural interfaces aim to provide natural and efficient human-computer interaction as in human-human interaction. In addition multimodal natural interaction is used to enable human computer interaction by enabling more than one interaction style like gestures and voice. In the scope of this study, multimodal natural interaction has been implemented to provide interaction with three dimensional images without the use of any traditional or touch-based input devices. A system was developed by the use of Kinect, which includes a three dimensional camera. System used three different voice commands for starting and stopping the system and resetting the three dimensional image into its initial state. In addition, gestures were used for rotating, panning and zooming the image.

In the study, usability evaluation was conducted to determine the usability of this multimodal natural interaction based system. A user test was applied with 12 participants to determine the effectiveness and efficiency. Participants were asked to conduct 7 tasks both with mouse and with Kinect and their task completion times were measured. Afterwards they were required to fill in a satisfaction questionnaire based on ISO 9241-9 standard. While experienced mouse users performed better in mouse interaction, user who had an experience with Kinect had similar task completion times in two interaction styles. Developed system can be considered a fair alternative to keyboard and mouse where accessibility of these traditional input devices is hard.

Keywords: Multimodal Natural Interaction, Natural User Interface, Human Computer Interaction, Kinect, Usability Testing.

GİRİŞ

Günlük hayatta insanlar birbirleri ile sözel iletişimi tercih ederler. Bununla birlikte vermek istediği mesajı etkili kılmak için de birtakım jestler ve mimiklerle mesajı pekiştirirler. Kurulan bu etkileşim biçimi insanların birbirleri ile olan en doğal etkileşimini ifade eder. Bir insan sahip olduğu bilgiyi paylaşmak veya bir isteğini diğer bir insana belirtmek için onunla diyalog kurar. Karşı taraf isteğini anlar ve cevabı verir. Bu tipik insan-insan etkileşimini ifade eder. İnsan benzer isteklerini veya bilgi paylaşımlarını veya diğer kuracağı diyalogları/etkileşimleri bir bilgisayar ile de kurabilir. İşte kurulan bu etkileşim ile alakalı çalışmaları ve tasarımları bilgisayar bilimi, davranış bilimi ve diğer birtakım çalışma alanları çerçevesinde inceleyen bilim dalı İnsan Bilgisayar Etkileşimi (İBE) (Human Computer Interaction - HCI) kapsamına girmektedir (URL-3, 2014). İBE'nin alt çalışma alanlarından olan kullanıcı arayüzleri açısından baktığımız zaman arayüzlerin evriminin insan-insan etkileşimine yaklaşmaya çalıştığı görülebilir (URL-21, 2014). Bu çerçevede kullanıcı arayüzlerinin evrimi teknolojik olanaklar çerçevesinde en doğal etkileşim şekline doğrudur.

Grafik Kullanıcı Arayüzleri (Graphical User Interface - GUI) 1990'larda dünya çapında yaygın şekilde kullanılmaya başlanmıştır. GUI'ler ilk çıktıkları zamanda Komut Satırı Arayüzlerine (Command Line Interface - CLI) göre daha anlaşılır, kullanımı ve öğrenmesi kolay arayüzler olarak karşımıza çıkmıştır (URL-4, 2014). Teknolojik olanakların gelişmesi ile bilgisayarlarla insana daha yakın etkileşim şekilleri geliştirilmeye başlanmıştır. Bunlardan biri de çok modlu

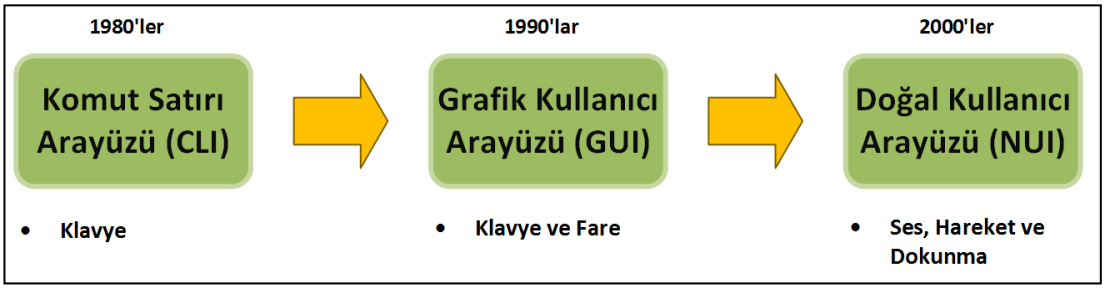
etkileşim'dir (Mutimodal Interaction). Çok modlu etkileşim insanın beş duyusuna (mode) hitap edecek şekilde etkileşim kurulmasını tanımlar (Multimodal Interaction, 2014) ve doğal kullanıcı arayüzleri geliştirilirken dikkat edilen bir etkileşim biçimidir.

Gerçeklenen sistemde başlatıp sonlandırma ve üç boyutlu görüntünün ilk durumuna getirilmesi (reset) işlemleri ses komutları ile gerçekleştirilmiştir. Ayrıca üç boyutlu görüntünün döndürülmesi (rotate), taşınması/sürüklenmesi (pan) ve yakınlaştırılıp uzaklaştırılması (zoom) işlemleri öğrenilmiş birtakım jestlerle gerçekleştirilmiştir. Bunların yanında sistemin başarısını ölçmek maksadıyla kullanıcı testine dayalı deneysel yaklaşımla 12 katılımcıya kullanıcı testi uygulanmıştır. Katılımcılardan ses ve jestlerden oluşan 7 görevi hem Kinect ile hem de fare ile tamamlamaları istenmiş ve katılımcıların jest görevlerini tamamlama süreleri ölçülmüştür. Ardından katılımcılara ISO 9241-9 (2000) dökümanında yer alan memnuniyet anketini doldurmaları istenmiştir. Sonrasında hem kullanıcı testi hem de kullanıcı anketi sonuçları yorumlanmıştır. Gerçeklenen sistem klasik girdi aygıtı olan klavye ve fare kullanımının zor olduğu koşullarda klavye ve fareye iyi bir alternatif olabileceği düşünülmektedir.

1. GENEL BİLGİLER

1.1. Tez Çalışmasının Amacı ve Başlatılma Sebepleri

Teknolojik imkanların artması, jest kontrol cihazlarına erişimin kolaylaşması Doğal Kullanıcı Arayüzü uygulamaları üzerine ilgiyi artırmıştır. Doğal Kullanıcı Arayüzleri sadece klavye ile etkileşim kurulan Komut Satırı Arayüzleri ile başlayıp klavye ve fare ile etkileşim kurulan Grafik Kullanıcı Arayüzleri ile devam eden gelişmenin son halkasıdır. Doğal Kullanıcı Arayüzlerinde diğer iki etkileşim yönteminin aksine daha kolay öğrenilebilir ve daha doğal etkileşim kurulabilmektedir (Şekil 1.1). Bu tez çalışmasında klavye ve fare gibi dokunma gerektiren hiçbir girdi aygıtı kullanma ihtiyacı olmadan üç boyutlu görüntülerin kontrolü sağlanabilmektedir.



Şekil 1.1. Kullanıcı arayüzlerinin evrimi

Bu tezin amacı üç boyutlu görüntülerin doğal ve temassız etkileşimini sağlamak ve ortaya koyulan bu sistemin verimliliğini, etkililiğini ve kullanıcılar tarafından memnuniyet derecesini ölçmektir. Tez çalışmasında öncelikli hedef temas gerektiren klasik girdi aygıtları olan klavye ve fareden tamamen bağımsız çok modlu doğal etkileşim sağlayan bir sistem geliştirmektir. Bunun için sistem başlatılmasından sonlandırılmasına kadar hem jestlerle hem de ses ile komut verilebilir şekilde tasarlanmış ve gerçeklenmiştir.

1.2. Tez Çalışmasının Katkıları

Bu tez çalışması aşağıdaki katkıları sağlamak amacıyla başlatılmıştır:

- Klasik girdi aygıtı olan klavye ve fare gibi cihazlardan farklı olarak jest ve ses tabanlı doğal insan-bilgisayar etkileşimi sağlayan Kinect teknolojisinin kullanım potansiyelini belirlemektir.

- Klasik girdi aygıtı olan klavye ve fare kullanımının zor veya uygun olmadığı sunum veya ameliyat gibi özel durumlarda herhangi bir girdi aygıtı ile fiziksel temas gerekmeden üç boyutlu görüntü kontrolünde Kinect'in kullanımını sağlamaktır.

- Kinect kullanarak sağlanan çok modlu doğal etkileşimin klasik girdi aygıtlarıyla karşılaştırıldığında ne kadar kullanılabilir olduğunu değerlendirmektir.

1.3. Tezin Yapısı

Bu tez çalışması beş kısımdan oluşmaktadır. Öncelikle birinci bölümde tezin amacı ve başlatılma sebeplerinden bahsedilerek tez çalışmasının katkılarından bahsedilmiştir. Tezin ikinci bölümünde çok modlu doğal etkileşim ve jest tabanlı kontrol cihazları tanıtılmış ve bu cihazlar ile alakalı alanyazın araştırmasına değinilmiştir. Tezin üçüncü bölümünde gerçeklenen sistemin gerçekleme detaylarına yer verilmiştir. Önce Kinect'in teknik özelliklerinden bahsedilmiş ardından uygulama arayüzü tanıtılmıştır. Bunun arkasından da geliştirme yapmak için seçilen araç ve ortamlardan bahsedilmiş ve sonrasında yazılımın fonksiyonlarına değinilmiştir. Son olarak da yazılım gerçekleme detayı olarak yazılım modülleri, sınıf ve bileşenlerinden bahsedilmiştir. Dördüncü bölümde gerçeklenen sistem için kullanılabilirlik çalışmasının nasıl yapıldığı, verilerin nasıl bir süreçle toplandığı ve kullanıcı testi ve kullanıcı anketi sonuçlarına göre bulgulardan bahsedilmiştir. Son olarak beşinci bölümde sonuçlardan ve ileride yapılabilecek ilgili çalışmalardan bahsedilmiştir.

2. ALANYAZIN ARAŞTIRMASI

Doğal kullanıcı arayüzleri (Natural User Interaface) tasarlanırken jest tabanlı kontrol cihazları yaygın şekilde kullanılmaktadır. Bu tez kapsamında kullanılan Kinect cihazı da doğal kullanıcı arayüzleri tasarlanırken kullanılan jest tabanlı kontrol cihazlarından biridir. Alanyazın araştırması kapsamında ise öncelikle bu çalışmayı dolaylı veya doğrudan ilgilendiren jest tabanlı kontrol cihazları tanıtılmıştır. Bunun arkasından tez kapsamında gerçeklenen sistemi ilgilendiren alanyazın araştırmasından bahsedilmiştir. Son olarak da bu tez kapsamında gerçeklenen kullanılabilirlik değerlendirmesi ile ilgili alanyazın çalışmalarına değinilmiştir.

2.1. Doğal Etkileşim

Doğal Etkileşim (Natural Interaction) insan beş duyusuna hitap ederek bilgisayarlarla etkileşim kurulması konseptine dayalıdır (URL-14, 2014). Bu etkileşim şekli birtakım girdi aygıtları kullanılarak ancak kurulabilmektedir. Ancak günümüzde klavye ve fare gibi yaygın kullanılan girdi aygıtları bu etkileşim için yetersiz kalmaktadır. Doğal Etkileşim kavramı İnsan Bilgisayar Etkileşimi'nde son yıllarda ön plana çıkan Doğal Kullanıcı Arayüzleri altında ayrı bir alan olarak işlev görmektedir çünkü bu çalışma alanı birçok farklı alan, paradigma ve teknolojileri kapsamaktadır. Bu ayrı alan şu üç farklı etkileşim ve Doğal Kullanıcı Arayüz yaklaşımlarına odaklanmıştır: vücut hareket takibi (body motion tracking), el jesti tanıma (hand gesture recognition) ve ses ve komut tanıma (speech and command recognition) (Ronchetti ve Avancini, 2011). Kinect hem jest hem de ses komutları algılayabildiği için bilgisayarlarla doğal etkileşimde yaygın olarak kullanılan bir cihazdır.

2.1.1. Çok modlu doğal etkileşim

Çok modlu etkileşimin kesin bir tanımı olmamakla birlikte konuşma, jest, yazma ve diğer iletişim kanallarından birden fazlasını kullanarak yapılan girdilere uygun karşılık verilmesi şeklinde tanımlanmıştır (Jaimes A. ve Sebe N., 2007). Çok modlu

etkileşim kullanılarak bir sistem gerçekleştirilmesi üzerine bilinen en eski çalışma Bolt R. A. tarafından gerçeklenen “Put That There” sistemidir (1980). Çalışmada ekranda görünen nesneleri bir yerden başka bir yere taşımak için hem jest hem de ses komutlarını beraber kullanır. Kullanıcı büyükçe bir ekranın karşısına oturarak ekranda herhangi bir yeri gösterir ve “create a blue rectangle there” şeklinde komut vererek gösterilen yere mavi dörtgen çizilmesini sağlar. Benzer şekilde kullanıcı var olan şekilleri de taşıyabilmektedir. Örneğin mavi dörtgeni işaret parmağı ile gösterdikten sonra “move that” der ve taşıyacağı yeri parmağı ile gösterdikten sonra “there” diyerek şekli taşıyabilmektedir. Bu çalışma daha sonraları yapılan çok modlu etkileşim içeren çalışmalara öncülük etmiştir.

Vo ve Waibel (1993) ise jest ve ses ile komut verilebilir bir arayüz üzerine çalışma yapmışlardır. Çalışmada dokunmatik ekranlar üzerinde stylus kalem ile jestlerin algılanması ve yapay sinir ağları (neural networks) ile jestlerin sınıflandırılması yapılmıştır. Ses algılama kısmında ise çok durumlu zaman gecikmeli yapay sinir ağları (multi-state time delay neural network) kullanılmıştır. Detaylar Vo ve Waibel'in (1993) çalışmasında verilmiştir. Ardından da her iki etkileşim yöntemi birleştirilerek bir komut haline getirilir. Mesela okunan bir metinde bir kelime stylus kalem ile daire içine alınır ve ses ile İngilizce olarak "lütfen bu kelimeyi sil" komutu verilir. Bunun ardından daire içine alınan kelime silinir.

Çok modlu etkileşim yöntemi ile geliştirilen bir sistemin kullanılabilirliği üzerine yapılan çalışmaların en eskilerinden biri de Hauptmann'ın (1989) gerçekleştirdiği deneydir. Çalışmada 36 katılımcıdan jest ve ses komutları kullanarak bilgisayar ekranındaki küpü manipüle etmeleri istenmiştir. Manipülasyon şekilleri de döndürme (rotation), sağa sola kaydırma (transposition) ve büyütme/küçültme (scaling) şeklindedir. Katılımcılardan bu manipulasyon şekillerinden oluşan 7 görev tamamlamaları istenmiştir. Bu görevleri de 3 farklı iletişim modu (communication mode) ile gerçekleştirmişlerdir. Bunlardan birincisi sadece jestleri kullanıp ses ile etkileşimi kullanmadan komut vermek. İkincisi jestleri kullanmayıp sadece ses ile komut vermek. Üçüncüsü de hem jest hem de ses ile komut vermek olarak anlatılmıştır. Deneyin sonuçları ANOVA (Analysis of Variance) tekniği ile analiz edilmiş ve katılımcıların çoğunun görüntüyü manipüle etmede hem jest hem de ses kontrolünü tercih ettiği gözlenmiştir.

2.2. Jest Tabanlı Kontrol Cihazları

Jest tabanlı etkileşim (Gesture-based Interaction) insanlarla bilgisayarların yalnızca GUI kullanarak etkileşme şekline göre daha doğal bir etkileşim şekli sunmaktadır. Son yıllarda jest tabanlı etkileşim alanında birçok ürün ve çalışma ortaya konulmuştur. Bunlardan biri Sony firmasına ait Playstation Move (URL-15, 2014) cihazıdır (Şekil 2.1a ve Şekil 2.1b). Cihaz yine Sony tarafından dağıtılan kamera ile beraber kullanılır. Şekli tıpkı bir mikrofona benzeyen cihazın tepesindeki yuvarlak kısım farklı renklerde aydınlatılarak kamera tarafından takip edilmesi sağlanır. Ayrıca aygıtın içerisinde magnetometre, jiroskop ve akselerometre bulunmaktadır. Farklı sensörler yardımı ile yuvarlanma eksenini algılama, x, y ve z koordinatlarında hareketi algılama ve Playstation Move cihazının kameranın neresinde olduğunu algılama işlemleri yapılabilmektedir. Böylece cihaz imleci hareket ettirmek için kullanılabilmektedir. Cihaz Playstation cihazları ile uyumlu şekilde çalışabilmektedir ve haberleşme bluetooth üzerinden yapılmaktadır. Sony, Playstation Move cihazını akademik çalışmaları teşvik etmek ve yaygın kullanımını artırmak için Move.me (URL-10, 2014) yazılım sunucu uygulamasını kullanıma sunmuştur. Ancak sadece Playstation platformlarında geliştirmeye izin verilmektedir. Bundan dolayı alanyazında az sayıda üzerinde yapılmış çalışmaya rastlanmıştır (Tanaka et al., 2012), (Williamson et al., 2011).

(a) (b)

Şekil 2.1. Playstation a) Move kontrolcüsü b) Playstation Eye kamerası

Ticari olarak piyasaya sürülen diğer bir jest tabanlı ürün de Nintendo Wii (URL-23, 2014)'dir. Şekil 2.2a ve Şekil 2.2b'de görüldüğü gibi daha çok televizyon kumandasına benzeyen bir görünüşü vardır. Playstation Move ile benzer şekilde akselerometre, jiroskop, kızılötesi sensör gibi sensörlere sahiptir. Playstation Move'dan farklı olarak kamera ile değil kızılötesi sensör barı ile cihazın konumu çıkarılmaktadır. Haberleşme yine bluetooth üzerinden yapılmaktadır. Wii Remote cihazı farklı platformlar için geliştirmeye açık olduğu için alanyazında özellikle sağlık&rehabilitasyon ve eğitim alanında çalışmalar yapılmıştır (Gallo ve Ciampi, 2009), (Şimşek ve Durdu, 2012).

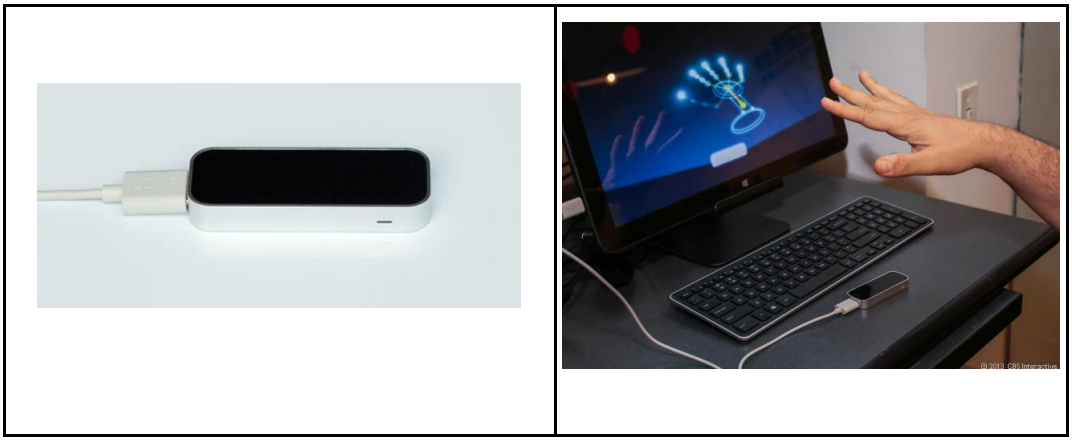
(a) (b)

Şekil 2.2. Wii a) Remote b) Wii kızılötesi sensör barı

Thalmic Labs (URL-20, 2014) tarafından geliştiriliyor olan ve 2014 ikinci yarısında piyasaya sürülecek olan MYO kol bandı (Şekil 2.3) da diğer bir jest tabanlı etkileşim sağlayan üründür. Ürün kalınca bir bilezik şeklinde dirsek ile el arasında dirseğe yakın tarafa takılarak kullanılır. Farklı el ve kol hareketleri yaptıkça kol üzerindeki kasların elektromiyografik aktivitesini ölçerek hangi jestin yapıldığını tespit eder. Cihaz haberleşmek için düşük güç tüketimi sağlayan Bluetooth 4.0 protokolünü kullanmaktadır. Önümüzdeki aylarda yazılım geliştirme kitinin geliştiricilere dağıtılması ile MYO ile ilişkili çalışmaların artması beklenmektedir.

Şekil 2.3. MYO kol bandı

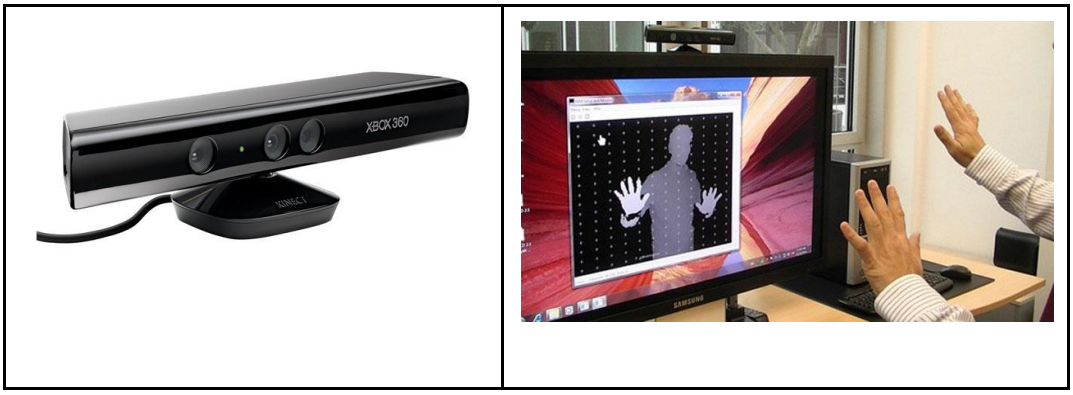
Bir diğer jest tabanlı kontrol cihazı da Leap Motion cihazıdır (URL-8, 2014). Cihaz Şekil 2.4b'de görüldüğü gibi küçük ve taşınabilirdir. Cihazın içerisinde yukarı doğru bakan iki adet monokromatik kızılötesi kamerası ve üç adet kızılötesi led bulunmaktadır. Cihaz yukarıya doğru 1 metrelik yarı küresel bir alanı tarayarak bu alanda meydana gelen hareketleri gözlemler. Leap Motion ile geliştirilen uygulamalar Airspace (URL-8, 2014) adı verilen bir uygulama mağazası üzerinden satışa sunulmuştur.



(a) (b)

Şekil 2.4. Leap Motion a) jest tabanlı kontrol cihazı b) kullanım esnasında

Evoluce firması, Kinect kullanılarak geliştirilmiş ticari bir medya navigasyon yazılımı geliştirmiştir. Bu yazılım, görüntü, video, PowerPoint sunum ve pdf gibi medyaların temassız şekilde jest tabanlı kontrol edilmesine olanak sağlamaktadır. Ayrıca 2011 yılında firma Kinect SDK'sına alternatif bir SDK yayınlamıştır (URL-2, 2013).



(a) (b)

Şekil 2.5. Kinect a) jest tabanlı kontrol cihazı b) kullanım esnasında

Microsoft'un 2010 yılında piyasaya sürdüğü Kinect jest tabanlı etkileşim sağlayan cihazlar arasında en başarılı ve en popüler olanıdır. Öyle ki cihazın ilk çıktığı günden 60 gün içinde 8 milyon, ilk çıktığı günden 24 şubat 2013 tarihine kadar 24 milyon adet satılmasından anlaşılabilir (URL-7, 2014). Cihazı kullanmak için iki ayrı yazılım kütüphanesi mevcuttur. Biri Microsoft firmasının çıkarttığı ve platform bağımlı yazılım geliştirme kütüphanesi (URL-9, 2014), diğeri de PrimeSense (URL-16, 2014) firmasının dağıttığı açık kaynak kodlu yazılım geliştirme kütüphanesidir. Ancak 24 Kasım 2013 tarihinde PrimeSense'in Apple firması (URL-1, 2014) tarafından satın alınması ile birlikte kütüphanenin kaynak kodları kapatılmıştır (URL-14, 2014). Bir sonraki kısımda bahsedilecek olan alanyazın çalışmalarda Kinect yoğun şekilde kullanılmıştır. Bunun gerekçesi ise zengin geliştirme seçeneği ve farklı platformlarda yazılım geliştirmeye olanak sağlaması olarak verilebilir. Şekil 2.5a ve Şekil 2.5b'de cihazın kullanım esnası görülebilir.

2.3. Jest Tabanlı Kontrol Cihazları ile Doğal Etkileşim Uygulamaları

Yukarıda bahsedilen jest tabanlı kontrol cihazları ile doğal etkileşim ve jest tabanlı etkileşim daha çok, sağlık, eğitim vb. alanlarında araştırmacılar tarafından etkinlikleri değerlendirilmiştir. Bu çalışmaların bazıları sadece jest komutları alabiliyorken bazıları da hem jest hem de ses ile komut alabilmeyi yeteneğine sahiptir. Aşağıda jest tabanlı kontrol cihazlarının bu alanlarda kullanıldığı çalışmalara değinilmiştir.

Gallo ve arkadaşları (2008) Wii Remote kullanarak üç boyutlu medikal verileri manipule edecek bir sistemi geliştirmişlerdir. Sistem MR (magnetic resonance) ve CT (computed tomography) görüntülerinden oluşturulan üç boyutlu görüntüleri kontrol etmek için var olan etkileşim yöntemlerine alternatif bir yöntem sunmuştur. Sunulan yöntemde oluşturulan üç boyutlu görüntülerle etkileşim kurmak için Wii Remote kumandasındaki tuşlarla beraber belirli hareketler yapılmış ve görüntü ile bu şekilde etkileşim kurulmuştur. Çalışmanın klinikte eğitim maksatlı kullanılabileceğine değinilmiştir.

Gallo ve arkadaşlarının (2011) diğer bir çalışması da 2008 yılında Wii Remote kumandası ile gerçekledikleri sistemin bir benzerini Kinect ile gerçeklemişlerdir. Açık kaynak kodlu Kinect kütüphanesini kullanarak MR ve CT görüntülerini üç boyutlu görüntü formatına dönüştürmeden sadece el kol hareketleri ile kontrolü sağlanmıştır. Sterilizasyonun önemli olduğu ameliyathane gibi ortamlarda dokunma gerektirecek herhangi bir aygıta temas etmek için steril ortamı terk etme ihtiyacı doğmaktadır. Bu da ameliyatın sürekliliğini sekteye uğratmaktadır. Kinect kullanarak herhangi bir girdi aygıtına dokunma ihtiyacı olmadan medikal görüntüler kontrol edilebilmektedir. Çalışmanın özellikle bu bağlamda önemli olduğu ileri sürülmektedir.

Bin Sidik ve arkadaşları da (2011) Kinect kullanarak insan vücut hareketleri ile doğal etkileşim (natural interaction) geliştirilmesi üzerine çalışma yapmışlardır. Çalışmalarında insan vücut hareketlerinin fonksiyonel bir taksonomisine yer verilmiş olup bunlar genel bir yapı içerisinde sunulup özetlenmiştir. Ardından Kinect'in başarılı bir doğal etkileşim sistemi olduğundan bahsedilmiş ve yakın zamanda yapılmış araştırmalara değinilerek çalışma sonuca bağlanmıştır.

Medikal alanda yapılan diğer bir çalışma da Tuntakurn ve arkadaşları (2012) tarafından gerçeklenmiştir. Kinect kullanılarak üç boyutlu medikal görüntüleme yazılımı geliştirilmiş ve bu yazılım üzerinde el ve parmak jestleri ile üç boyutlu medikal görüntünün kontrol edilmesi sağlanmıştır. Gallo ve arkadaşlarının (2011) çalışmasında olduğu gibi bu çalışmada da ameliyathane gibi ortamlarda klavye ve fare'ye dokunmaya ve ameliyathaneyi terk etmeye gerek kalmadan medikal görüntünün incelenmesine olanak sağlanmaktadır. Geliştirilen sistemde iki arka iş

parçacığı (background thread) ve ana iş parçacığı (main thread) olmak üzere üç iş parçacığı bulunmaktadır. Bunlardan birisi Kinect üzerinden görüntülerin alınmasından diğeri de görüntülerin işlenmesinden sorumludur. Ana iş parçacığı da görüntü ile etkileşimin sağlanmasından sorumludur. Gerçeklenen sistemde hem vücut hem de parmak jestlerinin bir arada kullanıldığına vurgu yapılmaktadır.

Kinect cihazının rehabilitasyon alanındaki çalışmalarından biri de Yeh ve arkadaşlarının (2012) felçli hastaların daha hızlı iyileşmesine yönelik geliştirdikleri egzersiz uygulamasıdır. Geleneksel rehabilitasyon ekipmanları büyük, pahalı ve erişilmesi güçtür. Bu yüzden felç hastalarının üst ekstermite (el ve kollar) egzersizlerini ev ortamında gerçekleştirmek üzere sanal ortamda top yakalama ve topa erişme gibi görevler tanımlanmıştır. Bu tanımlanan görevler de bir oyun çerçevesinde hastaya sunulmuştur.

Kinect gibi temassız jest kontrol cihazı kullanmadan sadece RGB kamera ile steril ortamda temassız medikal görüntü kontrolünü sağlamak üzere yapılan çalışmalara örnek olarak Wachs ve arkadaşlarının (2007) gerçeklediği sistem örnek verilebilir. Wach ve arkadaşları (2007) tarafından kullanıcılar için steril bir jest arayüzü (gesture interface) geliştirildiği belirtilmiştir. Medikal görüntüleri manipüle etmek için kullanıcı el hareketlerine ait renk-hareket işaretleri (color-motion cue) kullanılmıştır. Gerçeklenen sistem medikal görüntüyü navige etme, döndürme ve yakınlaştırıp uzaklaştırma yeteneklerine sahiptir.

Lange ve arkadaşları (2012) tarafından gerçeklenen sistem de oyun tabanlı rehabilitasyon çalışmalarına örnek olarak verilebilir. Kinect jest kontrol cihazı kullanılarak geliştirilen "JewelMine" adındaki oyun ile statik denge geliştirme egzersizleri yapılabildiği belirtilmiştir. Oyunda bir tren rayı üzerinde giderken sabit oyuncunun etrafında bulunan mücevherlere uzanarak toplamaları beklenmektedir. Kullanım sonrasında kullanımı analiz etmek üzere sofistike bir analiz aracı tasarlanacağından bahsedilmiştir.

Schwaller ve arkadaşlarının (2010) geliştirdiği Protatif Jest Arayüzü olarak tanımlanan (PyGmI - Portable Gesture Interface) araç bilinen jest tabanlı kontrol cihazlarından biraz farklıdır. Kullanıcı üzerine portatif bir ekipman giyer. Bu ekipmanda göğsünden önüne doğru bakan bir kamera ve bel kısmında önüne doğru

bakan bir projektör bulunur. Her iki elin işaret ve baş parmağının ucuna renkli işaretçiler giyilir. Kamera yardımı ile her iki elin iki parmağı ile yapılan jestler yorumlanarak projektör ile yapılan işlemler yansıtılır.

Wilson (2010) Kinect cihazı kullanarak dokunma sensörü (touch sensor) geliştirmiştir. Kinect kamerası dokunmatik alan haline getirilecek yüzeye bakacak şekilde ayarlanması ile yazılıma entegre edilmesi ile sistem çalışır. Derinlik çözünürlük hassasiyeti noktasında cihazın belirli limitleri olması sebebi ile kapasitif bir ekran kadar hassas çalışamamaktadır. Ancak derinlik kamerası ile dokunma sensörü oluşturmanın avantajları vardır. Bunlar ekranın dokunmatik özellikli olmasına gerek duyulmaması ve düz olmayan (non-flat) yüzeyleri de dokunmatik alan haline getirilebilir olmasıdır.

Rydén ve arkadaşları (2011) Kinect'i robotik ve tele-robotik ameliyatlarda güvenlik amaçlı kullanmışlardır. Yaptıkları çalışmada amaçları kaçınılması gereken ameliyat bölgelerini bir haptik cihaz yardımı ile sakındırılmasıdır. Önceki benzer çalışmalardaki eksiklik olarak MR ve CT görüntülerinden elde edilen sanal fikstürlerin (fixtures) gerçek ortamla ilişkilendirilme zorluğuna değinilmiş. Ardından çalışmada bu problemin Kinect ile aşıldığı dile getirilmiş. Cihaz tarafından üretilen üç boyutlu ve gerçek zamanlı ortamlarda haptik kuvvetlerin başarılı şekilde işlendiği (render) belirtilmiştir.

2.4. Doğal Etkileşim ve Jest Tabanlı Etkileşimde Kullanılabilirlik Değerlendirmesi Çalışmaları

Kullanılabilirlik çalışması olarak bu tez kapsamında detayları daha ileriki kısımlarda verilen kullanılabilirlik testi ve kullanılabilirlik anketi uygulanmıştır. Alanyazında de ses ve jest kontrolü üzerinde yapılan çalışmaların kullanılabilirlik, etkililik ve verimliliğini ölçen farklı çalışmalar mevcuttur.

Kirmizibayrak ve arkadaşları (2011) üç boyutlu medikal görüntüleri hem jest tabanlı etkileşim yöntemi hem de fare ile kontrolünü sağlayacak bir sistem hazırlamıştır. Ardından her iki girdi yöntemi ile iki farklı kullanıcı çalışması yapılmıştır. Birinci deneyde katılımcılara üç boyutlu medikal görüntüyü döndürme görevleri verilerek jest tabanlı kontrol cihazı ile farenin performansı karşılaştırılmıştır. İkinci deneyde de

üç boyutlu medikal görüntüde bulunan iç yapıların yerlerinin belirlenmesi görevleri her iki girdi yöntemi ile denenmiştir. Deney sonuçlarına göre jest tabanlı kontrol cihazı fareye göre hem doğruluk hem de zaman açısında daha üstün performans gösterdiği belirtilmiştir. İkinci deneyde doğruluk oranı fare girdi aygıtı ile daha yüksek çıktığı belirtilmiştir. Bununla Sihirli Lens (Magic Lens) (Kirmizibayrak 2011) kullanılarak hedef yeri belirleme işlemi jest tabanlı kontrol cihazı ile birlikte en hızlı şekilde gerçekleştirilmiştir.

Haartman ve Schlaefer (2013) yaptıkları çalışmada ameliyathane ışıklarının dokunmadan bağımsız şekilde kontrol edilmesinin uygulanabilir (feasible) olup olmadığı araştırılmıştır. Ameliyathane ışıkları robot bir kola bağlanmış ve kullanıcı etkileşimlerini gözlemek için Kinect kullanılmıştır. Çeşitli jestlerle de ışık kaynağının kontrolü sağlanmıştır. 18 katılımcı tarafından denenen sistem için tüm katılımcılar rahatça sistemi kullanıp düz bir yüzeye ışığı yöneltmeyi kolayca öğrenebildiğini belirtmiştir. 12 katılımcılar jest kontrolünü kolay bulmakla birlikte katılımcıların büyük çoğunluğu sistemi verimli bulduğunu belirtmiştir.

Martins ve arkadaşları (2012) Second Life (URL-18, 2014) adındaki üç boyutlu sanal dünya simülatörü üzerinde 13 farklı görev hem klasik klavye ve fare ile, hem 3DConnection firmasına ait Space Navigator ve yine aynı firmaya ait Space Pilot 3B fareleri ile 10 katılımcı tarafından denenmiştir. Katılımcıların hepsi omurganın farklı bölgelerinde (C5-D11) medüller lezyon (medullary lesions on vertebrae C5-D11) kaynaklı fiziksel engeli bulunan engellilerden oluşmaktadır. 5 katılımcı klavye ve fare kullanırken diğer 5 katılımcı 3B fareyi kullanmıştır. Sonuç olarak klavye ve fare ile gerçekleştirilen en zor 5 görevin 2-3 tanesi 3B fare ile daha kolay yapılabilmiştir. Katılımcıların görüşlerine göre 3B fareye daha pozitif yaklaştıkları gözlenmiştir.

Cuccurullo ve arkadaşları (2012) Microsoft PowerPoint sunumlarını hem Wip adını verdikleri kablosuz kumanda hem de geliştirdikleri KiP adını verdikleri Kinect tabanlı doğal kullanıcı arayüzü (natural user interface - NUI) ile kontrol edilmesi üzerine kullanılabilirlik çalışması yapmışlardır. Kullanılabilirlik çalışmasında ise iki farklı görev iki girdi yöntemi ile katılımcılar tarafından denenmiştir. Deneme sürelerine göre her iki yöntem için ampirik analiz uygulanmıştır. Ardından katılımcılardan işe yararlık (usefulness), kullanım kolaylığı (easy of use), öğrenim

kolaylığı (easy of learning) ve memnuniyet (satisfaction) derecelerini araştıran bir anket doldurmaları istenmiştir. Anket sonuçlarına göre KiP kullanım memnuniyeti daha yüksek, ampirik analiz sonucuna göre de KiP, WiP'ye göre daha performanslı çıkmıştır.

Ryu ve arkadaşları (2011) kızılötesi yakınlık sensörü (Infrared Proximity Array sensor) kullanarak T-less fare adını verdikleri el ve parmak jestleri ile işaret aygıtı (pointing device) geliştirmişlerdir. Ayrıca geliştirilen girdi yöntemi ile klasik fare ile karşılaştırarak kullanılabilirlik analizi uygulanmıştır. ISO 9241-9 (2000) dökümanında bahsedilen Fitts' kanununa göre 18 katılımcının kullanım verimliliği (throughput) ölçülmüştür. Bu ölçümlerin ardından yine ISO 9241-9 (2000) dökümanında sunulan kullanıcı memnuniyet anketi katılımcılar tarafından doldurulmuştur. Sonuçta T-less fare kullanarak işlemleri yapmak klasik fare ile işlemleri yapmaktan ortalamada daha uzun sürmüştür. Ancak T-less fare ile hatalı işlem sayısı klasik fareden çok farklı çıkmamıştır. T-less fare böylelikle klasik farenin kullanılamadığı durumlarda kullanışlı bir alternatif olabileceğinden bahsedilmiştir.

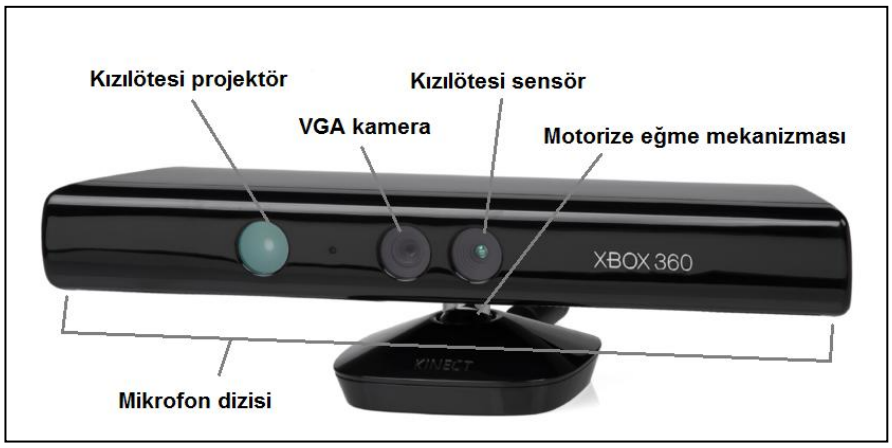
Natapov ve arkadaşları (2009) Nintendo oyun kumandası, Nintendo Wii Remote ve klasik fare ile işaretleme ve seçme işlemlerinden oluşan belirli görevleri 15 katılımcıya denetmiştir. Deneme süreci ISO 9241-9 (2000) dokümanında belirtilen yönergelere uygun şeklide gerçekleştirilmiş, öncelikle girdi aygıtlarının verimliliği (throughput) analiz edilmiş kullanım sonrasında da katılımcılardan memnuniyet anketi doldurmaları talep edilmiştir. Analiz sonucunda fare 3,78 bps (bit per second) ile en yüksek verimliliğe sahip aygıt olarak seçilmiştir. Wii Remote 2,59 bps ve Nintendo oyun kumandası da 1,47 bps ile en düşük verimliliğe sahip olduğu gözlenmiştir. Wii Remote aygıtı Nintendo oyun kumandasına göre %75 daha yüksek verimliliğe sahip olduğu gözlenmiştir. Girdi aygıtlarının hata oranların da fare, Nintendo oyun kumandası ve Wii Remote için sırasıyla %3,53, %6,58 ve %10,2 olarak ölçülmüştür.

3. GERÇEKLENEN SİSTEM

Bu kısımda teze konu olan ve kullanılabilirlik değerlendirmesi yapılan yazılımın gerçekleme süreçleri ve detaylarından bahsedilecektir. Öncelikle Kinect'in teknik özelliklerine değinilecek ardından uygulamanın arayüzünden bahsedilecektir. Bunun arkasından yapılan araç ve ortam seçiminden bahsedilecektir. Sonrasında gerçeklenen yazılımın fonksiyonlarına değinildikten sonra gerçekleme detayları olarak yazılım modüllerinden, yazılım sınıf ve bileşenlerinden bahsedilecektir.

3.1. Kinect'in Teknik Özellikleri

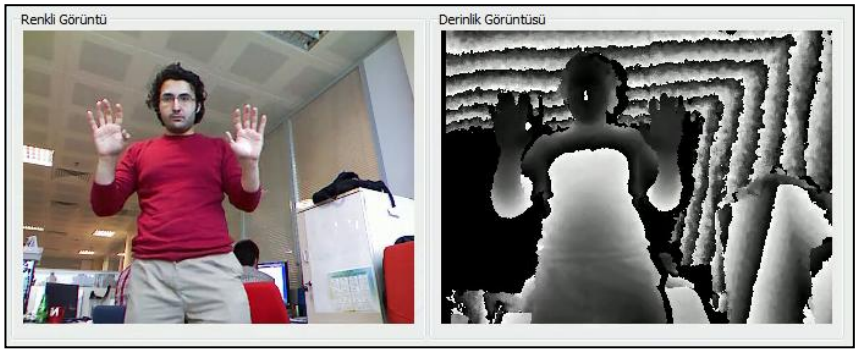
Kullanıcı tarafından yapılan jestleri tanımak üzere kullanılan girdi aygıtı Kinect'tir. Şekil 3.1'de görüldüğü gibi aygıtta bir adet RGB kamera ve paralel dizilmiş mikrofonlar, bir adet lazer tabanlı kızılötesi projektör ve monokrom CMOS (Complementary metal–oxide–semiconductor) sensörü bulunmaktadır.



Şekil 3.1. Microsoft Kinect

Kinect üzerinde yer alan projektör sahneye kızılötesi ışın demetleri yollar ve aygıt üzerindeki kızılötesi sensör de bu ışın demetleri üzerinden mesafe ölçümü yaparak sahnenin derinlik görüntüsünü çıkarır. Bu yöntem ile stereo kameralarda karşılaşılan maliyetli uyuşma/örtüşme problemi (correspondence problem) elimine edilmiş olur. Şekil 3.2a'da Kinect üzerindeki RGB kamera ile alınmış bir görüntü ile Şekil 3.2b'de Kinect ile oluşturulmuş bir derinlik resmi bulunmaktadır. Derinlik görüntüsü siyah

renkten beyaz renge doğru tonlama yapılarak gösterilmiştir. Ayrıca aygıtın yatay eksende eğilmesini sağlayan motorize bir mekanizma da mevcuttur. Aygıtın yatay görüş açısı 57º, dikey görüş açısı 43º'dir. Kızılötesi projeksiyon kısıtından dolayı ancak 0,8m-3,5m arasındaki derinlik bilgisi çıkarılabilmektedir. Ayrıca sensörün 2 metre uzaklığında X ve Y eksenlerdeki çözünürlüğü 3mm, Z eksenindeki çözünürlüğü ise 10mm olacak şekilde tasarlanmıştır (Gallo ve diğ., 2011).

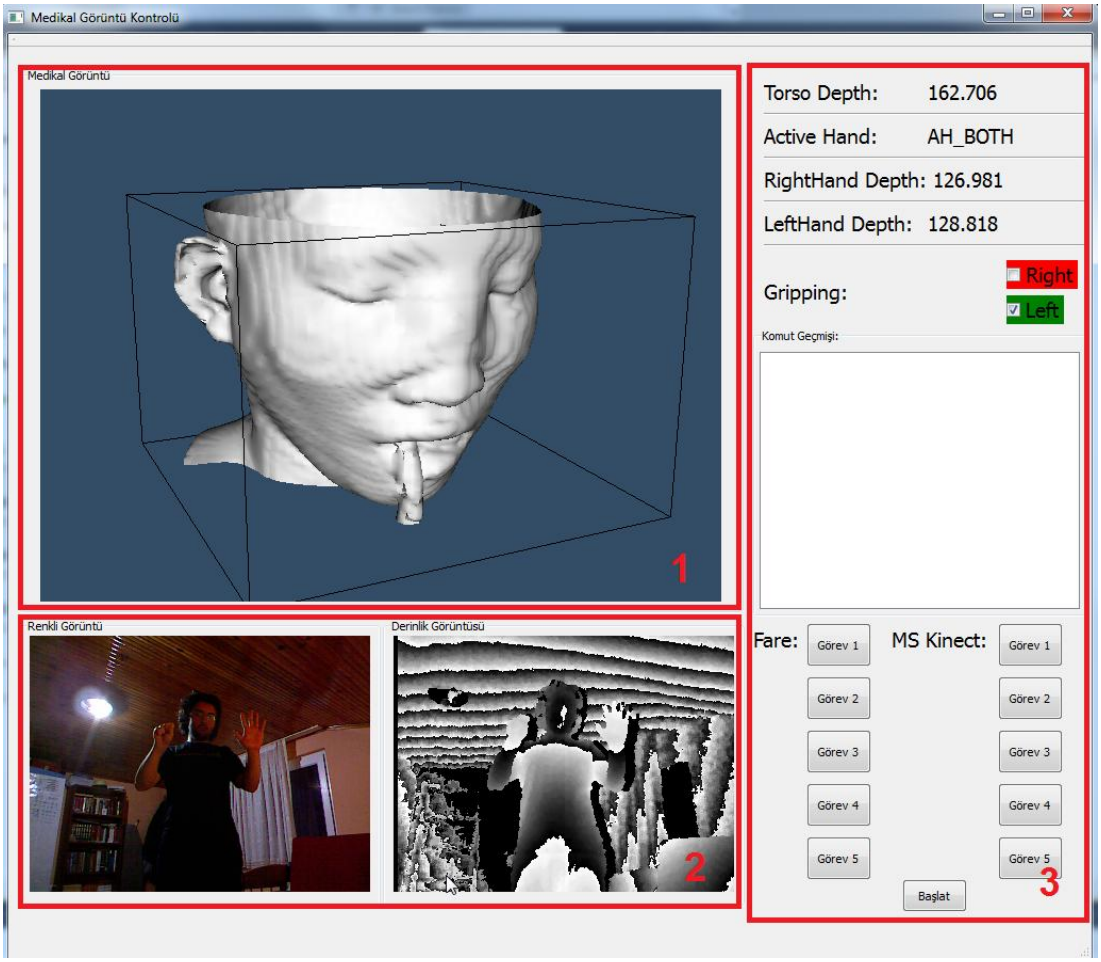


(a) (b)

Şekil 3.2. Kinect ile elde edilmiş a) RGB kamera ve b) derinlik görüntüsü

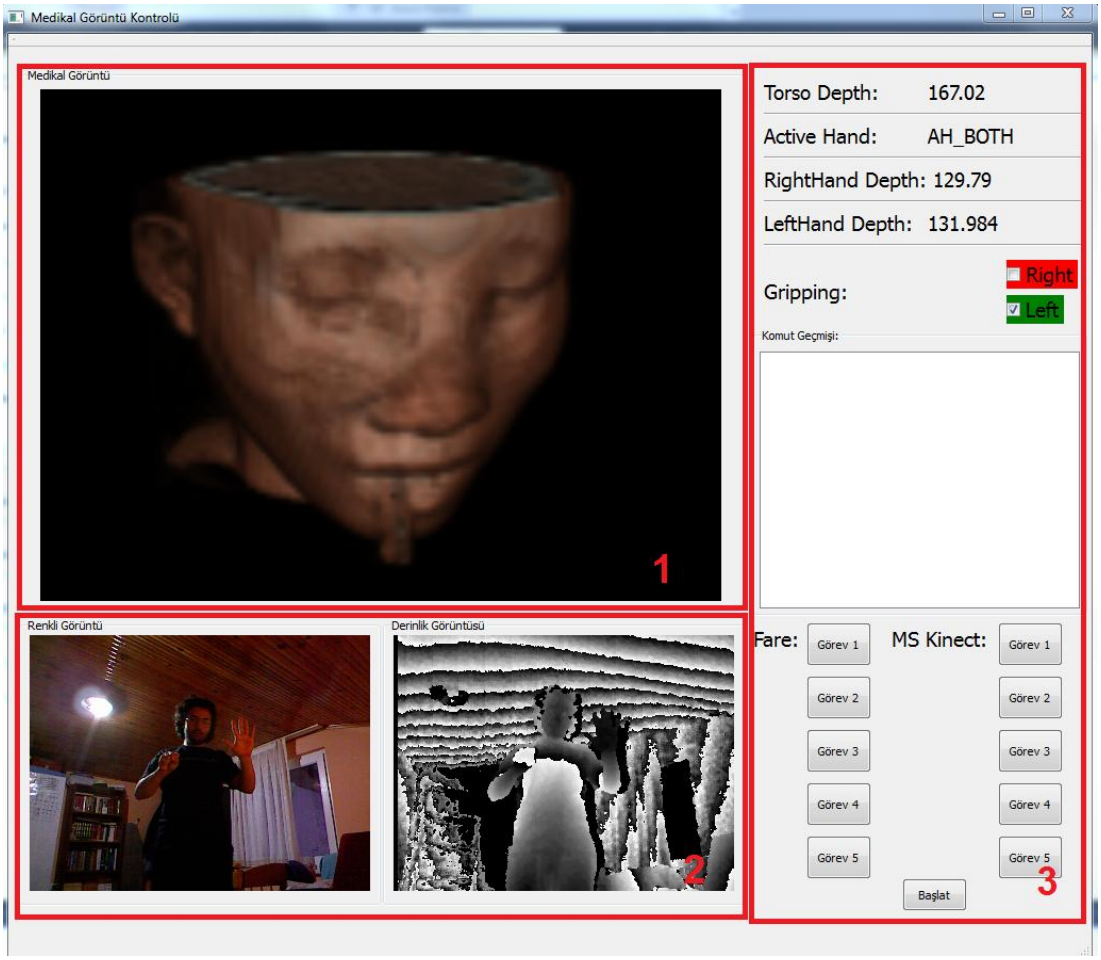
3.2. Uygulama Arayüzü

Üç boyutlu görüntülerin navigasyonu için geliştirilen arayüz Şekil 3.3'te gösterilmiştir. Arayüz işlevleri üç ana kısımda toplanmıştır. Birincisi üç boyutlu görüntünün gösterildiği kısım, ikincisi Kinect aygıtından alınan ve kullanıcı jestlerinin gösterildiği video ve derinlik görüntüleri, üçüncüsü de kullanıcıya geribildirimde bulunmak için komut geçmişi ve kullanıcının anlık durumu hakkında bilgilendirme ekranıdır. Uygulamada klavye ve fare kullanılarak gerçekleştirilen hiçbir komut yoktur ekrandaki butonlar kullanılabilirlik testinin yapılabilmesi için eklenmiştir. Etkileşim tamamıyla ses ve jestler ile sağlanmaktadır.



Şekil 3.3. Uygulama Arayüzü (izoyüzey görüntüsü)

Şekil 3.3'te görülen üç boyutlu görüntü iki boyutlu görüntülerden oluşturulmuş izoyüzey (isosurface) verisidir. İzo yüzey verisi basınç, sıcaklık, hız ve yoğunluk gibi sabit bir verinin üç boyutlu uzaydaki noktalarını ifade eder (URL-5, 2014). Şekil 3.4'te görülen görüntü ise bir hastaya uygulanan CT (computed tomography – bilgisayarlı tomografi) (URL-24, 2014) taraması sonucunda yansıyan x-ışınlarının oluşturduğu nokta uzayından elde edilmiştir. Bu görünüme ek olarak uygulamada üç boyutlu görüntüler ayrıca hacim kaplama (volume rendering) şeklinde görselleştirilebilmektedir. Şekil 3.4'te görüldüğü gibi üç boyutlu nesnenin hacimce resmedilmiş şekli kontrol edilebilmektedir.



Şekil 3.4. Uygulama Arayüzü (hacim kaplama görüntüsü)

3.3. Geliştirme İçin Yapılan Araç ve Ortam Seçimi

Planlanan sistemi geliştirmek için üç boyutlu görüntülerin görüntülenmesinde kullanılan arayüz, kullanılacak aygıt ile bağlantı kurulabilmesi için gerekli bir sürücü, görüntülerin aygıttan alınması ve diğer yardımcı işlemler için yardımcı kütüphaneler ve geliştirilmenin yapılacağı ortam gereklidir. Hâlihazırdaki kütüphane ve araçlar incelenerek birkaç farklı yolla bu sistemin geliştirebileceği tespit edilmiştir. Bunlar, Slicer3D (URL-19, 2014) adında bir üç boyutlu görüntüleme aracına plug-in yazmak, sistemin arayüzü olarak Qt (URL-17, 2014) ve üç boyutlu görüntülerin görüntülenmesi için VTK (Visualization Toolkit) (URL-22, 2014) kütüphanesinin açık kaynak kodlu Kinect sürücüsü olan OpenNI (URL-14, 2014) kütüphanesini kullanmak ve son olarak da yine Qt ve VTK'yı Kinect SDK'sı ile birlikte kullanmaktır. Birinci yöntem sistemin arayüzü Slicer3D adında bir üç boyutlu görüntüleme aracına plug-in yazarak Kinect ile haberleşebilir hale

getirilmesini içermektedir. Aygıt ile iletişim kurulabilmesi için kullanılan sürücü açık kaynak kodlu Kinect sürücüsü olan OpenNI kullanılır. Yardımcı kütüphane olarak OpenCV (URL-13, 2014) ve Slicer3D programına plug-in yazabilmek için öncelikle cmake aracı kullanılarak plug-in projesi oluşturulması gerekmektedir. Plug-in projesi de versiyon kısıtından dolayı ancak Visual Studio 2008 ortamında yapılabilmektedir. Bu yöntemle ilk aşamada yazılımı geliştirmek daha zahmetsiz gözükse de Slicer3D programına plug-in yazma kısıtları ve plug-in oluşturma projesi üretimi ve geliştirme ortamı versiyon kısıtları bulunmaktadır.

İkinci yöntem olarak sistemin arayüzü Qt ve üç boyutlu görüntülerin görüntülenmesi için VTK (Visualization Toolkit) (URL-22, 2014) kütüphanesi, aygıt ile bağlantı kurulması için açık kaynak kodlu Kinect sürücüsü OpenNI (URL-14, 2014) ve geliştirme ortamı olarak Visual Studio 2010 seçilebilir. Böylelikle açık kaynak kodlu ve platformdan bağımsız yazılım geliştirilebilmektedir. Ancak OpenNI kütüphanesi ses ile komut verme yeteneğine sahip değildir. Bu bağlamda Kinect SDK'sı, Microsoft Speech SDK'sı ile entegre çalışması sayesinde önemli bir avantaja sahiptir (URL-11, 2014).

Son yöntem olarak sistemin arayüzü Qt, üç boyutlu görüntülerin görüntülenmesi için VTK kütüphanesi, aygıt ile bağlantı kurulması için Kinect SDK ve sürücüsü ve geliştirme ortamı olarak Visual Studio 2010 kullanılabilir. Bu yöntem ihtiyaç duyulan ses komutlarını yorumlayacak bir yazılım geliştirme kütüphanesini içinde barındırmaktadır. Microsoft Speech SDK'sı da Kinect SDK'sı ile sorunsuz şekilde çalışabildiği için bu SDK tercih edilmiştir. Bununla beraber Kinect SDK'sı, açık kaynak kodlu Kinect SDK'sına nazaran daha hızlı ve istikrarlı iskelet takibi yapabilmektedir (Shotton ve diğ., 2013). Bu yöntemin tek dezavantajı platform bağımlılığını da beraberinde getirmesidir.

Bahsedilen tüm bu yöntemlerin sahip oldukları avantaj ve dezavantajlar Tablo 3.1'de özetlenmektedir. Bu çalışmada geliştirme için Qt ve VTK'yı Kinect SDK'sıyla beraber kullanmaya bağlı olan son konfigürasyon kullanılmaktadır.

Tablo 3.1. Geliştirme yöntemlerinin avantaj ve dezavantajları

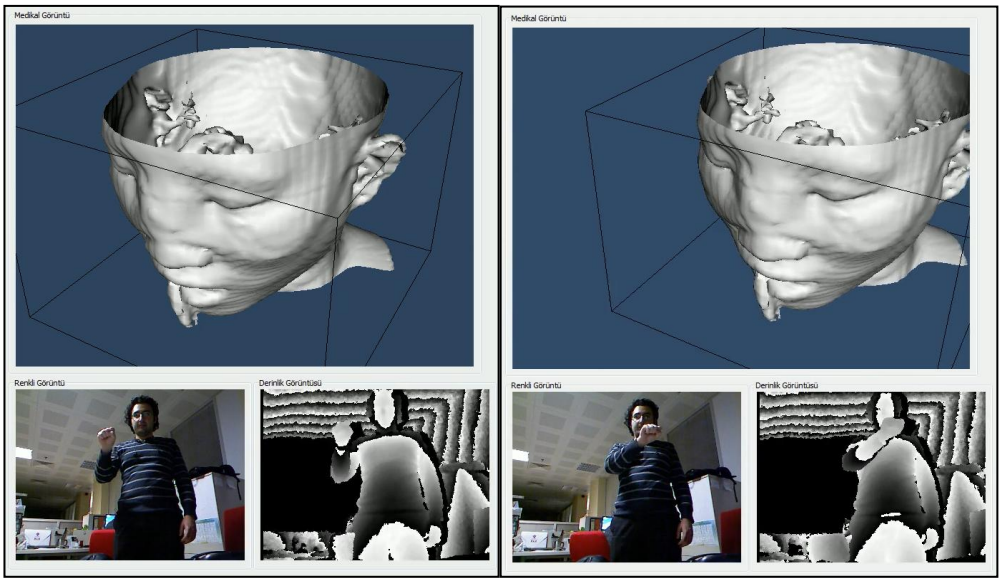
Geliştirme Seçenekleri	**Platform Bağımlılığı**	**Ses ile Etkileşim**	**Daha az kod ile geliştirme**
Slicer3D + OpenNI + VS_2008	**yok**	yok	**evet**
Qt + VTK + OpenNI + VS_2010	**yok**	yok	hayır
Qt + VTK + Kinect SDK + VS_2010	var	**var**	hayır

3.4. Gerçeklenen Yazılımın Fonksiyonları

Gerçeklenen sistem kullanıcıların tek eli aktifken veya iki eli aktifken jest komut alabilmektedir. Tatbik edilen jestler kullanılan elin yönünden bağımsızdır; sol veya sağ el ile komut verilebilmektedir. Jest komutlarının yanında ayrıca sisteme ses ile de komut verilebilmektedir.

Kullanıcı ayakta durur pozisyonda Kinect sensörünün görüş açısı içerisine girdikten sonra sensör kullanıcıyı takip etmeye başlayarak iskelet bilgilerini çıkaracaktır. Bundan sonra kullanıcı sensörün karşısında sesle İngilizce olarak "START" kelimesini telaffuz ettiğinde üç boyutlu görüntü kontrolü başlar.

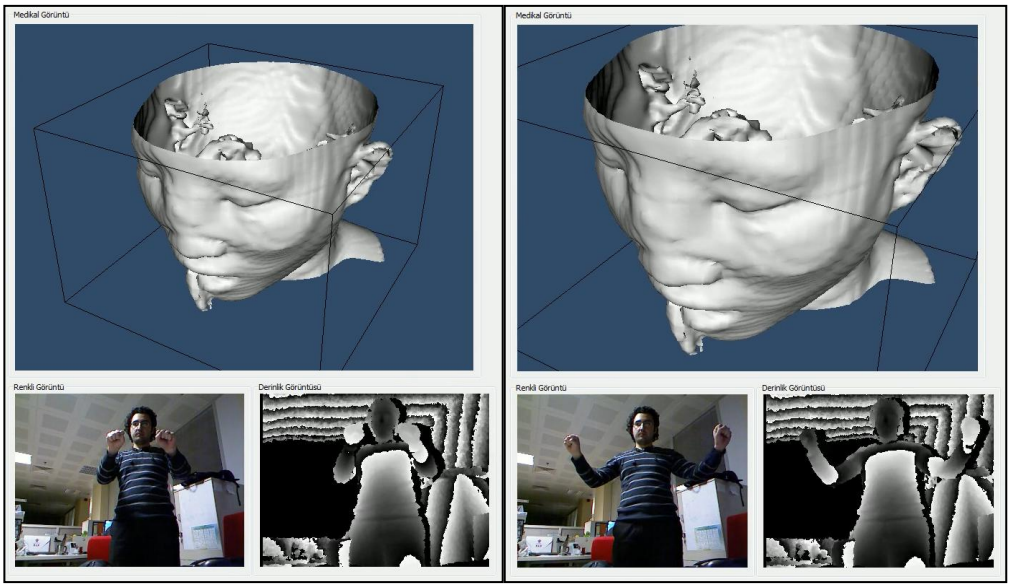
Üç boyutlu görüntülerin manipülasyonu daha önce de bahsedildiği gibi tutup sürükleme (pan), yakınlaştırıp uzaklaştırma (zoom) ve döndürme (rotate) işlemlerini kapsar.



Şekil 3.5. Sürükleme (Pan) işlemi

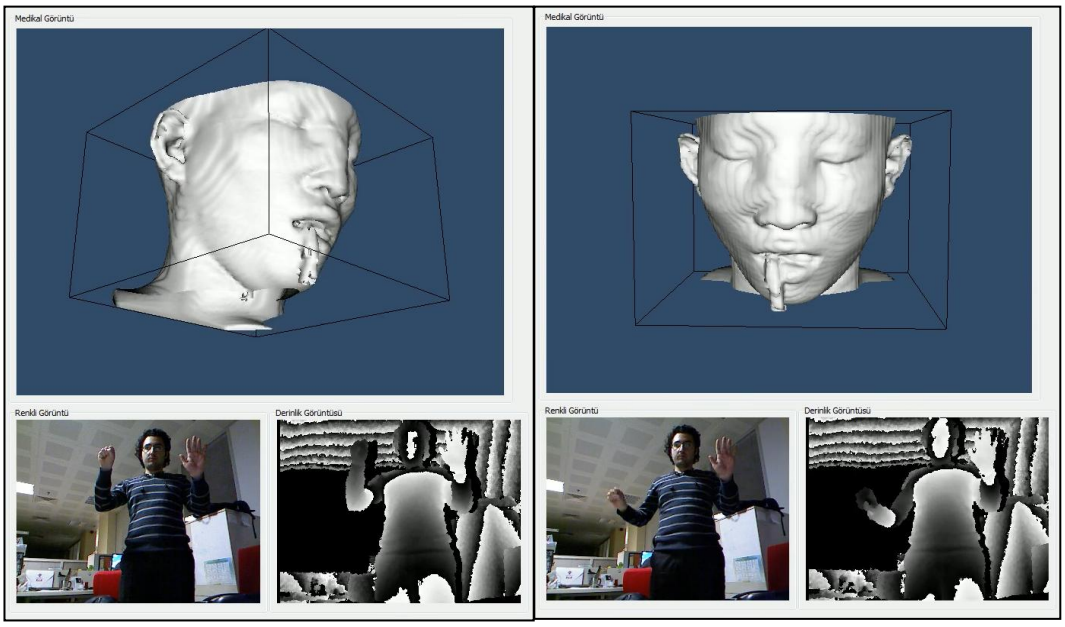
Tutup sürükleme (pan) işlemi hangi elin kullanıldığından bağımsız olarak tek el ile gerçekleştirilir Şekil 3.5'te görüldüğü gibi elle tutma (grip) işleminden sonra sola kaydırıldığında üç boyutlu görüntü sola, sağa kaydırıldığında üç boyutlu görüntü sağa, yukarıya kaydırıldığında üç boyutlu görüntü yukarı ve aşağı kaydırıldığında üç boyutlu görüntü aşağı doğru hareket eder. Elle tutma bırakıldığında etkileşim devre dışı kalır.

Yakınlaştırıp uzaklaştırma (zoom) işleminde iki el birlikte kullanılır. Şekil 3.6'da görüldüğü gibi her iki elle tutma işleminden (grip) sonra her iki el birbirlerinden yana doğru açılırsa üç boyutlu görüntü büyümekte, Her iki elle tutma işleminden sonra her iki el birbirlerine doğru yaklaştırılırsa da üç boyutlu görüntü küçülmektedir. Elle tutma bırakıldığında veya eller serbest şekilde aşağı doğru salındığında etkileşim devre dışı kalır.



Şekil 3.6. Yakınlaştırıp uzaklaştırma (Zoom) işlemi

Döndürme (rotate) işleminde de iki el birlikte kullanılır. Şekil 3.7'de görüldüğü gibi bir el açık şekilde dururken diğer elle tutarak (grip) sağa, sola, yukarı ve aşağı doğru hareket ettirilerek üç boyutlu görüntü ile etkileşim gerçekleşir.



Şekil 3.7. Döndürme (Rotate) işlemi

Yukarıdaki kontrollere ek olarak yanlış bir işlem yapıldığında üç boyutlu görüntüyü eski düzgün haline getirmek için sesle komut verilebilmektedir. Kullanıcı İngilizce olarak “RESET” kelimesini telaffuz ettiğinde bu işlem gerçekleştirilir. Yazılım başlatıldığında etkileşim aktif değildir. Kullanıcı İngilizce olarak “START”

kelimesini telaffuz ettiğinde üç boyutlu görüntü ile etkileşim aktif hale gelir. Kullanıcının yazılım ile işi bittiğinde de İngilizce olarak “CLOSE” komutu vererek sistemi sonlandırabilmektedir.

Özetle gerçeklenen sistem Kinect kullanılarak aşağıda belirtilen işlevlere sahiptir:

- “START” ses komutu verilerek üç boyutlu görüntü kontrolünün başlatılması
- “RESET” komutu verilerek üç boyutlu görüntünün ilk oryantasyonuna geri dönmesi
- “CLOSE” komutu verilerek sistemin kapatılması
- Tek el aktifken tutma işlemi ile sürükleme işlevinin gerçekleştirilmesi
- Çift el aktifken iki el birden tutma işlemi yaparak ellerin birbirinden uzaklaştırılması ile yakınlaştırma, iki el birden tutma işlemi yaparak ellerin birbirine yakınlaştırılması ile uzaklaştırma işlevinin gerçekleştirilmesi
- Çift el aktifken tek el tutar pozisyonda ve diğer el açık pozisyonda iken tutan elin hareket ettirilmesi ile döndürme işlevinin gerçekleştirilmesi

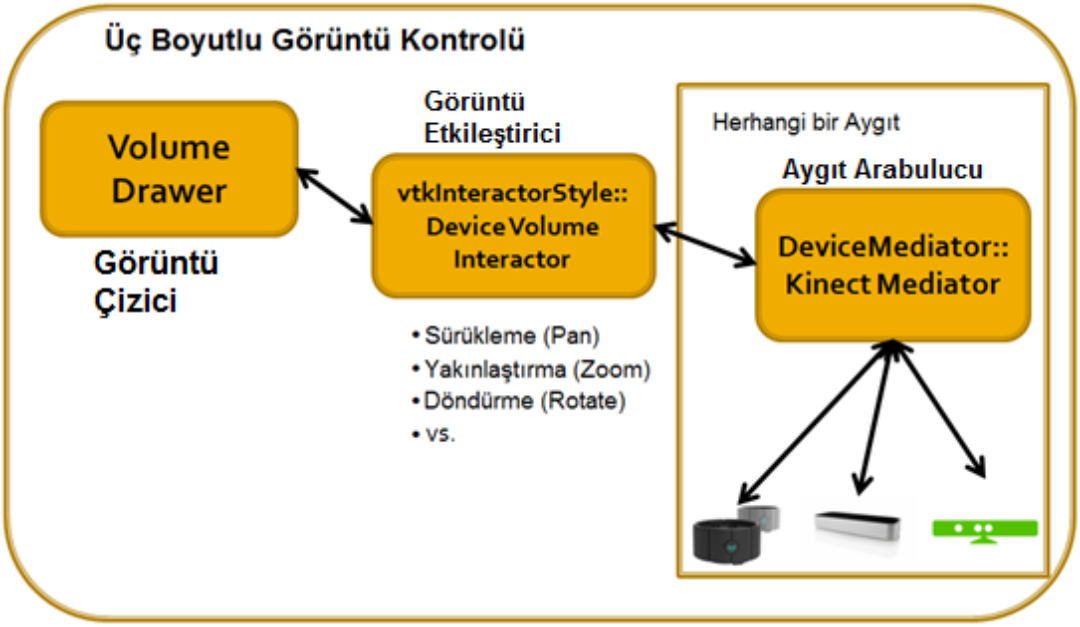
3.5. Yazılım Gerçekleme Detayları

Çalışmanın tümünde geliştirme ortamı olarak Visual Studio 2010 seçilmiştir. Ayrıca VTK kütüphanesinin C++ dili ile gerçeklenmiş olması sebebi ile programlama dili olarak C++ tercih edilmiştir. Yani Tablo 3.1'de gösterilen Qt, VTK, Kinect SDK ve geliştirme ortamı olarak da Visual Studio 2010 tercih edilmiştir.

3.5.1. Yazılım modülleri

Gerçeklenen yazılım birbirlerine ilişkili üç ana modülden meydana gelmektedir. Şekil 3.8’de gösterildiği gibi Bunlar VolumeDrawer, DeviceVolumeInteractor ve DeviceMediator olmak üzere üç ana kısma ayrılır.

Üç boyutlu görüntünün gösterilmesinden sorumlu modül, VolumeDrawer modülüdür. Bu modülde VTK tabanlı sınıflar yardımı ile üç boyutlu görüntünün dosyadan yüklenmesini, ardından pencere oluşturulmasını ve pencerede imge oluşturma (render) işlemini gerçekleştiren modüldür.



Şekil 3.8. Yazılım Modülleri

İmgesi oluşturulan (render) üç boyutlu görüntü DeviceVolumeInteractor modülü vasıtasıyla kullanıcı tarafından kontrol edilebilmektedir. vtkInteractorStyle sınıfından türetilmiş KinectVolumeInteractor sınıfı üç boyutlu görüntüye tutup sürükleme (pan), yakınlaştırıp uzaklaştırma (zoom), döndürme (rotate) işlemlerini uygulamaktan sorumludur.

Kinect cihazından elde edilen veri akışlarının anlamlı hale getirilmesinden sorumlu modül DeviceMediator modülüdür. Bu modül, cihazla yazılım arasında bir nevi arabulucu durumundadır. DeviceMediator modülünün bir bileşeni olan KinectMediator sınıfı bu işlemi gerçekleştirir. Kullanıcı tarafından gerçekleştirilen jest ve ses etkileşimlerinin birtakım komutlar halinde KinectVolumeInteractor sınıfına aktarılmasından sorumludur.

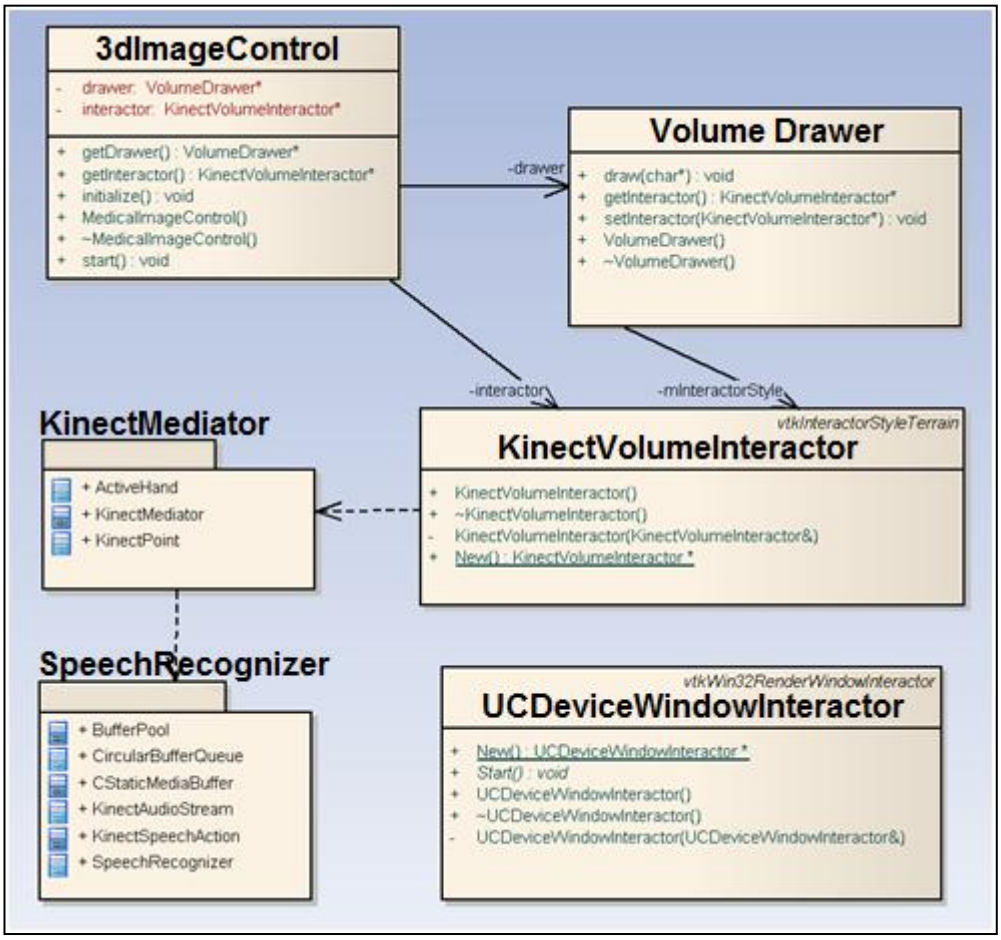
DeviceMediator modülü içinde barındırdığı KinectMediator sınıfı sayesinde Kinect cihazı ile etkileşime olanak sağlayabilmektedir. Ancak Şekil 2.4'te de görüldüğü üzere eğer MYO (URL-20, 2013), Leap (URL-8, 2014) ve benzeri farklı jest kontrol cihazları ile de entegre edilmek istenirse DeviceMediator sınıfından ilgili arabulucu sınıfı türetmek yeterli olacaktır.

Yazılım modüler bir yapıdadır, bu sayede her modül birbirinden bağımsız şekilde değerlendirilebilmektedir. Bu yaklaşımla yazılım mantıksal olarak bölümlendirilerek

daha entegre edilebilir, daha kolay değiştirilebilir ve daha sürdürülebilir hale getirilmiştir.

3.5.2. Yazılım sınıf ve bileşenleri

Üç ana modülden meydana gelen yazılımda sınıflar arasındaki ilişki Şekil 3.9'da gösterildiği gibidir. Tüm sınıfların üstünde bir Qt sınıfı olan ve ayrıca grafik arayüzü de tanımlayan 3dImageControl sınıfı bulunmaktadır. Bu sınıfın oluşturulup başlatılması ile tüm Drawer ve Mediator sınıfları ve buna bağlı sınıflar oluşturulur ve başlatılır.

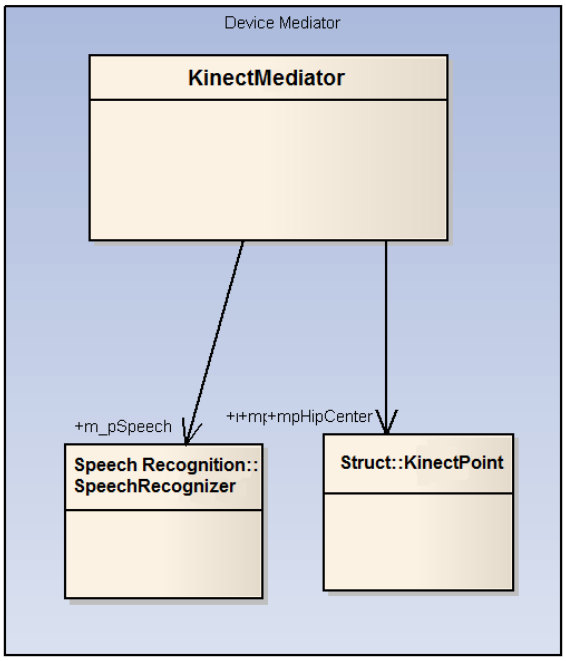


Şekil 3.9. Yazılımın genel sınıf diyagramı

UCDeviceWindowInteractor sınıfı üç boyutlu görüntünün pencere içerisinde gösterildiği ve pencere ile ilgili klavye/fare veya Kinect etkileşimlerinin ele alındığı sınıftır ve VTK sınıfı olan RenderWindowInteractor sınıfından türemiştir. Bu sınıf ana iş parçacığı (main thread) üzerinde çalışmaktadır. Yardımcı iş parçacıkları (background thread) üzerinde yorumlanan jest ve ses komutları bu sınıfa Windows

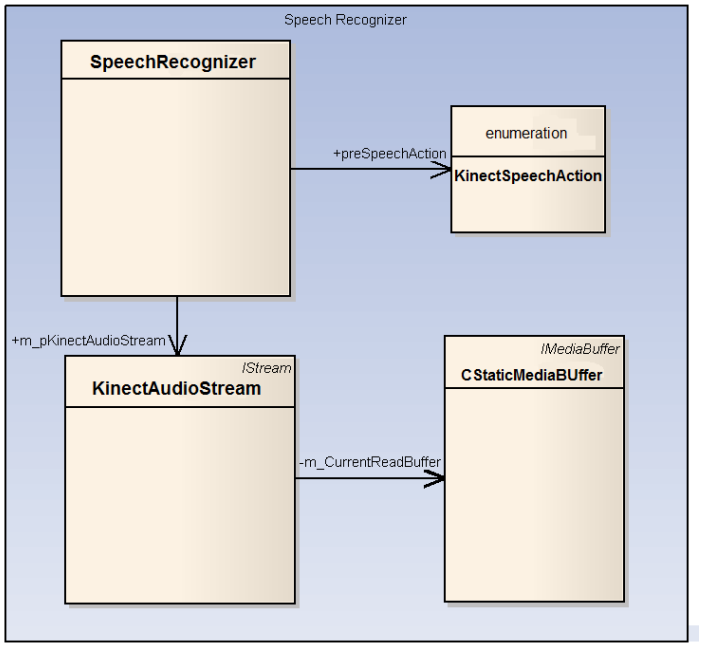
mesajları olarak gönderilir. Bu sınıf vasıtasıyla ilgili mesajlar VTK mesajlarına çevrilerek jest ve ses komutlarını üç boyutlu görüntü üzerinde işletmek üzere ilgili sınıfın ilgili fonksiyonuna gönderilir.

Yazılım, Kinect üzerinden birden fazla veri akışından bilgi alınması ve bunları işleme ihtiyacından dolayı çok iş parçacıklı (multi-threaded) bir yapıda tasarlanmıştır. Ana akışın sürdürüldüğü iş parçacığının yanında ses verilerinin işlendiği ve veri tablosu üzerinden kontrollerinin yapılarak ses komutu olarak yorumlanması işlemi KinectAudioStream içerisinde ayrı bir iş parçacığı (thread) vasıtasıyla yapılır. Aynı şekilde Kinect üzerinden kullanıcıya ait iskelet koordinatlarını çıkarmak için KinectMediator içerisinde ayrı bir iş parçacığı oluşturulur. Bunlarla beraber kullanıcının jestlerini yorumlayıp bunları komutlar haline getirmek için KinectVolumeInteractor içerisinde ayrı bir iş parçacığı oluşturulmuştur. Bu son iş parçacığının oluşturulması sebebi VTK kütüphanesinin iş parçacığı güvenli (thread-safe) olmamasıdır. VTK metotları kullanarak gönderilemeyen mesajları farklı bir iş parçacığı üzerinden ana iş parçacığı üzerinde çalışan UCDeviceWindowInteractor sınıfına gönderilmesi ile sorun aşılmıştır.



Şekil 3.10. KinectMediator sınıfının bileşenleri

KinectVolumeInteractor sınıfı tarafından oluşturulan KinectMediator sınıfı, ses tanıma işleminden sorumlu olan SpeechRecognizer sınıfını oluşturur ve çağırır. SpeechRecognizer ayrı bir iş parçacığı üzerinden KinectAudioStream sınıfı vasıtasıyla Kinect'ten elde edilen ses etkileşimlerini dinler ve önceden tanımlanmış ses etiketleri ile örtüşüp örtüşmediğinin kontrolünü yapar. Ses komutları İngilizce dilini kullanarak verilebilmektedir. Şekil 3.10 ve Şekil 3.11'de Mediator (arabulucu) ve Speech Recognition (ses tanıma) ile ilgili kısımlara ait sınıf bileşenleri verilmiştir.



Şekil 3.11. Ses tanıma kısmının sınıf bileşenleri

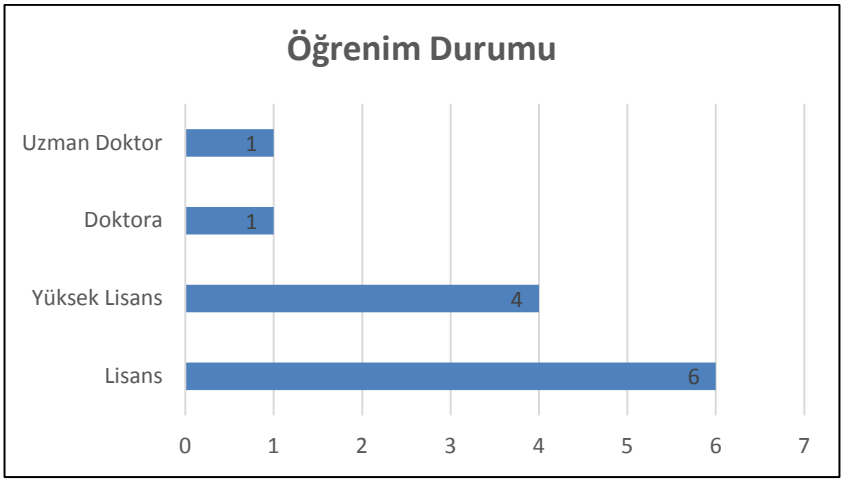
Özetle gerçeklenen sistem, üç boyutlu görüntüleri daha doğal bir etkileşim yöntemiyle kontrol edilmek üzere tasarlanmıştır. Sistem klavye ve fare gibi klasik girdi cihazları olmadan başlatılıp kullanılabilmeye ve ardından sonlandırılmaya olanak sağlamaktadır. Sistem özellikle ameliyathane, ders sunumu gibi klavye ve fare kullanımının güç olduğu ortamlarda kullanıcılara önemli fayda sağlayabilmektedir.

4. KULLANILABİLİRLİK DEĞERLENDİRMESİ

Bu tez çalışmasındaki kullanılabilirlik değerlendirmesi klavye ve fare kullanımının zor olduğu durumlarda Kinect kullanılarak üç boyutlu görüntüleri çok modlu doğal etkileşim ile kontrol etmenin uygun bir seçim olduğunu test etmek için gerçekleştirilmiştir. Çalışma kapsamında gerçeklenen sistemle Kinect aracılığı ile temassız şekilde üç boyutlu görüntülerin kontrolü gerçekleştirilerek doğal etkileşim sağlanmaktadır. Sağlanan bu etkileşimin kullanıcılar tarafından ne kadar etkili ve verimli olduğunun belirlenmesine yönelik olarak kullanılabilirlik değerlendirmesi gerçekleştirilmiştir. Bu kapsamda fare ve Kinect cihazları girdi aygıtı olarak kullanılarak gerçeklenen sistem bir grup katılımcı tarafından kullanılabilirlik açısından değerlendirilmiştir. Her iki girdi aygıtıyla gerçekleştirilen görev performansları kıyaslanarak analiz edilmiştir. Bu bölümde bu değerlendirme çalışmasında yer alan katılımcıların nasıl seçildiği ve özellikleri, değerlendirme prosedürü, değerlendirmede kullanılan araçlar, değerlendirme sonuçları açıklanmaktadır.

4.1. Katılımcılar

Kullanılabilirlik değerlendirme çalışması 12 katılımcı ile gerçekleştirilmiştir. Kullanılabilirlik çalışmalarında yer alması gereken katılımcı sayısı konusu tartışmalıdır. Virzi (1992) ve Nielsen (1993) 5 kişi ile sistemdeki kullanılabilirlik problemlerinin % 75'inin tespit edilebileceğini savunurken Woolrych ve Cockton (2001) ise bu sayının her türlü durumda ideal olmadığını iddia etmektedir. Ancak, uygun katılımcıların seçimi, uygun görevlerin tanımlanması ve bunlar arasındaki ilişkiler katılımcı sayısından daha önemlidir (Çağıltay, 2011). Bu çalışma 12 katılımcı ile gerçekleştirilmiştir. Katılımcıların öğrenim durumları en düşük lisans seviyesinde ve en yüksek doktora seviyesindedir. Erişim kısıtından dolayı katılımcılardan yalnızca bir tanesi uzman doktor, diğerleri çoğunlukla yoğun fare kullanan mühendislerden oluşmaktadır. Şekil 4.1'de katılımcılara ait öğrenim durumu dağılımı görülebilir.



Şekil 4.1. Katılımcıların öğrenim durumu

Katılımcılar k1, k2, k3, k4, k5, k6, k7, k8, k9, k10, k11, k12 şeklinde kodlanmış ve bulguların sunulmasında bu kodlar kullanılmıştır. Katılımcılara ait yaş, cinsiyet, İngilizce bilgisi, hangi eli kullandıkları ve varsa temassız kontrol cihazı aylık deneyimleri gibi sorular test öncesinde sorulmuştur. Tablo 4.1'de katılımcılara ait bu bilgiler sunulmaktadır. Katılımcıların yaşları 27 ile 60 arasında değişmektedir. 12 katılımcının yaş ortalamaları 33,6 olup yaşların standart sapması 8,72'dir.

Katılımcılardan 11 kişi sağ elini kullanarak etkileşimi gerçekleştirirken yalnızca 1 kişi sol elini kullanmaktadır. Ne var ki daha önceden de bahsedildiği gibi gerçeklenen sistemi Kinect ile kullanmak için sağ el veya sol el kullanma şartı aranmamakta, sağ veya sol elle üç boyutlu görüntüler kontrol edilebilmektedir. Bu soru sağ veya sol elle kullanımın başarıma etkisi olup olmadığını gözlemlemek için sorulmuştur ancak anlamlı bir çıkarım yapılacak kadar solak katılımcıya erişilememiştir. Sistemi kullanan katılımcılardan 11'i erkek yalnızca 1'i kadındır. Kadın katılımcı sağ elini kullanmaktadır.

12 kullanıcının hepsi İngilizce bilmekle birlikte Tablo 4.1'de gösterildiği gibi katılımcıların 9'u ileri derecede, 3'ü ise orta seviyede İngilizce bildiğini söylemektedir. Kinect kontrolü için basit düzeyde de olsa İngilizce sesle komut verilmesi gerektiği için katılımcıların İngilizce bilmesi tercih edilmiştir.

Tablo 4.1. Katılımcıların yaş, cinsiyet, ingilizce bilgisi, hangi eli kullandığı ve temassız kontrol cihazı deneyimi verileri

Katılımcılar	Yaş	Cinsiyet	İngilizce Bilgisi	Hangi Eli Kullanır?	Temassız Kontrol Cihazı Aylık Deneyimi	Günlük Bilgisayar Kullanım Süreleri (Saat)
k1	32	e	ileri seviye	sağ	Yok	10
k2	32	e	ileri seviye	sağ	Yok	8
k3	28	e	orta seviye	sağ	Yok	10
k4	33	e	ileri seviye	sağ	Yok	6
k5	28	e	ileri seviye	sağ	Yok	10
k6	29	e	ileri seviye	sol	aylık 2 saat	10
k7	35	e	ileri seviye	sağ	Yok	12
k8	35	e	ileri seviye	sağ	Yok	7
k9	27	e	orta seviye	sağ	Yok	10
k10	60	e	orta seviye	sağ	Yok	1
k11	33	k	ileri seviye	sağ	Yok	3
k12	32	e	ileri seviye	sağ	Yok	6

Katılımcıların en düşük günlük bilgisayar kullanım süresi 1 saat ile en yüksek kullanım süresi 12 saat arasında değişmektedir. Katılımcıların günlük bilgisayar kullanım ortalaması 7,75 saat olup aralarındaki standart sapma değeri 3,28'dir.

Katılımcılara eğer varsa Kinect veya Kinect benzeri temassız bir kontrol cihazını aylık kullanımlarının ne olduğu sorulmuştur. Ancak katılımcılardan yalnızca 1 kişinin temassız kontrol cihazı deneyimi olduğu, geri kalan 11 kişinin Kinect cihazını ilk defa deneyecekleri görülmüştür.

4.2. Veri Toplama Araçları

Geliştirilen sistem üzerinde kullanılabilirlik değerlendirmesi sistemin kullanımı sırasında kullanıcıların gerçek görevleri gerçekleştirmesini içeren kullanıcı testine dayalı deneysel yaklaşım ile yapılmıştır. Kullanıcı testi için öncelikle geliştirilen

sistemin fonksiyonları olan sürükleme (pan), yakınlaştırma/uzaklaştırma (zoom) ve döndürme (rotate) işlemlerini içeren görev listesi tanımlanmış ve bu görevleri katılımcıların hem fare ile hem de Kinect ile tamamlamaları istenmiştir. Bu görevleri yerine getirirken de sesli-düşünme protokolü (think-aloud protocol) (Nielsen J., 1993) gereği verdikleri olumlu veya olumsuz tepkiler gözlenmiştir. Bunun için katılımcılar uygulamayı kullanırken ekran ve ses kaydı alınması suretiyle verdikleri jest, mimik ve sözlü tepkiler kayıt altına alınmıştır. Ekran ve ses kayıtları toplanırken gizliliğe önem verilmiş olup üçüncü kişilerle paylaşılmayacağı temin edilmiştir. Kullanıcı testine ek olarak bu görevleri tamamlamadan önce katılımcıların demografik bilgilerini öğrenmek için sorular sorulmuştur. Katılımcı görevleri tamamladıktan sonra da ISO 9241-9:2000 dokümanında yer alan ve kullanıcı memnuniyetini ölçmeye yönelik yedili likert ölçeğine göre bir anketi tamamlamaları istenmiştir.

4.2.1. Görev listesi

Katılımcılara sistemin nasıl kullanılacağını anlatan yazılı ve görsel bir doküman sunulmuş ve ayrıca da sistemin nasıl kullanılacağı kendilerine anlatılmıştır. Tablo 4.2'de fare ile yapılacak görevler, Tablo 4.3'de de Kinect ile yapılacak görevler verilmiştir. Görev 0 ve Görev 6 üç boyutlu görüntünün kontrolü ile alakalı olmayıp sistemin başlatılıp sonlandırılması ile alakalıdır. Bu yüzden işlemlerin ne kadar hızlı gerçekleştirildiğine yönelik analizler bu görevler için yapılmamıştır. Bu görevler sistemin baştan sona temassız şekilde başlatılıp sonlandırılabildiğini geçerlemek için koyulmuştur.

Görevler zorluk derecelerine göre iki kısma ayrılmıştır. Birincisi kolay, ikincisi de zor görevlerdir. Bir görevin zor veya kolay olduğunu belirten kıstas ise görevlerin kendi içerisinde barındırdığı işlev sayısıdır. Başka bir deyişle bir görev eğer bir işlev içeriyorsa kolay, eğer iki işlevi birden barındırıyorsa zor olarak nitelendirilmiştir. Bu tanıma göre Tablo 4.1 ve Tablo 4.2'de belirtilen görevlerden Görev 1, Görev 2 ve Görev 3 kolay görev, Görev 4 ve Görev 5 de zor görev olarak nitelendirilmektedir. Görev 0 ve Görev 6 ise üç boyutlu görüntü kontrolü ile alakalı olmadığı için bu kategorizasyonun dışında bırakılmıştır.

Tablo 4.2. Fare ile yapılacak görevler

Fare ile yapılacak görevler		**Görev Zorluk Derecesi**
Görev 0:	"Başlat" butonuna basarak sistemi başlatınız.	-
Görev 1:	Üç boyutlu görüntüyü fare ile sağ üst köşedeki çerçeve içerisine **sürükleyin**.	Kolay
Görev 2:	Üç boyutlu görüntüyü fare ile çerçeveye sığacak şekilde **büyütün**.	Kolay
Görev 3:	Üç boyutlu görüntüyü fare ile yüzü bize bakacak şekilde **döndürün**.	Kolay
Görev 4:	Üç boyutlu görüntüyü fare ile **çerçeve içerisine sürükleyin** ve **çerçeve içerisine sığacak şekilde küçültün**.	Zor
Görev 5:	Üç boyutlu görüntüyü fare ile **sağ alt köşeye sürükleyin** ve **yüzü bize bakacak şekilde döndürün.**	Zor
Görev 6:	Fare ile ekranın sağ üst köşesinde bulunan "X" işaretine basarak pencereyi kapatınız.	-

Tablo 4.3. Kinect ile yapılacak görevler

MS Kinect ile yapılacak görevler		**Görev Zorluk Derecesi**
Görev 0:	Sesle ingilizce olarak "start" diyerek sistemi başlatınız.	-
Görev 1:	Üç boyutlu görüntüyü MS Kinect sağ üst köşedeki çerçeve içerisine **sürükleyin**.	Kolay
Görev 2:	Üç boyutlu görüntüyü MS Kinect ile çerçeveye sığacak şekilde **büyütün**.	Kolay
Görev 3:	Üç boyutlu görüntüyü MS Kinect ile yüzü bize bakacak şekilde **döndürün**.	Kolay
Görev 4:	Üç boyutlu görüntüyü MS Kinect ile **çerçeve içerisine sürükleyin** ve **çerçeve içerisine sığacak şekilde küçültün**.	Zor
Görev 5:	Üç boyutlu görüntüyü MS Kinect ile **sağ alt köşeye sürükleyin** ve **yüzü bize bakacak şekilde döndürün.**	Zor
Görev 6:	Sesle ingilizce olarak "close" diyerek sistemi kapatınız.	-

Görev 0 ile katılımcıdan beklenen İngilizce olarak “start” kelimesini telaffuz ederek çok modlu doğal etkileşimi başlatmasını sağlamasıdır. Görev 1 ile katılımcıdan beklenen ise sol alt köşede bulunan üç boyutlu görüntüyü sağ üst köşeye

sürüklemektir. Görev 2'de ise katılımcıdan ekranın ortasında ve küçük olarak bulunan üç boyutlu görüntüyü ekrana sığacak şekilde büyütmesi beklenmiştir. Görev 3'te ise farklı bir yöne bakan üç boyutlu yüz görüntüsünün katılımcıdan ekranan bakacak şekilde döndürmesi beklenmiştir. Görev 4 ile katılımcıdan sağ alt köşede bulunan görüntüyü önce sol üst köşede bulunan çerçeve içerisine sürükleyip ardından çerçeve içerisine sığacak şekilde küçültmesi beklenmiştir. Görev 5'te ise katılımcıdan sol üst köşede bulunan üç boyutlu görüntüyü öncelikle sağ alt köşeye sürükleyip ardından da yüzü bize bakacak şekilde döndürmesi beklenmiştir.

4.2.2. Anketler

Katılımcılara kullanıcı testi öncesi ve kullanıcı testi sonrası olmak üzere 2 farklı anket doldurmaları talep edilmiştir. Kullanıcı testi öncesindeki sorular daha çok katılımcıların demografik bilgilerini edinmeye yönelik olarak hazırlanmıştır. Bu kapsamda, katılımcılara ait yaş, cinsiyet, öğrenim durumu, meslek, günlük bilgisayar kullanımı, varsa temassız kontrol cihazı aylık deneyimi, ne derecede İngilizce bildikleri ve hangi ellerini kullandıkları soruları yöneltilmiştir. Katılımcıların demografik özelliklerine yönelik olarak hazırlanan sorular Ek-A'da görülebilir.

Katılımcıları görevleri gerçekleştirmelerinin ardından ISO 9241-9 (2000) standardında yer alan kullanıcı memnuniyetini ölçmeye yönelik olarak hazırlanmış sorulardan adapte edilen ikinci bir anket sunulmuştur. Bu anket içerisinden temassız kontrol cihazı için gerekli olan sorular seçilmiş ve Türkçeleştirilmiştir. Ardından Türkçeleştirilen anket iki alan uzmanı tarafından değerlendirilerek Ek-A'daki son halini almıştır. Anket 3 kısımdan oluşmaktadır. Birinci kısım ilk 8 sorunun bulunduğu kısımdır. Bu sorular kurulan etkileşim ile ilgili sorulardır. İkinci kısımda 9, 10 ve 11. sorulardan oluşmaktadır. Bu sorular etkileşimin biyomekanik etkilerini ölçmeye yönelik sorulardır. Üçüncü kısım da katılımcı tarafından ek olarak belirtilmek istenen noktaların yazılı şekilde ifade edilmesi istenen bir sorudan oluşmaktadır.

Katılımcılardan anketteki ifadeleri yedili likert ölçeğine göre değerlendirmeleri istenmiştir. Bu ölçekten düşük puanlı seçenek kontrol cihazı hakkında en olumsuz, en yüksek puanlı seçenek de cihaz hakkında en olumlu seçeneği ifade etmektedir.

4.3. Veri Toplama Süreci

Bu kısımda veri toplama araçları ile ne şekilde verilerin toplandığından bahsedilmektedir. Veri toplama süreci Şekil 4.2'de gösterildiği gibi dört aşamadan oluşmaktadır. Öncelikle gerçeklenen sistemin kullanılabilirlik testine uygunluğunu araştırmak için bir pilot çalışması yapılmış ve katılımcılardan geri dönüşlerle sistem iyileştirilmiştir. Ardından katılımcıların kontrollere aşinalığını sağlamak için sistemi keşfetmelerine yönelik bir eğitim verilmiştir. Bunun arkasından da kullanılabilirlik testi uygulanmış, test esnasında katılımcıların ekran ve ses kayıtları alınmıştır. Son olarak da katılımcılardan sistem hakkındaki memnuniyetlerine yönelik bir anket doldurmaları istenmiştir.

Şekil 4.2. Veri toplama sürecinde katılımcıların rolleri

4.3.1. Pilot çalışması

Sistem gerçeklenip görev tanımları yazıldıktan sonra 3 farklı katılımcı tarafından denenmiştir. Pilot çalışmasına katılan katılımcıların geri döndükleri yorumlar doğrultusunda gerçeklenen sistem ve gerçekleme ortamı tekrar gözden geçirilmiştir. Gözden geçirilmiş sistem üzerinde 12 katılımcı ile gerçekleştirilen kullanılabilirlik testi uygulanmıştır.

Pilot çalışmasına katılan katılımcılar gerçeklenen sistemde üç boyutlu görüntüyü kontrol ederken hassasiyetin düşük olduğunu belirtmişlerdir. Döndürme, yakınlaştırıp uzaklaştırma ve sürükleme işlemlerini gerçekleştirirken fazlaca efor sarf etmek gerektiğini bu yüzden hassasiyetin artırılmasını istemişlerdir. Bu geri bildirim sonucunda döndürme, yakınlaştırıp uzaklaştırma ve sürükleme işlemleri gerçekleştirme hassasiyetleri yükseltilerek iyileştirme yapılmıştır.

Pilot çalışmasına katılan katılımcılar aynı zamanda Kinect kullanarak elle tutma işlemini kullanıcıya geri bildiren alanın arayüz üzerinde görünür olmadığı şeklinde geri bildirimde bulunmuştur. Bu geri bildirime binaen hangi el tutma işlemi gerçekleştirdi ise o el ile alakalı geri bildirimde bulunacak olan arayüzdeki ilgili alan

yeşil olarak, tutma işlemi gerçekleştirmedi ise arayüzdeki ilgili alan kırmızı olarak sergilenecek şekilde gerçeklenen sistem güncellenmiştir.

Gerçekleştirilen pilot çalışmasında kullanıcılar yüksek derecede aydınlanmış bir ortamda işlemleri gerçekleştirirken elle tutma işlemi, sürükleme ve döndürme gibi işlemleri hatalı şekilde gerçekleştirdiği veya gerçekleştiremediği gözlendi. Bunun sebebi ise Kinect cihazının yüksek derecede aydınlatılmış ortamlarda başarılı şekilde çalışamamasıdır. Bu durum cihaza bağlı bir kısıt olmakla birlikte kullanılabilirlik testlerini yüksek derecede aydınlatılmış ortamlarda gerçekleştirilmemesi gerektiğini göstermektedir. Son olarak da pilot çalışmasında ortaya çıkan yazılımsal sorunlar giderilerek kullanılabilirlik çalışmasına başlamak için sistem hazır hale getirilmiştir.

4.3.2. Katılımcıların eğitimi

Kullanılabilirlik çalışmasına katılacak olan katılımcılara hem fare hem de Kinect cihazını nasıl kullanmaları gerektiği konusunda bilgilendiren bir kılavuz verilmiştir. Bu kılavuz Ek-B'de yer almaktadır.

Kullanılabilirlik çalışmasına başlamadan önce katılımcılar bu kılavuzu incelemişlerdir. Ardından da sistemin nasıl çalıştığına yönelik sözlü bir bilgilendirme yapılmıştır. Bunun arkasından da katılımcılardan hem Fare ile hem de Kinect ile sistemi denemeleri istenmiş ve katılımcıların kontrollere aşina olması sağlanmıştır. Katılımcılar test sorumlusuna hazır olduklarını söyledikten sonra kullanılabilirlik testi gerçekleştirilmiştir.

4.3.3. Kullanılabilirlik testi

Öncelikle kullanılabilirlik çalışmasına katılacak olan kişilerden Ek-A'da sunulan Bilgilendirilmiş Onam Formu'nu okuyup onaylamaları istenmiştir. Ardından kullanılabilirlik testi öncesi ve kullanılabilirlik testi sonrası olmak üzere katılımcıların iki farklı anket doldurmaları istenmiştir. Test öncesi uygulanan ankette katılımcılara ait demografik bilgilerin elde edilmesine yönelik sorular bulunmaktadır. Test esnasında ise katılımcılar Tablo 4.2 ve Tablo 4.3'de verilen görevleri yerine getirmeleri istenmiş ve bu esnada ekran ve ses kaydı alınmıştır. Alınan ekran ve ses kayıtları ile katılımcıların sesli-düşünme protokolü (think-aloud protocol) gereği

verdikleri olumlu veya olumsuz tepkiler ve görevleri ne kadar sürede ve ne şekilde gerçekleştirdiği gözlenmiş ve ölçülmüştür. Görevleri gerçekleştirme zaman hesabı fare için düğmeye basma ile başlayıp görev sonlanıncaya kadar devam eden süre olarak hesaplanmıştır. Kinect için ise elle tutma işlemi yaparak başlayıp görev sonlanıncaya kadar devam eden süre olarak hesaplanmıştır. Katılımcılar kullanılabilirlik testini 23 inch ekran üzerinde gerçekleştirmiştir.

Gerçeklenen sisteme Kinect kullanılarak ses ile üç farklı komut, jestlerle de üç farklı komut verilebilmektedir. Bu komutlar esas alınarak katılımcılara hem fare ile hem de Kinect ile yapabilecekleri görevler tanımlanmıştır. Bu görevler her iki etkileşim yöntemi için de aynıdır. Katılımcılardan aşağıdaki görevleri her iki etkileşim yöntemi için gerçekleştirmeleri talep edilmiştir:

- Etkileşimi başlatma
- Üç boyutlu görüntüyü bir noktadan bir noktaya taşıma
- Üç boyutlu görüntüyü büyütme
- Üç boyutlu görüntüyü döndürme
- Üç boyutlu görüntüyü bir noktaya taşıma ve bir çerçeve içerisine sığacak şekilde küçültme
- Üç boyutlu görüntüyü bir noktaya taşıma ve döndürme
- Sistemi kapatma

Bu görevlerin yanı sıra katılımcılara eğer beklenmedik bir durumla karşılaştıkları zaman ses ile "RESET" komutu vererek üç boyutlu görüntünün varsayılan ilk durumuna gelmesini sağlayabilecekleri belirtilmiştir. Tablo 4.2 ve Tablo 4.3'te belirtilen görevler her bir katılımcıya farklı sırada uygulanmıştır. Kimi katılımcılar fare görevleri ile, kimi katılımcılar da Kinect görevleri ile başlamıştır. Bunun yanı sıra her bir katılımcıdan görevleri farklı sırada yerine getirmesi istenmiştir. Böylece öğrenme faktörünün önüne geçilmesi hedeflenmiştir (Tullis & Albert, 2008). Her katılımcının oturumu yaklaşık 15-20 dakika sürmüştür.

4.4. Veri Analizi

Veri analizi sürecinde kullanılabilirlik testinde uygulanan sesli-düşünme protokolü ile elde edilen notlar incelenmiştir. Buna ek olarak memnuniyet anketinde son soru

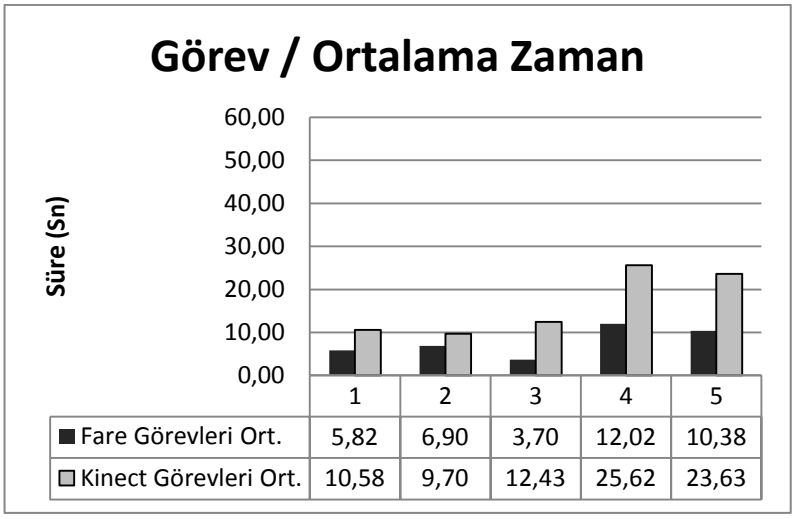
olarak sorulan “varsa belirtmek istediğiniz diğer noktalar” kısmına düşülen notlar da incelenmiştir. Memnuniyet anketine verilen cevaplar her katılımcı için ayrı ayrı ve ortalama olarak hesaplanmıştır. Ardından ekran kayıtları üzerinden katılımcıların görevleri gerçekleştirme süreleri hesaplanmıştır. Görevlerin gerçekleştirme zamanları görev başına ortalama süre hesaplanarak karşılaştırılmıştır. K10 katılımcısının en düşük kullanım oranına sahip olması ve K6 katılımcısının da en yüksek jest kontrol cihazı deneyimi olması sebebiyle bu katılımcılara yönelik detay analizlere yer verilmiştir. Elde edilen veriler betimsel analiz (Lazar 2010) yoluyla yorumlanmış ve elde edilen sonuçlar bulgular bölümünde sunulmaktadır.

4.5. Bulgular

Kullanılabilirlik değerlendirmesi sonucunda elde edilen bulgular iki ayrı kısımda ele alınmıştır. Birincisi katılımcılar tarafından sistemin denenmesi esnasında kaydedilen veriler üzerinden elde edilen bulgular. İkincisi de katılımcılara ait demografik veriler ve memnuniyet verileri üzerinden elde edilen bulgulardır.

4.5.1. Kullanılabilirlik testi bulguları

Toplamda 12 kullanıcıya ait görevlerle alakalı kullanım deneyimi incelenmekle beraber kullanıcıların eğitimi ve kontrollere alışma süreci de izlenmiş ve bu süreçteki katılımcılara ait geri bildirimler de dikkate alınmıştır. Şekil 4.3'te tüm görevlerin tüm katılımcılar tarafından hem fare ile hem de Kinect ile ortalama tamamlama süreleri gösterilmiştir. Şekil 4.3'te görüldüğü gibi 5 görevin hepsi için fare ile ortalama tamamlama süreleri Kinect ile ortalama tamamlama sürelerinden daha kısadır.

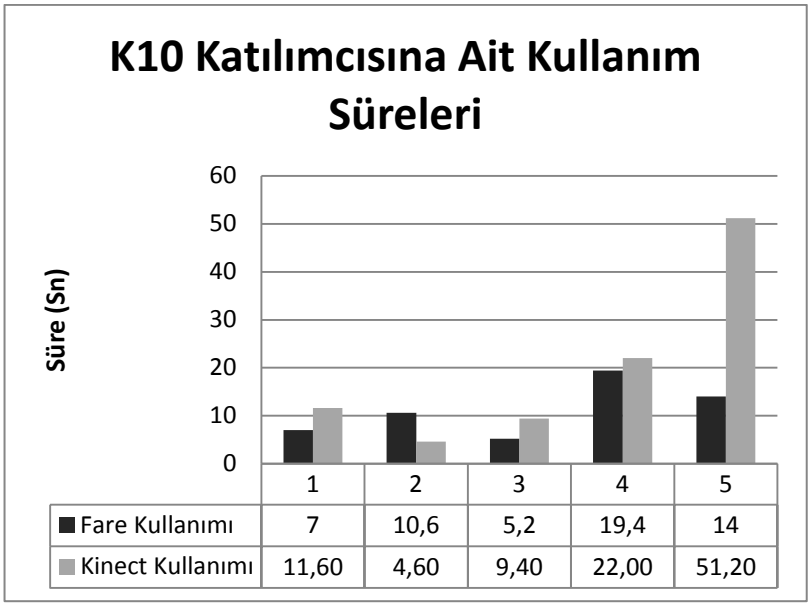


Şekil 4.3. Görevleri ortalama tamamlama süreleri

Şekil 4.3 incelendiğinde kolay görev olan Görev 1, Görev 2 ve Görev 3 görevlerini gerçekleştirme ile zor görev olan Görev 4 ve Görev 5 görevlerini gerçekleştirmek arasında en az yarı yarıya fark olduğu görülebilir. Bunun sebebi olarak zor görevlerin aslında iki kolay görevin birleşimi olduğu gösterilebilir. Ayrıca grafikte fare ile gerçekleştirme süreleri ile Kinect ile gerçekleştirme sürelerinde Görev 2 hariç en az yarı yarıya fark görülebilmektedir.

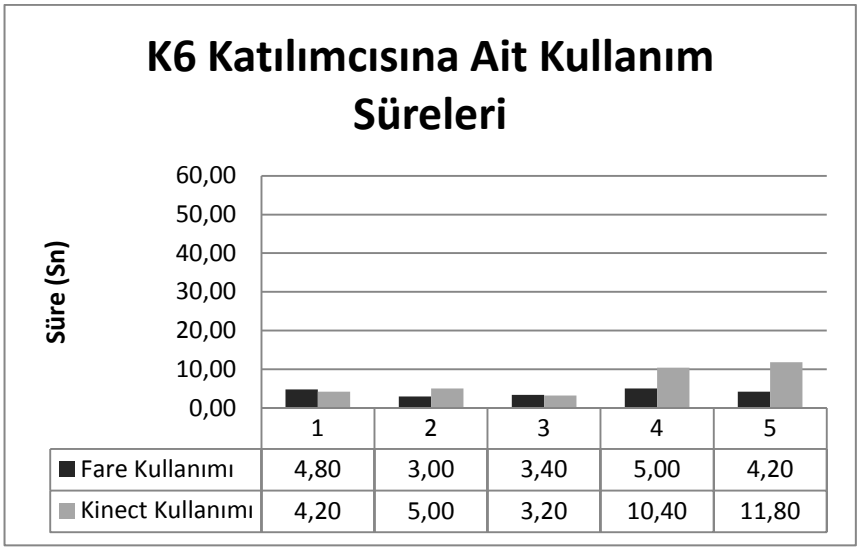
Katılımcılardan yalnızca bir kişinin (K10) günlük 1 saat bilgisayar kullanımı varken bütün katılımcıların ortalama günlük bilgisayar kullanımı 7,75 saattir. Yani katılımcıların bir kişi dışında fare kullanma konusunda uzman sayılabilir. Bunun aksine katılımcılardan bir kişi (K6) dışında hiç kimsenin Kinect veya benzeri temassız kontrol cihazı deneyimi yoktur. Bu durum da fare ile görevleri tamamlama sürelerinin Kinect ile tamamlama sürelerinin neden yarısına yakın olduğuna delil gösterilebilir.

Üç boyutlu görüntülerin kontrolünün alışılmış ve öğrenilmiş bir kontrol cihazı olan Fare ile ve alışılmamış ve tecrübe edilmemiş temassız kontrol cihazı olan Kinect ile gerçekleştirilmesi arasındaki ilişkiyi incelendiğinde birinci olarak fare deneyimi az olan ve Kinect deneyimi olmayan K10 katılımcısının kullanım ortalamaları Şekil 4.4'te görüldüğü gibidir.



Şekil 4.4. K10 Katılımcısına Ait Kullanım Süreleri

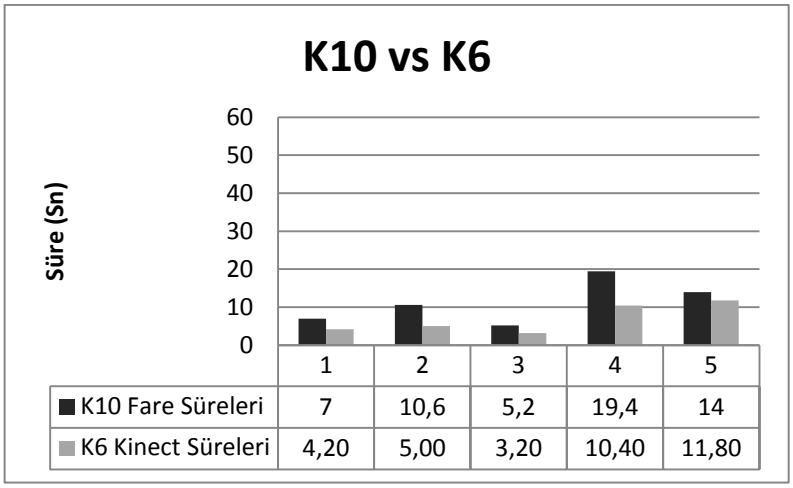
Beşinci görevdeki kullanım süresinin uzadığını göz ardı edersek 1, 3 ve 4. görevleri bitirme süreleri birbirlerine çok yakın olmanın yanında katılımcı Görev 2'yi gerçekleştirirken fareye nazaran 2 kattan daha hızlı şekilde Kinect ile işlemi gerçekleştirmiştir. Fare kullanım standart sapması 5,68 iken Kinect kullanım standart sapması ise 18,69 olarak kaydedilmiştir. Yani cihazı kullanım kararlılığında düşüş gözlenmektedir. İkinci olarak hem fare deneyimi yüksek hem de Kinect deneyimi olan K6 katılımcısına ait kullanım ortalamaları Şekil 4.5'te verilmiştir. Şekil 4.5'te görüldüğü gibi kontrol cihazlarıyla ortalama işlem süreleri en fazla 11,8 saniyedir. K6 katılımcısına ait ortalama Fare kullanımı 4,08 saniye standart sapması 0,86 olarak, ortalama Kinect kullanımı ise 6,92 saniye ve standart sapması 3,9 olarak ölçülmüştür. K10 katılımcısı ile karşılaştırıldığında standart sapmada dramatik bir düşüş gözlenmiştir. Yani cihazı kullanım kararlılığı K10 katılımcısına göre yüksektir.



Şekil 4.5. K6 Katılımcısına Ait Kullanım Süreleri

Şekil 4.5'te dikkat çeken diğer bir nokta ise K6 katılımcısının fare deneyimi fazla olmasına rağmen Görev 1 ve Görev 3'ü Kinect ile fareye göre biraz daha kısa sürede yapmış olmasıdır. Yani temassız kontrol cihazı ile ve jestlerle deneyimi olan katılımcılar Kinect ile kolay görevler daha hızlı gerçekleştirmiştir. Ancak hiçbir katılımcı zor görevleri Kinect ile fareden daha kısa sürede yerine getirememiştir. Ayrıca K10 ve K6 katılımcısının görevleri gerçekleştirme süreleri göz önünde bulundurulduğunda kullanılan cihaza aşinalığın ve alışkanlığın, görüntü kontrolünü etkilediğinin söyleyebiliriz.

Katılımcılar arasında günlük bilgisayar deneyimi en az olan K10 katılımcısının Fare süreleri ile temassız kontrol cihazı deneyimi olan K6 katılımcısının Kinect süreleri Şekil 4.6'da görülebilir. Burada en dikkat çekici nokta Fare kullanmasına rağmen K10 katılımcısının görevleri yerine getirme süreleri K6 katılımcısının Kinect ile görevleri yerine getirme sürelerine göre daha uzun sürmüştür. Bu durum daha önce de bahsedilen etkileşim cihazları ile deneyimin görevleri gerçekleştirmede önemli bir etken olduğunu göstermektedir.



Şekil 4.6. K6 ve K10 katılımcılarının deneyim karşılaştırması

Sesli-düşünme protokolü çerçevesince gerek katılımcı eğitimi esnasında gerekse görevlerin yerine getirilmesi esnasında katılımcılardan geri bildirimler alınmıştır. Katılımcıların önemli bir kısmının kontrol jestlerini hatırlamakta güçlük çektiği gözlenmiştir. Aslında bu durum sadece birkaç dakikalık eğitimle görevleri yapmaları talep edilen katılımcılar için normal sayılabilir.

Sadece K8 numaralı katılımcı jestlerin hassasiyetinin düşük olduğunu belirtmekle birlikte aynı katılımcı görevleri yapmaya başlarken jestleri şaşırdığı gözlenmiştir. K7 numaralı katılımcı ise tutma (grip) işlemi yapmadan eller açık şekilde görüntü kontrolünün gerçekleştirilebilmesi gerektiğini dile getirmiştir. Hemen hemen tüm katılımcılarda obje sürüklenirken, sürükleme ekseninden ufak sapmalar meydana gelmiştir. Bu durum cihazın bir kısıtı olmakla birlikte ortamın daha aydınlık olması durumunda bu sapmalar arttığı ve büyüdüğü gözlenmiştir. Tablo 4.4'te tüm katılımcılara ait fare ve Kinect kullanım süreleri verilmiştir.

Tüm katılımcılar en az orta seviyede İngilizce bildiklerini söylemekle birlikte orta seviye İngilizce bilgisine sahip olanlar sesli komut vermede başarılı olamadığı gözlenmiş ve ileri seviyede İngilizce bilgisine sahip olanlar da komutları bazen zorlanarak vermiştir. Bunun başlıca sebebi olarak sistemin kabul edebileceği aksanda İngilizce komut verilememesi gösterilebilir.

Tablo 4.4. Tüm katılımcıların fare ve Kinect ile görevleri bitirme süreleri

Fare	**K1**	**K2**	**K3**	**K4**	**K5**	**K6**	**K7**	**K8**	**K9**	**K10**	**K11**	**K12**
Görev 1	2,20	3,00	18,20	6,60	2,00	4,80	6,2	2,4	3,8	7	8,6	5
Görev 2	4,60	5,60	12,20	8,00	2,80	3,00	4,6	4,6	10,6	10,6	9,6	6,6
Görev 3	2,20	3,80	4,60	3,00	2,60	3,40	1,2	2,4	8,4	5,2	3,2	4,4
Görev 4	6,00	10,00	13,20	8,00	11,80	5,00	12,2	9,4	24	19,4	10,8	14,4
Görev 5	5,80	7,00	13,80	7,40	8,20	4,20	14,8	4,2	21,6	14	11,4	12,2
Kinect	**K1**	**K2**	**K3**	**K4**	**K5**	**K6**	**K7**	**K8**	**K9**	**K10**	**K11**	**K12**
Görev 1	11,00	7,40	15,40	15,80	8,80	4,20	12,60	6,6	11,4	11,60	5,40	16,80
Görev 2	18,00	13,40	8,80	27,80	12,40	5,00	3,40	4,6	5,8	4,60	7,40	5,20
Görev 3	12,00	9,40	11,00	16,80	11,20	3,20	14,20	6,4	13	9,40	28,80	13,80
Görev 4	24,40	42,60	58,60	12,80	23,60	10,40	11,40	11,2	30	22,00	42,60	17,80
Görev 5	26,60	13,00	23,40	47,40	13,00	11,80	17,40	15,2	22	51,20	17,80	24,80

4.5.2. Kullanıcı memnuniyet anketi bulguları

Tüm katılımcılar tarafından doldurulan anket sorularına verilen cevaplar Tablo 4.5'te gösterilmiştir. Tabloda gösterildiği üzere K2 katılımcısı en düşük memnuniyet oranına sahiptir. K2 katılımcısı operasyonları gerçekleştirmek için harcanan fiziksel çabanın yüksek olduğunu belirttiği gibi operasyonlar esnasındaki akıcılığın da az olduğunu belirtmiştir. K9 ve K6 katılımcısı ise en yüksek memnuniyet oranına sahip olmakla birlikte K9 katılımcısı hangi kontrol cihazını seçerdiniz sorusuna Fare kontrol cihazını tercih edeceğini belirtmiştir.

Tablo 4.5. Her bir katılımcının memnuniyetine yönelik bulgular

Sorular \ Katılımcılar	**K1**	**K2**	**K3**	**K4**	**K5**	**K6**	**K7**	**K8**	**K9**	**K10**	**K11**	**K12**
Operasyonları gerçekleştirmek için harcanan mental/zihinsel çaba *(yüksektir=1 düşüktür=7)*	3	6	3	3	3	6	7	2	4	3	6	5
Operasyonlar esnasındaki akıcılık *(kabadır/azdır=1 yumuşaktır/yüksektir=7)*	4	2	6	5	6	4	5	5	6	5	4	5
Doğru/Tam işlem yapmak *(zordur=1 kolaydır=7)*	5	3	6	7	6	6	7	5	7	5	6	6
Operasyonları gerçekleştirmek için harcanan fiziksel çaba *(yüksektir=1 düşüktür=7)*	2	4	5	3	5	6	4	3	6	3	5	6
Operasyonu gerçekleştirme sürati *(yavaştır=1 hızlıdır=7)*	5	3	6	6	6	5	6	6	7	5	4	5
Genel kullanım rahatlığı/konforu *(rahatsızdır=1 rahattır=7)*	6	2	6	6	5	6	6	4	7	5	5	4
Bütün olarak düşünürsek, etkileşim *(zordur=1 kolaydır=7)*	6	3	5	6	6	6	5	7	6	5	6	5
Hangisini daha çok tercih ederdiniz? MS Kinect kontrolünü mü yoksa Fare kontrolünü mü? *(Fare=1 Jest=7)*	4	3	6	2	4	5	5	4	1	4	7	3
Bilek yorgunluğu hissi *(çoktur=1 azdır =7)*	4	6	7	7	7	7	5	6	7	6	5	6
Kol yorgunluğu hissi *(çoktur=1 azdır =7)*	7	5	5	7	7	7	5	5	7	5	5	6
Omuz yorgunluğu hissi *(çoktur=1 azdır =7)*	7	6	7	7	7	7	5	6	7	6	7	6
Ortalama:	4,8	**3,9**	5,6	5,3	5,6	**5,9**	5,4	4,8	**5,9**	4,7	5,4	5,2
Toplam:	53	**43**	62	59	62	**65**	60	53	**65**	52	60	57

Test sonrası memnuniyet anketinde her bir soru için ortalama (mean), ortanca (median), standart sapma, en çok seçilen seçenek ve seçilmiş minimum ve maksimum seçenekler Tablo 4.6'da verilmiştir. Tablo 4.6 incelendiğinde anketin ikinci kısım soruları olan son üç soruya ortalama 7 üzerinden 6 verilmiştir. Bu da

bilek, kol ve omuz yorgunluğunun genel olarak az olduğunu göstermektedir. Bununla birlikte K1, K4, K6, K7 ve K8 katılımcıları bilek yorgunluğunu ölçebilecek kadar uzun süreli sistemi denemediklerini söylemiştir.

Sorular arasında ortalamada en düşük cevap " Hangisini daha çok tercih ederdiniz? MS Kinect kontrolünü mü yoksa Fare kontrolünü mü?" sorusuna verilen cevaptır. Katılımcılar bu soruya ortalama 4,00 vermişlerdir. Buradan katılımcıların alışmış oldukları kontrol cihazından ayrılıp yeni bir kontrol cihazı kullanmaya istekli olmadıkları çıkarılabilir. Tablo 4.6'da görüldüğü gibi birinci ve sekizinci sorulara verilen cevapların standart sapmaları en yüksektir.

Katılımcılardan sadece bir kişi kadın olduğu (K11), sadece bir kişi solak olduğu (K6) ve sadece bir kişi jest tabanlı kontrol cihazı deneyimi olduğu (K6) için bu parametrelere dayalı dayanaklı bir yorum çıkarılamamaktadır. Ayrıca katılımcılardan 11 tanesinin yaşları 27 ile 35 arasında değişmekle birlikte yalnızca bir katılımcının (K10) yaşı 60'dır. Yani katılımcıların hemen hepsinin genç ve fare ile uzun yıllar deneyimi olduğu görülebilir.

Tablo 4.6. Memnuniyet anketi ile ilgili istatistiksel bilgiler

No:	Sorular \ Seçenekler	Ortalama	Ortanca	Std Sap	En çok seçilen	Min	Maks
1	**Operasyonları gerçekleştirmek için harcanan mental/zihinsel çaba** *(yüksektir=1 düşüktür=7)*	4,25	3,5	1,658	3	2	7
2	**Operasyonlar esnasındaki akıcılık** *(kabadır/azdır=1 yumuşaktır/yüksektir=7)*	4,75	5	1,138	5	2	6
3	**Doğru/Tam işlem yapmak** *(zordur=1 kolaydır=7)*	5,75	6	1,138	6	3	7
4	**Operasyonları gerçekleştirmek için harcanan fiziksel çaba** *(yüksektir=1 düşüktür=7)*	4,33	4,5	1,371	5	2	6
5	**Operasyonu gerçekleştirme sürati** *(yavaştır=1 hızlıdır=7)*	5,33	5,5	1,073	6	3	7
6	**Genel kullanım rahatlığı/konforu** *(rahatsızdır=1 rahattır=7)*	5,16	5,5	1,337	6	2	7
7	**Bütün olarak düşünürsek, etkileşim** *(zordur=1 kolaydır=7)*	5,5	6	1,000	6	3	7
8	**Hangisini daha çok tercih ederdiniz? MS Kinect kontrolünü mü yoksa Fare kontrolünü mü?** *(Fare=1 Jest=7)*	4,00	4	1,651	4	1	7
9	**Bilek yorgunluğu hissi** *(çoktur=1 azdır =7)*	6,08	6	0,996	7	4	7
10	**Kol yorgunluğu hissi** *(çoktur=1 azdır =7)*	5,92	5,5	0,996	5	5	7
11	**Omuz yorgunluğu hissi** *(çoktur=1 azdır =7)*	6,50	7	0,674	7	5	7

Katılımcıların anketin son bölümünde yer alan belirtmek istedikleri diğer noktalar sorusuna verdikleri cevapları değerlendirilmiştir. K4 ve K6 katılımcıları fiziksel

yorgunluğa sebep olacak kadar fazla kullanılmadığını söylemişlerdir. K11 katılımcısı sistemin farklı alanlarda da kullanılabilmesi gerektiğini söylemiştir. K10 katılımcısı karşısında açıklayıcı bir kullanma talimatı olursa daha hızlı ve etkin kullanabileceğini söylemiştir. K7 katılımcısı yakınlaştırıp uzaklaştırmak için elle tutma yerine normal açık elle algılaması gerektiğini, sistemin daha hızlı tepki vermesini beklediğini son olarak da sesle komut vermenin detaylandırılması gerektiğini belirtmiştir. K5 katılımcısı görüntü akıcılığının yavaş olduğunu belirtip hızlandırılması gerektiğini belirtmiştir. K12 katılımcısı ilk kullanımda acemilik çektiğini belirtip ses komutlarının Türkçe olarak verilmesinin daha iyi olabileceğini belirtmiştir. Ayrıca katılımcıların görev bazında karşılaştıkları güçlükler de Tablo 4.7'de görülebilir.

Tablo 4.7. Katılımcılar ve karşılaştıkları güçlükler

Güçlük veya Karşılaşılan Sorun	**Görev**	**Katılımcılar**
Üç boyutlu görüntü kontrolünün akıcılığı	Görev 4 Görev 5	K1, K5, K6, K7, K8
İngilizce komut verme güçlüğü	Görev 0 Görev 6	K1, K2, K3, K5, K9, K10, K12
Maruz kalınan fiziksel yorgunluk	Görev 4 Görev 5	K1, K7, K11
Kullanılan jestlere alışma güçlüğü	Görev 4 Görev 5	K3, K12
Elle tutma (grip) işleminin algılanması güçlüğü	Görev 2 Görev 4 Görev 5	K3, K5, K8, K11

Tablo 4.7.'de tanımlı karşılaşılan güçlükler incelenmiştir. Katılımcıların maruz kaldığı fiziksel yorgunlu ve jestlere alışma güçlüğü problemlerinin aslında katılımcıların jest tabanlı bir sistemin ilk defa kullanmış olduklarından kaynaklandığı, deneyim kazandıkça bu problemlerin aşılabileceği düşünülmektedir. Katılımcıların yarısından fazlasının yaşadığı İngilizce komut verme güçlüğü sistemin ses komutlarını algılamada vurgu ve telaffuz hassasiyetinden kaynaklandığı düşünülmektedir. Sistemin Türkçe komut kabul etmesi ile bu problemin çözümü

sağlanabilir. Elle tutma ve görüntü kontrolünün akıcılığında yaşanan problemler cihazın aydınlık ortamlarda daha düşük performans göstermesinden kaynaklandığı düşünülmektedir.

Kullanılabilirlik çalışması iki aşamada yapılmıştır. Öncelikle kullanıcı testine dayalı deneysel gözlem yapılmış, katılımcılardan birtakım görevleri tamamlamaları istenmiştir. Bunun arkasından da katılımcılardan bir memnuniyet anketi doldurmaları istenmiştir. Ardından hem gözlem sonuçları hem de memnuniyet anketi bulguları yorumlanmıştır. Sonuç olarak fare daha yüksek performans göstermekle birlikte tecrübenin girdi aygıtı kullanım performansını olumlu etkilediği görülmüştür. Klasik girdi aygıtı olan klavye ve fare kullanımının zor olduğu veya uygun olmadığı durumlarda sunulan çok modlu doğal etkileşim kullanılarak üç boyutlu görüntüleri kontrol etmenin uygun bir seçim olduğu gözlenmiştir.

5. SONUÇLAR VE ÖNERİLER

5.1. Sonuç

Doğal kullanıcı arayüzleri teknolojinin gelişmesi ile daha yaygın kullanılmaya başlanmıştır. Bu tez çalışmasında hem sesle hem jestlerle etkileşim kurularak çok modlu etkileşime (URL-12, 2014) olanak sağlayan bir sistem geliştirilmiştir. Ardından geliştirilen bu sisteme kullanılabilirlik çalışması yapılmıştır. Kullanıcı testine 12 katılımcı katılmış, bunlardan 11'i fare deneyimi yüksek kullanıcılardan oluşmaktadır. Ayrıca sadece bir katılımcının Kinect deneyimi bulunmaktadır. Katılımcıların çoğu fare kullanarak görevleri daha hızlı yerine getirmiştir. Bununla birlikte Kinect deneyimi olan kullanıcının fare başarım süreleri ile Kinect başarım süreleri birbirine yakın çıktığı gözlenmiştir. Bazı katılımcılar bazı görevleri Kinect ile daha hızlı şekilde gerçekleştirmişlerdir ancak bu görevlerin hepsi kolay görevlerdir. Zor görevlerin hiçbiri Kinect ile daha hızlı gerçekleştirilememiştir. Kullanıcı testi sonucunda tecrübe ve alışkanlıkların kullanım performansını etkilediği gözlenmiştir.

Kullanıcı anketi sonuçlarına bakıldığında ise farklı bir girdi yöntemi ile başarılı şekilde işlem yapabildikleri için katılımcıların hemen hepsi olumlu yorumlarda bulunmuştur. Katılımcıların belirtmek istedikleri noktalara verdikleri cevaplar genellikle iyileştirmeye yönelik tavsiyeler şeklinde olduğu gözlenmiştir. Katılımcılara yöneltilen "Kinect kontrolünü yoksa fare kontrolünü mü tercih ederdiniz?" sorusuna 7'li likert skalasında 1,65 standart sapma ile 4 verildiği, en çok seçilen seçeneğin yine 4 olduğu görülmüştür. Kullanım boyunca katılımcıların fiziksel yorgunluk hissi oluşmadığını belirmeleri ile birlikte bazı katılımcılar yorgunluk hissi oluşacak kadar sistemi denemediğini belirtmiştir. Ayrıca katılımcılar "Genel kullanım rahatlığı nedir?" sorusuna verdikleri ortalama puan 1,33 standart sapma ile 5,16 çıkmıştır. Yani sistem kullanım rahatlığı konusunda katılımcıları tatmin ettiği söylenebilir. Katılımcılarda gözlenen diğer bir durum da sunulan doğal etkileşim yöntemini rahat ve kolay şekilde öğrenip uygulayabiliyor olmalarıdır.

Kırmızıbayrak ve arkadaşlarının (2011) çalışmasındaki sunulan yöntemde döndürme işlemleri için jest tabanlı arayüz fareye göre daha başarılı olmakla birlikte nesneleri seçme noktasında fare ile daha doğru işlem yapılmıştır. Bu çalışmada ise K3, K6, K7, K8, K9, K10, K12 katılımcıları görevleri Kinect ile yakın veya daha hızlı sürede gerçekleştirmekle birlikte genel başarıma bakıldığında fare ile daha başarılı işlem yapılmıştır. Ayrıca katılımcılar sistemi kullanmaktan memnun olduklarını ifade etmişlerdir. Gerçeklenen sistemin klavye ve fare kullanımının zor olduğu (ameliyathane ve steril ortamlar, ders ve sunum vb) koşullarda klavye ve fare'ye iyi bir alternatif olabileceği düşünülmektedir.

5.2. Yapılacak Çalışma Önerileri

Gerçeklenen sistem için iyileştirme ve zenginleştirme yapılarak sistem daha fonksiyonel hale getirilebilir. İyileştirme anlamında sistemin pencere arayüzünde her eksen takımı için üç boyutlu görüntüyü gösteren küçük pencereler eklenerek ve farklı üç boyutlu görüntüleri açmaya olana sağlanarak daha işlevsel hale getirilebilir. Zenginleştirme anlamında ise ses komutları genişletilip çoğaltılarak daha çeşitli hale getirilebilir. Örneğin “reset to x axis” denilerek x ekseninden bakacak şekilde üç boyutlu görüntü ilk duruma getirilir. Benzer şekilde “save orientation” dendiği zaman göntünün oryantasyonu belirli bir kimlikle kaydedildikten sonra aynı kimliği söyleyerek örneğin “load orientation 1” diyerek görüntünün kaydedilen oryantasyonda geri yüklenmesi sağlanabilir. Ses komutlarını Türkçe yapılması için farklı bir konuşma motoru (speech engine) ile çalışılabilir. ITK (URL-6, 2014) yazılım kütüphanesi yardımı ile bir veritabanı üzerinden veya herhangi bir lokasyondan farklı formattaki üç boyutlu görüntüler yüklenip incelenebilir. Medikal görüntüler için özelleştirilmiş bir sistem yapılabilir. Medikal görüntüler için üç boyutlu görüntü yanında her eksen takımındaki CT görüntüleri kesit olarak gösterilebilir. Kirmizibayrak ve arkadaşlarının (2011) bahsettiği Sihirli Lens özelliği eklenerek Kirmizibayrak ve arkadaşlarının (2011) gerçeklediği sistemle karşılaştırma yapılabilir.

Kullanılabilirlik anlamında yapılabilecek çalışmalardan bahsedecek olursak ISO 9241-9 (2000) dökümanında yer alan Fitts' kanunu (MacKenzie, 1992) baz alınarak kullanıcı testine dayalı deney düzenlenerek verimlilikleri (throughput) ölçülebilir. Gerçeklenen sistemin fare yerine dokunmatik ekran ile karşılaştırması yapılabilir.

Sistemin kullanımında cinsiyet ya da katılımcıların hangi ellerini kullandıklarının bir etkisinin olup olmadığını tespit etmek için bu iki faktör göz önünde bulundurularak eşit dağılımla oluşturulmuş bir katılımcı grubu ile çalışma tekrar edilebilir. Kinect deneyimi veya başka jest tabanlı kontrol deneyimi olan katılımcıların sayısı artırılarak cihaz hakkındaki bulgular sağlamlaştırılıp çeşitlendirilebilir.

KAYNAKLAR

Bin Mohd Sidik M. K., Bin Sunar M. S., Bin Ismail I., Bin Mokhtar M. K., Jusoh N. B. M., A study on natural interaction for human body motion using depth image data, *Digital Media and Digital Content Management (DMDCM),* DOI: 10.1109/DMDCM.2011.26.

Bolt R. A., "Put-that-there": Voice and gesture at the graphics interface, *ACM,* 1980, **14**, 3, 262-270.

Cuccurullo S., Francese R., Murad S., Passero I., Tucci M., A gestural approach to presentation exploiting motion capture metaphors, *Proceedings of the International Working Conference on Advanced Visual Interfaces,* DOI:10.1145/2254556.2254584

Çağıltay K., *İnsan bilgisayar etkileşimi ve kullanılabilirlik mühendisliği: Teoriden pratiğe,* ODTÜ Yayıncılık, Ankara, 2011.

Folmer E., Bosch J., Architecting for usability: a survey. *Journal of systems and software,* 2004, **70**(1), 61-78.

Gallo L., De Pietro G., Marra I., 3D interaction with volumetric medical data: experiencing the Wiimote, *Proceedings of the 1st international conference on Ambient media and systems,* Brussel, Belgium, 11-14 February 2008.

Gallo L., Ciampi M., Wii Remote-enhanced Hand-Computer interaction for 3D medical image analysis, *Current Trends in Information Technology (CTIT),* DOI: 10.1109/CTIT.2009.5423137.

Gallo L., Placitelli A. P., Ciampi M., Controller-free exploration of medical image data: Experiencing the Kinect. In *Computer-Based Medical Systems (CBMS),* DOI: 10.1109/CBMS.2011.5999138.

Hartmann F., Schlaefer A., Feasibility of touch-less control of operating room lights, *International journal of computer assisted radiology and surgery,* 2013, **8**(2), 259-268.

Hauptmann A. G., Speech and gestures for graphic image manipulation, *ACM SIGCHI Bulletin,* 1989, **20**, 241-245.

Howard A., 7 areas beyond gaming where Kinect could play a role, http://radar.oreilly.com/2010/12/dancing-with-kinects-future-in.html, (Ziyaret tarihi: 15 Mayıs 2014).

ISO, 9241-9 Ergonomic requirements for office work with visual display terminals (VDTs) - Part 9: Requirements for non-keyboard input devices, International Organization for Standardization, 2000.

Jaimes A., Sebe N., Multimodal human computer interaction: A survey. *Computer vision and Image Understanding*, 2005, **108**(1-2), 116-134.

Kirmizibayrak C., Radeva N., Wakid M., Philbeck J., Sibert J., Hahn J., Evaluation of gesture based interfaces for medical volume visualization tasks, *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*, DOI: 10.1145/2087756.2087764.

Lange B., Koenig S., McConnell E., Chang C., Juang R., Suma E., Rizzo A., Interactive game-based rehabilitation using the Microsoft Kinect, *Virtual Reality Short Papers and Posters (VRW)*, DOI: 10.1109/VR.2012.6180935.

Lazar J., Feng J. H., Hochheiser H., *Research methods in human-computer interaction*, John Wiley & Sons, Glasgow, England, 2010.

MacKenzie I. S., Fitts' law as a research and design tool in human-computer interaction, *Human-computer interaction*, 1992, **7**(1), 91-139.

Martins M., Cunha A., Morgado L., Usability test of 3Dconnexion 3D mice versus keyboard+mouse in Second Life undertaken by people with motor disabilities due to medullary lesions, *Procedia Computer Science*, 2012, **14**, 119-127.

Natapov D., Castellucci S. J., MacKenzie I. S., ISO 9241-9 evaluation of video game controllers, *Proceedings of Graphics Interface 2009*, Toronto, Canada, 25-27 May 2009.

Nielsen J., *Usability Engineering*, Academic Press, Boston, 1993.

Ren Z., Meng J., Yuan J., Depth camera based hand gesture recognition and its applications in human-computer-interaction, *8th International Conference on Information, Communications and Signal Processing (ICICS)*, DOI: 10.1109/ICICS.2011.6173545.0

Ronchetti M., Avancini M., Using Kinect to emulate an Interactive Whiteboard, Master Thesis, Trento University, Mathematics, Physics and Natural Science Department, Trento, 2011.

Rydén F., Chizeck H. J., Kosari S. N., King H., Hannaford B., Using kinect and a haptic interface for implementation of real-time virtual fixtures, *2nd Workshop on Proceedings of the RGB-D: Advanced Reasoning with Depth Cameras*, Washington, USA, 27 June 2011.

Ryu Y. S., Do Hyong Koh D. R., Um D., Usability Evaluation of Touchless Mouse Based on Infrared Proximity Sensing. *Journal of Usability Studies*, 2011, **7**(1), 31-39.

Schwaller M., Lalanne D., Khaled O. A., PyGmI: creation and evaluation of a portable gestural interface, *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries,* DOI: 10.1145/1868914/1869026.

Shotton J., Sharp T., Kipman A., Fitzgibbon A., Finocchio M., Blak, A., Moore R., Real-time human pose recognition in parts from single depth images, *Communications of the ACM*, 2013, **56**(1), 116-124.

Şimşek S., Durdu P. O., Whiteboard Teknolojisi İle Etkileşimli Öğrenme Ortamı Gerçekleştirilmesi, *e-Journal of New World Sciences Academy (NWSA),* 2012, **7**(2).

Tanaka K., Parker J. R., Baradoy G., Sheehan D., Holash J. R., Katz L., A comparison of exergaming interfaces for use in rehabilitation programs and research, *The Journal of the Canadian Game Studies Association.*, 2012, **6**(9).

Albert B., Tullis T., *Measuring the user experience*, Elsevier, USA, 2008.

Tuntakurn A., Thongvigitmanee S. S., Sa-Ing V., Makhanov S. S., Hasegawa S., Natural interaction on 3D medical image viewer software, *Biomedical Engineering International Conference (BMEiCON),* DOI: 10.1109/BMEiCon.2012.6465424.

URL-1: http://www.apple.com/iphone/, (Ziyaret tarihi: 15 Mayıs 2014).

URL-2: http://www.evoluce.com/, (Ziyaret tarihi: 15 Mayıs 2014).

URL-3: http://en.wikipedia.org/wiki/Human%E2%80%93computer_interaction, (Ziyaret tarihi: 15 Mayıs 2014).

URL-4: http://en.wikipedia.org/wiki/History_of_the_graphical_user_interface, (Ziyaret tarihi: 15 Mayıs 2014).

URL-5: http://en.wikipedia.org/wiki/Isosurface, (Ziyaret tarihi: 18 Mayıs 2014).

URL-6: http://en.wikipedia.org/wiki/Insight_Segmentation_and_Registration_Toolkit, (Ziyaret tarihi: 18 Mayıs 2014).

URL-7: http://en.wikipedia.org/wiki/Kinect, (Ziyaret tarihi: 11 Mart 2014).

URL-8: http://en.wikipedia.org/wiki/Leap_Motion, (Ziyaret tarihi: 12 Mart 2014).

URL-9: http://research.microsoft.com/en-us/collaboration/focus/nui/default.aspx, (Ziyaret tarihi: 22 Mayıs 2014).

URL-10: http://wiki.etc.cmu.edu/unity3d/index.php/Sony_PlayStation_Move, (Ziyaret tarihi: 22 Mayıs 2014).

URL-11: http://social.msdn.microsoft.com/Forums/en-US/d79abef7-b13d-41ed-a11a-04ba63c134be/kinect-sdk-vs-openni?forum=kinectsdk, (Ziyaret tarihi: 22 Mayıs 2014).

URL-12: http://en.wikipedia.org/wiki/Multimodal_interaction, (Ziyaret tarihi: 23 Mayıs 2014).

URL-13: http://en.wikipedia.org/wiki/OpenCV, (Ziyaret tarihi: 23 Mayıs 2014).

URL-14: http://en.wikipedia.org/wiki/OpenNI, (Ziyaret tarihi: 23 Mayıs 2014).

URL-15: http://en.wikipedia.org/wiki/PlayStation_Move, (Ziyaret tarihi: 23 Mayıs 2014).

URL-16: http://en.wikipedia.org/wiki/PrimeSense, (Ziyaret tarihi: 27 Mayıs 2014).

URL-17: http://qt-project.org/, (Ziyaret tarihi: 23 Mayıs 2014).

URL-18: http://secondlife.com/, (Ziyaret tarihi: 27 Mayıs 2014).

URL-19: http://slicer.org/pages/Introduction, (Ziyaret tarihi: 27 Mayıs 2014).

URL-20: https://www.thalmic.com/en/myo/, (Ziyaret tarihi: 27 Mayıs 2014).

URL-21: http://en.wikipedia.org/wiki/User_interface, (Ziyaret tarihi: 27 Mayıs 2014).

URL-22: http://www.vtk.org/Wiki/VTK, (Ziyaret tarihi: 28 Mayıs 2014).

URL-23: http://en.wikipedia.org/wiki/Wii_Remote, (Ziyaret tarihi: 1 Haziran 2014).

URL-24: http://en.wikipedia.org/wiki/X-ray_computed_tomography, (Ziyaret tarihi: 28 Mayıs 2014).

Virzi R. A., Refining the test phase of usability evaluation: how many subjects is enough?, *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 1992, **34**(4), 457-468.

Vo M. T., Waibel A., Multi-modal HCI: combination of gesture and speech recognition, *INTERACT'93 and CHI'93 Conference Companion on Human Factors in Computing Systems*, DOI: 10.1145/259964.260076.

Wachs J., Stern H., Edan Y., Gillam M., Feied C., Smith M., Handler J., Gestix: a doctor-computer sterile gesture interface for dynamic environments, *Soft Computing in Industrial Applications*, 2007, **39**, 30-39.

Williamson B. M., Wingrave C., LaViola J. J., Roberts T., Garrity P., Natural full body interaction for navigation in dismounted soldier training, *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC),*Orlando, Florida, 1-4 December 2011.

Wilson A. D., Using a depth camera as a touch sensor, *ACM international conference on interactive tabletops and surfaces*, DOI: 10.1145/1936652.1936665.

Woolrych A., Cockton G., Why and when five test users aren't enough, *Proceedings of IHM-HCI 2001 conference*, 2001, **2**, 105-108.

Yeh S. C., Hwang W. Y., Huang T. C., Liu W. K., Chen Y. T., Hung Y. P., A study for the application of body sensing in assisted rehabilitation training, *2012 International Symposium on Computer, Consumer and Control (IS3C)*, DOI: 10.1109/IS3C.2012.240.

Zhang Z., Overview of usability evaluation methods, http://www.cs.umd.edu/~zzj/UsabilityHome.html, (Ziyaret tarihi: 27 Mayıs 2014).

EKLER

Ek−A BİLGİLENDİRİLMİŞ ONAM FORMU VE KULLANICILARA SORULMUŞ SORULAR

Kocaeli Üniversitesi

İnsan Bilgisayar Etkileşimi
Çalışma Grubu

Üç Boyutlu Görüntü Manipulasyon Aracı Kullanılabilirlik Testi

Giriş ve Amaç

Öncelikle bizlere zaman ayırdığınız için çok teşekkür ederiz! Bizler Kocaeli Üniversitesi İnsan Bilgisayar Etkileşimi Çalışma Grubu'ndan Fatih Ergüner. Başlamadan önce sizlere bugün deneyeceğimiz sistem hakkında biraz bilgi vermek istiyoruz.

Üç boyutlu görüntüleri jest ve ses komutlarıyla kontrol etmek amacıyla bir araç geliştirilmiştir. Amacımız geliştirilen bu aracın klasik girdi yöntemlerine göre ne kadar kullanılabilir olduğunu tespit edebilmektir.

Test Katılımcısının Rolü

Katılımcı olarak sizlerden talep edilen şey geliştirilen bu aracı hem klasik girdi aygıtı olan fare hem de MS Kinect kullanarak verilen görevleri yerine getirmektir. Maksadımız fare kullanımı ile MS Kinect kullanımı arasında kullanılabilirlik açısından bir karşılaştırma yapmaktır. Test sırasında kaydettiğimiz bilgiler sadece akademik çalışmada kullanılacak, hiçbir şekilde başka kurum ya da kişilerle paylaşılmayacaktır..

Öncelikle burada test ettiğimiz şey aracın başarısıdır, yani sizleri test etmiyoruz. Bu sebeple üzerinizde hiçbir baskı veya zorlayıcı etken hissetmeyin. Ölçümler yapabilmemiz için sizler aracı kullanırken ses, ekran ve video kaydı yapılacaktır. Bu yüzden yapmaya çalıştığınız işlemi sesli düşünerek yapmanızı istiyoruz. Sözgelimi "şimdi elimi yumruk yapıp çekiyorum, görüntü hareket etmiyor", "elimi hareket ettiriyorum, hop! görüntü kayboldu" gibi sesli şekilde geribildirimde bulunmanız çalışmayı değerlendirmemiz açısından son derece önem arz etmektedir. Kayıtların hiçbir şekilde üçüncü şahıslarla paylaşılmayacağının tekrar altını çizmek isteriz.

Bu teste katılım tamamen gönüllülük esasına dayanmaktadır ve aracı denemekte hiçbir fiziksel ve psikolojik risk bulunmamaktadır.

Çalışmaya Katılım Onayı

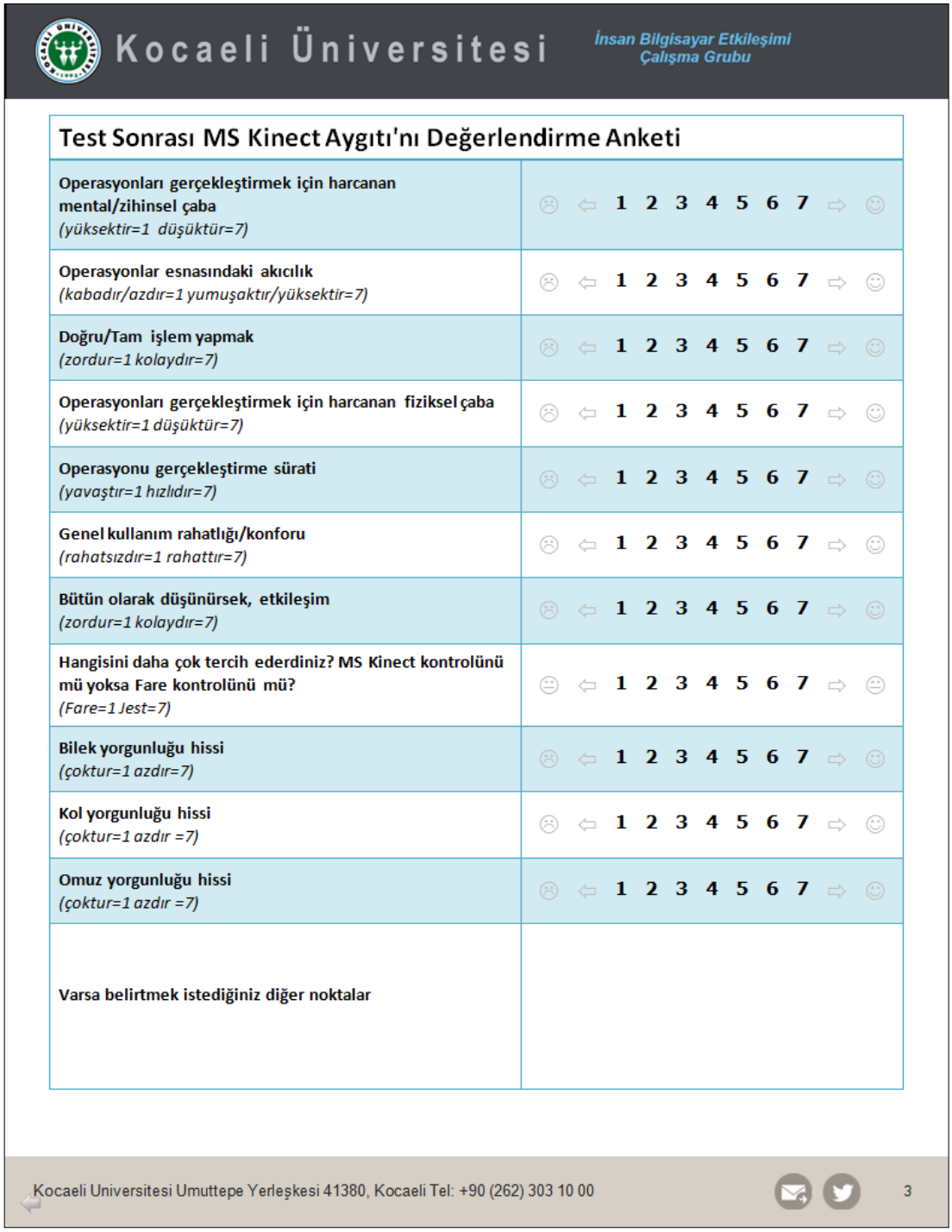
Çalışmaya dair açıklamaları okudum ve katılımcı olarak hak ve sorumluluklarımın farkındayım. Çalışmayı gerçekleştirenler tarafından kimlik bilgilerimin özel olduğu ve üçüncü şahıslarla paylaşılmayacağı temin edilmiştir.
Çalışmaya katılmayı kabul ediyorum.

Ad-Soyad_____

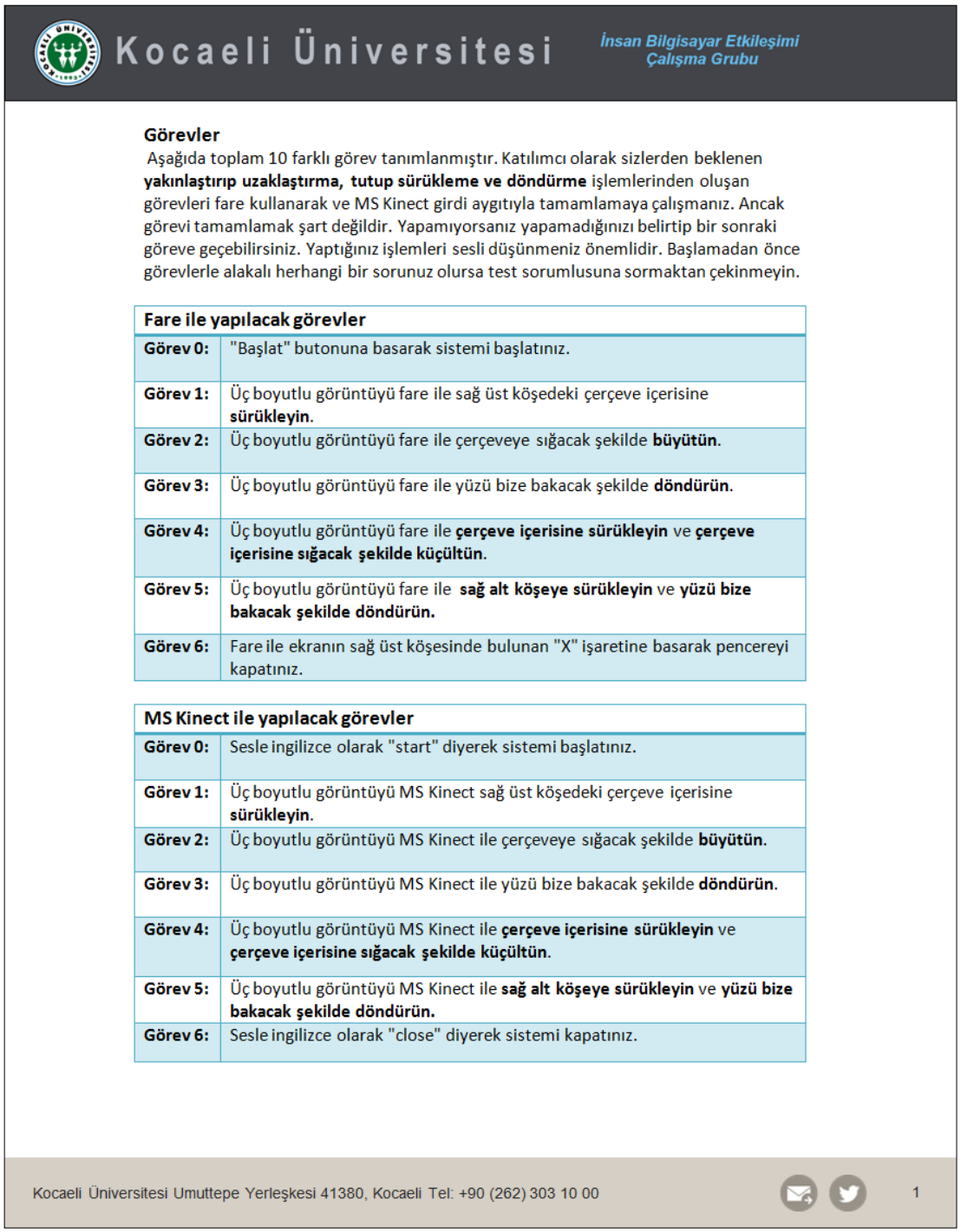
İmza_____

Kocaeli Üniversitesi Umuttepe Yerleşkesi 41380, Kocaeli Tel: +90 (262) 303 10 00 1

Teste Başlamadan Önce	
Yaş	
Cinsiyet	☐ Erkek ☐ Kadın
Öğrenim Durumu *(Lise, Lisans, Y. Lisans vs.)*	
Meslek	
Günlük Bilgisayar kullanımı *(örn. günde 5 saat)*	
Varsa temassız kontrol cihazı aylık deneyiminiz *(örn. aylık 8 saat)*	
Ne derecede İngilizce biliyorsunuz? *(bir tanesini seçin)*	1- Bilmiyorum 2-Giriş Seviyesi 3-Orta Seviye 4-İleri Seviye 5-Ana dili gibi
Sağ elinizi mi, yoksa sol elinizi mi kullanırsınız?	☐ Sağ El ☐ Sol El


Kocaeli Üniversitesi
İnsan Bilgisayar Etkileşimi
Çalışma Grubu
Test Sonrası MS Kinect Aygıtı'nı Değerlendirme Anketi
Operasyonları gerçekleştirmek için harcanan mental/zihinsel çaba
(yüksektir=1 düşüktür=7)
Operasyonlar esnasındaki akıcılık
(kabadır/azdır=1 yumuşaktır/yüksektir=7)
Doğru/Tam işlem yapmak
(zordur=1 kolaydır=7)
Operasyonları gerçekleştirmek için harcanan fiziksel çaba
(yüksektir=1 düşüktür=7)
Operasyonu gerçekleştirme sürati
(yavaştır=1 hızlıdır=7)
Genel kullanım rahatlığı/konforu
(rahatsızdır=1 rahattır=7)
Bütün olarak düşünürsek, etkileşim
(zordur=1 kolaydır=7)
Hangisini daha çok tercih ederdiniz? MS Kinect kontrolünü mü yoksa Fare kontrolünü mü?
(Fare=1 Jest=7)
Bilek yorgunluğu hissi
(çoktur=1 azdır=7)
Kol yorgunluğu hissi
(çoktur=1 azdır =7)
Omuz yorgunluğu hissi
(çoktur=1 azdır =7)
1 2 3 4 5 6 7
Varsa belirtmek istediğiniz diğer noktalar
Kocaeli Universitesi Umuttepe Yerleşkesi 41380, Kocaeli Tel: +90 (262) 303 10 00
3

Ek–B GERÇEKLENEN SİSTEME AİT KULLANIM KLAVUZU VE GÖREVLER LİSTESİ



Kocaeli Üniversitesi

İnsan Bilgisayar Etkileşimi Çalışma Grubu

Görevler

Aşağıda toplam 10 farklı görev tanımlanmıştır. Katılımcı olarak sizlerden beklenen **yakınlaştırıp uzaklaştırma, tutup sürükleme ve döndürme** işlemlerinden oluşan görevleri fare kullanarak ve MS Kinect girdi aygıtıyla tamamlamaya çalışmanız. Ancak görevi tamamlamak şart değildir. Yapamıyorsanız yapamadığınızı belirtip bir sonraki göreve geçebilirsiniz. Yaptığınız işlemleri sesli düşünmeniz önemlidir. Başlamadan önce görevlerle alakalı herhangi bir sorunuz olursa test sorumlusuna sormaktan çekinmeyin.

Fare ile yapılacak görevler	
Görev 0:	"Başlat" butonuna basarak sistemi başlatınız.
Görev 1:	Üç boyutlu görüntüyü fare ile sağ üst köşedeki çerçeve içerisine **sürükleyin**.
Görev 2:	Üç boyutlu görüntüyü fare ile çerçeveye sığacak şekilde **büyütün**.
Görev 3:	Üç boyutlu görüntüyü fare ile yüzü bize bakacak şekilde **döndürün**.
Görev 4:	Üç boyutlu görüntüyü fare ile **çerçeve içerisine sürükleyin** ve **çerçeve içerisine sığacak şekilde küçültün**.
Görev 5:	Üç boyutlu görüntüyü fare ile **sağ alt köşeye sürükleyin** ve **yüzü bize bakacak şekilde döndürün.**
Görev 6:	Fare ile ekranın sağ üst köşesinde bulunan "X" işaretine basarak pencereyi kapatınız.

MS Kinect ile yapılacak görevler	
Görev 0:	Sesle ingilizce olarak "start" diyerek sistemi başlatınız.
Görev 1:	Üç boyutlu görüntüyü MS Kinect sağ üst köşedeki çerçeve içerisine **sürükleyin**.
Görev 2:	Üç boyutlu görüntüyü MS Kinect ile çerçeveye sığacak şekilde **büyütün**.
Görev 3:	Üç boyutlu görüntüyü MS Kinect ile yüzü bize bakacak şekilde **döndürün**.
Görev 4:	Üç boyutlu görüntüyü MS Kinect ile **çerçeve içerisine sürükleyin** ve **çerçeve içerisine sığacak şekilde küçültün**.
Görev 5:	Üç boyutlu görüntüyü MS Kinect ile **sağ alt köşeye sürükleyin** ve **yüzü bize bakacak şekilde döndürün.**
Görev 6:	Sesle ingilizce olarak "close" diyerek sistemi kapatınız.

Kocaeli Üniversitesi Umuttepe Yerleşkesi 41380, Kocaeli Tel: +90 (262) 303 10 00 1

Kullanım Şekli

Deneyeceğiniz aracı iki farklı yöntemle kontrol edebilirsiniz. Birincisi **fare**, ikincisi de **MS Kinect**'tir. Aşağıda iki farklı yöntemin nasıl çalıştığını anlatan açıklamalar mevcuttur.

Fare:

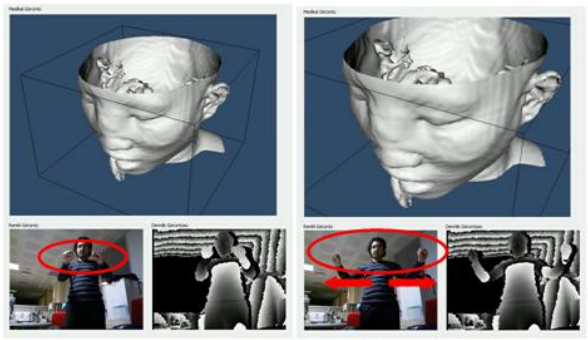
- **Yakınlaştırma/Uzaklaştırma:**
  Yakınlaştırıp uzaklaştırma (Zoom) işlemi şu şekilde yapılır:
  Yakınlaştırmak için **fare sağ tuşuna basılı tutularak imleç yukarı**, uzaklaştırmak için **fare sağ tuşuna basılı tutularak imleç aşağı** doğru hareket ettirilir.

- **Sürükleme**
  Sürükleme (Pan) işlemi şu şekilde yapılır:
  Fare 3. tuşuna basılı tutularak imleç sağa, sola, yukarı ve aşağı doğru hareket ettirilir.

- **Döndürme:**
  Döndürme (Rotate) işlemi de şu şekilde yapılır:
  Fare sol tuşuna basılı tutularak imleç sağa, sola, yukarı ve aşağı doğru hareket ettirilir.

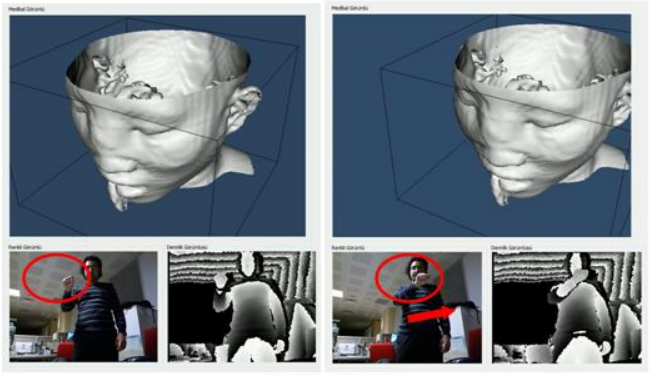
MS Kinect:

- **Yakınlaştırma/Uzaklaştırma:**
  Yakınlaştırıp uzaklaştırma (Zoom) işleminde iki el birlikte kullanılır. Şekil 1'de görüldüğü gibi her iki elle tutma işleminden sonra her iki el birbirlerinden yana doğru açılırsa üç boyutlu görüntü büyümekte, her iki elle tutma işleminden sonra her iki el birbirlerine doğru yaklaştırılırsa da üç boyutlu görüntü küçülmektedir. Elle tutma bırakıldığında veya eller serbest şekilde aşağı doğru salındığında etkileşim deaktif olur.



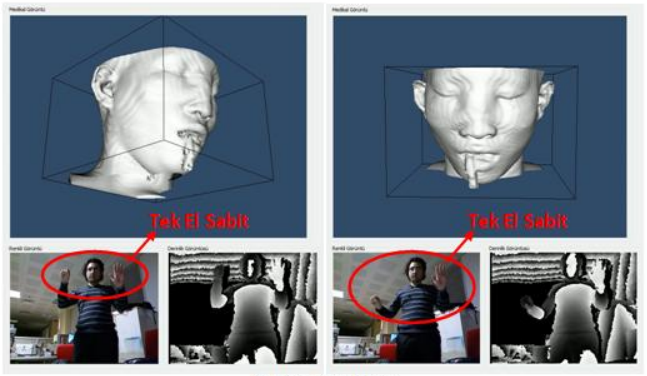
Şekil 1 - Yakınlaştırma / Uzaklaştırma

- **Sürükleme**
  Sürükleme (Pan) işlemi tek el ile icra edilir ve hangi elle jestin icra edildiğinden bağımsızdır. Şekil 2'te görüldüğü gibi elle tutma (grip) işleminden sonra sola kaydırıldığında üç boyutlu görüntü sola, sağa kaydırıldığında üç boyutlu görüntü sağa, yukarıya kaydırıldığında üç boyutlu görüntü yukarı ve aşağı kaydırıldığında üç boyutlu görüntü aşağı doğru hareket eder. Elle tutma bırakıldığında etkileşim deaktif olur.



Şekil 2 - Sürükleme

- **Döndürme:**
  Döndürme (Rotate) işleminde de iki el birlikte kullanılır. Şekil 3'de görüldüğü gibi bir el açık şekilde dururken diğer elle tutarak (grip) sağa, sola, yukarı ve aşağı doğru hareket ettirilerek üç boyutlu medikal görüntü manipule edilir.



Şekil 3 - Döndürme

Ek–C GELİŞTİRİLEN SİSTEM İÇİN YAZILAN KAYNAK KODLAR

void UCDeviceWindowInteractor::Start()

```
// Let the compositing handle the event loop if it wants to.
if (this->HasObserver(vtkCommand::StartEvent) && !this-
>HandleEventLoop)
{
this->InvokeEvent(vtkCommand::StartEvent,NULL);
return;
}

MSG msg;
while (GetMessage(&msg, NULL, 0, 0))
{
KinectVolumeInteractor* interactor =
reinterpret_cast<KinectVolumeInteractor*>(GetInteractorStyle());
//setting the window size
int* winSize = this->GetRenderWindow()->GetSize();
interactor->setWindowSize(winSize);

if(msg.message > 1025)
{
        int x = 0;
}

//interpreting the windows messages, map them into vtk messages
switch ( msg.message )
{
case INVOKE_RESET_EVENT:
        interactor->InvokeEvent(interactor->ResetEvent, 0);
        break;
case INVOKE_PAN_EVENT:
        interactor->InvokeEvent(interactor->PanEvent, 0);
        break;
case INVOKE_ZOOM_EVENT:
        interactor->InvokeEvent(interactor->ZoomEvent, 0);
        break;
case INVOKE_ROTATE_EVENT:
        interactor->InvokeEvent(interactor->RotateEvent, 0);
        break;
case INVOKE_fare_g1_Event:
        interactor->loadMouseTask1();
        break;
case INVOKE_fare_g2_Event:
        interactor->loadMouseTask2();
        break;
} ...
```

void KinectMediator::updateKinectST()

```
DWORD dwWaitResult;
bool continueProcessing = true;
while ( continueProcessing )
{
        if ( WAIT_OBJECT_0 ==
WaitForSingleObject(m_hNextColorFrameEvent, 0) )
        {
                ProcessColor();
        }
        if ( WAIT_OBJECT_0 ==
WaitForSingleObject(m_hNextDepthFrameEvent, 0) )
        {
                ProcessDepth();
        }
        // Wait for 0ms, just quickly test if it is time to process
a skeleton
        if ( WAIT_OBJECT_0 ==
WaitForSingleObject(m_hNextSkeletonEvent, 0) )
        {
                dwWaitResult = WaitForSingleObject(
                                                m_hMutex,    //
handle to mutex
                                                INFINITE);  // no
time-out interval

                switch(dwWaitResult)
                {
                case WAIT_OBJECT_0:

                        __try
                        {
                                m_FoundSkeleton = false;
                                NUI_SKELETON_FRAME skeletonFrame =
{0};

                                // Get the skeleton frame that is
ready
                                if (SUCCEEDED(m_pNuiSensor-
>NuiSkeletonGetNextFrame(0, &skeletonFrame)))
                                {
                                        //// Process the skeleton
frame
                                        for ( int i = 0 ; i <
NUI_SKELETON_COUNT ; i++ )
                                        {

        NUI_SKELETON_TRACKING_STATE trackingState =
skeletonFrame.SkeletonData[i].eTrackingState;

                                                if ( trackingState ==
NUI_SKELETON_TRACKED )
                                                {
                                                        m_SkeletonId =
```

```
i;

        m_FoundSkeleton = true;


        handleSkeletonPoints( skeletonFrame.SkeletonData[i]);
                                                }
                                        }
                                }

                                // no skeletons!
                                if( !m_FoundSkeleton )
                                {
                                        //std::cout << "No Skeleton
TRACKED" << std::endl;
                                }

                                // smooth out the skeleton data
                                HRESULT hr = m_pNuiSensor-
>NuiTransformSmooth(&skeletonFrame,NULL);
                                if ( FAILED(hr) )
                                {
                                        //std::cout << "Smoothing is
NOT SUCCESSFUL" << std::endl;
                                }


                                Vector4 v;
                                m_pNuiSensor-
>NuiAccelerometerGetCurrentReading(&v);
                                hr =m_nuiIStream-
>ProcessSkeleton(NUI_SKELETON_COUNT,



                                if( FAILED( hr ) )
                                {
                                        cout<<"Process Skeleton
failed"<<endl;
                                }

                                // we found a skeleton, re-start the
skeletal timer
                                m_bScreenBlanked = false;


        UpdateTrackedSkeletons(skeletonFrame);

                        }//end of __try

                        __finally
                        {

                                // Release ownership of the mutex
object
                                if (! ReleaseMutex(m_hMutex))
```

```
                                {
                                        std::cout << "Mutex CAN NOT
be released in updateKinectST()" << std::endl;
                                }

                        }//end of __finally

                        break; //break of "case WAIT_OBJECT_0"

                // The thread got ownership of an abandoned mutex
                // The database is in an indeterminate state
                case WAIT_ABANDONED:
                        return;

                }//end of switch

        } //WaitForSingleObject m_hNextSkeletonEvent


        if (WAIT_OBJECT_0 ==
WaitForSingleObject(m_hNextInteractionEvent, 0))
        {
                ProcessInteraction();
        }

        if ( WAIT_OBJECT_0 == WaitForSingleObject(m_pSpeech-
>getSpeechEvent(), 0) )
        {
                m_pSpeech->ProcessSpeech();

                if(getSpeechAction() ==
KinectSpeechAction::KSA_START )
                {
                        emit signalStartGestureCommand();
                }
                else if(getSpeechAction() ==
KinectSpeechAction::KSA_CLOSE )
                {
                        emit signalExitProgram();
                }

        }//WaitForSingleObject m_pSpeech

}// main Loop
```

void KinectMediator::handleSkeletonPoints(const NUI_SKELETON_DATA & skel)

```
//setting the previous points
mpPrevHandLeft = mpHandLeft;
```

```
mpPrevHandRight = mpHandRight;
mpPrevWristLeft = mpWristLeft;
mpPrevWristRight = mpWristRight;

//setting the new points
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_HEAD],
mpHead, mWindowWidth, mWindowHeight);
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_SHOULD
ER_CENTER], mpShoulderCenter, mWindowWidth, mWindowHeight);
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_SHOULD
ER_LEFT], mpShoulderLeft, mWindowWidth, mWindowHeight);
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_SHOULD
ER_RIGHT], mpShoulderRight, mWindowWidth, mWindowHeight);
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_ELBOW_
LEFT], mpElbowLeft, mWindowWidth, mWindowHeight);
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_ELBOW_
RIGHT], mpElbowRight, mWindowWidth, mWindowHeight);
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_WRIST_
LEFT], mpWristLeft, mWindowWidth, mWindowHeight);
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_WRIST_
RIGHT], mpWristRight, mWindowWidth, mWindowHeight);
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_HAND_L
EFT], mpHandLeft, mWindowWidth, mWindowHeight);
mpHandLeftWorldCoord =
skel.SkeletonPositions[NUI_SKELETON_POSITION_HAND_LEFT];
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_HAND_R
IGHT], mpHandRight, mWindowWidth, mWindowHeight);
mpHandRightWorldCoord =
skel.SkeletonPositions[NUI_SKELETON_POSITION_HAND_RIGHT];
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_SPINE]
, mpSpine, mWindowWidth, mWindowHeight);
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_HIP_CE
NTER], mpHipCenter, mWindowWidth, mWindowHeight);
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_HIP_LE
FT], mpHipLeft, mWindowWidth, mWindowHeight);
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_HIP_RI
GHT], mpHipRight, mWindowWidth, mWindowHeight);
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_KNEE_L
EFT], mpKneeLeft, mWindowWidth, mWindowHeight);
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_KNEE_R
IGHT], mpKneeRight, mWindowWidth, mWindowHeight);
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_ANKLE_
LEFT], mpAnkleLeft, mWindowWidth, mWindowHeight);
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_ANKLE_
RIGHT], mpAnkleRight, mWindowWidth, mWindowHeight);
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_FOOT_L
EFT], mpFootLeft, mWindowWidth, mWindowHeight);
SkeletonToScreen(skel.SkeletonPositions[NUI_SKELETON_POSITION_FOOT_R
IGHT], mpFootRight, mWindowWidth, mWindowHeight);

//checking which hand is active hand : left,right,both,none
checkActiveHand();


//calculating the torso depth
calcAverageTorsoDepth();
```

```
/*
logCoords << "Avg Torso Depth: " << mpTorsoDepth << endl;
logCoords << "----------------------" << endl;
*/
//emit signalWriteMessage(QString::number(mpTorsoDepth));

// if hands are 40cm front of torso, then gesture commanding is
activated
determineGestureCommandActivated();

//checking whether hand is fixed or not.
determineIsHandMoving();


float left_depth, right_depth;

left_depth =
skel.SkeletonPositions[NUI_SKELETON_POSITION_HAND_LEFT].z*100;
right_depth =
skel.SkeletonPositions[NUI_SKELETON_POSITION_HAND_RIGHT].z*100;

/*
logCoords << "LeftDepth: " << left_depth << endl;
logCoords << "RightDepth: " << right_depth << endl;
logCoords << "----------------------" << endl;
*/

emitSignalKinectUpdate();

//      cout << "LeftDepth: " << left_depth << endl;
//      logCoords << "LeftDepth: " << left_depth << endl;
//      cout << "RightDepth: " << right_depth << endl;
//      logCoords << "RightDepth: " << right_depth << endl;
//      cout << "----------------------" << endl;
//      logCoords << "----------------------" << endl;
//}
```

void KinectMediator::SkeletonToScreen(Vector4 skeletonPoint, KinectPoint& pKinectPoint, int width, int height)

```
LONG x, y;
USHORT depth;

// calculate the skeleton's position on the screen
// NuiTransformSkeletonToDepthImage returns coordinates in
NUI_IMAGE_RESOLUTION_320x240 space
NuiTransformSkeletonToDepthImage( skeletonPoint, &x, &y, &depth );

//width&height is 320x240 and windowWidth&Height is 640x480 default
```

```
and can be changed
pKinectPoint.x = static_cast<float>(x * g_ScreenWidth) / width;
pKinectPoint.y = (g_ScreenHeight - (static_cast<float>(y *
g_ScreenHeight)) / height);
pKinectPoint.depth = skeletonPoint.z * 100; // depth is meter,
converting to cm
```

void KinectMediator::UpdateTrackedSkeletons(const NUI_SKELETON_FRAME & skel)

```
DWORD nearestIDs[2] = { 0, 0 };
USHORT nearestDepths[2] = { NUI_IMAGE_DEPTH_MAXIMUM,
NUI_IMAGE_DEPTH_MAXIMUM };

// Purge old sticky skeleton IDs, if the user has left the frame,
etc
bool stickyID0Found = false;
bool stickyID1Found = false;
for ( int i = 0 ; i < NUI_SKELETON_COUNT; i++ )
{
        NUI_SKELETON_TRACKING_STATE trackingState =
skel.SkeletonData[i].eTrackingState;

        if ( trackingState == NUI_SKELETON_TRACKED || trackingState
== NUI_SKELETON_POSITION_ONLY )
        {
                if ( skel.SkeletonData[i].dwTrackingID ==
m_StickySkeletonIds[0] )
                {
                        stickyID0Found = true;
                }
                else if ( skel.SkeletonData[i].dwTrackingID ==
m_StickySkeletonIds[1] )
                {
                        stickyID1Found = true;
                }
        }
}

if ( !stickyID0Found && stickyID1Found )
{
        m_StickySkeletonIds[0] = m_StickySkeletonIds[1];
        m_StickySkeletonIds[1] = 0;
}
else if ( !stickyID0Found )
{
        m_StickySkeletonIds[0] = 0;
}
else if ( !stickyID1Found )
{
```

```
        m_StickySkeletonIds[1] = 0;
}

// Calculate nearest and sticky skeletons
for ( int i = 0 ; i < NUI_SKELETON_COUNT; i++ )
{
        NUI_SKELETON_TRACKING_STATE trackingState =
skel.SkeletonData[i].eTrackingState;

        if ( trackingState == NUI_SKELETON_TRACKED || trackingState
== NUI_SKELETON_POSITION_ONLY )
        {
                // Save SkeletonIds for sticky mode if there's none
already saved
                if ( 0 == m_StickySkeletonIds[0] &&
m_StickySkeletonIds[1] != skel.SkeletonData[i].dwTrackingID )
                {
                        m_StickySkeletonIds[0] =
skel.SkeletonData[i].dwTrackingID;
                }
                else if ( 0 == m_StickySkeletonIds[1] &&
m_StickySkeletonIds[0] != skel.SkeletonData[i].dwTrackingID )
                {
                        m_StickySkeletonIds[1] =
skel.SkeletonData[i].dwTrackingID;
                }

                LONG x, y;
                USHORT depth;

                // calculate the skeleton's position on the screen
                NuiTransformSkeletonToDepthImage(
skel.SkeletonData[i].Position, &x, &y, &depth );

                if ( depth < nearestDepths[0] )
                {
                        nearestDepths[1] = nearestDepths[0];
                        nearestIDs[1] = nearestIDs[0];

                        nearestDepths[0] = depth;
                        nearestIDs[0] =
skel.SkeletonData[i].dwTrackingID;
                }
                else if ( depth < nearestDepths[1] )
                {
                        nearestDepths[1] = depth;
                        nearestIDs[1] =
skel.SkeletonData[i].dwTrackingID;
                }
        }
}

m_pNuiSensor->NuiSkeletonSetTrackedSkeletons(nearestIDs);
```

void KinectMediator::checkActiveHand( )

```
//if skeletons are not tracking then no hand is active
if( !m_FoundSkeleton )
{
        m_ActiveHand = ActiveHand::AH_NOT_ACTIVE;
        return;
}

double leftHandDiff  = mpHandLeft.y - mpElbowLeft.y;
double rightHandDiff = mpHandRight.y - mpElbowRight.y;

//if both hand is raised (upper than elbow)
if( ( leftHandDiff > -3 ) && ( rightHandDiff > -3 ) )
{
        m_ActiveHand = ActiveHand::AH_BOTH;
}
//if only left hand is raised (upper than elbow)
else if ( leftHandDiff > -3 )
{
        m_ActiveHand = ActiveHand::AH_LEFT;
}
//if only right hand is raised (upper than elbow)
else if (  rightHandDiff > -3 )
{
        m_ActiveHand = ActiveHand::AH_RIGHT;
}
//if no hand is raised
else
{
        m_ActiveHand = ActiveHand::AH_NOT_ACTIVE;
}
```

void KinectMediator::determineIsHandMoving( )

```
float dX1,dY1,dZ1,dX2,dY2,dZ2;
//threshold of 1.0 is acceptable smaller is sensitive, greater is
inacceptable
const float threshold = 1.0;

dX1 = abs ( mpHandLeft.x - mpPrevHandLeft.x );
dY1 = abs ( mpHandLeft.y - mpPrevHandLeft.y );
dZ1 = abs ( mpHandLeft.depth - mpPrevHandLeft.depth );

dX2 = abs ( mpHandRight.x - mpPrevHandRight.x );
dY2 = abs ( mpHandRight.y - mpPrevHandRight.y );
dZ2 = abs ( mpHandRight.depth - mpPrevHandRight.depth );

//check for left hand
((dX1 + dY1 + dZ1) < threshold) ?     mIsHandLeftFixed = true :
```

```
mIsHandLeftFixed = false;
//check for right hand
((dX2 + dY2 + dZ2) < threshold) ?     mIsHandRightFixed = true :
mIsHandRightFixed = false;
```

void KinectMediator::ProcessColor()

```
HRESULT hr;
NUI_IMAGE_FRAME imageFrame;

// Attempt to get the color frame
hr = m_pNuiSensor->NuiImageStreamGetNextFrame(m_pColorStreamHandle,
0, &imageFrame);
if (FAILED(hr))
{
        return;
}

INuiFrameTexture * pTexture = imageFrame.pFrameTexture;
NUI_LOCKED_RECT LockedRect;

// Lock the frame data so the Kinect knows not to modify it while
we're reading it
pTexture->LockRect(0, &LockedRect, NULL, 0);

// Make sure we've received valid data
if (LockedRect.Pitch != 0)
{
        // Draw the data with Direct2D
        m_ColorRGB = static_cast<BYTE *>(LockedRect.pBits);
        emit signalColorImageUpdated( m_ColorRGB, cCWxCHxBPP);

}

// We're done with the texture so unlock it
pTexture->UnlockRect(0);

// Release the frame
m_pNuiSensor->NuiImageStreamReleaseFrame(m_pColorStreamHandle,
&imageFrame);
```

void KinectMediator::ProcessDepth()

```
HRESULT hr;
NUI_IMAGE_FRAME imageFrame;
```

```
// Attempt to get the depth frame
hr = m_pNuiSensor->NuiImageStreamGetNextFrame(m_pDepthStreamHandle,
0, &imageFrame);
if (FAILED(hr))
{
        return;
}

BOOL nearMode = FALSE;
INuiFrameTexture* pTexture;

// Get the depth image pixel texture
hr = m_pNuiSensor->NuiImageFrameGetDepthImagePixelFrameTexture(
                m_pDepthStreamHandle, &imageFrame, &nearMode,
&pTexture);
if (FAILED(hr))
{
        goto ReleaseFrame;
}

NUI_LOCKED_RECT LockedRect;

// Lock the frame data so the Kinect knows not to modify it while
we're reading it
pTexture->LockRect(0, &LockedRect, NULL, 0);

// Make sure we've received valid data
if (LockedRect.Pitch != 0)
{
        hr = m_nuiIStream-
>ProcessDepth(LockedRect.size,PBYTE(LockedRect.pBits),imageFrame.liT
imeStamp);
        if( FAILED( hr ) )
        {
                writeMessage("Process Depth failed");
        }

        // Get the min and max reliable depth for the current frame
        int minDepth =  NUI_IMAGE_DEPTH_MINIMUM >>
NUI_IMAGE_PLAYER_INDEX_SHIFT;
        int maxDepth = NUI_IMAGE_DEPTH_MAXIMUM >>
NUI_IMAGE_PLAYER_INDEX_SHIFT;

        BYTE * rgbrun = m_depthRGBX;
        const NUI_DEPTH_IMAGE_PIXEL * pBufferRun =
reinterpret_cast<const NUI_DEPTH_IMAGE_PIXEL *>(LockedRect.pBits);

        // end pixel is start + width*height - 1
        const NUI_DEPTH_IMAGE_PIXEL * pBufferEnd = pBufferRun +
(cDepthWidth * cDepthHeight);

        while ( pBufferRun < pBufferEnd )
        {
                // discard the portion of the depth that contains
only the player index
```

```
                USHORT depth = pBufferRun->depth;

                if(depth < 1.5)
                {
                        int x_debug = 0;
                }

                // To convert to a byte, we're discarding the most-
significant
                // rather than least-significant bits.
                // We're preserving detail, although the intensity
will "wrap."
                // Values outside the reliable depth range are mapped
to 0 (black).

                // Note: Using conditionals in this loop could
degrade performance.
                // Consider using a lookup table instead when writing
production code.
                BYTE intensity = static_cast<BYTE>(depth >= minDepth
&& depth <= maxDepth ? depth % 256 : 0);

                // Write out blue byte
                *(rgbrun++) = intensity;

                // Write out green byte
                *(rgbrun++) = intensity;

                // Write out red byte
                *(rgbrun++) = intensity;

                // We're outputting BGR, the last byte in the 32 bits
is unused so skip it
                // If we were outputting BGRA, we would write alpha
here.
                ++rgbrun;

                // Increment our index into the Kinect's depth buffer
                ++pBufferRun;
        }

        emit signalDepthImageUpdated( m_depthRGBX, cDWxDHxBPP);

}

// We're done with the texture so unlock it
pTexture->UnlockRect(0);

pTexture->Release();

ReleaseFrame:
// Release the frame
m_pNuiSensor->NuiImageStreamReleaseFrame(m_pDepthStreamHandle,
&imageFrame);
```

void KinectMediator::ProcessInteraction()

```
NUI_INTERACTION_FRAME Interaction_Frame;
m_nuiIStream->GetNextFrame( 0,&Interaction_Frame );
int trackingID = 0;
int event=0;
int indexTrackId = 0;

for(int i=0; i < NUI_SKELETON_COUNT; i++)
{
        trackingID =
Interaction_Frame.UserInfos[i].SkeletonTrackingId;
        if(trackingID > 1)
        {
                indexTrackId = i;
                break;
        }
}

for(int i=0; i < NUI_USER_HANDPOINTER_COUNT; i++)
{
        //represents the hand
        NUI_HANDPOINTER_INFO hand =
Interaction_Frame.UserInfos[indexTrackId].HandPointerInfos[i];

        //which hand? left or right -> RIGHT
        if ( NUI_HAND_TYPE::NUI_HAND_TYPE_RIGHT == hand.HandType )
        {
                //store the type of the hand. REFACTOR?
                mGripHandRight.type =
NUI_HAND_TYPE::NUI_HAND_TYPE_RIGHT;

                //store the gripping state
                if ( NUI_HAND_EVENT_TYPE::NUI_HAND_EVENT_TYPE_GRIP
== hand.HandEventType /*&&     m_ActiveHand ==
ActiveHand::AH_RIGHT*/)
                {
                        mGripHandRight.isGripActive = true;
                        //writeMessage("Right Hand Grip: GRIP");
                }
                if (
NUI_HAND_EVENT_TYPE::NUI_HAND_EVENT_TYPE_GRIPRELEASE ==
hand.HandEventType /*|| m_ActiveHand != ActiveHand::AH_RIGHT*/)
                {
                        //writeMessage("Right Hand Grip: RELEASE");
                        mGripHandRight.isGripActive = false;
                }
        }
        //which hand? left or right -> LEFT
```

```
        if ( NUI_HAND_TYPE::NUI_HAND_TYPE_LEFT == hand.HandType )
        {
                //store the type of the hand. REFACTOR?
                mGripHandLeft.type =
NUI_HAND_TYPE::NUI_HAND_TYPE_LEFT;

                //store the gripping state
                if ( NUI_HAND_EVENT_TYPE::NUI_HAND_EVENT_TYPE_GRIP
== hand.HandEventType /*&& m_ActiveHand == ActiveHand::AH_LEFT*/ )
                {
                        mGripHandLeft.isGripActive = true;
                        //writeMessage("Left Hand Grip: GRIP");
                }
                if (
NUI_HAND_EVENT_TYPE::NUI_HAND_EVENT_TYPE_GRIPRELEASE ==
hand.HandEventType /*|| m_ActiveHand != ActiveHand::AH_LEFT*/)
                {
                        mGripHandLeft.isGripActive = false;
                        //writeMessage("Left Hand Grip: RELEASE");
                }
        }

}
```

SpeechRecognizer-D2D.grxml

```
<grammar version="1.0" xml:lang="en-US" root="rootRule" tag-
format="semantics/1.0-literals"
xmlns="http://www.w3.org/2001/06/grammar">
  <rule id="rootRule">
    <one-of>
                <item>
                        <tag>HELLO</tag>
                        <one-of>
                                <item> hello </item>
                        </one-of>
                </item>
                <item>
                        <tag>ZOOM</tag>
                        <one-of>
                        <item> zoom </item>
                        </one-of>
                </item>
                <item>
                        <tag>START</tag>
                        <one-of>
                                <item> start </item>
                        </one-of>
                </item>
                <item>
```

```
                <tag>STOP</tag>
                <one-of>
                        <item> stop </item>
                        <item> stub </item>
                </one-of>
        </item>
        <item>
                <tag>SCALE</tag>
                <one-of>
                        <item> scale </item>
                </one-of>
        </item>
        <item>
                <tag>ROTATE</tag>
                <one-of>
                        <item> rotate </item>
                </one-of>
        </item>
        <item>
                <tag>PAN</tag>
                <one-of>
                        <item> pan </item>
                        <item> pen </item>
                </one-of>
        </item>
        <item>
                <tag>RESET</tag>
                <one-of>
                        <item> reset </item>
                </one-of>
        </item>
        <item>
                <tag>GRAB</tag>
                <one-of>
                        <item> grab </item>
                </one-of>
        </item>
        <item>
                <tag>REVERT</tag>
                <one-of>
                        <item> revert </item>
                </one-of>
        </item>
        <item>
                <tag>RUBBER_BAND</tag>
                <one-of>
                        <item> rubber band </item>
                        <item> rubber bandth </item>
                        <item> rubber bandth </item>
                        <item> robber bandth </item>
                </one-of>
        </item>
        <item>
                <tag>OK</tag>
                <one-of>
                        <item> okay </item>
```

```
                                <item> oh key </item>
                                <item> oh kay </item>
                        </one-of>
                </item>
                <item>
                        <tag>CALIBRATE</tag>
                        <one-of>
                                <item> calibrate </item>
                                <item> kelly braight</item>
                                <item> cally braight</item>
                                <item> kally braight</item>
                                <item> kelly braid</item>
                                <item> cally braid</item>
                        </one-of>
                </item>
        </one-of>
   </rule>
</grammar>
```

DataStructures.h

```
////////////////////////////////
//              VARIABLES
////////////////////////////////
const int g_ScreenWidth = 320;
const int g_ScreenHeight = 240;
const double FIELDNOTSET = -45678;

////////////////////////////////
//              ENUMERATIONS
////////////////////////////////

enum KinectSpeechAction
{
    KSA_ROTATE,
    KSA_ZOOM,
    KSA_PAN,
    KSA_RESET,
    KSA_RUBBER_BAND,
    KSA_OK,
    KSA_GRAB,
    KSA_CALIBRATE,
    KSA_RELEASE,
    KSA_STOP,
    KSA_START,
    KSA_CLOSE,
    KSA_NONE
};

enum DeviceControlState
```

```
{
    CS_NONE = 0,
    CS_ZOOM = 1,
    CS_PAN = 2,
    CS_ROTATE = 3
};

namespace DeviceMediator
{

enum ActiveHand
{
    AH_NOT_ACTIVE = 0,
    AH_LEFT = 1,
    AH_RIGHT = 2,
    AH_BOTH = 3
};


////////////////////////////////
//              STRUCTURES
////////////////////////////////

/*Defines the Kinect point

 x,y:  the coordinates for frame. Left-Bottom side (0,0) and
 Right-Top side is (windowWidth,windowHeight).

 depth: range of the point from the Sensor

 valid: point is currently tracking or not

*/
struct KinectPoint
{
    KinectPoint()
    {
        x              = FIELDNOTSET;
        y              = FIELDNOTSET;
        depth   = FIELDNOTSET;
        valid   = false;
    }

    void ResetPoint()
    {
        x              = FIELDNOTSET;
        y              = FIELDNOTSET;
        depth   = FIELDNOTSET;
        valid   = false;
    }

    const KinectPoint& operator= ( const KinectPoint& pValue )
    {
        x = pValue.x;
        y = pValue.y;
        depth = pValue.depth;
```

```
        valid = pValue.valid;

        if ( &pValue == this )
        {
            return *this;
        }
    }

    double x;
    double y;
    double depth;
    bool valid;
};

struct GripHand
{
    GripHand()
    {
        type = NUI_HAND_TYPE_NONE;
        isGripActive = false;
    }

    NUI_HAND_TYPE type;
    bool isGripActive;
};


static void normalizeValue(double& pVal, double pNormalizeTo,
double pTol)
{
    if (pVal > pTol || pVal < -1*pTol)
    {
        pVal = pNormalizeTo;
    }
}

} //namespace DeviceMediator
```

DWORD WINAPI KinectAudioStream::CaptureThread()

```
HANDLE mmHandle = NULL;
DWORD mmTaskIndex = 0;
HRESULT hr = S_OK;
bool bContinue = true;
BYTE *pbOutputBuffer = NULL;
CStaticMediaBuffer outputBuffer;
DMO_OUTPUT_DATA_BUFFER OutputBufferStruct = {0};
OutputBufferStruct.pBuffer = &outputBuffer;
DWORD dwStatus = 0;
ULONG cbProduced = 0;
```

```
// Set high priority to avoid getting preempted while capturing
sound
mmHandle = AvSetMmThreadCharacteristics(L"Audio", &mmTaskIndex);

while (bContinue)
{
        if (WaitForSingleObject(m_hStopEvent, 0) == WAIT_OBJECT_0)
        {
                bContinue = false;
                continue;
        }

        do
        {
                outputBuffer.Init(0);
                OutputBufferStruct.dwStatus = 0;
                hr = m_pKinectDmo->ProcessOutput(0, 1,
&OutputBufferStruct, &dwStatus);
                if (FAILED(hr))
                {
                        bContinue = false;
                        break;
                }

                if (hr == S_FALSE)
                {
                        cbProduced = 0;
                }
                else
                {

        outputBuffer.GetBufferAndLength(&pbOutputBuffer,
&cbProduced);
                }

                // Queue audio data to be read by IStream client
                if (cbProduced > 0)
                {
                        QueueCapturedData(pbOutputBuffer,
cbProduced);
                }
        }
        while (OutputBufferStruct.dwStatus &
DMO_OUTPUT_DATA_BUFFERF_INCOMPLETE);

        Sleep(10); //sleep 10ms
}

SetEvent(m_hDataReady);
AvRevertMmThreadCharacteristics(mmHandle);

if (FAILED(hr))
{
        return 0;
}
```

```
return 1;
```

void SpeechRecognizer::ProcessSpeech()

```
const float ConfidenceThreshold = 0.3f;

SPEVENT curEvent;
ULONG fetched = 0;
HRESULT hr = S_OK;

m_pSpeechContext->GetEvents(1, &curEvent, &fetched);

while (fetched > 0)
{
        switch (curEvent.eEventId)
        {
        case SPEI_RECOGNITION:
                if (SPET_LPARAM_IS_OBJECT == curEvent.elParamType)
                {
                        // this is an ISpRecoResult
                        ISpRecoResult* result =
reinterpret_cast<ISpRecoResult*>(curEvent.lParam);
                        SPPHRASE* pPhrase = NULL;

                        hr = result->GetPhrase(&pPhrase);
                        if (SUCCEEDED(hr))
                        {
                                if ((pPhrase->pProperties != NULL) &&
(pPhrase->pProperties->pFirstChild != NULL))
                                {
                                        const SPPHRASEPROPERTY*
pSemanticTag = pPhrase->pProperties->pFirstChild;
                                        if (pSemanticTag-
>SREngineConfidence > ConfidenceThreshold)
                                        {
                                                KinectSpeechAction
tmpSpeechAction;
                                                std::wcout <<
pSemanticTag->pszValue << std::endl;
                                                tmpSpeechAction =
MapSpeechTagToAction(pSemanticTag->pszValue);
                                                //accept only
available commands
                                                if ( tmpSpeechAction
!= KinectSpeechAction::KSA_NONE &&

        (tmpSpeechAction == KinectSpeechAction::KSA_PAN ||

tmpSpeechAction == KinectSpeechAction::KSA_ZOOM ||
```

```
tmpSpeechAction == KinectSpeechAction::KSA_ROTATE ||

tmpSpeechAction == KinectSpeechAction::KSA_RESET ||

tmpSpeechAction == KinectSpeechAction::KSA_START ||

tmpSpeechAction == KinectSpeechAction::KSA_CLOSE ||

tmpSpeechAction == KinectSpeechAction::KSA_STOP))
                                                    {
                                                            speechAction =
tmpSpeechAction;
                                                    }

                                            }
                                    }
                                    ::CoTaskMemFree(pPhrase);
                            }
                    }
                    break;
            }

            m_pSpeechContext->GetEvents(1, &curEvent, &fetched);
}

return;
```

KinectSpeechAction SpeechRecognizer::MapSpeechTagToAction(LPCWSTR pPszSpeechTag)

```
struct SpeechTagToAction
{
        LPCWSTR pszSpeechTag;
        KinectSpeechAction action;
};
const SpeechTagToAction Map[] =
{
        {L"ROTATE", KSA_ROTATE},
        {L"ZOOM", KSA_ZOOM},
        {L"SCALE", KSA_ZOOM},
        {L"PAN", KSA_PAN},
        {L"GRAB", KSA_PAN},
        {L"RESET", KSA_RESET},
        {L"RUBBER_BAND", KSA_RUBBER_BAND},
        {L"OK", KSA_OK},
        {L"GRAB", KSA_GRAB},
        {L"CALIBRATE", KSA_CALIBRATE},
        {L"STOP", KSA_STOP},
        {L"START", KSA_START},
```

```
        {L"CLOSE", KSA_CLOSE},
        {L"RELEASE", KSA_RELEASE}

};

//store the pre commanded action
if (speechAction != KinectSpeechAction::KSA_NONE)
{
        preSpeechAction = speechAction;
}

KinectSpeechAction action = KSA_NONE;

for (int i = 0; i < _countof(Map); ++i)
{
        if (0 == wcscmp(Map[i].pszSpeechTag, pPszSpeechTag))
        {
                action = Map[i].action;
                break;
        }
}

return action;
```

DWORD WINAPI KinectVolumeInteractor::GestureEventThread( )

```
DWORD dwWaitResult;
bool continueProcessing = true;
while ( continueProcessing )
{
        dwWaitResult = WaitForSingleObject(
                                        mKm->m_hMutex,    //
handle to mutex
                                        INFINITE);  // no time-out
interval

        if(dwWaitResult == WAIT_OBJECT_0)
        {
                //for user testing
                if( mKm->GetUserTaskNo() > 0 )
                {
                        switch(mKm->GetUserTaskNo())
                        {
                        case 1:
                                PostThreadMessage( mainThreadId,
INVOKE_fare_g1_Event, 0, 0 );
                                break;
                        case 2:
                                PostThreadMessage( mainThreadId,
INVOKE_fare_g2_Event, 0, 0 );
```

```
                                break;
                        case 3:
                                PostThreadMessage( mainThreadId,
INVOKE_fare_g3_Event, 0, 0 );
                                break;
                        case 4:
                                PostThreadMessage( mainThreadId,
INVOKE_fare_g4_Event, 0, 0 );
                                break;
                        case 5:
                                PostThreadMessage( mainThreadId,
INVOKE_fare_g5_Event, 0, 0 );
                                break;
                        case 6:
                                PostThreadMessage( mainThreadId,
INVOKE_kinect_g1_Event, 0, 0 );
                                break;
                        case 7:
                                PostThreadMessage( mainThreadId,
INVOKE_kinect_g2_Event, 0, 0 );
                                break;
                        case 8:
                                PostThreadMessage( mainThreadId,
INVOKE_kinect_g3_Event, 0, 0 );
                                break;
                        case 9:
                                PostThreadMessage( mainThreadId,
INVOKE_kinect_g4_Event, 0, 0 );
                                break;
                        case 10:
                                PostThreadMessage( mainThreadId,
INVOKE_kinect_g5_Event, 0, 0 );
                                break;

                        }

                        //return to initial value
                        mKm->SetUserTaskNo(-1);

                }

                checkSpeechCommands();

                // PAN
                //either left or right hand is activated and gesture
commanding activated
                if ( mKm->m_ActiveHand ==
DeviceMediator::ActiveHand::AH_LEFT && mKm-
>mGripHandLeft.isGripActive && !mKm->mGripHandRight.isGripActive
/*mKm->m_GesComActiveL*/ ||
                                mKm->m_ActiveHand ==
DeviceMediator::ActiveHand::AH_RIGHT && mKm-
>mGripHandRight.isGripActive && ! mKm-
>mGripHandLeft.isGripActive/*mKm->m_GesComActiveR*/ )
                {
                        //TODO: activate panning
```

```
                        setCurrentGestureState(
DeviceControlState::CS_PAN );
                        PostThreadMessage( mainThreadId,
INVOKE_PAN_EVENT, 0, 0 );
                        Sleep(50);
                }
                else if ( getCurrentGestureState() ==
DeviceControlState::CS_PAN )
                {
                        setCurrentGestureState(
DeviceControlState::CS_NONE );
                        PostThreadMessage( mainThreadId,
INVOKE_PAN_EVENT, 0, 0 );
                }
                //-----------------------------------------

                // ZOOM
                if ( mKm->m_ActiveHand ==
DeviceMediator::ActiveHand::AH_BOTH &&
                                mKm->mGripHandLeft.isGripActive &&
                                mKm->mGripHandRight.isGripActive &&
                                (!mKm->mIsHandLeftFixed && !mKm-
>mIsHandRightFixed) )
                {
                        //TODO: activate image zoom (both hand
raises and one of the hand is active)
                        setCurrentGestureState(
DeviceControlState::CS_ZOOM );
                        PostThreadMessage( mainThreadId,
INVOKE_ZOOM_EVENT, 0, 0 );
                        Sleep(50);
                }
                else if ( getCurrentGestureState() ==
DeviceControlState::CS_ZOOM )
                {
                        mPrevDistHandX = mDistHandX;
                        mPreDistDiff = mDistDiff = 0.0;
                        setCurrentGestureState(
DeviceControlState::CS_NONE );
                        PostThreadMessage( mainThreadId,
INVOKE_ZOOM_EVENT, 0, 0 );
                }
                //-----------------------------------------

                // ROTATE
                if ( mKm->m_ActiveHand ==
DeviceMediator::ActiveHand::AH_BOTH &&
                                ( (mKm->mGripHandLeft.isGripActive &&
!mKm->mGripHandRight.isGripActive && mKm->mIsHandRightFixed ) ||
                                  (mKm->mGripHandRight.isGripActive
&& !mKm->mGripHandLeft.isGripActive && mKm->mIsHandLeftFixed) )
                   )
                {
                        if(!mHasGotHandPoint )
                        {
                                mHasGotHandPoint = true;
```

```
                                //LEft hand is fixed, rotate hand
will be right hand
                                if(mKm->mIsHandLeftFixed && mKm-
>mGripHandRight.isGripActive )
                                {
                                        mRotateHandPoint.x = mKm-
>mpHandRight.x;
                                        mRotateHandPoint.y = mKm-
>mpHandRight.y;
                                        mRotateHandPoint.depth = mKm-
>mpHandRight.depth;
                                }
                                //Right hand is fixed, rotate hand
will be left hand
                                else if(mKm->mIsHandRightFixed &&
mKm->mGripHandLeft.isGripActive)
                                {
                                        mRotateHandPoint.x = mKm-
>mpHandLeft.x;
                                        mRotateHandPoint.y = mKm-
>mpHandLeft.y;
                                        mRotateHandPoint.depth = mKm-
>mpHandLeft.depth;
                                }
                        }
                        setCurrentGestureState(
DeviceControlState::CS_ROTATE);
                        PostThreadMessage( mainThreadId,
INVOKE_ROTATE_EVENT, 0, 0 );
                        Sleep(50);
                }
                else if ( getCurrentGestureState() ==
DeviceControlState::CS_ROTATE )
                {
                        mHasGotHandPoint = false;
                        setCurrentGestureState(
DeviceControlState::CS_NONE );
                        PostThreadMessage( mainThreadId,
INVOKE_ROTATE_EVENT, 0, 0 );
                }
                //-----------------------------------------------

                // Release ownership of the mutex object
                if (!ReleaseMutex(mKm->m_hMutex))
                {
                        writeMessage( "Mutex CAN NOT be released in
updateKinectST()" );
                }
        }
        // The thread got ownership of an abandoned mutex
        // The database is in an indeterminate state
        else if (dwWaitResult == WAIT_ABANDONED)
                return 0;
}
```

```
return 0;
```

void KinectVolumeInteractor::ZoomCallback(vtkObject* caller, long unsigned int eventId, void* clientData, void* callData)

```
int winW = 640;
int winH = 480;
bool triggerCopyDistHand = true;

// We can get the calling object like this:
KinectVolumeInteractor *iStyle =
static_cast<KinectVolumeInteractor*>(caller);
vtkRenderWindowInteractor* tmpInteractor = iStyle->GetInteractor();

iStyle->debug_wentIntoZoomCallback = true;

tmpInteractor->GetRenderWindow()->MakeCurrent();
tmpInteractor->FindPokedRenderer(winW/2, winH/2);

if ( iStyle->getCurrentGestureState() != DeviceControlState::CS_ZOOM
)
{
        tmpInteractor->SetEventPosition(winW/2, winH/2);
        tmpInteractor-
>InvokeEvent(vtkCommand::RightButtonReleaseEvent, 0);
        tmpInteractor-
>InvokeEvent(vtkCommand::MiddleButtonReleaseEvent, 0);
        tmpInteractor-
>InvokeEvent(vtkCommand::LeftButtonReleaseEvent, 0);
        iStyle->isZoomActive = false;
        return;
}

if ( !iStyle->isZoomActive )
{
        triggerCopyDistHand = false;
        tmpInteractor-
>InvokeEvent(vtkCommand::RightButtonPressEvent, 0);
        iStyle->isZoomActive = true;
}

int cursorX = winW/2;
int cursorY = winH/2;
tmpInteractor->SetEventPosition(winW/2, winH/2);

if ( triggerCopyDistHand )
{
        iStyle->mPrevDistHandX = iStyle->mDistHandX;
}
```

```
iStyle->mDistHandX = abs(iStyle->getKinectMediator()->mpHandLeft.x -
iStyle->getKinectMediator()->mpHandRight.x);
DeviceMediator::normalizeValue(iStyle->mPrevDistHandX,iStyle-
>mDistHandX,999999);

if ( !triggerCopyDistHand )
{
        iStyle->mPrevDistHandX = iStyle->mDistHandX;
}

if ( triggerCopyDistHand )
{
        iStyle->mPreDistDiff = iStyle->mDistDiff;
}
iStyle->mDistDiff = iStyle->mDistHandX - iStyle->mPrevDistHandX;
DeviceMediator::normalizeValue(iStyle->mPreDistDiff,iStyle-
>mDistDiff,35);

if ( !triggerCopyDistHand )
{
        iStyle->mPreDistDiff = iStyle->mDistDiff;
}

//LOGGING
#if TEXTLOG
TxtLogger::getInstance()->logCoords << "mPrevDistHandX: " << iStyle-
>mPrevDistHandX << endl;
TxtLogger::getInstance()->logCoords << "mDistHandX: " << iStyle-
>mDistHandX << endl;
TxtLogger::getInstance()->logCoords << "----------------------" <<
endl;
TxtLogger::getInstance()->logCoords << "mPreDistDiff: " << iStyle-
>mPreDistDiff << endl;
TxtLogger::getInstance()->logCoords << "mDistDiff: " << iStyle-
>mDistDiff << endl;
TxtLogger::getInstance()->logCoords << "----------------------" <<
endl;
TxtLogger::getInstance()->logCoords << "----------------------" <<
endl;
#endif

//DEBUG
if(iStyle->mDistDiff != 0.0 || (!cursorX && !iStyle->mDistDiff) )
{
        int i = 0;
}
tmpInteractor->SetEventPosition(cursorX, (iStyle-
>getKinectMediator()->mWindowHeight/2 + iStyle->mDistDiff*1.4) );
//4 katsayısı sensitivity sini ayarlıyor
tmpInteractor->SetEventPosition(cursorX, (winH/2 + iStyle-
>mDistDiff*4.0) );

tmpInteractor->InvokeEvent(vtkCommand::MouseMoveEvent, 0);
```

void KinectVolumeInteractor::PanCallback(vtkObject* caller, long unsigned int eventId, void* clientData, void* callData)

```
// We can get the calling object like this:
KinectVolumeInteractor *iStyle =
static_cast<KinectVolumeInteractor*>(caller);
vtkRenderWindowInteractor* rwi = iStyle->GetInteractor();
int curX, curY, preX, preY;
int winW = 640;
int winH = 480;

//for debugging
if(iStyle->debug_wentIntoZoomCallback)
{
        int i = 0;
}



int cursorX = winW/2, cursorY = winH/2;

rwi->GetRenderWindow()->MakeCurrent();
rwi->FindPokedRenderer(cursorX, cursorY);

if( iStyle->getCurrentGestureState() != DeviceControlState::CS_PAN
)
{
        rwi->SetEventPosition(winW/2, winH/2);
        rwi->InvokeEvent(vtkCommand::MiddleButtonReleaseEvent, 0);
        rwi->InvokeEvent(vtkCommand::RightButtonReleaseEvent, 0);
        rwi->InvokeEvent(vtkCommand::LeftButtonReleaseEvent, 0);
        iStyle->isPanActive = false;
        return;
}
else if ( iStyle->getKinectMediator()->m_ActiveHand ==
DeviceMediator::ActiveHand::AH_LEFT /*&& iStyle-
>getKinectMediator()->m_GesComActiveL*/ )
{
        curX = iStyle->getKinectMediator()->mpHandLeft.x;
        curY = iStyle->getKinectMediator()->mpHandLeft.y;
        preX = iStyle->getKinectMediator()->mpPrevHandLeft.x;
        preY = iStyle->getKinectMediator()->mpPrevHandLeft.y;
}
else if ( iStyle->getKinectMediator()->m_ActiveHand ==
DeviceMediator::ActiveHand::AH_RIGHT /*&& iStyle-
>getKinectMediator()->m_GesComActiveR*/ )
{
        curX = iStyle->getKinectMediator()->mpHandRight.x;
        curY = iStyle->getKinectMediator()->mpHandRight.y;
        preX = iStyle->getKinectMediator()->mpPrevHandRight.x;
        preY = iStyle->getKinectMediator()->mpPrevHandRight.y;
}
else
```

```
{
        return;
}

rwi->SetEventPosition(cursorX, cursorY);
if( !iStyle->isPanActive )
{
        rwi->InvokeEvent(vtkCommand::MiddleButtonPressEvent, 0);
        iStyle->isPanActive = true;
}

int tmpX = cursorX+5*(curX-preX);
int tmpY = cursorY+5*(curY-preY);
rwi->SetEventPosition(tmpX, tmpY);

// Get the vector of motion
double fp[3], focalPoint[3], pos[3], v[3], p1[4], p2[4];

vtkCamera *camera = iStyle->CurrentRenderer->GetActiveCamera();
camera->GetPosition( pos );
camera->GetFocalPoint( fp );

iStyle->ComputeWorldToDisplay(fp[0], fp[1], fp[2],
                                                        focalPoint);

iStyle->ComputeDisplayToWorld(rwi->GetEventPosition()[0],
                                                        rwi-
>GetEventPosition()[1],

focalPoint[2],
                                                        p1);

iStyle->ComputeDisplayToWorld(rwi->GetLastEventPosition()[0],
                                                        rwi-
>GetLastEventPosition()[1],

focalPoint[2],
                                                        p2);

for (int i=0; i<3; i++)
{
        v[i] = p2[i] - p1[i];
        pos[i] += v[i];
        fp[i] += v[i];
}

camera->SetPosition( pos );
camera->SetFocalPoint( fp );

if (rwi->GetLightFollowCamera())
{
        iStyle->CurrentRenderer-
>UpdateLightsGeometryToFollowCamera();
}

rwi->Render();
```

void KinectVolumeInteractor::RotateCallback(vtkObject* caller, long unsigned int eventId, void* clientData, void* callData)

```
int winW = 640;
int winH = 480;

KinectVolumeInteractor *iStyle =
static_cast<KinectVolumeInteractor*>(caller);
vtkRenderWindowInteractor* rwi = iStyle->GetInteractor();
//int curXL, curYL, curXR, curYR;
float worldCoordHandXL, worldCoordHandYL, worldCoordHandZL,
worldCoordHandXR, worldCoordHandYR, worldCoordHandZR;

rwi->GetRenderWindow()->MakeCurrent();
rwi->FindPokedRenderer(winW/2, winH/2);

std::stringstream ss;
string s;

if( iStyle->getCurrentGestureState() ==
DeviceControlState::CS_ROTATE  )
{
        worldCoordHandXL = iStyle->getKinectMediator()-
>mpHandLeftWorldCoord.x;
        worldCoordHandYL = iStyle->getKinectMediator()-
>mpHandLeftWorldCoord.y;
        worldCoordHandZL = iStyle->getKinectMediator()-
>mpHandLeftWorldCoord.z;

        worldCoordHandXR = iStyle->getKinectMediator()-
>mpHandRightWorldCoord.x;
        worldCoordHandYR = iStyle->getKinectMediator()-
>mpHandRightWorldCoord.y;
        worldCoordHandZR = iStyle->getKinectMediator()-
>mpHandRightWorldCoord.z;

        /*
        ss << "LeftX: " << worldCoordHandXL << " LeftY: " <<
worldCoordHandYL << "LeftZ: " << worldCoordHandZL << endl
           << " RightX: " << worldCoordHandXR << " RightY: " <<
worldCoordHandYR << "RightZ: " << worldCoordHandZR;
        s = ss.str();
        iStyle->writeMessage(s);
        */
}
else
{
        rwi->SetEventPosition(winW/2, winH/2);
        rwi->InvokeEvent(vtkCommand::LeftButtonReleaseEvent, 0);
        rwi->InvokeEvent(vtkCommand::MiddleButtonReleaseEvent, 0);
        rwi->InvokeEvent(vtkCommand::RightButtonReleaseEvent, 0);
        iStyle->isRotateActive = false;
        return;
```

```
}

//iStyle->writeMessage("Rotate Active");

rwi->SetEventPosition( winW/2, winH/2 );
if ( !iStyle->isRotateActive )
{
        rwi->InvokeEvent(vtkCommand::LeftButtonPressEvent, 0);
        iStyle->isRotateActive = true;
}

int dx,dy;
//6 katsayısı sensitivity sini ayarlıyor
if(iStyle->getKinectMediator()->mIsHandRightFixed)
{
        //6 katsayısı sensitivity sini ayarlıyor
        dx = -6 * (iStyle->mKm->mpHandLeft.x - iStyle->mKm-
>mpPrevHandLeft.x);
        dy = -6 * (iStyle->mKm->mpHandLeft.y - iStyle->mKm-
>mpPrevHandLeft.y);
}
else if(iStyle->getKinectMediator()->mIsHandLeftFixed)
{
        dx = -6 * (iStyle->mKm->mpHandRight.x - iStyle->mKm-
>mpPrevHandRight.x);
        dy = -6 * (iStyle->mKm->mpHandRight.y - iStyle->mKm-
>mpPrevHandRight.y);
}
else
{
        //yukarida da bu is yapiliyor
        rwi->SetEventPosition(winW/2, winH/2);
        rwi->InvokeEvent(vtkCommand::LeftButtonReleaseEvent, 0);
        rwi->InvokeEvent(vtkCommand::MiddleButtonReleaseEvent, 0);
        rwi->InvokeEvent(vtkCommand::RightButtonReleaseEvent, 0);
        iStyle->isRotateActive = false;
        return;
}

int *size = iStyle->CurrentRenderer->GetRenderWindow()->GetSize();

double a = dx / static_cast<double>( size[0]) * 180.0;
double e = dy / static_cast<double>( size[1]) * 180.0;

// Move the camera.
// Make sure that we don't hit the north pole singularity.

vtkCamera *camera = iStyle->CurrentRenderer->GetActiveCamera();
camera->Azimuth( a );

double dop[3], vup[3];

camera->GetDirectionOfProjection( dop );
vtkMath::Normalize( dop );
camera->GetViewUp( vup );
vtkMath::Normalize( vup );
```

```
double angle = vtkMath::DegreesFromRadians( acos(vtkMath::Dot( dop,
vup) ) );
if ( ( angle + e ) > 179.0 ||
                ( angle + e ) < 1.0 )
{
        e = 0.0;
}

camera->Elevation( e );

if ( iStyle->AutoAdjustCameraClippingRange )
{
        iStyle->CurrentRenderer->ResetCameraClippingRange();
}

rwi->Render();
```

void KinectVolumeInteractor::ResetCallback(vtkObject* caller, long unsigned int eventId, void* clientData, void* callData)

```
// We can get the calling object like this:
KinectVolumeInteractor *iStyle =
static_cast<KinectVolumeInteractor*>(caller);
vtkRenderWindowInteractor* rwi = iStyle->GetInteractor();

iStyle->writeMessage("Reset Activated");

//stopping any action
rwi->InvokeEvent(vtkCommand::LeftButtonReleaseEvent, 0);
rwi->InvokeEvent(vtkCommand::RightButtonReleaseEvent, 0);
rwi->InvokeEvent(vtkCommand::MiddleButtonReleaseEvent, 0);
iStyle->isPanActive = false;
iStyle->isRotateActive = false;
iStyle->isZoomActive = false;

iStyle->GetInteractor()->GetRenderWindow()->MakeCurrent();
iStyle->FindPokedRenderer(320, 240);

vtkCamera* aCamera = iStyle->GetCurrentRenderer()-
>GetActiveCamera();
aCamera->SetViewUp (0, 0, -1);
aCamera->SetPosition (0, 1, 0);
aCamera->SetFocalPoint (0, 0, 0);
aCamera->ComputeViewPlaneNormal();
aCamera->Azimuth(30.0);
aCamera->Elevation(30.0);

iStyle->GetCurrentRenderer()->ResetCamera();
```

```
iStyle->GetInteractor()->Render();
```

void KinectVolumeInteractor::resetView()

```
this->GetInteractor()->GetRenderWindow()->MakeCurrent();
this->FindPokedRenderer(320, 240);

vtkCamera* aCamera = this->GetCurrentRenderer()->GetActiveCamera();
aCamera->SetViewUp (0, 0, -1);
aCamera->SetPosition (0, 1, 0);
aCamera->SetFocalPoint (0, 0, 0);
aCamera->ComputeViewPlaneNormal();
aCamera->Azimuth(30.0);
aCamera->Elevation(30.0);

this->GetCurrentRenderer()->ResetCamera();
this->GetInteractor()->Render();
```

void VolumeDrawer::displayVolumeRendering(char* pFilePath)

```
// The following reader is used to read a series of 2D slices
(images)
// that compose the volume. The slice dimensions are set, and the
// pixel spacing. The data Endianness must also be specified. The
reader
// uses the FilePrefix in combination with the slice number to
construct
// filenames using the format FilePrefix.%d. (In this case the
FilePrefix
// is the root name of the file: quarter.)
vtkSmartPointer<vtkVolume16Reader> v16 =
        vtkSmartPointer<vtkVolume16Reader>::New();
v16->SetDataDimensions(64, 64);
v16->SetImageRange(1, 93);
v16->SetDataByteOrderToLittleEndian();
v16->SetFilePrefix(pFilePath);
v16->SetDataSpacing(3.2, 3.2, 1.5);

// The volume will be displayed by ray-cast alpha compositing.
// A ray-cast mapper is needed to do the ray-casting, and a
// compositing function is needed to do the compositing along the
ray.
vtkSmartPointer<vtkVolumeRayCastCompositeFunction> rayCastFunction
=
        vtkSmartPointer<vtkVolumeRayCastCompositeFunction>::New();
```

```
vtkSmartPointer<vtkVolumeRayCastMapper> volumeMapper =
        vtkSmartPointer<vtkVolumeRayCastMapper>::New();
volumeMapper->SetInput(v16->GetOutput());
volumeMapper->SetVolumeRayCastFunction(rayCastFunction);

// The color transfer function maps voxel intensities to colors.
// It is modality-specific, and often anatomy-specific as well.
// The goal is to one color for flesh (between 500 and 1000)
// and another color for bone (1150 and over).
vtkSmartPointer<vtkColorTransferFunction>volumeColor =
        vtkSmartPointer<vtkColorTransferFunction>::New();
volumeColor->AddRGBPoint(0,    0.0, 0.0, 0.0);
volumeColor->AddRGBPoint(500,  1.0, 0.5, 0.3);
volumeColor->AddRGBPoint(1000, 1.0, 0.5, 0.3);
volumeColor->AddRGBPoint(1150, 1.0, 1.0, 0.9);

// The opacity transfer function is used to control the opacity
// of different tissue types.
vtkSmartPointer<vtkPiecewiseFunction> volumeScalarOpacity =
        vtkSmartPointer<vtkPiecewiseFunction>::New();
volumeScalarOpacity->AddPoint(0,    0.00);
volumeScalarOpacity->AddPoint(500,  0.15);
volumeScalarOpacity->AddPoint(1000, 0.15);
volumeScalarOpacity->AddPoint(1150, 0.85);

// The gradient opacity function is used to decrease the opacity
// in the "flat" regions of the volume while maintaining the
opacity
// at the boundaries between tissue types.  The gradient is
measured
// as the amount by which the intensity changes over unit distance.
// For most medical data, the unit distance is 1mm.
vtkSmartPointer<vtkPiecewiseFunction> volumeGradientOpacity =
        vtkSmartPointer<vtkPiecewiseFunction>::New();
volumeGradientOpacity->AddPoint(0,   0.0);
volumeGradientOpacity->AddPoint(90,  0.5);
volumeGradientOpacity->AddPoint(100, 1.0);

// The VolumeProperty attaches the color and opacity functions to
the
// volume, and sets other volume properties.  The interpolation
should
// be set to linear to do a high-quality rendering.  The ShadeOn
option
// turns on directional lighting, which will usually enhance the
// appearance of the volume and make it look more "3D".  However,
// the quality of the shading depends on how accurately the
gradient
// of the volume can be calculated, and for noisy data the gradient
// estimation will be very poor.  The impact of the shading can be
// decreased by increasing the Ambient coefficient while decreasing
// the Diffuse and Specular coefficient.  To increase the impact
// of shading, decrease the Ambient and increase the Diffuse and
Specular.
vtkSmartPointer<vtkVolumeProperty> volumeProperty =
```

```
        vtkSmartPointer<vtkVolumeProperty>::New();
volumeProperty->SetColor(volumeColor);
volumeProperty->SetScalarOpacity(volumeScalarOpacity);
volumeProperty->SetGradientOpacity(volumeGradientOpacity);
volumeProperty->SetInterpolationTypeToLinear();
volumeProperty->ShadeOn();
volumeProperty->SetAmbient(0.4);
volumeProperty->SetDiffuse(0.6);
volumeProperty->SetSpecular(0.2);

// The vtkVolume is a vtkProp3D (like a vtkActor) and controls the
position
// and orientation of the volume in world coordinates.
vtkSmartPointer<vtkVolume> volume =
        vtkSmartPointer<vtkVolume>::New();
volume->SetMapper(volumeMapper);
volume->SetProperty(volumeProperty);

// Finally, add the volume to the renderer
mRenderer->AddViewProp(volume);

// Set up an initial view of the volume.  The focal point will be
the
// center of the volume, and the camera position will be 400mm to
the
// patient's left (which is our right).
mCamera = mRenderer->GetActiveCamera();
double *c = volume->GetCenter();
mCamera->SetFocalPoint(c[0], c[1], c[2]);
mCamera->SetPosition(c[0] + 400, c[1], c[2]);
mCamera->SetViewUp(0, 0, -1);

// Increase the size of the render window
mRenWin->SetSize(640, 480);

// draws into the render window, the interactor enables mouse- and
// keyboard-based interaction with the data within the render
window.
//
mRenWin->AddRenderer(mRenderer);

mRenWin->SetInteractor(mIren);

mIren->SetInteractorStyle( getInteractor() );

// Interact with the data.
mIren->Initialize();
```

void VolumeDrawer::displayIsosurfaceOnly(char* pFilePath)

```
// The following reader is used to read a series of 2D slices
(images)
// that compose the volume. The slice dimensions are set, and the
// pixel spacing. The data Endianness must also be specified. The
reader
// uses the FilePrefix in combination with the slice number to
construct
// filenames using the format FilePrefix.%d. (In this case the
FilePrefix
// is the root name of the file: quarter.)
mV16->SetDataDimensions (64,64);
mV16->SetImageRange (1,93);
mV16->SetDataByteOrderToLittleEndian();
//"C:\\Tez\\vtk\\VTKData5.10.1\\Data\\quarter\\quarter"
//mV16->SetFilePrefix (argv[1]);

//char* filePath =
"C:\\Tez\\vtk\\VTKData5.10.1\\Data\\quarter\\quarter";
//mV16->SetFilePrefix (filePath);

mV16->SetFilePrefix (pFilePath);
mV16->SetDataSpacing (3.2, 3.2, 1.5);

// An isosurface, or contour value of 500 is known to correspond to
the
// mSkin of the patient. Once generated, a vtkPolyDataNormals
filter is
// is used to create normals for smooth surface shading during
rendering.
mSkinExtractor->SetInputConnection(mV16->GetOutputPort());
mSkinExtractor->SetValue(0, 500);

mSkinNormals->SetInputConnection(mSkinExtractor->GetOutputPort());
mSkinNormals->SetFeatureAngle(60.0);

mSkinMapper->SetInputConnection(mSkinNormals->GetOutputPort());
mSkinMapper->ScalarVisibilityOff();

mSkin->SetMapper(mSkinMapper);

// An mOutline provides context around the data.
//
mOutlineData->SetInputConnection(mV16->GetOutputPort());

mMapOutline->SetInputConnection(mOutlineData->GetOutputPort());

mOutline->SetMapper(mMapOutline);
mOutline->GetProperty()->SetColor(0,0,0);

// It is convenient to create an initial view of the data. The
FocalPoint
// and Position form a vector direction. Later on (ResetCamera()
method)
// this vector is used to position the camera to look at the data
in
// this direction.
```

```
mCamera->SetViewUp (0, 0, -1);
mCamera->SetPosition (0, 1, 0);
mCamera->SetFocalPoint (0, 0, 0);
mCamera->ComputeViewPlaneNormal();
mCamera->Azimuth(30.0);
mCamera->Elevation(30.0);

// Actors are added to the renderer. An initial camera view is
created.
// The Dolly() method moves the camera towards the FocalPoint,
// thereby enlarging the image.
mRenderer->AddActor(mOutline);
mRenderer->AddActor(mSkin);
mRenderer->SetActiveCamera(mCamera);
mRenderer->ResetCamera();
mCamera->Dolly(1.3);

// Set a background color for the renderer and set the size of the
// render window (expressed in pixels).
mRenderer->SetBackground(.2, .3, .4);
mRenWin->SetSize(640, 480);

// Note that when camera movement occurs (as it does in the Dolly()
// method), the clipping planes often need adjusting. Clipping
planes
// consist of two planes: near and far along the view direction.
The
// near plane clips out objects in front of the plane; the far
plane
// clips out objects behind the plane. This way only what is drawn
// between the planes is actually rendered.

mRenderer->ResetCameraClippingRange ();

// draws into the render window, the interactor enables mouse- and
// keyboard-based interaction with the data within the render
window.
//
mRenWin->AddRenderer(mRenderer);

mRenWin->SetInteractor(mIren);

mIren->SetInteractorStyle( getInteractor() );
mIren->Initialize();
```

KİŞİSEL YAYINLAR VE ESERLER

Baştürk T., **Ergüner F.**, Genç H. M., Life Cycle Of Weapon Control Systems For Wire Guided Torpedoes, *Simulation Interoperability Workshop International European Multi Conference*, San Diego, USA, 23-27 March 2009.

Baştürk T., Genç H. M., **Ergüner F.**, Okutan C. C., Algoritmadan Uygulamaya Modern Torpido Atış Kontrol Yazılımı, *SAVTEK*, Ankara, Türkiye, 23-25 Haziran 2010.

ÖZGEÇMİŞ

1985 yılında Ankara'da doğdu. Lise öğrenimini 2003 yılında İzmit'te tamamladı. 2003 yılında Doğu Akdeniz Üniversitesi'nde başladığı bilgisayar mühendisliği lisans eğitimini Gebze Yüksek Teknoloji Enstitüsü'nde 2008 yılında tamamladı ve aynı yıl Tübitak Bilişim Teknolojileri Enstitüsü'nde araştırmacı olarak göreve başladı. Denizaltı operatörlerinin taktik eğitim ve notlandırılması ihtiyaçlarını karşılayan, HLA mimarisi üzerine kurulu yüksek ölçekli ve yüksek sadakatli bir dağıtık simülatör olan DATAS projesinde atış kontrol ve silah güdümü paketinde görev aldı. Halen Tübitak BİLGEM bünyesinde DDS sistemi üzerinde geliştirilen dağıtık bir simülatör olan Sonar Tasarım Aracı Projesi'nde görev almaktadır. 2009 yılında başladığı Kocaeli Üniversitesi Fen Bilimleri Bilgisayar Mühendisliği Anabilim Dalı'ndaki Yüksek Lisans eğitimine devam etmektedir.