ÇOKLU SERBEST HAREKET KABİLİYETLİ SİMÜLATÖR GELİŞTİRİLMESİ

2010
YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ

Ferhat ATASOY

ÇOKLU SERBEST HAREKET KABİLİYETLİ SİMÜLATÖR GELİŞTİRİLMESİ

Ferhat ATASOY

Karabük Üniversitesi

Fen Bilimleri Enstitüsü

Bilgisayar Mühendisliği Anabilim Dalında

Yüksek Lisans Tezi

Olarak Hazırlanmıştır

KARABÜK

Haziran 2010

Ferhat ATASOY tarafından hazırlanan "ÇOKLU SERBEST HAREKET KABİLİYETLİ SİMÜLATÖR GELİŞTİRİLMESİ" başlıklı bu tezin Yüksek Lisans Tezi olarak uygun olduğunu onaylarım.

Prof. Dr. Abdullah ÇAVUŞOĞLU .....
Tez Danışmanı, Bilgisayar Mühendisliği Anabilim Dalı

Bu çalışma, jürimiz tarafından oy birliği ile Bilgisayar Mühendisliği Anabilim Dalında Yüksek Lisans tezi olarak kabul edilmiştir. 02/ 07/ 2010

Ünvanı, Adı SOYADI (Kurumu)	İmzası
Başkan : Prof. Dr. Abdullah ÇAVUŞOĞLU (KBÜ)	.....
Üye : Doç. Dr. Fatih ÇELEBİ (AÜ)	.....
Üye : Yrd. Doç. Dr. Baha ŞEN (KBÜ)	.....

...../..../2010

KBÜ Fen Bilimleri Enstitüsü Yönetim Kurulu, bu tez ile, Yüksek Lisans derecesini onamıştır.

Doç. Dr. Süleyman GÜNDÜZ .....
Fen Bilimleri Enstitüsü Müdürü

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

Ferhat ATASOY

ÖZET

Yüksek Lisans Tezi

ÇOKLU SERBEST HAREKET KABİLİYETLİ SİMÜLATÖR GELİŞTİRİLMESİ

Ferhat ATASOY

Karabük Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı:
Prof. Dr. Abdullah ÇAVUŞOĞLU
Haziran 2010, 80 sayfa

Bilgisayarlarda gerçekleştirilen 3 boyutlu sanal gerçeklik platformları, gerçek fiziksel ortamların olası risklerinden sakınmanın önemli olduğu birçok uygulamada kullanılabilir. Simülasyonlar için birçok fizik ve grafik kütüphanesi yazılımı geliştirilmiştir. Stewart Platformu gibi pahalı profesyonel sistemlerde, 6 veya daha çok hareket serbestisi sağlayan mekanik sistemlerin simülasyonu gerçekleştirilir. Daha az karmaşık sistemler daha az sayıda hareket serbestisi sağlarlar. Bu çalışmada, daha az maliyetli bir simülasyon sistemi geliştirilmiştir. Bu sistemde açık kaynak kodlu grafik ve fizik kütüphaneleri, (PCI) I/O kartları 5 hareket serbestliği sağlayan mekanik bir platformun kontrolü için kullanılmıştır. Bu sanal gerçeklik uygulamasında seçenekli olarak tespit edilen yükseklik haritaları üzerinde değişik karakteristik özelliklere sahip hareketli farklı taşıtların pozisyonlarının simülasyonu sağlanmıştır. Ayrıca, aracın hareket ve davranışları elektronik olarak gerçek zamanlı çalışmak üzere tasarlanan mekanik sisteme aktarılmıştır. Simulator gerçeğe uygun

çıktılar ürettiği için, kullanıcılar aracın gerçek ortama uygun olduğunu görebilmekte ve hissedebilmektedirler.

Anahtar Sözcükler : Araç simülasyonu, Delta3D, open dynamics engine, openscenegraph, sanal gerçeklik

Bilim Kodu : 902.1.014

ABSTRACT

M.Sc. Thesis

DEVELOPING MULTI FREEDOM MOTION TALENTED SIMULATOR

Ferhat ATASOY

Karabük University
Graduate School of Natural and Applied Sciences
Department of Computer Engineering

Thesis Advisor:
Prof. Dr. Abdullah ÇAVUŞOĞLU
June 2010, 80 pages

3D Virtual Reality platforms developed on computers can be used in several applications where avoiding the possible risks of real physical environments is essential. A lot of physics and graphics software libraries have been developed for simulations. In expensive professional systems, such as the Stewart Platform, providing six or more degrees of freedoms are simulated. Less sophisticated systems offer fewer degrees of freedoms. In this study, an inexpensive system has been developed. In the system open source graphics and physics libraries, (PCI) I/O cards are used for controlling a mechanical platform providing five degrees of freedom. In this virtual reality application, positions of various mobile vehicles with different characteristics on a selectively chosen height maps are simulated. In addition the behaviour of the vehicle motions are transferred electronically onto the mechanical systems, that has been constructed, in real time. Since the simulator provides tangible outcomes, the users have the feeling of the real atmosphere of the vehicle.

Key Words : Vehicle simulation, Delta3D, open dynamics engine, openscenegraph, virtual reality

Science Code : 902.1.014

TEŞEKKÜR

Bu tez çalışmasının planlanmasında, araştırılmasında, yürütülmesinde ve oluşumunda ilgi ve desteğini esirgemeyen, engin bilgi ve tecrübelerinden yararlandığım, yönlendirme ve bilgilendirmeleriyle çalışmamı bilimsel temeller ışığında şekillendiren sayın hocalarım Prof. Dr. Abdullah ÇAVUŞOĞLU ve Yrd. Doç. Dr. Baha ŞEN'e sonsuz teşekkürlerimi sunarım.

Mekanik sistemlerin tasarlanmasında değerli tecrübelerini esirgemeyen sayın hocam Doç. Dr. Cevdet GÖLOĞLU ve öğrencisi, ülkemizdeki çok boyutlu sinema teknolojisindeki deneyimlerinden faydalandığım Mustafa MERTCAN'A teşekkürü borç bilirim.

Sistemin mekanik platformunun prototipinin hazırlanmasında her türlü imkanlarını sunan sayın hocam Yrd. Doç. Dr. İsmail KARACAN ve öğrencisi Orhan ÇEVİK' e teşekkür ederim.

Hayatımın her güzelliğinde emeği olan sevgili aileme, Pelin BİLİR, Kutlu KAPTAN ve Nesrin AYDIN'a manevi hiçbir yardımı esirgemeden yanımda olarak güç verdikleri için tüm kalbimle teşekkür ederim.

İÇİNDEKİLER

Sayfa

ŞEKİLLER DİZİNİ

Sayfa

ÇİZELGELER DİZİNİ

Sayfa

SİMGELER VE KISALTMALAR DİZİNİ

SİMGELER

e : Hata

K_d : Türev katsayısı

K_i : İntegral katsayısı

K_p : Oransal katsayı

PY : Piston Yüksekliği

minY : En küçük piston yüksekliği

µs : Mikro saniye

Ω : Ohm

^oC : Santigrat

KISALTMALAR

3B : 3 Boyutlu

3D : 3 Dimension

Ar-Ge : Araştırma – Geliştirme

CPU : Central Processing Unit

Eş. : Eşitlik

FPS : Frame Per Second

FSR : Full-scale range

GLSL : GL Shading Language

GNU : GNU Not Unix

GPU : Graphics Processing Unit

HMD : Head Mounted Display

ICEMT : Iowa Center for Emerging Manufacturing Technology

KMS : Kaideye Monteli Stinger

LGPL : Lesser General Public License (Kısıtlı Genel Kamu Lisansı)

LLC : Land cover classification

LSB : Least Significant Bit

mA	: Miliamper
mm	: Milimetre
mV	: Milivolt
Max.	: Maximum
Min.	: Minumum
ODE	: Open Dynamics engine
OpenGL	: Open graphics library
OSG	: OpenSceneGraph
P	: Proportional (Oransal)
PCI	: Peripheral Component Interconnect
PD	: Proportional Derivative (Oransal Türev)
PI	: Proportional Integral (Oransal İntegral)
PID	: Proportional Integral Derivative (Oransal İntegral Türev)
ppm	: Parts-per milion
SG	: Sanal Gerçeklik
SGI	: Silicon Graphics International
STAGE	: Simulation, Training And Game Editor
V	: Volt
VR	: Virtual Reality
XML	: Extensible Markup Language

BÖLÜM 1

GİRİŞ

İlk hesap makinesi abaküsten, günümüzde işlem gücü saniyede milyonlarla ifade edilen işlemcilere kadar geçen süreç, insanoğlunun bilim dünyasındaki hayallerini gerçekleştirebilmesi için önemli bir araç olmuştur. Elektronik ve malzeme alanındaki gelişmeler sonucunda, bilgisayar teknolojisi bilimsel araştırma yapan kurumların dışında artık günlük hayatta sıradan insanların kullandığı bir cihaz haline gelmiştir. Bu gelişmeler sonucunda bilgiye erişim, bilginin yorumlanması, matematiksel ve istatistiksel yaklaşımlar hızla artmıştır. Bunun yanı sıra güncel ve doğru bilginin elde edilmesi, uygun ve gerçeğe dayalı yorumlanması, sonuçların kabul edilebilir olması gibi gereklilikler ortaya çıkmıştır.

Teknolojiye erişimin ucuzladığı ve kolaylaştığı günümüzde hemen hemen her evde, işletmede, sınıfta en az bir bilgisayar kullanılabilir olmuştur. Bu gelişmelerin yanı sıra bilgisayar artık hesap yapan bir cihaz olmaktan çıkmış, çoklu ortam uygulamalarının yapıldığı, görsel ve işitsel duyulara hitap eden karmaşık bir sistem haline gelmiştir. Artık kitaplarla dolu kütüphanelerde gezmek yerine internette araştırma yapmak, bankalarda saatlerce sıra bekleyerek işlem yapmak yerine online bankacılık hizmetlerinden faydalanmaya kadar hayatın her alanında bilgisayarlar kullanılır olmuştur. Öyle ki bazı alanlar için özel donanımlara gereksinim duyulmuştur. Bunlardan en çok bilinen ve kullanılanı hiç şüphesiz grafik işlemcilerdir. Grafik yongalarının işi de hesaplamaktan farklı değildir. Ancak iş sırası, akışı, kontroller ve bazı özelleştirmeler işlerin hızlanmasını sağlamıştır. Örnek vermek gerekirse bir merkezi işlem biriminde (cpu'da) 4 çekirdek bulunurken, bir grafik işlemcide 240 çekirdek vardır. Farklı veri paketlerindeki işlemlerde gpu yavaş kalırken aynı veri paketleri için gpu önemli bir hız farkı oluşturabilmektedir [1,2].

Bilgisayar teknolojisinin çoklu ortam öğeleriyle kullanıldığı alanlardan bir tanesi de gerçek dünyada ulaşılması zor, tehlikeli, kavranması zor durumların modellenerek simüle edildiği sanal gerçeklik uygulamalarıdır. Sanal gerçeklik, geçmişi yaklaşık elli yıla dayanan bir teknolojidir. Sanal gerçeklik ortamlarında, genellikle simülasyon amacı ile kullanıcılar ve simülasyondaki nesneler arasında etkileşim söz konusudur. Sanal ortamlar önceleri büyük birkaç firmanın ticari ürünleriyken, günümüzde gelişen teknolojiyle birlikte ticari sır olmaktan çıkmış, birçok geliştirici tarafından hazırlanabilen ortamlar olmuştur.

İnsanların her şeyi yaşayarak tecrübe etmesi imkansızdır. Buradan hareketle sanal dünyalar ve matematiksel altyapıya dayanan sanal modeller oluşturmak bir ihtiyaç haline gelmiştir. Artık eğlence sektöründen eğitime, basit fen deneylerinden karmaşık tıp uygulamalarına kadar birçok alanda sanal gerçeklik kullanılmaya başlanmıştır. Sanal gerçeklik uygulamalarında kullanıcılar, kendilerini sanki o anı, durumu yaşıyormuş gibi hissederler. Bu durum gerçeğin birebir aynısı olmasa da kazanılan tecrübeler küçümsenemeyecek kadar ciddidir. Öyle ki artık askeri alanda kullanılan uçuş simülatörlerinden, sürücü kurslarında kullanılan sürüş simülatörlerine, eğlence dünyasına hitap eden çok boyutlu sinemalardan, turistik gezilerin yapılabildiği sanal şehirlere kadar sayılamayacak kadar çok uygulaması mevcuttur.

Ayrıca askeri alandaki birçok araç hem temin hem de kullanım açısından maliyeti yüksek araçlardır. Bu anlamda askeri personelin tecrübe kazanması ve verilen görevleri yerine getirmesi için yapılan simülasyonlar büyük önem arz etmektedir. Delta3D ile hızlı bir şekilde geliştirilecek yeni simülasyonlar simülatörün kullanımını esnek hale getirmektedir.

Çalışma sürüş eğitimlerinde kullanılabilecek araç simülatörü üzerine yapılmıştır. Çalışma, sanal ortam oluşturma, sanal ortamdaki nesnelere fizik kütüphanesinin sunduğu olanaklarla fiziksel kurallar yükleme, sanal ortam verilerinin somut bir çıktısı olan platformun diğer bileşenlerle birlikte uyumlu çalışması için yapılan entegrasyonu konu almaktadır. Açık kaynak kodlu grafik, fizik kütüphanelerinden faydalanılarak oluşturulmuş, yine açık kaynak kodlu bir oyun motoru ile oluşturulan

sanal dünyadaki aracın yükseklik ilişkileri fiziksel bir platformda oluşturulacak şekilde planlanmıştır.

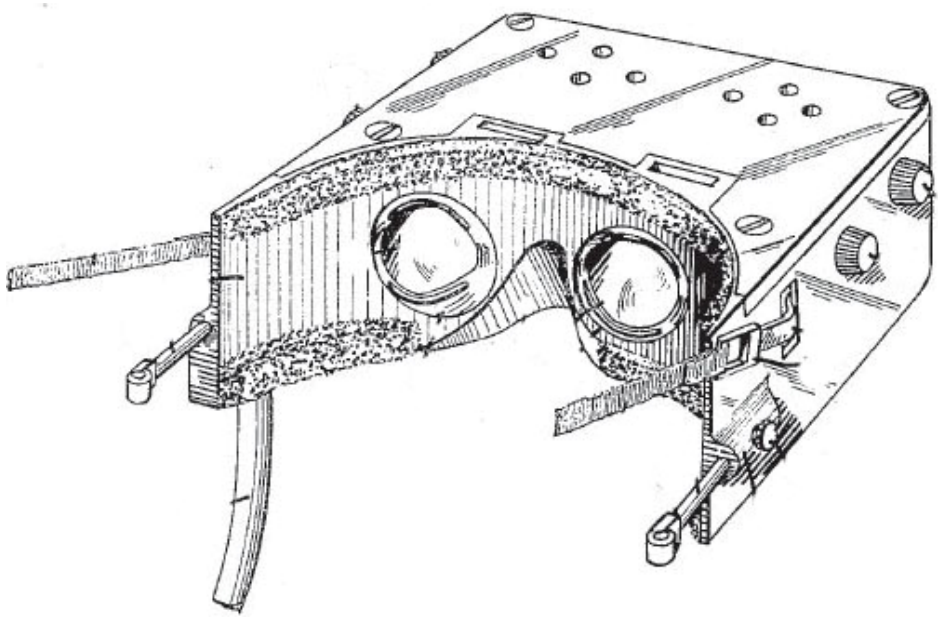
Çalışmanın ikinci bölümünde teknolojinin gelişmesiyle birlikte hayatımıza giren yeni bir kavram olan sanal gerçeklik, tarihçesi, sanal gerçeklik ortamlarında kullanılan aygıtlar ve sanal gerçekliğin eğitimdeki önemi üzerinde durulmaktadır. Çalışmanın üçüncü bölümünde simülasyon ve simülatör kavramları açıklanarak araç simülatörleri hakkında genel bilgiler verilmektedir. Dördüncü bölümde simülatörlerde kullanılan yazılımlar hakkında tanıtıcı bilgiler verildikten sonra çalışmada kullanılan Delta3D oyun motorunun tarihçesi ve özellikleri hakkında bilgiler verilmektedir. Beşinci bölümde mekanik platformun kontrolünde kullanılabilecek kontrol yöntemleri hakkında bilgi verildikten sonra ilerleyen bölümlerde simülatörün fiziksel ve yazılımsal yapısı hakkında bilgiler verilmiştir. Sonuç bölümünde hedefe ne kadar ulaşıldığı ve çalışmanın üzerine yapılabilecek geliştirmelere değinilmiştir.

BÖLÜM 2

SANAL GERÇEKLİK

Sanal gerçeklik (SG), bilgisayar ortamında oluşturulan üç boyutlu resim ve animasyonların teknolojik araçlarla insanların zihninde gerçekmiş hissi yarattığı, bir ya da daha çok bilgisayar ve özel yazılım sisteminden oluşan, insan – bilgisayar etkileşiminin olduğu üç boyutlu benzetim modelidir [3-5].

İlk sanal gerçekliğe uygun makine 1950'lerin sonunda bir film yapımcısı olan Morton Heiling tarafından tasarlandı. İki boyutlu resimler izleyicinin görme alanının %18'ini doldururken, Heiling üç boyutlu görüntülerle görme alanının %100'ünü doldurmak istedi. 1957'de, başa yerleştirilen, her göz için ayrı birer ekrandan oluşan siyah-beyaz bir TV ünitesi için patent aldı. Bu tasarım stereo sesler için kulaklıktan, farklı hız ve sıcaklıklar için kokulu ya da kokusuz hava veren borudan oluşmaktadır [6]. Şekil 2.1'de bu tasarım görülmektedir.



Şekil 2.1. Bireysel kullanım için başa takılan Stereoskopik televizyon aparatı [6].

Daha sonra yaşanan bazı gelişmelerin ardından Birleşik Devletler Hükümeti, Tom Furness ve grubu tarafından tasarlanan ileri savaş kokpiti, havacılık simülatörü ile ilgilenmişlerdir. Bu tasarımda pilot pencerenin dışında oluşturulan grafik ekranlı cihazı başına takmaktadır. Nihayet 1980'lerin başında NASA bir mini-bilgisayar kullanarak gerçek bir SG sistemi yapmıştır [6]. Bütün bu gelişmelerin ardından 3B ortamlar için veri eldiveni, vücut sensörleri, vücut kitleri gibi çeşitli donanımlar geliştirilmiştir.

Teknolojinin gelişmesi, bilgisayarların ticari birer cihaz olmaları ve oyun oynanacak kadar ucuzlamaları, bu teknolojinin gelişmesini sağlamıştır. Artık üç boyutlu görüntülerin yanında ses, koku, rüzgar ve su gibi çeşitli kontrol ve eklemelerle sanal gerçeklik daha da ileriye gitmektedir [7]. Halen bu alanda ciddi çalışmalar devam etmektedir.

Sanal gerçekliğin geçmişi eski olmasına rağmen günümüzün popüler konularındandır. Öyle ki bu alanda çalışan bilim insanı ve ticari firmalar azımsanamayacak kadar çoktur. Popülaritesini korumasının en büyük sebeplerinden bir tanesi bilgisayar teknolojisinin hala istenilen hıza ulaşmamış olmasıdır. Gerek grafiksel işlemlerde, gerek matematiksel ve fiziksel modellerin çalıştırılmasında yavaşlamalar söz konusu olmaktadır. Üniversitelerde çeşitli gruplar tarafından sanal gerçeklik üzerine geniş çalışmalar yapılmaktadır. Çalışma konularından bazıları sıralanacak olursa [4]:

1. Görüntüleme teknolojisi
2. Etkileşim aygıtları
3. Ara yüz teknolojisi
4. Uygulama tasarımı

bunlardan önemli başlıkları oluşturur.

Iowa State Üniversitesinde de mühendislik uygulamaları için, disiplinler arası, işbirlikçi sanal gerçeklik araştırmasına yoğunlaşılmıştır. Iowa Center for Emerging Manufacturing Technology (ICEMT) adını verdikleri tesiste birçok konu üzerine

çalışma yapılmaktadır. Burada bilgisayarlardaki hesaplama işlerinin alt yapısı süper bilgisayar üreticisi olan, askeri ve sivil birçok alanda hizmet veren SGI'ye dayalıdır [4].

2.1. SANAL GERÇEKLİKTE KULLANILAN DONANIMLAR

2.1.1. Başa Takılan Ekran (Head Mounted Display)

Kullanıcının başına taktığı bir visör ya da miğfer şeklindeki bileşendir (HMD). HMD kullanıcının kendini sanal ortamda hissetmesini sağlamak amacıyla bir bilgisayara bağlı olarak çalışır. HMD, her göz için ayrı bir görüntü ünitesi ve seslerin algılanabilmesi için hoparlör içerir. Kullanıcının başını döndürdüğünde görüntünün değiştirilebilmesi, hareketi algılayıp sisteme aktaran bir donanımla gerçekleştirilebilmektedir. Şekil 2.2'de HMD görülmektedir.

Şekil 2.2. HMD [3].

2.1.2. Veri Eldiveni

Veri eldiveni sanal gerçeklik ortamları için giriş aygıtı olan bir eldivendir. Parmakların fiziksel hareketlerini yakalamak için çeşitli sensör teknolojileri

kullanılmaktadır. Sıklıkla hareket izleyici, manyetik izleyici aygıt veya eylemsiz izleme aygıtları, eldivenin evrensel pozisyonunu ve yönünü yakalamak için bağlanır. Bu hareketler aynı zamanda bir yazılım tarafından yorumlanır. Hareketler işaret dilindeki gibi ya da farklı sembolik fonksiyonlar olarak kullanılabilir bilgiler olarak kategorilendirilebilir. “Datagloves” ya da “cybergloves” terimleri Sun Microsystems ve Immersion Corporation’a ait ticari markalar olduğundan kablolu eldiven olarak anılırlar [8]. Şekil 2.3’de veri eldiveni görülmektedir.

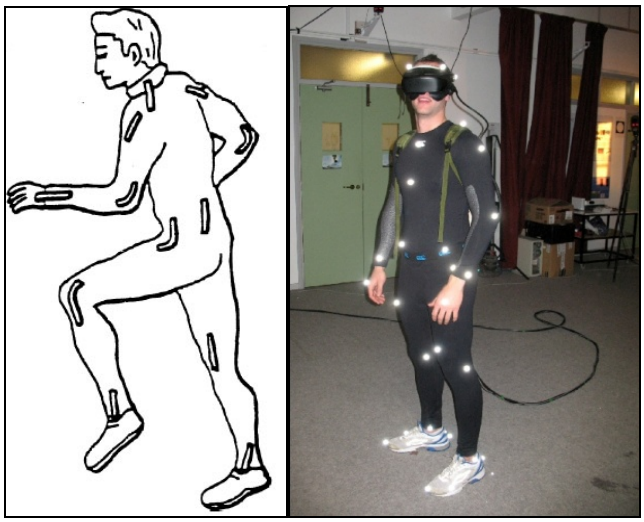
Dokunma hissini simule eden pahalı eldivenler hassas geri dönüşler de sağlayabilir, hem de bir çıkış aygıtı olarak kullanılabilir. Geleneksel olarak veri eldivenleri muazzam maliyetli donanımlardır. Parmak hareketlerini algılayan sensörler ve izleme aygıtları ayrı ayrı satılırlar [8].

Şekil 2.3. Windows Mouse olarak kullanılan P5 veri eldiveni [8].

2.1.3. Vücut Kitleri ve Sensörleri

İlki Thomas Zimmerman tarafından yapılarak patenti alınan vücut kiti, veri eldiveni yanı sıra optik, esnek sensörlerden oluşmaktadır. Patenti alınan vücut kiti Şekil

2.4'de gösterilmektedir. Bu sayede bedensel hareketler sanal ortama aktarılmaktadır. Günümüzde özellikle oyun sektöründe ve matematiksel olarak modeli çıkarılamayan durumlar için benzer teknolojiler kullanılmaktadır [6,9].



Şekil 2.4. Vücut kitleri ve sensörleri [3,9].

2.1.4. Kabin Simülatörleri

Kabin simülatörleri, kullanıcıların gerçeğiyle benzer ya da aynı şekilde tasarlanmış kokpit, sürücü koltuğu vb. gibi fiziksel bileşenlerdir. Bunun yanı sıra kullanıcının sanal gerçekliği yaşadığı ortam, bilgisayar aracılığıyla oluşturulur. Kullanıcı joystick, direksiyon, buton vb gibi aygıtlarla etkileşimli olarak sanal ortamı yaşama fırsatı bulur [3]. Şekil 2.5'de örnek kabin simülatörü görülmektedir.

Şekil 2.5. Kabin Simülatörü [3].

Bazı kabin simülatörleri aynı zamanda fiziksel olarak da çıkış verebilirler. Örnek olarak uçuş simülatörleri askeri alanda, sürüş simülatörleri sivil alanda fiziksel risklerin azaltılması bakımından önemli bir kullanıcı kitlesine sahiptir. Bu alanda ülkemizde de faaliyet gösteren firmalar da bulunmaktadır. Bunlardan en kayda değerlerinden biri şüphesiz daha çok askeri alanda faaliyet gösteren Aselsan tarafından üretilen Kaideye Monteli Stinger Sistemi Eğitim Simülatörüdür. KMS simülatörü ile sistemin genel fonksiyonlarının yanı sıra oluşturulmuş çeşitli senaryolara göre kullanıcıların tecrübe kazanmasını sağlamaktadır [10]. Şekil 2.6'da KMS görülmektedir.

Şekil 2.6. KMS Simülatörü [10].

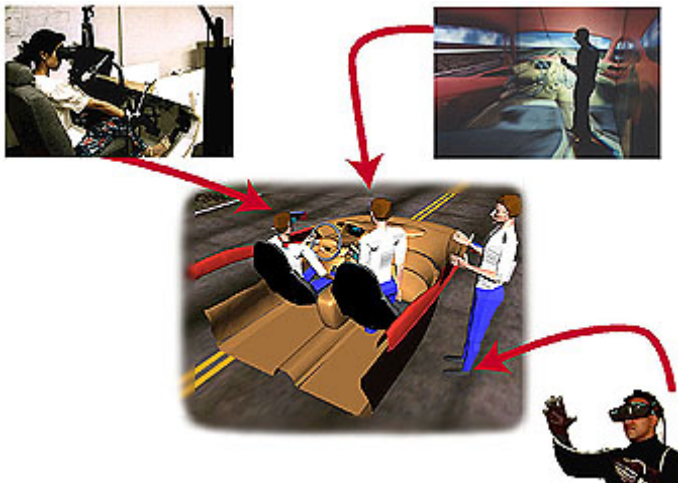
2.1.5. Masaüstü Sanal Gerçeklik

Masaüstü sanal gerçeklik ortamında bilgisayar monitörü yanında klavye, fare ya da joystick gibi bir giriş aygıtları kullanılır. Modellenmiş bir şehirde dolaşmak masaüstü sanal gerçekliğe iyi bir örnektir. Son yıllarda oyun sektöründe bu teknoloji oldukça fazla kullanılmıştır. Şekil 2.7'de örnek bir masaüstü sanal gerçeklik uygulaması görülmektedir.

Şekil 2.7. Masaüstü sanal gerçeklik [3].

2.2. PAYLAŞTIRILMIŞ SANAL ORTAMLAR

Farklı yerlerdeki kullanıcılar aynı sanal ortamı paylaşabilirler. Kullanıcıların aynı araç-gereçlere sahip olma ya da kullanma zorunlulukları olmadığı paylaşılmış sanal ortamlarda, kullanıcılar diğer kullanıcılarla etkileşim kurabilirler [3]. Şekil 2.8'de paylaşılmış sanal ortam resimlerle açıklanmaktadır. Paylaşılmış sanal ortamlar, günümüz en çok oyun sektöründe kullanılmaktadır. Son dönemlerde popüler hale gelen online oyunlar sayesinde, tüm kullanıcılar evlerinden, ofislerinden hatta kafeteryalardan, kısaca internete girme imkanının olduğu hemen her yerden paylaştırılmış sanal ortamları kullanmaktadır.



Şekil 2.8. Paylaşılmış sanal ortamlar [3].

2.3. SANAL GERÇEKLİK UYGULAMALARI

Sanal gerçeklik, öncelikle beklentinin az, hatanın en kolay tolere edilebileceği oyun ve eğlence sektöründe uygulama alanı bulmuştur. Günümüzde Amerika ve Japonya'da SG teknolojisine dayalı oyun salonları kurulmuştur [5].

Gerçek mekanların modellenmesi ile SG teknolojisi turistik amaçlarla da kullanılmıştır. Bunların en önemlilerinden biri, Fransız devrimi sonrasında yok edilen 'The abbey of Cluncy', arşiv kayıtlarına göre SG yardımıyla bilgisayar ortamında tekrar can bulmuştur [5].

SG teknolojisi şehir planlama konusunda da kullanım imkanı bulmuştur. İnsanların yaşadığı ortamı değiştirmek, en uygun şekilde yeniden planlama isteği nüfusun artması ile artık bir gereklilik haline gelmiştir. Şehir planlama ve inşa etme çok zaman alan ve maliyetli bir olaydır. Bu sebeple yapılmış çalışmalardan biri Çin'in Pekin şehri için yapılmıştır. Yapılan çalışmada Pekin'in seçilmesi nedeni, modern bir kent görünümüne sahip olabilmesi için sosyal, çevresel, ekonomik ve planlama boyutlarının göz önünde bulundurularak bir çalışma yapılmasına duyulan ihtiyaçtır [5].

Ayrıca simülasyon sistemlerini kullanan kullanıcıların bir süre sonra sıkılması sürekli aynı ortamı kullanmalarından kaynaklanmaktadır. Çünkü simülasyon sistemlerinin

çoğunda ya mevcut ya da sanal olarak üretilmiş sınırlı sayıda harita kullanılmaktadır. Bu sebeple belirtilen şartlara uygun haritalar üretilmesini sağlayan simülasyon sistemleri için sanal şehirler üreten çalışmalara literatürde rastlamak mümkündür [11,12]. Şekil 2.9'da simülasyon sistemleri için belirli kriterlere göre üretilmiş 3B şehir görülmektedir.

Şekil 2.9. Simülasyon sistemleri için belirli kriterlere göre üretilmiş 3B şehir [11].

Dünyanın önemli ağır – iş makineleri üreticilerinden Caterpiller Inc., Peoria, IL., tasarımları gözden geçirmede sanal prototipleri tercih etmektedir. Bu, para ve zaman kazancı anlamına gelmektedir. Dolayısı ile son ürünlerin rekabette daha güçlü olmasını sağlamıştır. Bunun yanında Boeing Computer Sytems Inc., Boeing 777 uçaklarının iç tasarımını SG teknolojisini kullanarak gerçekleştirmiştir [5].

SG, öğrenme süreçlerine de önemli katkılar sunmaktadır. Örneğin kayakla ilgili geliştirilen sistem, kullananların ciddi kazalar geçirmeden, fiziksel ortamın risklerinden uzak bir öğrenme imkanı sağlar [13]. Şekil 2.10'da kayak simülatörü görülmektedir. Benzer şekilde matematik, fen, tıp, asker ve havacılık eğitimlerinde SG kullanılması hem maliyetleri hem de riskleri azaltmaktadır. Ayrıca fen

derslerinde bir bileşiğin oluşmasını, moleküller arasındaki çekim kuvveti, cisme etki eden kuvvetlerin gösterilmesi gibi gözle görülmesi pek de kolay olmayan konuların kavranmasını kolaylaştırmaktadır [3,5]. Tarih, coğrafya gibi derslerde gerçek ortamların görünmesinin zor, tarihsel olayların canlandırılmasının, izli, kalıcı ve etkili bir öğrenmenin olmasının istendiği durumlarda SG eğitimde kaliteyi arttırmak için önemli bir yöntem olarak kendini kabul ettirmektedir.

Şekil 2.10. Kayak Simülatörü [13].

2.4. EĞİTİMDE SANAL GERÇEKLİĞİN FAYDALARI

Eğitimde sanal gerçekliğin sağladığı faydalar şüphesiz çok önemlidir. Bu konuda yapılmış çalışmalar ve tecrübeler bu önemin en gerçek göstergesidir. Faydanın ölçüsü şüphesiz kullanıcıdan kullanıcıya, gruptan gruba değişiklik gösterecektir. Bu anlamda beklenen fayda da önemlidir. Erişkin bir kişi için bir cam nesnenin yere düştüğünde kırılması ve tehlikeli sonuçlar doğurması yaşanarak öğrenilmiş bir tecrübe olabilir. Oysa henüz bu durumu tecrübe etmemiş çocuklar için bilinmeyen bir durumdur. Yetişkinlerin hiç ilgisini çekmeyecek bu durum çocuklar için iyi bir öğrenme fırsatı olabilmektedir.

Sanal gerçeklik hayatın her alanında olduğu gibi, eğitimde de önemli faydalar sağlamaktadır. Bu faydaların bir kısmı şunlardır:

1. Motivasyonu arttırarak kalıcı öğrenme sağlar.
2. Gözle görülmesi mümkün olmayan hususların açıklanmasını kolaylaştırır.
3. Malzeme temini, deneylerin tehlikeli olması, öğrencinin çeşitli sebeplerle fiziksel olarak katılamadığı derslere ve konulara uzaktan katılma olanağı sunar.
4. Öğrencinin kavrama/anlama düzeyine göre tekrarlanabilir uygulama imkanı sunarken eğitim maliyetlerini arttırmaz.
5. Bilgisayar ve teknoloji kullanım becerilerinin artmasına yardımcı olur.

BÖLÜM 3

SİMÜLASYON VE SİMÜLATÖR

Simülasyon, doğal sistemlerin ya da insanlar tarafından oluşturulan sistemlerin bir benzerinin farklı bir ortamda kurularak sistemin nasıl çalıştığının anlaşılması yolunda yapılan faaliyetlerdir. Şekil 3.1'de I. Dünya savaşı döneminde kullanılan at simülatörü görülmektedir. Diğer teknolojik simülasyonlar performans artışı, güvenlik, test, eğitim ve öğretim konularını içerir. Çoğunlukla bilgisayar teknolojileri kullanılarak grafiksel ortamda simülasyonlar yapılmaktadır [14,15].

Şekil 3.1. Birinci Dünya Savaşı zamanında kullanılan at simülatörü [15].

Düşük maliyet ve hız, simülasyon sistemlerinin kullanılmalarının en temel gerekçeleridir. Ayrıca tasarım sırasında sistemin gerçekleştirilebilir olduğunu görmek, gerçekleştirildiğinde sistemin davranışlarının incelemek, parametrelerdeki değişimin sisteme etkisini görmek amacıyla, tasarım aşamasında simülasyon sistemlerinden faydalanılır. Ek olarak, var olan sistemlerin performansını anlamak, yapılabilecek değişikliklerin etkilerini ve performansın arttırılması konularında simülasyon sistemlerinden faydalanılır [14].

Örnek olarak elektronik devre tasarımında kullanılan Multisim, Isis Professional, Orcad vb. programlarla uygulama geliştirmek, denemek ve hata bulmak, gerçek

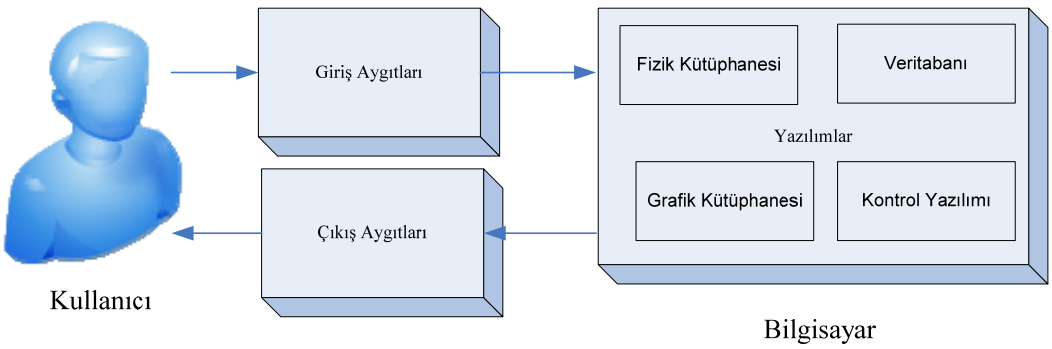
hayatla kıyaslanamayacak kadar hızlıdır. Özellikle teknolojiyi üretmeden dışarıdan satın alan ülkemizde malzeme temini her zaman kolay olmamaktadır. Bunun yanında yüksek maliyetli malzemelerle çalışmak ve denemeler yapmak riskli ve maliyetlidir.

İyi bir simülasyon programından, gerçek sisteme, duruma ya da olaya yakın sonuçları hızlı bir şekilde üretmesi beklenir. Ayrıca simülasyon parametrelerinin değiştirilebilir olması, kullanıcıya iyi bir arayüz sunması, sonuçları etkin bir şekilde raporlayabilmesi ve standartlarla uyumlu çalışması programdan beklenen diğer özelliklerdir. Simülasyonun doğru çalışması için matematiksel modellerin ve girilen verilerin doğru olması en önemli gerekliliklerdendir [14].

Yukarıda anlatılan gerekçelerle simülasyonlar her alanda kullanılabilmektedir. Buradan hareketle araçların gerçek hayattaki durumlarının modellenerek simüle edilmesi çalışmanın başlangıç noktasını oluşturmaktadır. Çalışmanın araç simülasyonu üzerine yapılması sebebiyle araç simülatörlerine göre bir yol izlenecektir.

3.1. BİLGİSAYARLI SİMÜLATÖRLERİN YAPISI

Simülatörler, kullanıcıya simüle edilen sistemin görüntüsünü, simüle edilen sistemle iletişim kurabilmek için giriş aygıt veya aygıtlarını ve varsa sistemin mekanik olarak çıkışını sunarlar. Bu özellikler açısından simülatörleri Şekil 3.2'deki gibi ilişkisel bileşenlere ayırabiliriz.



Şekil 3.2. Simülatör bileşenlerinin ilişkisel gösterimi.

3.1.1. Giriş Aygıtları

Simülasyon yazılımının parametre aldığı klavye, fare, joystick ya da simülasyonu yapılan sisteme özgü kontrol birimi giriş aygıtıdır. Girişler simülasyon anında verilebildiği gibi simülasyon başlamadan önce belirlenen parametreler doğrultusunda da işlemler yapılabilir.

3.1.2. Çıkış Aygıtları

Simülasyon yazılımı tarafından giriş parametrelerine göre hesaplanan/üretilen sonuçlar çıkış aygıtları ile kullanıcıya iletilir. Monitör, gösterge ve yazılım tarafından kontrol edilen platform çıkış aygıtlarına örnek gösterilebilir.

3.1.3. Yazılımlar

Yazılımlar simülasyon sisteminin en önemli bileşenleridir. Şekil 3.2' de temsili olarak gösterildiği gibi simülasyon yazılımı birden çok bileşenden oluşur. Kullanıcıya sunulan görsel ortam, grafik kütüphanesi ile gerçekleştirilir. Görsel ortamdaki nesnelerin rengi, sayısı, ortamdaki konumu gibi özellikleri veritabanında saklanır. Nesnelerin grafiksel ortamdaki hareketlerinin ve birbirleri arasındaki etkileşimin gerçek dünyaya uygunluğu matematiksel modellere göre oluşturulmuş kurallara göre yapılır –ki bu görev fizik kütüphanesi vasıtası ile gerçekleştirilir. Ayrıca simülatör, titreşim veya platform hareketlendirilmesi gibi fiziksel bir çıkış üretiyorsa bu hareketler kontrol yazılımı tarafından yapılır.

3.2. ARAÇ SİMÜLATÖRLERİ

İhtiyaçlar doğrultusunda, simülasyonlar birçok alanda olduğu gibi araçlar için de geliştirilmiştir. Geliştirilen simülatörler en çok eğlence/oyun ve eğitim amaçlı olarak kullanılmaktadır. Bunlardan en çok bilinen ve kullanılanları şöyle sıralayabiliriz [16]:

1. Uçuş Simülatörü (Flight Simulator)
2. Yarış/Sürüş Simülatörü (Racing Simulator)
3. Uzay Gemisi Simülatörü (Spacecraft Simulator)
4. Tren Simülatörü (Train Simulator)
5. Mücadele Taşıtı Simülatörü (Vehicular Combat Simulator)
6. Deniz Uçağı Simulatörü (Watercraft Simulator)
7. Kamyon Simülatörü (Trucking Simulator)

Uçuş simülatörleri askeri ve sivil amaçlı olarak sınıflandırılabilir. Askeri alandaki simülatörler kullanıcının verilen görevleri başarmasını, genellikle havadan yapılan saldırıyı temel alır. Sivil uçuş simülatörlerinde savaşa dayalı simülasyonlar olsa da amaç çoğunlukla yarış, hız ve hızlanma testi gibi özel turnuvalardır. Bunlardan sivil alanda en bilinenleri FlightGear, Microsoft Flight Simulator, X-Plane, askeri alanda ise Lock-On'dur [16].

Yarış simülasyonlarını organize edilmekte olan ve sanal olarak organize edilen simülasyonlar olarak sınıflandırılabilirler. Organize edilmekte olan yarışlar için yarış kategorilendirmesine göre sürüş deneyimi imkanı verilir. Indycar, NASCAR, Formula 1 vb uygulamalar buna örnektir. Diğer yandan sanal organizasyonlar araziden şehre fantastik bir eğlence imkanı sunarlar. Yarış simülatörlerin öyle ciddileri vardır ki, simülasyonda araçlarda modifikasyon yapılabilir, araçlar fiziksel olarak hasar alabilir, yakıtı bitebilir. Diğer simülasyonlar bu faktörleri gözetmezler [16].

Tren simülatörleri, raylı sistem taşımacılığını simule ederler [16]. Bu alanda bir diğer simülatör de raylı sistemler simülatörüdür. Buradaki amaç hata kabul etmeyen raylı sistem tasarımının simüle edilerek yapılabilecek değişikliklerin inşa aşamasından önce tasarlanmasını sağlamaktır [14].

Mücadele taşıtı simülatörleri tank ve diğer güvenlik araçlarının simülasyonlarını içerir. Bu simülasyonlarda doğruluk, hız, görüş mesafesi, ve geniş kitlelere karşı silahlar kullanımı işlenmektedir.

Kamyon simülatöründe kullanıcı şoför veya nakliyeci pozisyonundadır. Genellikle amaç para kazanmak, daha üst sınıftaki kamyonlara sahip olmaktır. 18 Wheels of Steel veya Euro Truck Simulator bu konudaki en iyi örneklerdendir [14]. Bunların yanı sıra sivil alanda ve Şekil 3.3'de görüldüğü gibi askeri alanda eğitim amaçlı ağır vasıta simülatörleri de kullanılmaktadır.

Şekil 3.3. Askeri alanda kullanılan ağır vasıta simülatörü [15].

3.2.1. Sürüş Simülatörlerinin Yapısı

Dünyada son zamanlarda ileri profesyonel sürüş simülatörlerinde küçük arabalar ya da kamyonlar kullanılmaktadır. Yüksek çözünürlüklü gerçeğe yakın görüntüler için üçten fazla grafik işlemci, yüksek performanslı paralel çalışan bilgisayarlar ya da bir grup sunucu ve 3B sesler simülasyonlar için kullanılmaktadır. Hareketlerin oluşturulduğu mekanik sistemler için altı serbestlik dereceli Stewart Platform yapısı tercih edilmektedir. [17].

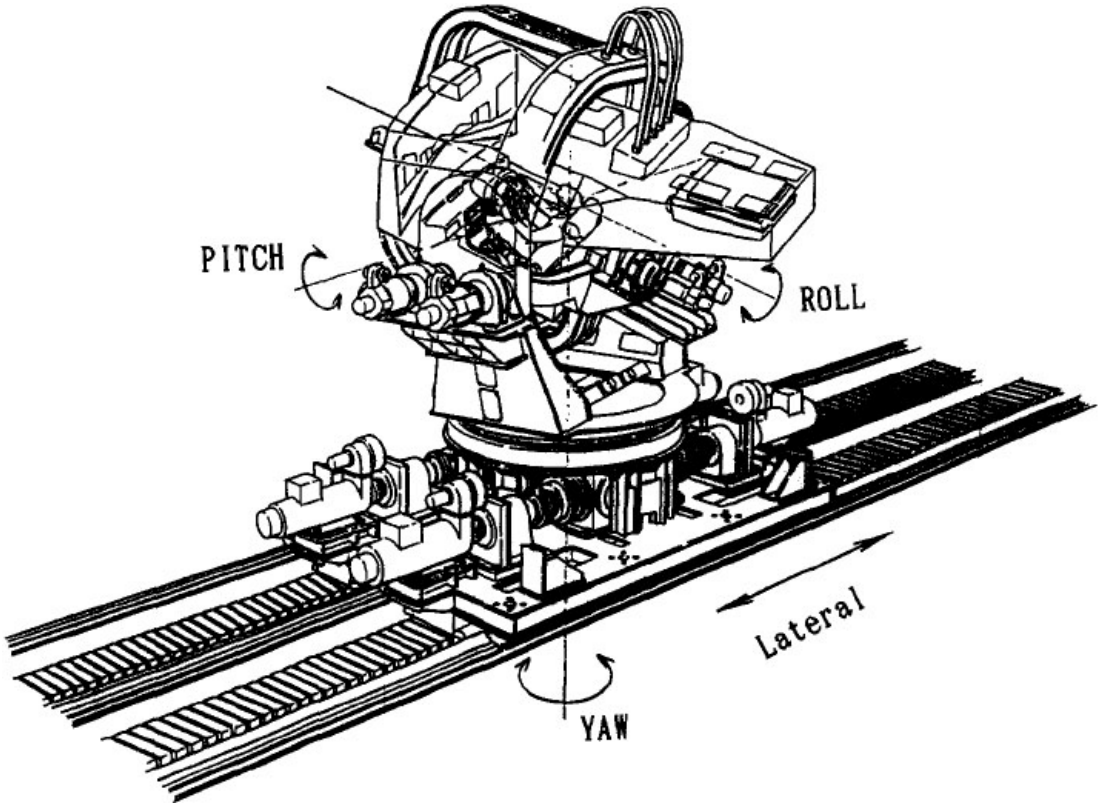
Hareketli platforma sahip olanlar kullanıldıkları yerlere göre farklılık göstermektedirler. Oyun salonlarında kullanılan simülatörlerin bir kısmı iki serbestlik derecesine sahip simülatörlerdir. Yalnızca düşme, sekme ve yana yatma durumlarında çıkış verirler. Hareketlerin sertliği açısından çoğunun matematiksel bir dayanağı yoktur. Bir diğer, iki serbestlik derecesine sahip simülatör tipinde yalnızca öne, arkaya, sağa ve sola yatma simule edilir. Diğerinde olduğu gibi bu tip simülatörlerde de hareketlerdeki şiddetin matematiksel bir dayanağı yoktur. Eğlence

amacı güdülerek yapıldığı için kullanıcıya verilen her yeni çıkış kullanıcının aldığı keyfi arttırmaktadır.

Bunlardan biraz daha gelişmiş üç tahrik noktalı beş serbestlik derecesine sahip platformlara rastlamak da mümkündür. Bu tip simülatörler oyun dışında eğitim amaçlı olarak da kullanılmaktadır.

Dört tahrik noktalı beş hareket kabiliyetli sürüş simülatörleri hakkında literatürde çalışma olmamasına karşın, bireysel hobi amaçlı çalışmalar bulunmaktadır. Bunlarda yana doğru olan ötelenme hareketleri üç tahrik noktalı olanlara oranla daha iyi simule edilebilmektedir.

Profesyonel simülatörler altı tahrik noktalı altı serbestlik derecesine sahip Stewart Platformu yapısını kullanırlar. Bu tip simülatörler de hem eğlence, hem eğitim amaçlı olarak kullanılmaktadırlar [17].



Şekil 3.4. Mazda Araştırma Merkezi'nde geliştirilen sürüş simülatörü [18].

Daha gelişmiş simülatörler altıdan çok serbestlik derecesine sahip hareket kabiliyetine sahiptir [18-20]. Şekil 3.4. ve Şekil 3.5'de otomobil üreticilerinin kullandığı çok gelişmiş simülatörler görülmektedir.

Şekil 3.5. Toyota tarafından kullanılan sürüş simülatörü [20].

BÖLÜM 4

SİMÜLATÖR YAZILIMLARI

Simülasyon sistemlerinin en önemli bileşenleri hiç şüphesiz yazılımlardır. Ülkemizde bu bağlamda yapılmış çalışmalar mevcuttur [21,22]. Bunun dışında yazılım geliştirme sürecinde, geliştiricilere yardımcı olan kütüphanelerin öne çıkanları bu bölümde incelenecektir. Bu bölümde incelenen yazılımlar yaptıkları işlere göre sınıflandırılmış bir şekilde tanıtılacaktır. Daha sonra çalışmada kullanılanlar detaylandırılacaktır.

4.1. GRAFİK KÜTÜPHANELERİ

4.1.1. Open Graphics Library (OpenGL)

Silicon Graphics Inc. tarafından tasarlanan OpenGL, grafik donanımı için gelişmiş destek sunan, iki ve üç boyutlu grafikleri ekrana çizdirmek için kullanılan bir yazılım ara yüzüdür. Bu ara yüz yazılımcıya üç boyutlu, yüksek kaliteli grafik özellikli objeler üretmesini sağlayan yüzlerce yordam ve fonksiyondan oluşmaktadır. Windows, Linux, MacOS ve Solaris gibi birçok işletim sisteminde ve Playstation3 başta olmak üzere birçok oyun konsolunda desteklenen, çoklu ve cross platform desteği içeren bu kütüphane, özellikle deneysel, bilimsel araçlarda başarılı ve standart olarak kullanılmaktadır. Ada, C, C++, C# (SharpGL adı verilen sınıflar sayesinde), Fortran, Python, Perl ve Java programlama dilleri kullanılarak OpenGL kitaplığından faydalanılabilir [23,24].

4.1.2. DirectX

Microsoft tarafından Microsoft'un video oyunları başta olmak üzere çoklu ortam yazılımlarının rahat, hızlı ve uyumlu bir şekilde hazırlanabilmesi için oluşturulmuş

yazılım programlama ara yüzüdür. Microsoft tarafından geliştirilmesi, platform bağımlı çalışmasının en büyük gerekçelerinden biri olan DirectX, grafik, ses, giriş/çıkış aygıtı yönetimi ve ağ üzerinden iletişim özellikleri ile göz doldurur. Linux sistemlerde direkt olarak kullanılamayan DirectX ancak farklı yazılımların kullanılması ile çalışmaktadır. OpenGL, DirectX'in Windows Grafik Temelleri Direct3D'nin en büyük rakibidir. Playstation'ın ciddi rakibi XBox DirectX kullanmaktadır [25].

4.1.3. OpenSG

Sanal gerçeklik gibi gerçek zamanlı grafiksel programlar oluşturmak için taşınabilir bir yazılımdır. Açık kaynak ve ücretsiz olan, OpenGL temeli üzerine dayalı kütüphane Windows, Linux, Solaris ve MacOS X platformlarında çalışmaktadır. Halen birçok projede ve yerde farklı amaçlarla kullanılmaktadır [26].

4.1.4. OpenSceneGraph

Linux sistemler için sahne grafiği (scene graph) yazmaya başlayan SGI yazılım danışmanı Don Burns, OpenGL SGI tarafından Linux'a port edildikten sonra projesini rafa kaldırmıştır.1990ların başında e-posta listesinden Don ile tanışan açık kaynak destekçisi Roberts Osfeld projenin yeniden hayat bulmasını sağlayan isim babasıdır. 2000li yılların ilk yarısında önemli yol kat eden proje 2006 yılında tamamı yenilenerek yayınlanmıştır [27]. OSG açık kaynak, cross platform, yüksek performanslı 3 boyutlu grafik aracıdır. Uygulama geliştiriciler tarafından görsel simülasyon, oyun, sanal gerçeklik, bilimsel modelleme ve görüntüleme alanları gibi pek çok alanda kullanılan OSG, ticari sahne grafiği araçlarının performans ve yararlılık bakımından önemli bir rakibidir. OpenGL ve standart C++ ile yazılan kütüphane bütün Windows platformlarında, OSX, GNU/Linux, IRIX, Solaris, HP-Ux, AIX ve FreeBSD işletim sistemlerinde çalışmaktadır. Zaten açık kaynak sahne grafiği araçları arasında lider ve popüler olan OSG, ticari sahne grafiği araçlarının alternatifi olarak çok sayıda üniversitede araştırmacılar tarafından kullanılmaktadır [28].

4.2. FİZİK KÜTÜPHANELERİ

Simülasyonlardaki nesnelerin hareketleri, birbirleri ile olan etkileşimleri, tepkileri, cisimlere etki eden kuvvetler gibi gerçek dünyanın fiziksel kurallarını sanal ortama taşıyan bileşenler fizik kütüphaneleridir. Mühendislik problemlerinin çözümü için üretilen matematiksel modellemeler fizik kütüphanelerinde yer almaktadırlar. Ayrıca bu alanda çeşitli firmaların yaptığı ar-ge faaliyetleri sonucunda elde ettikleri sonuçlar analiz programlarında matematiksel modelleri ile kıyaslanarak, matematiksel modellerin geçeğe yakınlığı da ölçülmektedir. Bu alanda ticari ve açık kaynak ücretsiz birçok yazılım mevcuttur.

4.2.1. Havok Physics

İrlanda da bir firma tarafından geliştirilmiş fizik kütüphanesi, konsol ve video oyunları için tasarlanmıştır. 3B gerçek zamanlı çarpışma ve katı nesne dinamiklerine izin vermektedir. Büyük ölçüde optimize edilen çarpışma tespit kütüphanesi ve birçok dinamik sınırlama tipi sunmaktadır. Dinamik simülasyon özelliği ile oyunlarda daha gerçekçi dünya sunan yazılım, firmanın yeni geliştirdiği yazılımlarla ters kinematik işlevleri de içermektedir [29].

Microsoft Windows, Xbox, Xbox360, Nintendo GameCube ve Wii, Sony Playstation Portable, Playstation2 ve Playstation3, Linux ve MacOS X'de çalışan 7.1 versiyonu 2009 yılında yayınlanmıştır. Bazı kütüphaneleri binary formatta verilen yazılımın C/C++ kaynak kodları, motor fonksiyonlarının rahatça değiştirilmesini ve farklı platformlarda çalışmasını mümkün kılmaktadır [29].

4.2.2. Open Dynamics Engine (ODE)

Proje sahipliğini Russel Smith'in yaptığı ODE 2000li yılların başında yayınlanmış, açık kaynak kodlu katı ve esnek nesne dinamiklerini simüle etmek için oluşturulmuş yüksek performanslı bir kütüphanedir. Özellikleri tam, kararlı, olgunlaşmış ve platformdan bağımsız olarak çalışan, C/C++ ile kolayca kullanılabilen uygulama programlama ara yüzüdür (api). Gelişmiş eklem tipleri ve sürtünme ile çarpışma

tespiti entegrasyonuna sahiptir. Araç simülasyonunda, sanal gerçeklik ortamlarındaki nesnelerde ve sanal nesnelerin simülasyonunda kullanışlıdır. 3B yazım araçları ve simülasyon araçları ile birçok bilgisayar oyununda kullanılmaktadır [30].

4.2.3. Newton Game Dynamics

Newton Game Dynamics ücretiz ama kapalı kod fizik motorudur. Julio Jerez ve Alain Suero'nun yazdığı kütüphane, oyunlarda ve diğer gerçek zamanlı uygulamalarda katı nesnelerin davranışlarını gerçekçi olarak simüle eder. Deterministik yöntemle çalışan uygulama tekrarlı yöntemlere göre biraz yavaş kalmaktadır. Windows, Mac OS X, iPhone, ve Linux platformlarında çalışan yazılım, yeni versiyonunda çok çekirdekli işlemci ve grafik işlemcileri ile uyumludur. Akademik ve ticari birçok uygulamada kullanılmaktadır. C uygulama geliştirme ara yüzü sunan yazılım kararlı ve güvenilir hızı ile temel fizik prensiplerini bilen kullanıcılar tarafından kolayca kullanılabilir [31,32].

4.3. OYUN MOTORLARI

Oyun motoru, oyun yapmak amacıyla kullanılan programlara verilen isimdir. Birçok araç oyun motoru olarak anılmaktadır. Aslında bu program, kütüphanelerden oluşmuş bir yapıdır. Genel olarak grafik, giriş, ağ ve diğer sistemleri kontrol eden çekirdek fonksiyonları kütüphaneleridir. Bu kütüphanelerin içinde daha önceden tanımlanmış fonksiyonlar, sınıflar vb. veriler bulunmaktadır. Oyun motorunun kullanım amacı daha önce tanımlanan verilerin tekrardan tanımlanma zahmetinden kurtarmasıdır. Bu durum yazılımın daha az karmaşık olmasını sağlar. Oyun tasarımcıları kolay ve hızlı oyun yazabilmek için bu araçlardan faydalanırlar. [33].

4.3.1. Delta3D

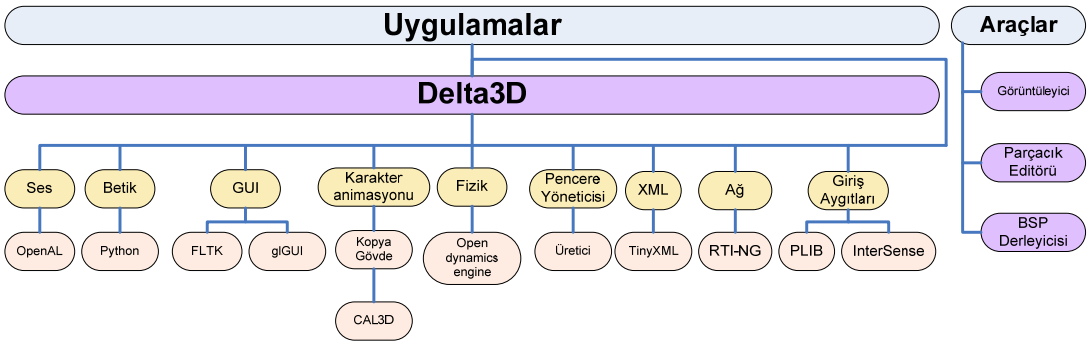
Açık kaynak kodlu oyun motoru, Monterey şehrindeki Amerika Deniz Kuvvetleri lisansüstü eğitim kurumu MOVES Enstitüsünde oluşturulmuştur. Delta3D oyun motoru inşa edilirken motora eklenecek işlevler ve özelliklerin, ne ve nasıl olduğu göz önünde bulundurulmuştur. Diğer açık kaynak kütüphaneler gibi LGPL lisansı ile

lisanslanan motorun grafiksel işleri için OpenSceneGraph, fiziksel modellemeleri ve ilişkileri için Open Dynamics Engine tercih edilmiştir. Projenin teknik yararı ve kullanıcı destek temeli, değer kriteri olarak düşünülmüştür. İlk olarak motive olunan husus Delta3D' nin modülerliğinin uzun ömürlü olmasıdır. Bu sebeple açık kaynak olmasının verdiği esneklik, kullanıcıların yazılımı tercih etmesine ve özgürce değiştirerek uygulamalar geliştirmesine imkan vermektedir [34].

Askeri bir okul tarafından geliştirilen uygulamanın amaçlarından önemli bir tanesi de kurumun fatura ödeme zorunluluğundan kurtarılmasıdır. Bu durum, yazılımcıları açık kaynak çözümlere yöneltmiştir. Açık kaynak projelerin gücü, çok sayıda geliştiricinin dev bir takım gibi sorunların üstesinden gelmek için var güçleriyle çalışmalarında yatar. Yapının iyi tasarlanması, insanların ilgisini çeker ve kendi uygulamalarında kullanmalarını, geliştirdiklerini orijinal sisteme eklemelerini sağlar. Belli bir zaman sonra bu eklentiler orijinal sistemden daha kıymetli olur. Ancak topluluk oluşturmak özveri ve zaman ister. Bu nedenle mevcut olan açık kaynak topluluklardan faydalanarak, onların yararına ve projenin temelleri göz önünde bulundurularak, dolaylı olarak Delta3D'nin bileşenlerini geliştiren açık kaynak kullanıcılarını sistemin altyapısında kullanmak mantıklı bir tercihtir. Buna ek olarak, büyük kullanıcı kitlesi olan projeler, güncel ve modern bir çizgide oldukları için küçük projelerdeki gibi bir modülü değiş tokuş etme ihtiyacı duyulma olasılığını azaltmaktadır. Bu durum, bir oyun motorunun hazırlanması için gereken bileşenlerin oluşturulması için toplam iş yükünü azaltırken, güçlü bir oyun motoru inşa edilmesini izin vermiştir [34].

Motora nelerin ekleneceği kadar, sadece uygulamalar için gereken özelliklerin Delta3D' ye eklenmesi üzerinde de aşırı derecede durulmuştur. Motorun kullanımının artması için ihtiyaç duyulan ilave fonksiyonlar ve gereken tüm modüller Delta3D' ye eklenmiştir [34].

Askeri amaçlı oyun temelli simülasyonlarda kullanılması düşüncesi ile tasarlanan Delta3D bu altyapıyı kullanan birçok oyun uygulamasında kullanılabilir. Delta3D'nin yapısı Şekil 4.1'de görülmektedir [34,35].



Şekil 4.1. Delta3D'nin yapısı ve bileşenleri [35].

OpenSceneGraph sahne grafiği yazılımı, grafik kütüphanesi olarak ve Open Dynamics Engine fizik kütüphanesi olarak Delta3D tarafından kullanılmaktadır. Bunun yanı sıra sesler için OpenAL, karakter animasyonları için de Cal3D yazılımları kullanılmaktadır. Şekilden 4.1' den de anlaşıldığı üzere Delta3D farklı işler için kullanılan yazılımları bir çatı altında toplayarak entegrasyonu sağlamaktadır. Bu, aynı yazılımların kullanıldığı projelerde entegrasyon için harcanan emeği ve zamanı kısaltır. Sonuç olarak geliştiricilerin hızlı uygulama geliştirmesi için uygulama programlama ara yüzü sunulmuştur.

4.3.1.1. Delta3D Özellikleri

Delta3D'nin özellikleri aşağıdaki şekilde sınıflandırılarak sayılabilir [36]:

dtCore

Aşağıdaki özelliklerin tamamını kapsar:

1. Giriş aygıtları (klavye, fare, joystick, trackers)
2. Hareketli modeller (uçma, UFO, yürüme, orbit, ilk insan)
3. Çevre görüntüleme (rendering) (bulutlar, sis, gökyüzü kutuları (skyboxes), günlük vakitler)
4. Parçacık sistem efektleri (duman, patlama, özel)
5. Arazi görüntüleme (yordamsal arazi, yükseklik haritası bazlı arazi)
6. Dosya yükleme

- .3dc, .3ds, .ac, .dw, .flt, .geo, .ive, .logo, .lwo, .lws, .md2, .obj, .osg, .tgz, .x, .zip
- .bmp, .dds, .gif, .jpg, .pic, .png, .pnm, .rgb, .tga, .tiff, .txp
- .wav

7. Kamera kontrolleri (alana bakış, üç ayak(tripod))
8. Çoklu kamera desteği
9. Çoklu pencere desteği
10. Fizik kuralları (katı gövde, çarpışma tespit etme, otomatik şekil sınırlandırma)
11. OpenGL ışıklandırma
12. Bezier yol düğümleri
13. Tam OpenGL 2.0 desteği
14. GLSL köşe ve parça gölgelendirme (GL Shading Language)

dtChar

1. Animasyon harmanlama (blending)
2. Zemin izleme
3. Avatar ayaklarını dünyaya göre konumlandırma

dtABC

Yüksek seviye, uygulama tabanlı bileşenler. Bazı uygulamaları geliştirmek için kullanışlıdır.

1. Uygulama şablonu
2. Hava arayüzü (görünürlük, bulut kaplama)
3. Hızlı ışık araç kiti ile pencere entegrasyonu
4. Diğer pencere araç kitleri ile kolay entegrasyon için buton vb unsurlar

dtHLA

Yüksek seviye mimari ağı için arayüz. Kullanıcı kontrollü gerçek zamanlı güncellemeler ile entegre dahili bileşenler.

1. Koordinat sistemi dönüşümü
2. Patlama araüzü
3. Yüsek seviye mimari varlık arayüzü

dtTerrain

Arazi yükleme, görüntüleme ve dekorasyon uygulamaları için çatı (framework) sağlar. Görüntüleyici, okuyucu ve dekoratörler için modüler mimari.

1. Sayfalanmış arazi kaplama, görüntü işlemleri ve gürültü üretimi
2. DTED okuyucu
3. Detayların devamlı seviyeleri ile SOARX görüntüleyici
4. Yordamsal bitki yerleştirme – arazi kaplaması sınıflama desteği (LCC) Görüntü üstü dekoratör – geoTIFF desteği

dtDAL

Dinamik aktör katmanı oluşturma, erişim ve aktör hareketleri için genel bir altyapı sağlar.

1. Aktör vekili (proxy) ve aktör özellikleri mimarisi
2. Kütüphane yönetimi (Aktör kayıtları)
3. Proje ve harita yükleme (XML)
4. Aktör özellik tipleri: Ses, arazi, karakter, kaplama, statik ağ, parçacık sistemi, sıralama, aktör, renk, vec2/3/4, string, int, double ve daha fazlası
5. Temel motor aktör vekilleri

dtGUI

Çılgın Edi'nin grafiksel arabirimi ile direkt arayüz,

1. Kullanıcı arabirimi çizebilen ve görüntüleyici
2. Açılır kabuk arabirimi
3. Varsayılan kullanıcı grafik arabirim resimleri

dtGame

Karmaşık oyun ve eğitim uygulamaları yapmak için komple bir mimari sağlar. Oyun yöneticisi, aktör ve bileşenler arasında yerel ya da istemci-sunucu iletişimi için yüksek seviye bir altyapıdır.

1. Oyun aktörü ve vekil nesnesi
2. Eklenti desteği ile oyun yöneticisi bileşen mimarisi – varsayılan mesaj işlemcisi, kural bileşeni ve raporlama bileşeni
3. Mesajlaşma altyapısı – mesaj ve parametre fabrikaları
4. Ağ tanılama – tek istemci, istemci/sunucu ve kayıttan oynatma desteği
5. Mesajları gönderme, alma ve kaydetme için mesaj yayınlama
6. Temel istemci ve sunucu oyun yöneticileri

dtUtil

Delta3d ile kullanılan temel nesneler.

1. Dosya loglama ve Xerces XML yardımcı programları
2. Kaplama, gürültü, matris ve string yardımcı programları
3. Kütüphane yönetimi
4. Güvenli sıralama oluşturma

dtBSP

Yüklenen nesneleri otomatik olarak optimize edilmiş BSP formatına çevirme.

1. İç sahnelerle en iyi çalışma
2. .osg ve . ive dışa aktarım

dtAudio

Ses oynatımı için yüksek seviye fonksiyonellik sağlar

1. 2B/3B sesler
2. Tam ses kontrolü (kazanç, eğim, pozisyon, yürütme, durdurma, geri sarma vs.)

3. Doppler efekti
4. Ses donanımını etkili kullanma

dtNet

Çok kullanıcı ağı ile kullanmak için yüksek seviye uygulama programlama arabirimi sunar.

1. Sunucu/istemci mimarisi
2. Güvenli/güvensiz paket iletimi

dtPython

Python betik arayüzüne, uygulama geliştirme arayüzü erişimi sağlar.

1. Tam uygulama geliştirme arayüzü erişimi
2. Python ve Boost için ek paket gereksinimleri vardır.

STAGE – Simulation, Training, and Game Editor

STAGE 3B harita inşa etmek için tam görsel bir editördür.

1. Harita düzenleme – proje kaynakları ve kütüphane yönetimi
2. Görsel olarak aktör oluşturma ve hareket ettirme
3. Özellik editörü ve ileri ve geri al ile aktör sıralama
4. Bölünmüş ekran görünümleri – Perspektif, üst, yan ve ön görünümler
5. Kafes, ışıklı ve kaplama modları
6. Tekrar düzenlenebilir kullanıcı arayüzü
7. Aktör arama ve evrensel aktör yönetimi
8. Kaplama, ağ, ses ve parça tarayıcılar

Tests

Delta3D başlangıç için ünite testleri ve kapsamlı örnekler sunar.

1. 29 örnek test uygulaması
2. 8000 satırdan fazla ünite testleri

Cross Platform

Windows, Linux ve Mac OS X'i destekler.

1. MS Visual Studio 2003/2005/2008 çözüm ve projeleri
2. Diğer platformlar için komple SCons betikleri

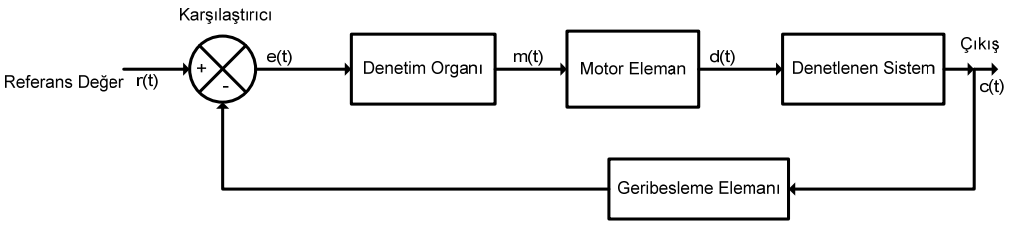
Utilities

1. Grafiksel parçacık efekt editörü – Parçacık efektlerini düzenleme ekranı ve kullanışlı grafiksel kullanıcı arabirimi sağlar.
2. 3B model görüntüleyici – Kullanımı kolay, 3B modellere ön bakış için özel görüntüleyici
3. BSP derleyici – Yükler, derler ve genel nesneleri optimize edilmiş BSP dosyası olarak kaydeder.
4. Gizli yüsek seviye mimari grafiksel görüntüleyici – Yüksek seviye mimari ağındaki tüm öğelerin temel 3B gösterimi.

BÖLÜM 5

DENETİM SİSTEMLERİ

Bir denetim sistemi, bir takım elemanların ilişkisel olarak belli bir düzene göre bağlanması ile meydana gelmektedir. Bu sistem elemanları arasındaki bağlantı, girişleri ve çıkışları yoluyla gerçekleştirilmektedir. Sistem elemanlarının işlevleri, bireysel giriş - çıkışları ve sistem elemanları arasındaki bilgi akışı, blok şemalar ile gösterilmektedir. Bu şemalar, sistem elemanlarının bağlantı gerekçelerini ve sistemin yapısının incelenmesini sağlamaktadır [37]. Şekil 5.1'de kapalı – döngü denetim sisteminin blok şeması gösterilmektedir.



Şekil 5.1. Kapalı-döngü denetim sistemi.

Konu ile ilgili terimlerin kısaca tanımı şöyledir:

Sistem: Bir bütün oluşturacak şekilde ilişkisel olarak belli bir düzene göre bağlı elemanlardan oluşan yapıdır [37].

Denetim: Değişken bir niceliğin ya da nicelikler kümesinin önceden tanımlanmış koşullara uygunluğunu sağlamak amacıyla gerçekleştirilen işlemler bütünüdür [37].

Denetim Sistemi: Kendisini ya da başka bir sistemi kumanda etmek, yönlendirmek ya da ayarlamak üzere oluşturulan yapıdır [37].

5.1. TEMEL DENETİM YÖNTEMLERİ VE DENETİM ORGANLARI

Kapalı döngü bir denetim organının görevi ölçme elemanı üzerinden geri beslenen çıkış büyüklüğünü referans giriş büyüklüğü ile karşılaştırmak ve karşılaştırma sonucunda ortaya çıkabilecek hata değerine ve kendi denetim etkisine bağlı olarak uygun bir kumanda denetim sinyali üretmektir. Denetim organlarında kullanılan belli başlı dört temel denetim etkisi vardır [37-39]. Bunlar:

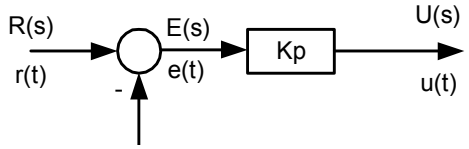
1. İkili veya aç kapa denetim etkisi
2. Oransal denetim etkisi (P etki)
3. İntegral denetim etkisi (I etki)
4. Türev denetim etkisi (D etki)

Bu temel denetim etkilerinin bir veya birkaçının bir arada uygun şekilde kullanılmasıyla değişik denetim etkilerinde çalışan denetim organları oluşturulur [37,38]. Bunlar:

1. Oransal denetim organı (P tipi)
2. Oransal integral tipi denetim organı (PI tipi)
3. Oransal türev tipi denetim organı (PD tipi)
4. Oransal integral türev tipi denetim organı (PID tipi)

5.1.1. Oransal Denetim Organı

Orantı etkide herhangi bir anda çıkış işareti u(t) hatanın büyüklüğüne bağlıdır ve o anda hata e(t) ne kadar büyük olursa denetim sinyali u(t) o oranda büyük olur. Hata çok küçük olduğunda ise denetim organı yeteri kadar etkili düzeltici sinyal üretmez. Bu nedenle orantı etki ile çalışan sistemler kalıcı durum hatası verirler. Kazanç katsayısı K_p yi artırmak sureti ile kalıcı durum hatasını azaltmak mümkündür. P denetim organının blok diyagramı Şekil 5.2'de verilmektedir [37-39].



Şekil 5.2. P tipi denetim organının blok diyagramı.

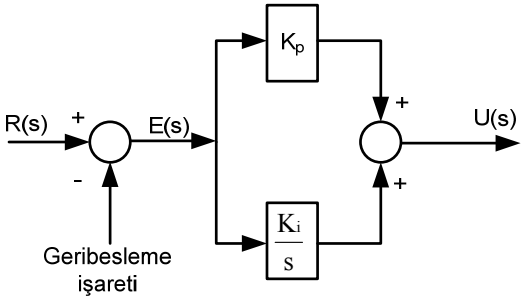
P tipi denetim organın s ve t düzleminde çıkış fonksiyonları Eş. 5.1 ve Eş. 5.2 ile hesaplanmaktadır.

t- düzleminde : $u(t) = K_p \text{ x } e(t)$ (5.1)

s- düzleminde : $u(s) = K_p \text{ x } E(s)$ (5.2)

5.1.2. Oransal İntegral Denetim Organı

Bu denetim organı oransal ve integral denetim etkilerinin birleştirilmesinden meydana gelmektedir. Orantı etkiye, integral etki ilavesi orantı etkinin tek başına kullanılması halinde sistemde ortaya çıkan kalıcı durum hatasını ortadan kaldırır. İntegrasyon işlemi kalıcı durum hatasını ortadan kaldırmakla beraber aynı bağıl kararlılık koşullarında sistemin cevap hızını azaltır. İntegral etki kazancını artırmak sureti ile cevap hızı artmakla beraber kazanç değerinin çok fazla artırılması sistemi kararsızlığa sürükleyebilmektedir [37,39]. PI denetim organının blok diyagramı Şekil 5.3'de verilmektedir.



Şekil 5.3. PI denetim organın blok diyagramı.

$$U(t) = K_p \text{ x } e(t) + K_i \int_0^t e(t) \text{ x } dt \tag{5.4}$$

$$\frac{u(s)}{E(s)} = \left(K_p + \left(\frac{K_i}{s} \right) \right) = \left(\frac{sK_p + K_i}{s} \right) = K_p \left(1 + \left(\frac{K_i}{sK_p} \right) \right) \tag{5.5}$$

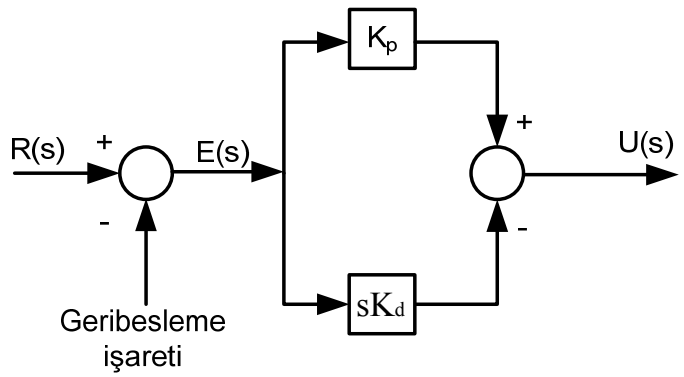
$T_i = \dfrac{K_p}{K_i}$ integral zamanıdır.

PI denetim organı yapısı basit olup özellikle süreç denetim sistemlerinde sıklıkla kullanılmaktadır.

5.1.3. Oransal Türev Denetim Organı

PD tipi denetim organın transfer fonksiyonu Eş. 5.7 kullanılarak bulunabilir. Burada K_d/K_p türev etki zamanıdır. PD denetim organının blok diyagramı Şekil 5.4'de verilmektedir.

$$\frac{U(s)}{E(s)} = K_p + sK_d = K_p \left(1 + \frac{sK_d}{K_p} \right) \tag{5.7}$$



Şekil 5.4. PD tipi denetimin blok diyagramı gösterimi.

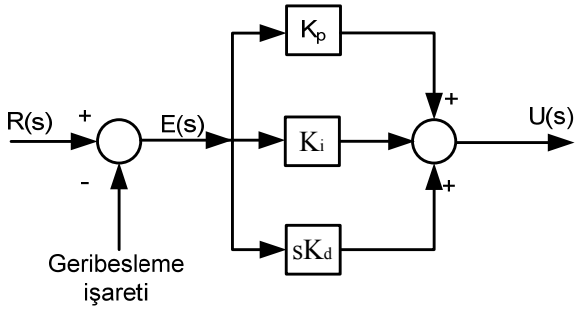
5.1.4. Oransal İntegral Türev Denetim Organı

PID denetimde K_p, K_d, K_i, T_d, T_i ayarlanabilir sabitlerdir. PID tipi denetim organın transfer fonksiyonu Eş. (5.9) ile hesaplanmaktadır.

$$\frac{E_o(s)}{E_i(s)} = \left(K_p + \frac{K_i}{s} + sK_d \right) = K_p \left(1 + \frac{K_i}{sK_p} + \frac{sK_d}{K_p} \right) \tag{5.9}$$

K_p arttığında sistem hızı artar (aşmaya doğru eğilimle) ve sabit durum hatası azalır, ancak yok edilemez. K_d arttığında, sönme faktörü, aşmayı azaltarak artar. Türev denetimi gürültüye hassastır ve asla yalnız kullanılmaz.

K_i arttığında sabit durum hatası sıfıra yaklaşır ve sistem dengesiz olma eğilimine girer. İntegral kontrolde hiçbir zaman tek başına kullanılmaz [40]. PID denetim organının blok diyagramı Şekil 5.5'de verilmektedir.



Şekil 5.5. PID tipi denetim organının blok diyagramı gösterimi.

5.2. YENİ DENETİM TEKNİKLERİ VE ORGANLARI

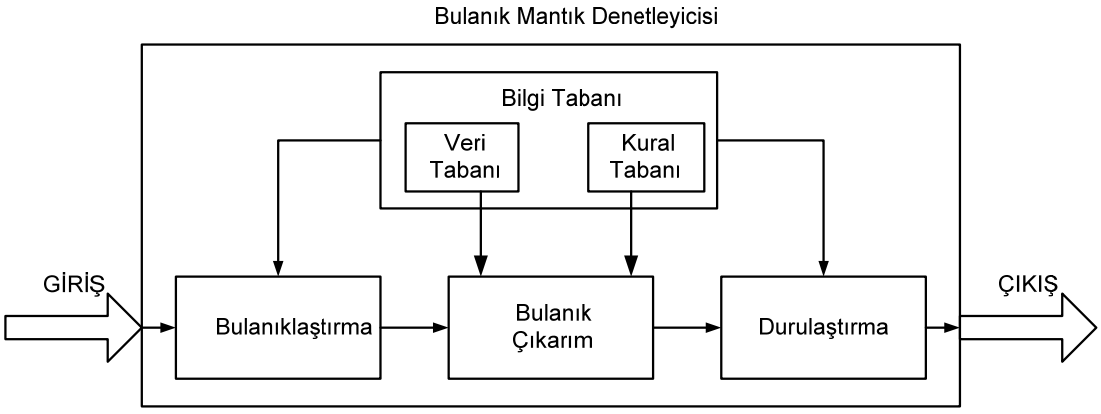
5.2.1. Bulanık Denetim

Bulanık mantığın ilk uygulaması, Mamdani' tarafından 1974 yılında bir buhar makinesinin bulanık denetiminin gerçekleştirilmesidir. 1980 yılında bir Hollandalı şirketinde, çimento fırınlarının denetiminde bulanık mantık denetimi uygulanmıştır. Üç yıl sonra Fuji elektrik şirketi, su arıtma alanları için kimyasal püskürtme aleti

üzerinde çalışmalar yapmıştır. 1987'de ikinci IFSA kongresinde ilk bulanık mantık denetleyicisi sergilenmiştir. Bu denetimler 1984 yılında araştırmalara başlayan Omron şirketinin 700'den fazla uygulamasını içermektedir. 1987 yılında ise Hitachi takımının tasarladığı Japon Sendai metrosu denetleyicisi çalışmaya başlamıştır. Bu bulanık mantık denetim; metro da daha rahat bir seyahat, düzgün bir yavaşlama ve hızlanma sağlamıştır. 1989 yılında Omron şirketi Japonya'nın Harumi şehrinde bulunan çalışma merkezinde yapmış olduğu depolama, tekrar etme ve bulanık sonuçlarını elde etmek için kullanılan bilgisayara dayalı olan çalışmaları tanıtmıştır [40,41].

Bulanık mantık, elektrikli ev aletlerinden oto elektriğine, gündelik kullandığımız iş makinelerinden, üretim mühendisliğine, endüstriyel teknolojilerden, otomasyona kadar aklımıza gelebilecek her yerde kendisine uygulama alanı bulmuştur.

Bulanık mantık, motorlarda hataların bulunması ve teşhis edilmesinde de başarıyla kullanılmaktadır. Bulanık mantık, uzman kişinin bilgisinin, dilsel ifadeler haline getirilerek kullanılmasını sağlar. Bu özellik bulanık mantığa büyük bir esneklik sağlar. Bulanık mantığın tek dezavantajı kesin sonuç vermemesidir. Bu bulanık mantığın doğal yapısından kaynaklanmaktadır [40,41]. Bulanık mantık denetleyicisi blok şeması Şekil 5.6'da verilmektedir.



Şekil 5.6. Bulanık mantık denetleyicisi blok şeması.

Bulanık mantık denetleyicisinin üstünlükleri;

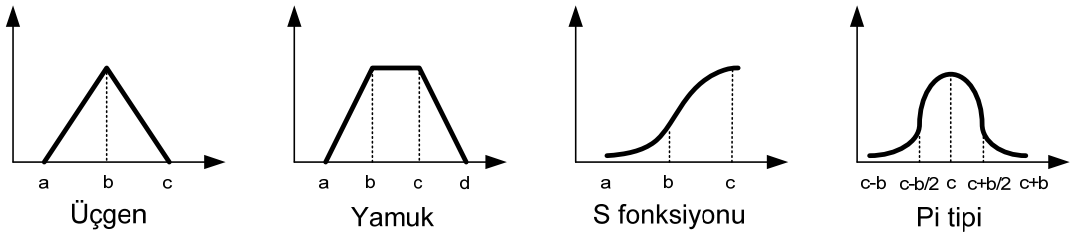
1. Giriş ile çıkış arasında kolayca anlaşılan bağlantı kurar.
2. Hızlı modellemeye izin verir, çünkü sistemi tasarlayanın kurulmadan önce sistem hakkında her şeyi bilmesine gerek yoktur.
3. Ucuzdur, çünkü tasarımı kolaydır.
4. Sistemin sağlamlılığını artırır.
5. Kazanılmış ve temsil edilen bilgiyi kolaylaştırır.
6. Bazı kurallar çok büyük zorluklara yol gösterir.
7. Daha az osilasyon ve tepkime ile sonuçlanır.
8. Daha kısa zamanda sabit durumda başarıyla bitirilebilir.

Bulanık mantık denetleyicisinin sınırlıkları;

1. Bir bulanık sistemden model geliştirmek zordur.
2. Modellemeden önce ince ayar benzetimi gerektirir.
3. Daha öncede söz edildiği gibi bulanık kelimesi (en azından batı dünyası için) kötü bir çağrışım yapmaktadır. Mühendisler ve insanların çoğu kesinliğe (crispness) alışkındırlar ve bulanık kontrolden ve bulanık karar verme işlemlerinin yapılmasından çekinmektedirler [38]. Bulanık mantık, her gün kullandığımız ve davranışlarımızı yorumladığımız yapıya ulaşmamızı sağlayan matematiksel bir disiplindir. Temelini doğru ve yanlış değerlerin belirlendiği bulanık küme kuramı oluşturur. Burada yine geleneksel mantıkta olduğu gibi "1" ve "0" değerleri vardır. Ancak bulanık mantık, bu değerlerle yetinmeyip, bunların ara değerlerini de kullanır.

Üyelik fonksiyonları

Bulanık kümenin her elemanı, bu küme içerisinde bir üyelik değerine sahiptir ve bulanık bir kümenin değerleri 0 ile 1 arasındaki gerçek sayılardan oluşur. Bulanık mantık süreçlerinin başlatılabilmesi için gerekli üyelik fonksiyonları (membership function), dilsel niteleyicilerden oluşan bir anlam gurubudur. Bu anlam üyelik fonksiyonlarının ağırlık merkezleriyle tanımlanırlar [40,42].



Şekil 5.7. Üyelik fonksiyonlarının şekilleri.

Üyelik fonksiyonları genellikle biçimsel olarak gösterilirler ve en çok kullanılan biçimler; üçgen (triangular), yamuk (trapezoidal), çan eğrisi (bell – shaped) ve Pi tipi (p type)' dir. Şekil 5.7'de bu fonksiyonların şekilleri verilmiştir. Eş. 5.11 - Eş. 5.14'de bu fonksiyonlara ait formüller verilmektedir [41,43].

$$\text{üçgen}\,(x;a,b,c) = \max(\min(\frac{x-a}{b-a},\frac{c-x}{c-b}) \tag{5.11}$$

$$\text{yamuk}(x;a,b,c,d) = \max(\min(\frac{x-a}{b-a},1,\frac{d-x}{d-c})) \tag{5.12}$$

$$\text{çan}(x;a,b,c) = \begin{cases} 0 & x < a \\ 2\cdot\left[\dfrac{x-a}{c-a}\right]^2 & b \le x \le c \\ 1-2\cdot\left[\dfrac{x-b}{c-x}\right]^2 & x > c \end{cases} \tag{5.13}$$

$$\text{pi}\,(x;b,c) = \begin{cases} 2\cdot\left[\dfrac{x-(c-b)}{b}\right]^2 & c-b \le x < c-\dfrac{b}{2} \\ 1-2\cdot\left[\dfrac{c-x}{b}\right]^2 & c-\dfrac{b}{2} \le x < c \\ 1-2\cdot\left[\dfrac{x-c}{b}\right]^2 & c \le x < c+\dfrac{b}{2} \\ 2\cdot\left[\dfrac{(c+b)-x}{b}\right]^2 & c+\dfrac{b}{2} \le x \le c+b \end{cases} \tag{5.14}$$

Bulanıklaştırma

Fiziksel giriş bilgilerinin, dilsel niteleyicilerle ifade edebileceğimiz bulanık mantık bilgileri şekline çevirme işlemine bulanıklaştırma (fuzzification) adı verilmektedir. Bulanıklaştırma işlemi önemli ölçüde kesin olmayan bilgiyi de içine almakta ve bulanıklaştırmaktadır. Bulanıklaştırma sonucu elde edilen değişkenlere dilsel değişkenler (linguistic variables) denir. İşlemle birlikte tüm giriş değişkenlerinin değerleri, üyelik derecesi olarak atanmaktadır. Örneğin, 100 km' lik bir uzaklık giriş bilgisi, dilsel niteleyici olarak "uzak" olarak ifade edilebilmektedir. Bununla beraber yine 100-150 km arası, tam kesin olmayan bir bilgi olarak yine "uzak" olarak ifade edilebilmektedir [43].

Bulanıklaştırma işlemi göreceli olarak bu kadar kolay olmasına karşın, daha önce de değinildiği gibi uzman sistem kalıplarından dolayı bu işlemlerin yapılması büyük ölçüde deneyime dayanmaktadır. Operatörün sistemde çalışırken gösterdiği davranışlar, sistemin matematiksel modelinden daha önemlidir. Dolayısıyla bulanıklaştırma aşamasına gelinebilmesi için gerekli süre bazen çok uzun olabilmektedir. Bununla birlikte kesin olmayan bilgileri kullanabilmesi, sürecin matematiksel bir modeline gereksinim duyulmaması ve uygulamaya çabucak geçilebilmesi, bütün bunlardan sonra da yüksek derecede verim alınabilmesi bulanık mantığın önemini açıkça ortaya koymaktadır.

Dilsel değişkenler

Bulanık kümeler genellikle üç, beş ya da yedi üyelik fonksiyondan oluşabilmektedirler. Örneğin hız giriş değişkeni için; yavaş, az hızlı, orta hızlı, hızlı, çok hızlı şeklinde beş üyelik fonksiyonuna sahip bir bulanık kümesi oluşturulabilmektedir. Burada da olduğu gibi, tanımlar uzmanın söylemlerine göre geliştirilmektedir. Bunların fonksiyonel olarak elde edilmeleri ve uygulama aşamasına getirilmeleri büyük ölçüde sistemde daha önce elde edilmiş deneyimlere bağlı olmaktadır.

Bulanık çıkarım

Bulanık mantıkta da geleneksel mantıkta olduğu gibi bazı mantık işlemleri yer almaktadır. Ancak bu işlemin komutları AND, OR, NOT ve IF, THEN (VE, VEYA, DEĞİL, EĞER, ÖYLEYSE) ile sınırlı çok basit ve aynı zamanda da kullanışlı olmaktadır. Bu kurallar bütünü, kurallar (rules) ya da bulanık mantık denetleyicisi üzerinde kural tabanı (rule base) olarak adlandırılmaktadır [41,43].

Durulaştırma

Bulanık küme çıkarımlarının, sistem üzerinde uygulanabilmesi için yeniden fiziksel ve kesin sayılara dönüştürülmesi gerekmektedir. Bu işleme durulaştırma (defuzzification) adı verilmektedir. Çeşitli durumlara göre durulama yöntemleri geliştirilmiştir. Bulanık işlemciden elde edilen mantıksal çıkarımların üyelik işlevleri, bir ya da birden fazla olmaktadır. Bulanık çıkarım bu kümelerin bileşkesi olmaktadır. Bulanık işlemcilerden elde edilen bulanık çıkarımlar, aslında ikiden çoktur. Üyelik işlevlerinin biçimleri farklı şekillerde de olabilmektedir [41].

Literatürde en çok kullanılan sekiz çeşit durulama yöntemi şunlardır:

1. Üyelik fonksiyonunun en yüksek noktası (Max-Membership Principle)
2. Üyelik fonksiyonunun en düşük noktası (Min-Membership Principle)
3. Merkez (Centroid Method)
4. Ağırlık ortalama yöntemi (Weighted Avarage Method)
5. Üyelik işlevinin en yüksek noktalarının ortalaması (Mean-Max Membership)
6. Toplamların ortası (Center of Sums)
7. Geniş alan merkezi (Center of Largest Area)
8. İlk veya son yükselti (First or Last of Maxima)

Bunlardan sisteme en uygun olanı seçilmelidir. Hellendorn ve Thomas, 1993 yılında uygun olanın seçilmesi için beş dayanak ortaya atmıştır. Bunlar; süreklilik (continuity), belirsiz olmama (disambiguity), uygunluk (plausibility), hesapsal

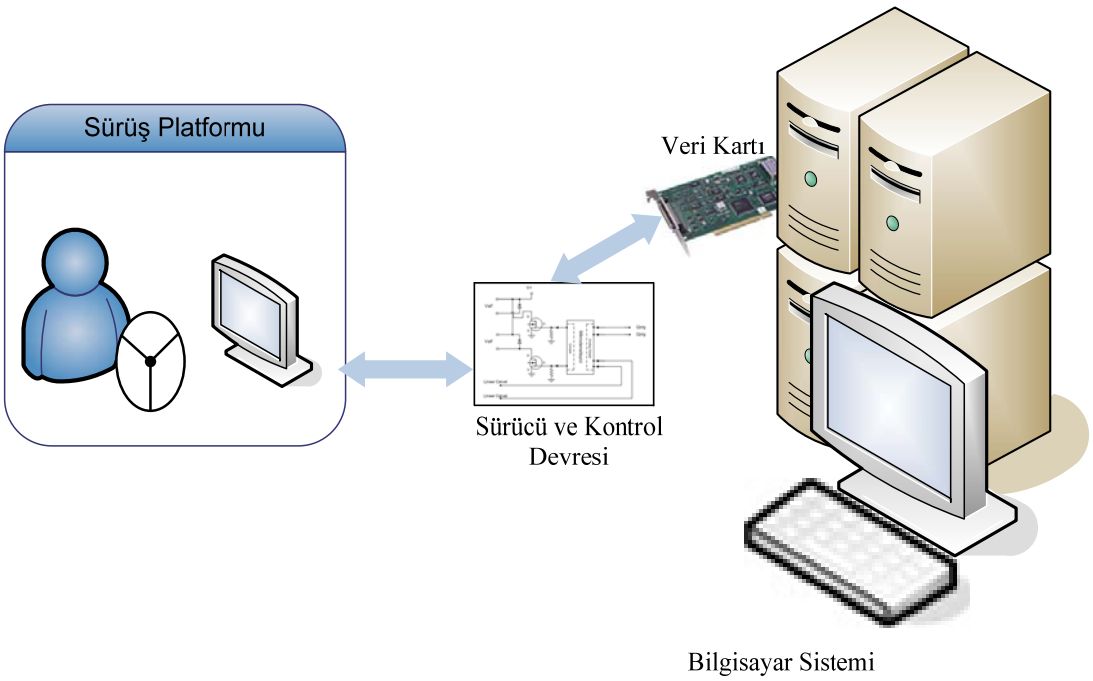
kolaylık (computational simplicitiy) ve ağırlaştırma yöntemidir (weighting method). Ayrıca fiziksel sistemin yapısı ve kullanıcıların deneyimleri de durulama yönteminin seçilmesi için birer büyük dayanak noktası oluşturmaktadır.

BÖLÜM 6

SİMÜLATÖRÜN GELİŞTİRİLMESİ

6.1. GENEL SÜRÜŞ SİMÜLATÖRÜ YAPISI

Bu çalışmada dört tahrik noktalı, beş serbestlik dereceli, açık kaynak kütüphaneler kullanılarak bir sürüş simülatörü tasarlanmıştır. Tasarlanan araç simülatörünün genel yapısı Şekil 6.1'de görülmektedir.



Şekil 6.1. Sürüş simülatörü genel yapısı.

Kullanıcı, sadece kendisine sunulan sürüş platformu üzerinden simülasyonda izin verilen ve kendi kontrolündeki parametreleri değiştirebilir. Arka planda çalışan simülasyon yazılımı kullanıcının yapması gerekenleri, işlem sırasını ve diğer parametreleri kontrol eder ve isteğe bağlı olarak kullanıcıya uyarı verebilir.

Bilgisayarın simülasyon yazılımını gerçek zamanlı çalıştırabilmesi ve kullanıcının gerçek hayattaymış gibi tecrübeler kazanabilmesi için işlemlerin kabul edilebilir süreler içerisinde yapılması ve kullanıcıya iletilmesi gerekmektedir. Matematiksel denklem eşitlikleri ile modellenen fizik kurallarının hızlı bir şekilde çözülmesi için paralel işlemler yapabilen çok çekirdekli mimariye sahip bilgisayar kullanılması uygundur. Grafiksel işlemlerin hızlı olması için kullanılan grafik motorunun ihtiyaçlarına cevap verebilen hızlı ekran kartları kullanılmalıdır. Bilgisayarın diğer bileşenlerinin de hızlı çalışmayı sağlayacak konfigürasyonda olması gerekmektedir. Bunun yanı sıra veri, sürücü ve kontrol kartları temel bilgisayar bileşenlerinin haricinde kullanılan giriş ve çıkışlar için kullanılır. Veri, sürücü ve kontrol kartları gibi bileşenler bütün simülatör sistemlerinde bulunmamakla birlikte hassasiyetin önemli olduğu, çok sayıda girişe ve çıkışa ihtiyaç duyulan uygulamalarda kullanılırlar.

Bilgisayarın giriş ve çıkışları harici aygıtlarla her zaman direkt olarak bağlanmazlar. Giriş ve çıkışların elektriksel olarak belli özellikleri vardır. Eğer bağlanacak aygıt bu özelliklere uygun değilse, sürücü ya da uygunlaştırıcı kartların kullanılması PCI kartların güvenliği açısından önemli bir gerekliliktir. Sürücü devresi mekanik olarak hareket eden platformun kontrolünü sağlamaktadır ve bilgisayardan alınan elektriksel sinyalleri, mekanik sistemi hareket ettirecek kadar güçlendirmektedir. Aynı zamanda istenilen yükseklik ile sistemin yüksekliğini kontrol ederek kontrolü sağlamaktadır.

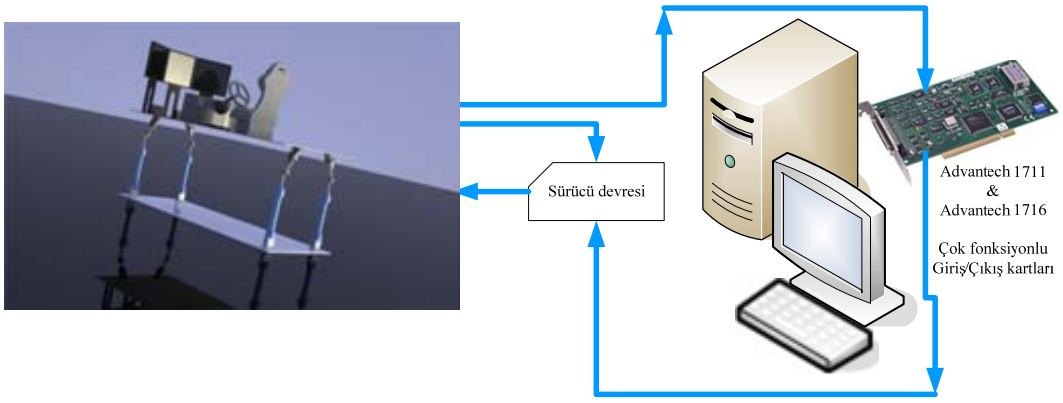
6.2. YAPILAN SÜRÜŞ SİMÜLATÖRÜNÜN YAPISI

Çalışmada yapılan simülatör, dört tahrik noktalı beş serbestlik derecesine sahip araç simülatörüdür. Çalışmada simülatör yazılımı olarak Delta3D 2.4 versiyonu kullanılmıştır. Mekanik sistemin kontrolü için Advantech 1711 ve 1716 PCI veri kartları ile sisteme özel olarak tasarlanan sürücü devreleri kullanılmıştır.

6.2.1. Fiziksel Konfigürasyon

Çalışmada dört tahrik noktalı, beş serbestlik derecesine sahip platformun sistemle eş zamanlı kontrolü planlanmıştır. Sistemin mekanik kontrolünde, geri besleme

sağlamak amacıyla lineer cetveller tercih edilmiştir. Piston yükseklikleri haritadan okunan yükseklik bilgisini simüle ederken, kendi süspansiyon sistemine sahip platform arazi üzerindeki aracın pozisyonunu canlandırmaktadır. Şekil 6.2. simülatörü temsili olarak göstermektedir.



Şekil 6.2. Tasarlanan simülatörün temsili gösterimi.

Çalışmada tek grafik kartı ve tek monitör kullanılmıştır, ancak birden çok ekrana görüntü çoğaltılabilmektedir. Monitör sayısının arttırılması, görüş alanını arttırarak, daha çok gerçeklik hissi sağlamaya yardımcı olacaktır. Bunun yanında, arttırılacak grafik kartı sayısı daha gerçekçi grafiklerin kullanılmasına olanak sağlarken, hız problemini azaltmaya yardımcı olacaktır.

Kabindeki hoparlörler ile simülasyonun farklı bir boyuttaki çıktısı olan ses kullanıcıya verilmektedir. Direksiyon, fren/gaz pedalları vb giriş aygıtları sisteme kolayca entegre edilebilmektedir. Advantech veri kartları gerek dijital, gerekse analog giriş/çıkış sayısı için sayılan özelliklere fazlasıyla yeterlidir. Kartlara ait özellikler aşağıda verilmiştir.

Advantech 1711 veri kartının kullanılan ya da kullanılabilecek temel özellikleri [44]:

1. 16 tek uçlu analog giriş
2. 12-bit A/D dönüştürücü, 100 kHz' e kadar örnekleme oranı
3. Programlanabilir kazanç
4. Otomatik kanal/kazanç tarama

5. 1024 örnekleme için kart üstü FIFO hafıza
6. İki tane 12-bit analog çıkış
7. 16 dijital giriş ve 16 dijital çıkış
8. Kart üstü programlanabilir sayıcı

Advantech 1716 veri kartının kullanılan ya da kullanılabilecek temel özellikleri [44]:

1. 16 tek uç veya 8 diferansiyel ya da analog girişlerin kombinasyonu
2. 16-bit A/D dönüştürücü, 250 kHz' e kadar örnekleme oranı
3. 1024 örnekleme için kart üstü FIFO hafıza
4. Otomatik kalibrasyon
5. 2 analog çıkış
6. 16 dijital giriş ve 16 dijital çıkış
7. Kart üstü programlanabilir sayıcı

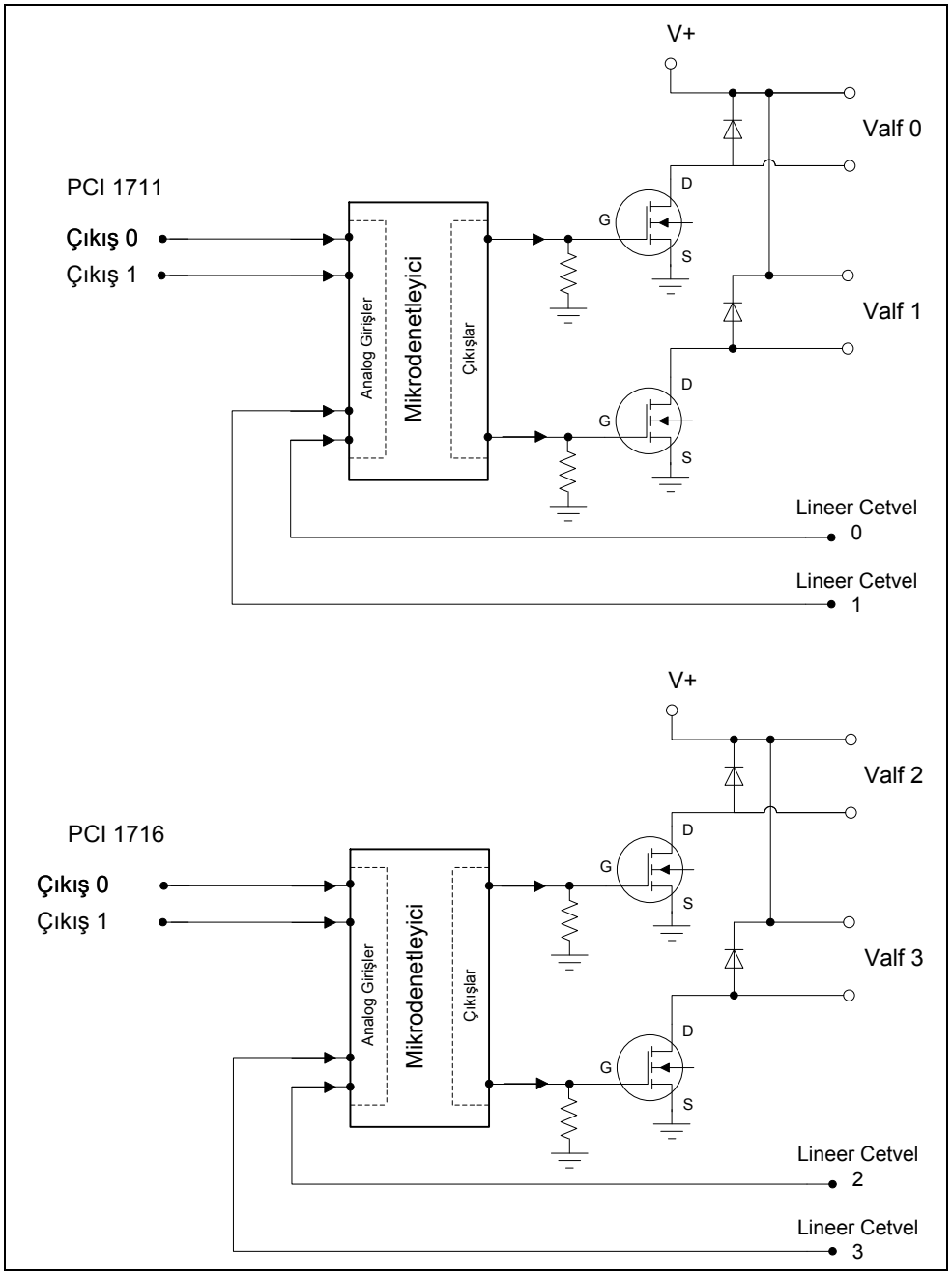
6.2.2. Sürücü Devresi

PCI kartların çıkışlarının, platform hareketlerini kontrol edebilecek yapıda olmaması sebebiyle sürücü devresi tasarlanmıştır. Sürücü devresi, PCI kartların çıkışından aldığı analog gerilim değerine göre valfler için gerekli elektriksel çıkışı sağlar. Ayrıca mikrodenetleyicinin analog girişlerine bağlı olan lineer cetvellerden, geri besleme bilgisi alınarak sisteme denetim yöntemi uygulanmaktadır. Denetim yönteminin sürücü devresinde yapılması, sistemin hızlı, kararlı ve daha basit olmasını sağlamaktadır. Tasarlanan sürücü devresine ait şema Şekil 6.3'de görülmektedir. Sürücü devresi, her PCI kart için bir mikrodenetleyici olacak şekilde tasarlanmıştır. Bunun nedeni, hem mikrodenetleyicinin tüm analog girişlerinin PCI çıkış ve geri besleme ile kullanılmasından, hem de işlemlerin bölünerek daha hızlı yapılabilmesindendir. Sistemin kontrolü için PID ve bulanık mantık denetim yöntemleri kullanılabilmektedir.

PID kontrol için kod taslağı şu şekildedir [45]:

```
onceki_hata=0
integral = 0
basla:
```

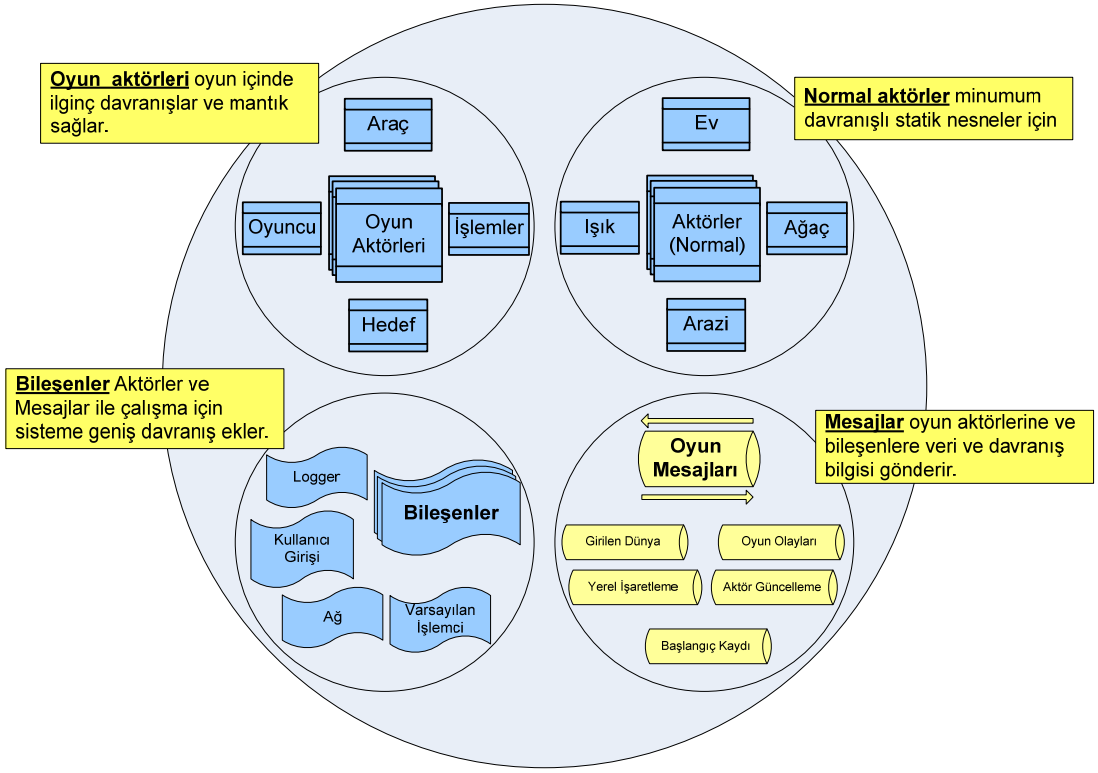
```
hata = referans – guncel_cikis
integral = integral + (hata*dt)
turev = (hata – onceki_hata)/dt
cikis = (Kp*hata) + (Ki*integral) + (Kd*turev)
onceki_hata = hata
bekle(dt)
basla'ya git
```



Şekil 6.3. Sürücü devresi şeması.

6.2.3. Yazılım Mimarisi

Sistemin yazılım mimarisi açık kaynak kodlu Delta3D oyun motoru kullanılarak oluşturulmuştur. Delta3D'nin seçilme sebebi: aynı ortamda, temel öğeler ile esnek programlama imkanı sunmasıdır. Sistemde 2.4 versiyonu kullanılan oyun motorunun en güncel versiyonu dosya deposundan indirilebilmektedir. Delta3D'ye ek olarak Advantech tarafından sağlanan .dll kütüphaneler kontrol işlemleri için yazılıma eklenmiş ve kullanıcı tanımlı fonksiyonlarla yazılıma entegre edilmiştir. Delta3D'nin oyun yöneticisinin yapısı Şekil 6.4'de gösterilmiştir.



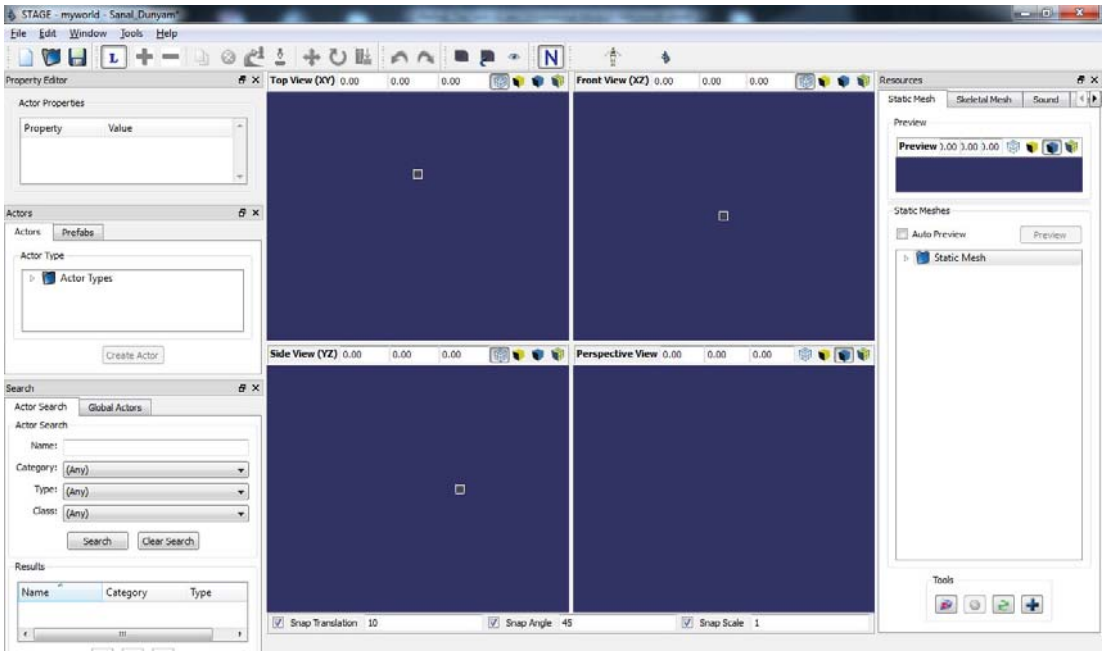
Şekil 6.4. Delta3D oyun yöneticisi [46].

Delta3D ile geliştirilen simülasyonun çekirdeği oyun yöneticisidir. Oyun yöneticisinin kendi aktörleri, bileşenleri ve mesajları vardır. Araç, hareketli insan, dinamik arazi gibi aktif nesneler ile ağaç, ev gibi hareketsiz pasif nesneler, ışıklandırma, sesler vb simülasyon yazılımına gönderilir. Bileşenler, oyun yöneticisi belirli bir aktöre mesaj göndermesi ihtimaline karşılık gelecek olan mesaja açık olarak beklerler. Burada, mesajlar kullanıcı tanımlı verilerdir ve aktörlerle diğer

bileşenler arasındaki iletişim için kullanılır. Aracın sanal dünyadaki durumu, Delta3D ve bileşenleri sayesinde gerçekleştirilir [46]. Delta3D ile görsel ve işitsel çıktılar alınırken platform için PCI kartlar ile gerçek zamanlı çıkışlar sağlanır.

6.2.3.1. Delta3D ile Sanal Dünya Oluşturma ve Örnek Uygulama Geliştirme

Delta3D, dünya oluşturmak için bir editör sunmaktadır. Şekil 6.5'de editör görülmektedir.



Şekil 6.5. Delta3D Editörü.

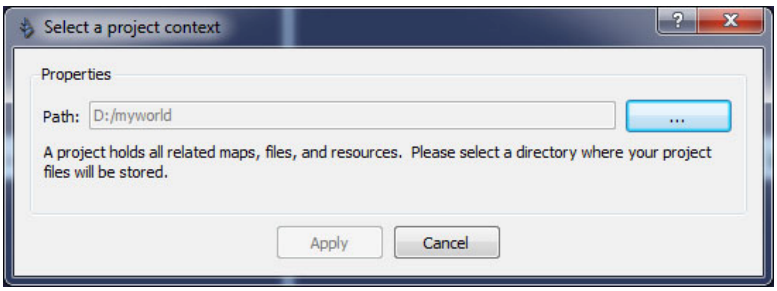
Editörün varsayılan ekranı Şekil 6.5'deki gibidir. Ekranın sağ tarafında kaynaklar bulunmaktadır. Bu kısım nesne, arazi, kaplama (texture) dosyası, ses ve animasyon gibi oyunda kullanılacak bileşenlerin veritabanını oluşturur.

Oyundaki her bileşen öncelikle aktör olarak tanımlanır; oyun aktörü (game actor) ya da normal aktör (non-game actor). Bu nedenle kaynaklar kısmında bulunan nesnelerin oyuna aktarılması için öncelikle aktör oluşturulması gerekmektedir. Bunun için ekranın sol kısmındaki aktörler kısmından belirtilen tipte aktör oluşturulur. Sol tarafta en üstte duran özellikler editörü oluşturulan aktöre ait özelliklerin tanımlanmasını sağlar.

Ekranın ortasında bulunan dörde bölünmüş pencere, oluşturulan dünyanın farklı açılardan görüntülenmesini sağlar.

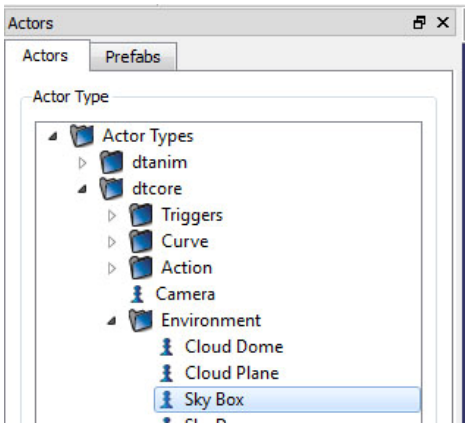
Örnek bir sanal dünya oluşturulmak istenirse izlenecek adımlar şöyledir:

1. File menüsünden Change Project seçilir ve açılan pencerede projenin saklanacağı yol belirtilir. Şekil 6.6'da proje yolunun tanımlandığı ekran görülmektedir.



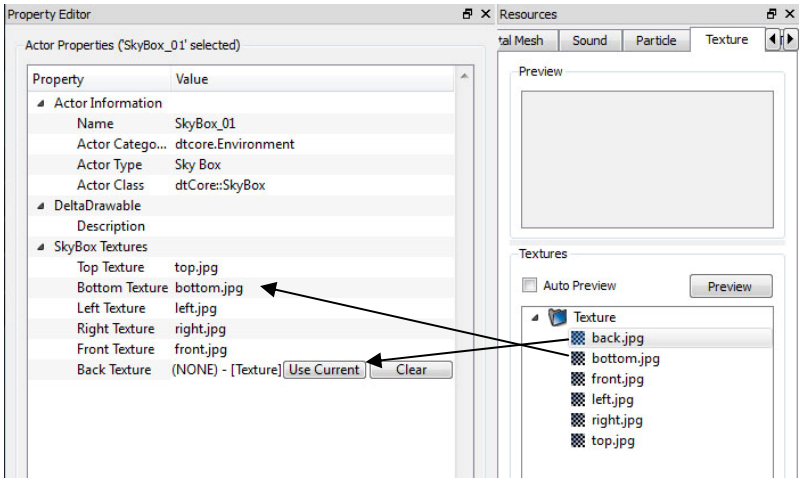
Şekil 6.6. Projenin saklanacağı yol

2. Ardından File menüsünden New Map'e tıklanıp harita ismi verilerek yeni harita oluşturmaya başlanır.
3. Sanal dünyadaki her şey aktör olarak tanımlanır. Aktör tanımlamak için Actors sekmesindeki kütüphanelerden faydalanılır. Dünya bir küp olarak düşünülürse; ortamın önü, arkası, sağ yanı, sol yanı, altı ve yukarısı için global bir aktör olarak eklenen Sky Box bileşeni, Actors sekmesinin aşağısında bulunan Create butonuna basılarak oluşturulur. Şekil 6.7'de aktörler bölümü ve Sky Box nesnesinin yeri görülmektedir.



Şekil 6.7. Aktörler penceresi ve Sky Box bileşeni

4. Sky Box bileşeninin resimlerle kaplanması için kaynaklar (resources) bölümünden Texture sekmesine sağ tıkla ya da altta bulunan araçlar bölümünden, kaplamada kullanılacak olan resimler eklenir.

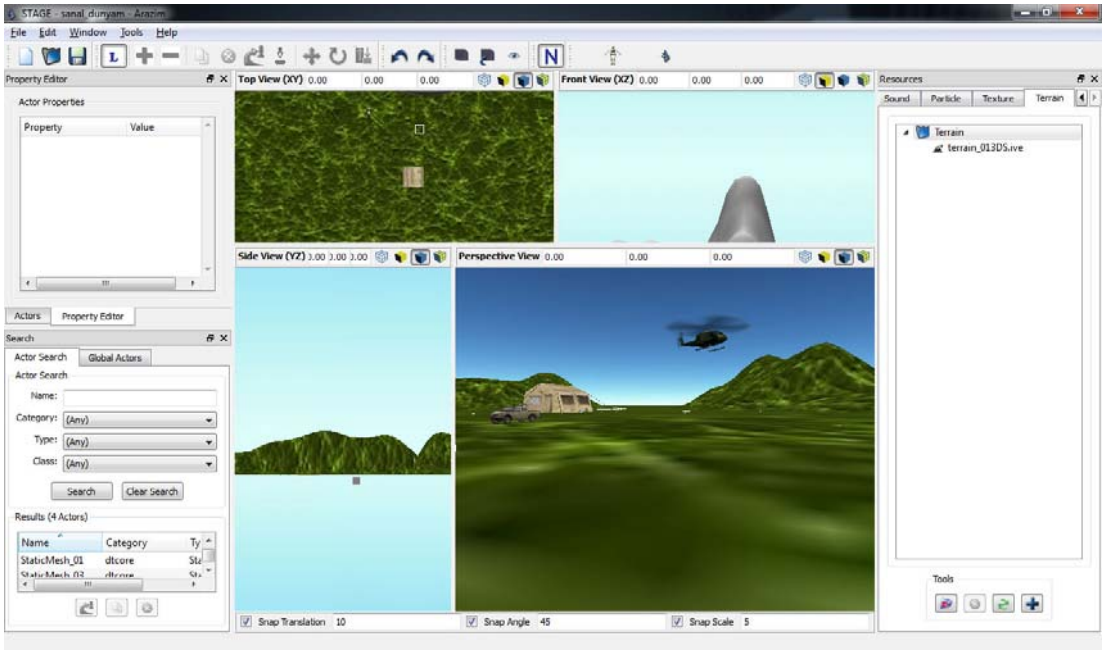


Şekil 6.8. Sky Box bileşeni özellikleri ve kaynaklar

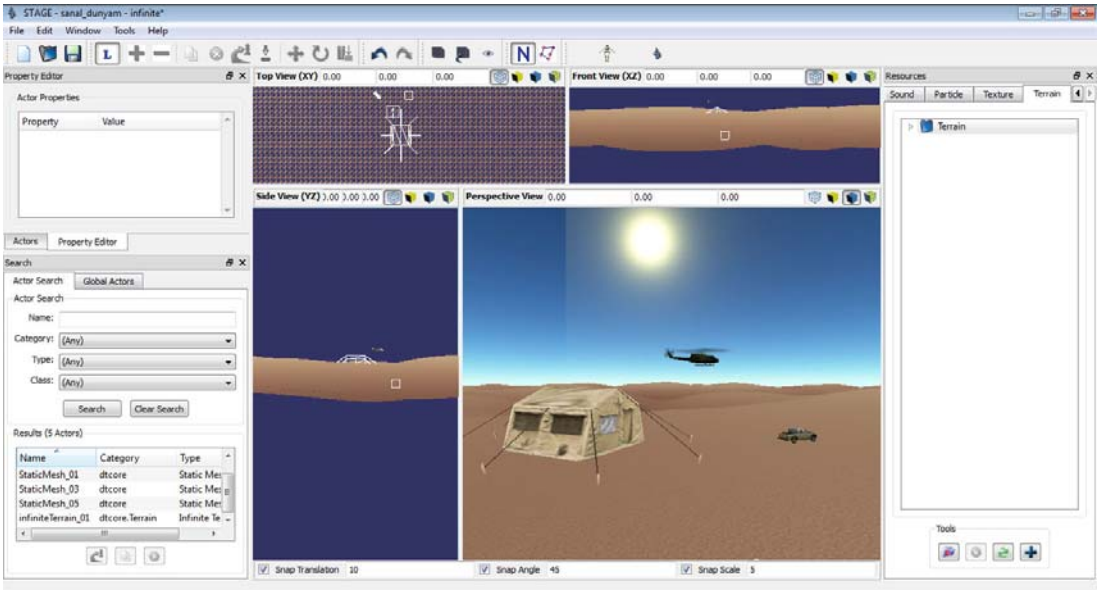
5. Ardından Sky Box bileşeninin özelliklerinden SkyBox Textures kısmına gerekli resim atamaları kaynaklardan seçilerek Şekil 6.8'deki gibi yapılır. (Kaplama dosyası seçildikten sonra özellik editöründen ilgili yüzey için "Use Current" butonuna basılır.)

Artık küp şeklinde görülebilen bir sanal dünya oluşmuş durumda. Bundan sonraki işlem arazi aktörü oluşturup, kaynaklar kısmından yukarıda anlatılanlara benzer şekilde arazi yüklemektir. Arazi aktörü eklemek için iki seçenek mevcuttur: "Mesh

Terrain" ya da "Infinite Terrain". İlk seçenekte üç boyutlu modellenmiş bir arazi yüklenerek ilgili özellikleri ayarlanır. Burada bina, ağaç ve yol gibi nesneler, harita üzerinde zaten bulunan nesneler (3D Max gibi programlarda oluşturulmuş arazi ve diğer nesneler) de olabilir, sonradan editör yardımıyla da eklenebilir. Şekil 6.9'da, "Mesh Terrain" olarak oluşturulan arazi, Sky Box ve Static Mesh aktörleri (çadır, kamyonet ve helikopter) kullanılarak, editör yardımıyla oluşturulmuş sanal dünya görülmektedir. İkinci seçenekte dışarıdan modellenmiş bir arazi yüklenmez. "Infinite Terrain" Delta3D'nin yordamsal arazi oluşturma sınıfıdır. Şekil 6.10'da "Infinite Terrain" olarak oluşturulan arazi, Sky Box ve Static Mesh aktörleri görülmektedir. Her iki arazi türü için parametreler özellikler editörü penceresinden düzenlenebilir.



Şekil 6.9. Editör yardımıyla oluşturulmakta olan sanal dünyanın ekran görüntüsü (Mesh Terrain).



Şekil 6.10. Editör yardımıyla oluşturulmakta olan sanal dünyanın ekran görüntüsü (Infinite Terrain).

Arazi üzerindeki nesneler rastgele aktör üreticisi ile belirlenen şartlara göre çoğaltılabilir. Ayrıca 3B aktörler yine editör vasıtası ile aktör olarak oluşturulabilir. İşlemler bittikten sonra yapılması gereken haritanın simülasyonda kullanılmasıdır. Editör kullanılarak oluşturulan haritanın simülasyonda kullanılması için gerekenler aşağıda adım adım listelenmiştir [47]:

1. Öncelikle Delta3d uygulama sınıflarından bir sınıf türetilir.

```
class Test_Car : public dtABC::Application
{

  public:
   Test_Model(): Application("config.xml")
   {
 }

   virtual ~Test_Model()
   {
   }
```

2. Daha sonra kamera için hareketli model ayarlamak, harita üzerinde taşıma ve Delta3D'yi ayarlamak için gerekir. Bu aşağıda verilen kısa kod parçası ile gerçekleştirilir.

```
 public:
  Test_Model()
  : Application("config.xml")
  {
```

```
      mMM = new dtCore::MotionModel(GetKeyboard(), GetMouse());
      mMM->SetTarget(GetCamera());
   }
private:
    dtCore::RefPtr<dtCore::MotionModel> mMM;
```

3. Bundan sonra yapılması gereken proje yolunu ayarlamak ve editördeki isme referans vermektir. Bu işlemin tamamı yapıcı sınıf içerisinde yapılabilir.

```
public:
   Test_Model()
   : Application("config.xml")
   {
      …

      dtDAL::Project::GetInstance().SetContext("sanal_dunyam");

      dtDAL::Map &myMap =
dtDAL::Project::GetInstance().GetMap("infinite");

      dtDAL::Project::GetInstance().LoadMapIntoScene(myMap,
*GetScene());
}
```

4. Sahneye harita yüklemek için LoadMapIntoScene() fonksiyonu kullanılır. Gereken parametreler şu şekilde sıralanabilir: yüklenecek harita için referans ve haritanın yükleneceği sahne. Hareketli model olarak helikopter olarak seçilmiş olsun. Harita, vekil nesnelerine göre sorgulanmak zorundadır. Doğru vekil nesnesi bulunana kadar aranır ve doğrulandıktan sonra model vekil nesnesine atanır. Harita sınıfı FindProxies olarak bilinen vektör ve string parametre alan bir fonksiyona sahiptir. Bu fonksiyon haritadaki vekil nesneler ile vektörleri doldurmaktadır ve string olarak belirtişmiş isimlerle vekil nesneleri eşleştirerek çağırır. Örneğin "*" parametresi haritadaki bütün vekil nesneleri döndürür. Bundan sonra istenilen aktör hareket ettirilebilir. Aşağıdaki kod parçası Test_Model sınıfının yapıcı kısmına gidebilir ve helikopter üye değişkeni sınıfın private bölümde tanımlanabilir. (Örnekte yordamsal harita kullanılmıştır.)

```
private:
  dtCore::RefPtr<dtCore::MotionModel> mMM;
  dtCore::RefPtr<dtCore::Object> mHelicopter;

public:
```

```
Test_Model()
   : Application("config.xml")
   {
      mHelicopter = NULL;
      mMM = new dtCore::MotionModel(GetKeyboard(), GetMouse());
      mMM->SetTarget(GetCamera());

      …

      typedef std::vector< dtCore::RefPtr<dtDAL::ActorProxy> >
ProxyVector;

      ProxyVector proxies;
      myMap.FindProxies(proxies, "Static*");

      ProxyVector::iterator end_iter = proxies.end();

      // Proxy process
      for (ProxyVector::iterator iter = proxies.begin(); iter
!= end_iter; ++iter)
      {
       if ((*iter)->GetName() == "StaticMesh_05")
         {
            mHelicopter =
dynamic_cast<dtCore::Object*>((*iter)->GetActor());
         }
      }

      if (mHelicopter == NULL)
      {
         return;
      }
   }

   virtual void PostFrame(const double deltaSimTime)
   {
      static double step = 0.0f;
      step += deltaSimTime;

      dtCore::Transform x(0, step, 0, 0, 0, 0);

      if(mHelicopter != NULL)
      {
        mHelicopter->SetTransform(x);
      }
   }
}
```

5. Son olarak ana fonksiyon yazılarak bu ortamda giriş aygıtlarından alınan girişlere göre hareket eden bir nesne(helikopter) ve üç boyutlu bir ortam oluşturulmuş olur.

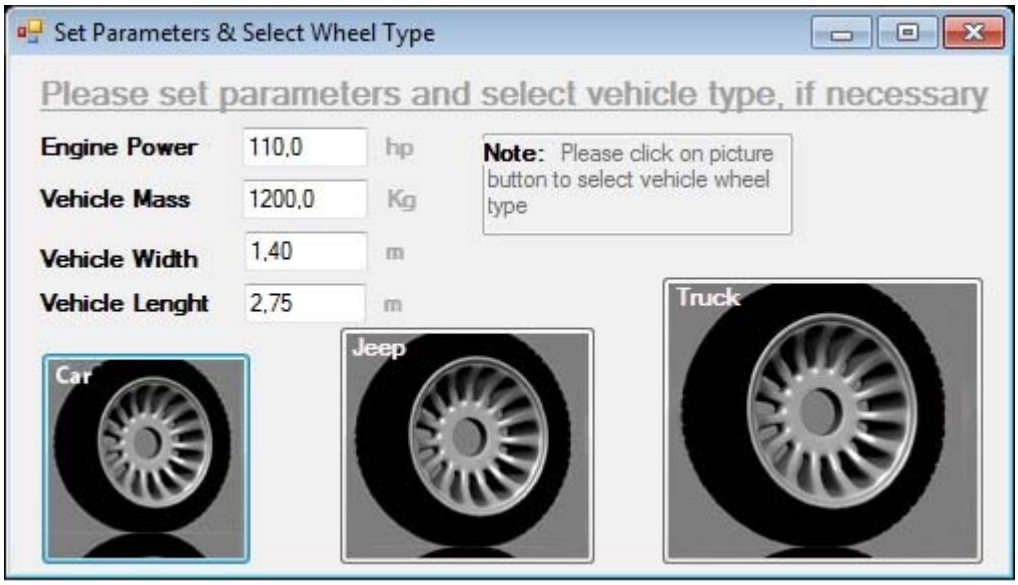
```
int main()
{
dtCore::SetDataFilePathList(".;"+dtcore::GetDeltaDataPathList()
(;
```

```
dtCore::RefPtr<Test_Model> app = new Test_Model();
app->Config();
app->Run();
    return 0;
}
```

Örnek uygulama ile editörde oluşturulan sanal dünya ve oluşturulan sanal dünyada nesne hareket ettirilmesi işlemleri gerçekleştirilmiştir.

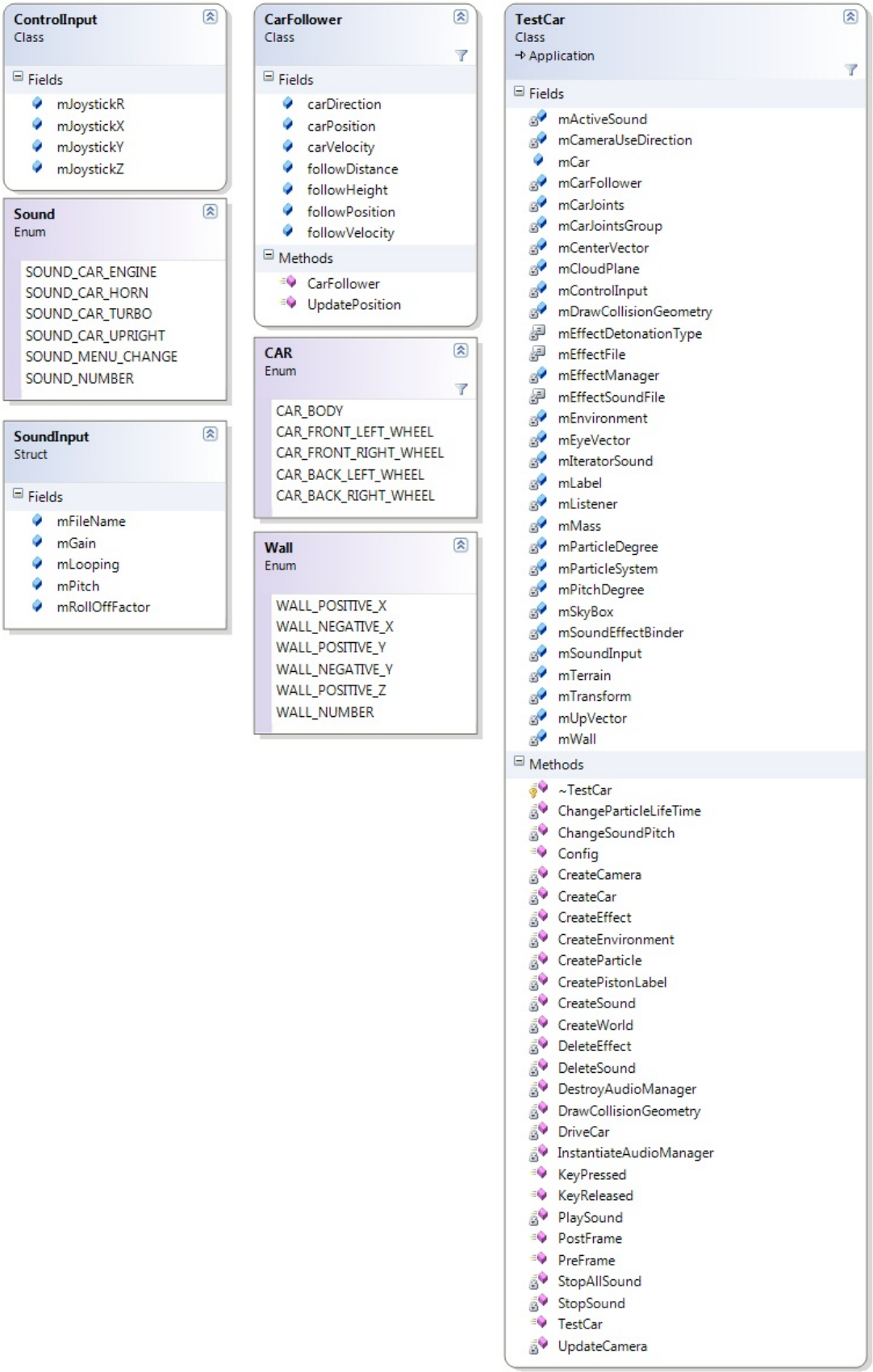
6.2.3.2. Uygulamanın Özelleştirilmiş Tarafları

Sistemin parametrik çalışabilmesi için uygulamanın başında kullanıcıdan parametrelerin alındığı grafiksel bir arayüz tasarlanmıştır. Parametre seçimi yapılan grafik arayüz Şekil 6.11'de gösterilmektedir. Grafik arayüz aracılığı ile kullanılacak aracın motor gücü, kütlesi, en ve boy uzunlukları sayısal değer olarak metin kutularına yazıldıktan sonra, tekerlek boyutu seçimi yapılır. Sanal dünyadaki aracın özellikleri belirlendikten sonra tekerlek boyutu seçimi ile uygulama başlamış olur.



Şekil 6.11. Parametre ayarlama ve tekerlek seçimi ekranı.

Geliştirilen uygulama daha karmaşık bir yapıda olması sebebiyle, işlemler fonksiyonlarla bölünmüş ve modüler hale getirilmiştir. Sanal dünyayı oluşturan yazılımın sınıf diyagramı Şekil 6.12'de gösterilmiştir.



Şekil 6.12. Uygulamanın sınıf diyagramı.

Sınıf diyagramında belirtildiği üzere TestCar sınıfı, sınıfa ait fonksiyonlar ve sınıf üyeleri programın temelini oluşturmaktadır. Bu fonksiyonlardan en çok kullanılan hiç şüphesiz aracın hareketi için giriş bilgilerinin alındığı KeyPressed fonksiyonudur. Bunun yanında aracın ekranda sürekli görüntülenmesi için kameranın cismi takip etmesi gerekir. Bu UpdateCamera fonksiyonu ile gerçekleştirilir. Bunların yanında özellikle belirtilmesi gereken kısım, aracın gövdesinin ve her bir tekerleğin ayrı birer parça olarak işlenmesidir. Bu durum Şekil 6.12'de açıkça görülmektedir. Bunun yanında oluşturulan Wall nesnesi arazi üzerindeki aracın boşluğa düşmesini engellemek için koyulmuş saydam duvarlardır.

Delta3D, simülasyon süresi boyunca yapılması gereken döngüleri sağlamaktadır. Program çalıştırıldığında ana fonksiyon işletilir. Ana fonksiyon bloğu aşağıdaki şekildedir:

```
int main()
{
   dtCore::SetDataFilePathList(dtCore::GetDeltaDataPathList() + ";"
+ dtCore::GetDeltaRootPath() + "/examples/data" + ";" + "./data" +
";");


   app = new TestCar("config.xml");
   assert(app.valid());

   app->Config(); //configuring the application
   app->Run(); // running the simulation loop

        set_piston_height(0,0,0.0);
        set_piston_height(0,1,0.0);
        set_piston_height(1,0,0.0);
        set_piston_height(1,1,0.0);
        Sleep(1000);

   return 0;
}
```

Burada öncelikle yollar tanımlandıktan sonra, uygulamanın başlangıcında işletilmek istenilen fonksiyonlar `app->Config();` satırı ile gerçekleştirilir.

```
void TestCar::Config()
{
   GetWindow()->SetWindowTitle("Test_Car_Screen");

   Application::Config();

   CreateEnvironment();
```

```
    CreateWorld();
    CreateCar();
    CreateCamera();
    CreatePistonLabel();
    …
    …
}
```

`CreateEnvironment()`fonksiyonu ile çevresel etmenler oluşturulur. Örneğin bulutların durumu, sis ve görünen mesafe ayarlanabilir. Bunun için Delta3D fonksiyonları kullanılmıştır.

```
void TestCar::CreateEnvironment()
{


   mEnvironment = new dtCore::Environment();
   assert(mEnvironment.valid());
…
   mEnvironment->SetFogEnable(true);
   mEnvironment->SetFogMode(dtCore::Environment::EXP);
  //Visibility
   mEnvironment->SetVisibility(500.0);
   mEnvironment->SetFogNear(50.0);

// CloudPlane(octaves, cutoff, frequency, amp, persistance, density, texSize,
height)
   mCloudPlane = new dtCore::CloudPlane(1, 0.5, 4, 1.0, 0.15,
0.5, 256, 100.0);
   assert(mCloudPlane.valid());
   mEnvironment->AddEffect(mCloudPlane.get());

   mSkyBox = new dtCore::SkyBox();
   assert(mSkyBox.valid());
   mSkyBox->SetTexture(dtCore::SkyBox::SKYBOX_RIGHT,
"models/world/skybox/right.jpg");
   …
   mEnvironment->AddEffect(mSkyBox.get());

   GetScene()->AddDrawable(mEnvironment.get());
}
```

`CreateWorld()`fonksiyonu ile oluşturulan sanal dünyanın arazisi yüklenir. Daha önce oluşturulmuş bir arazi kullanıldığı için sadece dosyanın alınacağı yol gösterilerek arazi yüklenir. Arazinin yüklenme şekli “6.2.2.1. Delta3D ile Sanal Dünya Oluşturma ve Örnek Uygulama Geliştirme” bölümünde anlatılmıştır.

`CreateCar()`fonksiyonu 3B arazi üzerinde hareket edecek aracın oluşturulduğu kısımdır. Uygulamada dikdörtgenler prizması şeklinde oluşturulan nesneye tekerlekler belirlenen bağlantı noktalarından ilişkilendirilir. Ayrıca oluşturulan araca ve tekerleklere ait fiziksel özellikler burada tanımlanır.

`CreateCamera()`fonksiyonu ile oluşturulan 3B nesnenin ekranda görüntülenmesi için kamera oluşturur ve takip edilecek nesne olarak araç seçilir.

`CreatePistonLabel()`fonksiyonu, aracın hesaplanan tekerlek yüksekliklerinin ekrana bastırılması için kullanılır.

Uygulamanın başlaması için sanal dünya oluşturulduktan sonra, `app->Run()`komutu ile uygulama boyunca tekrar etmesi gereken döngüler işletilir. Döngüler içerisinde tanımlanan bir fonksiyon ile klavyeden ESC tuşuna basıldığında döngü sonlanır. Son olarak program kapatılırken, yükseklik verisini çıkış olarak veren pci kartların gerilimi `set_piston_height()` fonksiyonu ile sıfırlanır. Bin milisaniye sonra programdan çıkılarak uygulama kapatılmış olur.

Delta3D oyun motoru ile az sayıda nesne görüntülenirken saniyede oluşturulan çerçeve sayısı yirmiye yakın olmaktadır. Şekil 6.13'de simülasyon esnasında saniyede görüntülenebilen çerçeve sayısı (19.19) sol üst köşede görülmektedir.

Şekil 6.13. Uygulama esnasında saniyede görüntülenebilen çerçeve sayısı (FPS).

Uygulama esnasında görüntülenen arazi ve nesne sayısı arttıkça ekranda oluşturulabilen çerçeve sayısı yaklaşık olarak ona kadar gerilemektedir. Nesne görüntüleri detaylandıkça saniyedeki çerçeve sayısı da azalmaktadır. Şekil 6.14'de sol üst köşede uygulama esnasında oluşturulan çerçeve sayısı (9.35) gösterilmektedir.

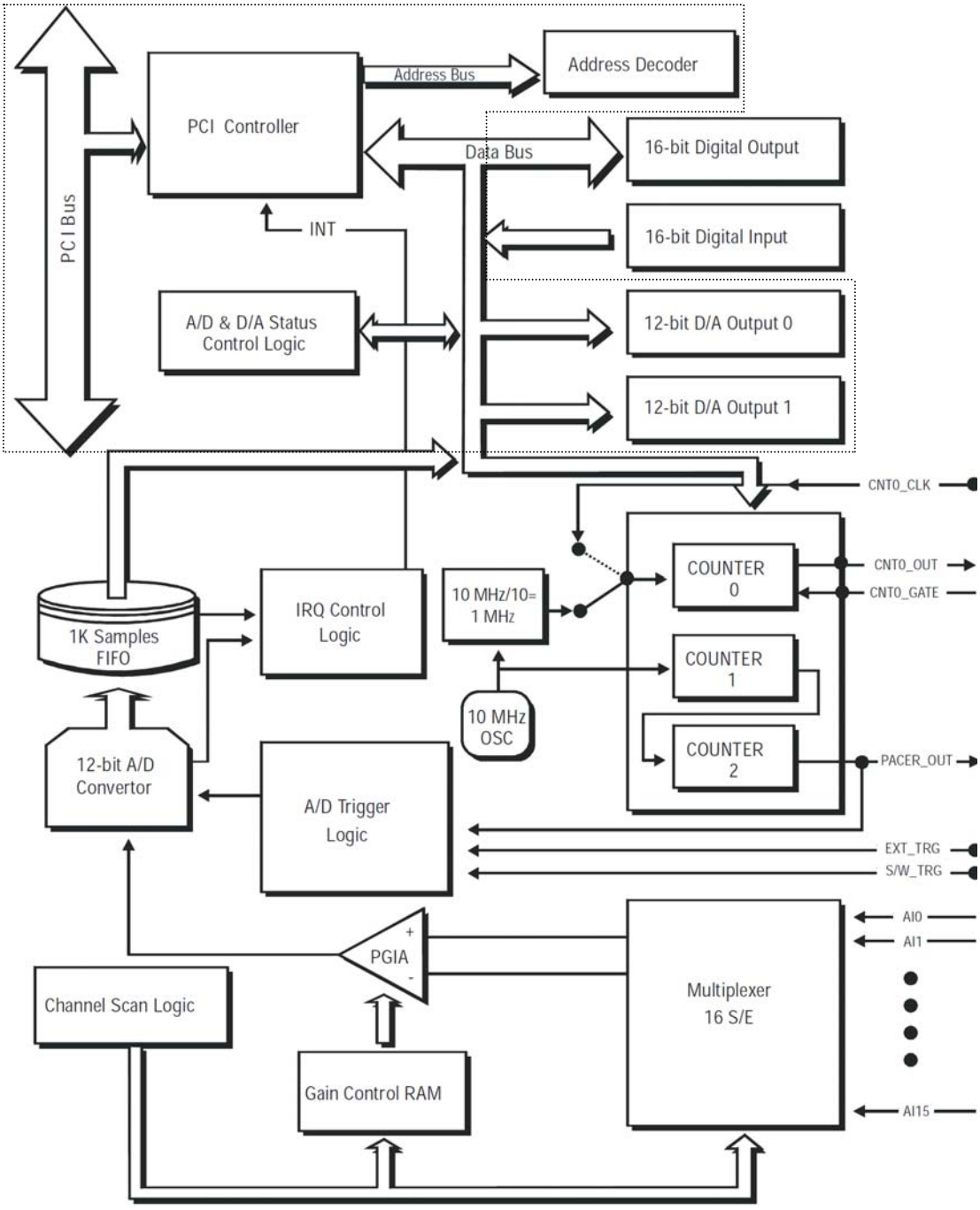
Şekil 6.14. Uygulama esnasında saniyede görüntülenebilen çerçeve sayısı (FPS).

6.2.4. Sistemin Kontrolü

Simülasyonda aracın gövdesi ve her bir tekerleği ayrı birer bileşen olarak tanımlanmıştır. Kullanılan haritada hareket eden aracın tekerlek merkezlerindeki yükseklik ilişkileri değerlendirilerek çıkışlar hesaplanır. Aracın tekerlek merkezlerindeki yükseklikler haritanın sıfır düzeyine göre hesapladığı için kartlara gönderilmeden uygunlaştırılması gerekmektedir. Bunun için aşağıdaki formül kullanılır.

$$PY[i] = | PY[i] - \min Y | + offset \tag{6.1}$$

Hesaplama sonucunda üretilen değerler `set_piston_height()` fonksiyonu ile kartlara gönderilmiştir. `set_piston_height (Ulong lDevNum, USHORT usChan, float fOutValue)` fonksiyonu üç parametre almaktadır. Bunlar sırası ile kart numarası, kanal numarası ve gönderilecek değerdir. Kartlara ait kaydedici adresleri direkt kullanılabileceği gibi, Advantech tarafından sağlanan başlık dosyalarında yapılmış tanımlarla da adreslere ulaşmak mümkündür.



Şekil 6.15. PCI-1711 veri kartı blok şeması [44].

Çizelge 6.1. PCI – 1711 veri kartının analog çıkış özellikleri [44].

Channels	2	
Resolution	12-bit	
Output Range (Internal & External Reference)	Internal Reference	0 ~ +5V, 0 ~ +10V
	External Reference	0 ~ +x V @ -x V (-10 ≤ x ≤10)
Accuracy	Relative	± 1/2 LSB
	Differential Non-linearity	± 1/2 LSB
Gain Error	Adjustable to zero	
Slew Rate	11V/μs	
Drift	40ppm/°C	
Driving Capability	3 mA	
Throughput	38 kHz (min.)	
Output Impedance	0.81Ω	
Settling Time	26 μs (to ± 1/2 LSB of FSR)	
Reference Voltage	Internal	-5 V or -10 V
	External	-10 V ~ + 10 V

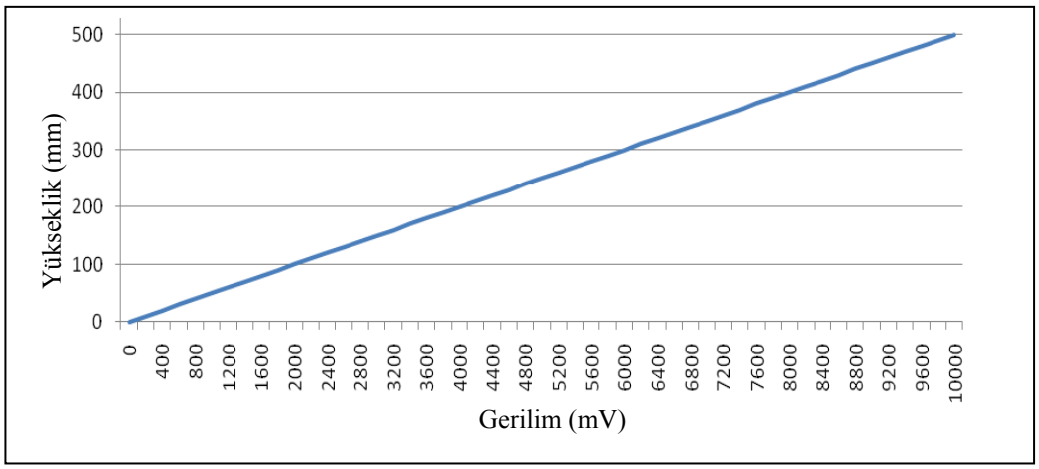
Şekil 6.15'de PCI-1711 veri kartının blok şeması ve kullanılan özellikler noktalı çizgiler ile gösterilmiştir. Yazılım tarafından dijital olarak gönderilen yükseklik bilgisi 12 bitlik D/A çevirici ile çıkışlara aktarılmaktadır. Kartın üzerinde bulunan diğer giriş ve çıkışlar, uygulamanın geniş yelpazede modifiye edilmesi esnekliğini sunar.

Çizelge 6.2. PCI – 1716 veri kartının analog çıkış özellikleri [44].

Channels	2	
Resolution	16-bit	
Operation mode	Single output	
Throughput	200 kS/s max. Per channel (FSR)	
Output Range (Internal & External Reference)	Using Internal Reference	0 ~ +5V, 0 ~ +10V, -5 ~ +5V, -10 ~ +10V
	Using External Reference	0 ~ +x V @ +x V (-10 ≤ x ≤10) -x ~ +x V @ +x V (-10 ≤ x ≤10)
Accuracy	DC	DNLE: ± 1 LSB (monotonic)
		INLE: ± 1LSB
		Zero (Offset) error: Adjustable to ± 1 LSB
		Gain (Full-scale) error: Adjustable to ± 1 LSB
Dynamic Performance	Settling Time	5 μs (to 4 LSB of FSR)
	Slew Rate	20 V/μs
Drift		10ppm/°C
Driving Capability	±20 mA	
Output Impedance	0.81Ω	

PCI-1711 veri kartının kendi referansı kullanılarak 0 ~ +10 V aralığında çıkış vermesi sağlanmıştır. Aracın ön tekerlekleri için kullanılan kartın hızı, yazılımın çıkış hızına cevap verebilecek yetenektedir.

Pistonların maksimum yüksekliği 500 mm ve PCI veri kartlarının çıkış aralığının 0 – 10 000 mV olduğu düşünülürse, gerilimdeki her 100 mV'luk artış piston yüksekliğinde 1 mm yükselişe tekabül etmektedir. Şekil 6.16'daki grafikte gösterildiği gibi pistonların yükselişi lineerdir.



Şekil 6.16. PCI veri kartı çıkışına göre piston yüksekliği grafiği.

Sürücü devresi, simülasyon yazılımından veri kartları aracılığı ile alınan piston yükseklik bilgisine göre valfleri kontrol etmektedir. Aynı zamanda pistonların kontrolü, lineer cetvellerden alınan geri besleme bilgisine göre sürücü devresinde yapılacak şekilde tasarlanmıştır.

BÖLÜM 7

SONUÇLAR VE ÖNERİLER

Yapılan çalışma sonucunda, üç boyutlu grafiksel ortamda fiziksel kurallara göre hareket eden aracın yükseklik verileri PCI veri kartından elektriksel çıkış olarak alınmıştır. Bu çıkışlar ile sürücü devresi vasıtasıyla mekanik platform kontrol edilebilmektedir. Mekanik platformun simülatör yazılımı ile eş zamanlı çalışarak hareket etmesi, kullanıcının gerçek bir araç kullanıyormuş gibi hissetmesini sağlamaktadır. Bu kapsamda geliştirilen simülatör, sürücü kurslarında direksiyon eğitimlerinde kullanılmaya uygundur.

Araç en ve boy uzunlukları; en uzunluğu en çok 3 metre, boy uzunluğu ise en çok 12 metre olarak sınırlandırılmıştır. Simülatörün bu özellikleri ile kişisel otomobillerden, kamyonetlere kadar çeşitli araçların simülasyonları yapılabilmektedir. Geliştirilen simülatörün dört tekerlekli araçlar için yapılmış olması dingilli araçların simülasyonundaki başarısını olumsuz yönde etkilemektedir. Bunun nedeni, pistonlar arazi yüksekliklerini simüle ederken, süspansiyon sistemine sahip aracın araziye göre konum almasından kaynaklanmaktadır. Bu bakış açısıyla, araç süspansiyonun konfora etkilerini test etmek için mekanik platformun kullanılması mümkün hale gelmekte ve olumsuz görünen özellik farklı alanda araştırmalar için çalışma ortamı oluşturmaktadır.

Şekil 6.13'de ve Şekil 6.14'de uygulama esnasında saniyede görüntülenebilen çerçeve sayısı gösterilmektedir. Uygulamanın başlaması ile nesneler sahnede görüntülenmeye başlamaktadır. Görüntülenen nesne sayısı arttıkça ve detaylandıkça beklendiği üzere görüntülenebilen çerçeve sayısı azalmaktadır. Bu, grafik işlemcinin görüntüleme için yaptığı işlemlerin zaman almasından kaynaklanmaktadır ve işlem yükü artışının sonucudur. Daha çok grafik belleğe ve işlem kapasitesine sahip grafik işlemcilerle bu durum giderilebilmektedir.

Çalışmanın önemli konularından bir tanesi de mekanik platformun tasarımıdır. Burada üzerinde durulması gereken husus, aracın fiziksel durumunun ne kadar simüle edilebileceğidir. Mekanik platform, kolay kontrol ve çok durum simüle edecek şekilde tasarlanmalı ve silindirler bu kriterler düşünülerek konumlandırılmalıdır.

Çalışma ile sistemi oluşturan bileşenlerin entegrasyonu sağlanmıştır. Bundan sonra grafiksel hızlandırma, fiziksel durumların modellenmesi ve diğer işlemlerin daha hızlı yapılabilmesi için yazılımda optimizasyona gidilebilir. Bunun yanı sıra mekanik platformun kontrolü ile ilgili yeni yöntemler üzerinde çalışmalar yapılabilir.

KAYNAKLAR

1. İnternet: Hardwaremania "GPU Computing(GPU Hesaplama)", **http://www.hardwaremania.com/makaleler/124-gpu-computing.html?showall=1** (2008).

2. Alkım, B. E., "Grafik cephesinden cpu'ya destek"Aralık, ***CHIP,*** 70-74 (2009).

3. Çavaş, B., Huyugüzel Çavaş, P. ve Taşkın Can, B., "Eğitimde sanal gerçeklik", ***TOJET,*** 3 (4): 110-116 (2004).

4. Cruz-Neira, C., "Making virtual reality useful: A report on immersive applications at Iowa State University", ***Future Generation Computer Systems,*** 14: 147-155 (1998).

5. Bayraktar, E., Kaleli, F., "Sanal gerçeklik ve uygulama alanları", ***Akademik Bilişim 2007,*** Kütahya, (2007).

6. Spear, B., "Virtual reality: patent review", ***World Patent Information,*** 24: 103-109 (2002).

7. İnternet: Simulator Makine Produksiyon "6D Sinema", **http://www.simulator.com.tr/6dsinema.html** (2010).

8. İnternet: Wikipedia "Wired Glove", **http://en.wikipedia.org/wiki/Wired_glove** (2010).

9. İnternet: The Official Website of The Irish Football Association "Virtual reality training programme", **http://www.irishfa.com/the-ifa/news/5438/virtual-reality-training-programme** (2010).

10. İnternet: Aselsan "KMS Eğitim Simülatörü", **http://www.aselsan.com.tr/urun.asp?urun_id=78&lang=tr#** (2010).

11. Göktaş, H. H., Çavuşoğlu, A., Şen, B. ve Görgünoğlu, S., "Simülasyon sistemleri için 3 boyutlu sanal şehirlerin sayısal coğrafik haritalar üzerinde üretilmesi", ***Teknoloji,*** 9 (1): 27-38 (2006).

12. Göktaş, H. H., Çavuşoğlu, A. ve Şen, B., "AUTOCITY : A System for generating 3D virtual cities for simulation systems on GIS map", ***AutoSoft - Intelligent Automation and Soft Computing***,.15 (1): 29-39 (2009).

13. İnternet: iTech News Net "SKIGYM Ski Simulator", **http://www.itechnews.net/2009/01/04/skigym-ski-simulator/** (2010).

14. İnternet: “SimuX Raylı Sistem Simülatörü”, **http://www2.itu.edu.tr/~soylemezm/simux/simux.pdf** (2010).

15. İnternet: Wikipedia “Simulation”, **http://en.wikipedia.org/wiki/Simulation** (2010).

16. İnternet: Wikipedia “Vehicle simulation game”, **http://en.wikipedia.org/wiki/Vehicle_simulation_game** (2010).

17. Ni, T., Zhao, D. and Zhang, H., “Realistic vehicle driving simulator with dynamic terrain deformation”, ***Proceedings of the 2009 IEEE International Conference on Mechatronics and Automation***, China, 4795-4800 (2009).

18. Horiguchi, A., Suetomi, T., “A kansei engineering approach to driver/vehicle system”, ***International Journal of Industrial Ergonomics,*** 15: 25-37 (1995).

19. İnternet: TOYOTA “Sürüş Simülatörü”, **http://www.toyota.com.tr/about/news_and_events/toyota_surus_simulatoru.aspx** (2010).

20. İnternet: TheCar Connection “Toyota’s Huge New Simulator”, **http://www.thecarconnection.com/tips-article/1012323_toyotas-huge-new-driving-simulator** (2010).

21. Çavuşoğlu A., Şen B. ve Vural M.,"Akıllı trafik eğitimi simülatörü için 3 boyutlu veritabanı üretilmesi" Proje Kodu: 07/2003-45, ***Gazi Üniversitesi Bilimsel Araştırmalar Projesi Sonuç Raporu***, 1-25 (2005).

22. Çavuşoğlu A., Şen B. ve Vural M., "Sürücü adayları için akıllı sanal sürüş eğitimi platformunun geliştirilmesi", ***DPT Projesi Raporu*** (2006).

23. Segal, M., Akeley, K., “What is the OpenGL graphics system?”, The OpenGL Graphics System: A Specification, ***Khonos Group Inc***., USA, 1-3 (2010).

24. İnternet: Wikipedia “OpenGL”, **http://tr.wikipedia.org/wiki/OpenGL** (2010).

25. İnternet: Wikipedia “DirectX”, **http://tr.wikipedia.org/wiki/DirectX** (2010).

26. İnternet: “OpenSG”, **http://www.opensg.org** (2010).

27. Martz, P., “History of OpenSceneGraph”, OpenSceneGraph quick start guide, ***Skew Matrix Software LLC***, Louisville-USA, 1-3 (2007).

28. Yuan, P., Wang, S., Zhang, J. and Liu, H., “Virtual reality platform based on open sourced graphic toolkit OpenSceneGraph”, ***10th Computer-Aided Design and Computer Graphics IEEE International Conference***, Beijing, 361-364 (2007).

29. İnternet: Wikipedia “Havok”, **http://en.wikipedia.org/wiki/Havok_(software)** (2010).

30. İnternet: “Open Dynamics Engine”, **http://www.ode.org/** (2009).

31. İnternet: Newton Game Dynamics “What is Newton Physics Engine”, **http://newtondynamics.com/forum/newton.php** (2010).

32. İnternet: Wikipedia “Newton Game Dynamics”, **http://en.wikipedia.org/wiki/Newton_Game_Dynamics** (2010).

33. İnternet: Wikipedia “Game Engines”, **http://gpwiki.org/index.php/Game_Engines** (2010).

34. Darken, C. J., Anderegg, B. G. and McDowell, P. L., “Game AI in Delta3D”, ***Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games,*** 312-319 (2007).

35. Darken, R., McDowell, P. and Johnson, E., “The Delta3D open source game engine”, ***IEEE Computer Graphics and Applications,*** 25 (3): 10-12 (2005).

36. İnternet: Wikipedia “Delta3D”, **http://en.wikipedia.org/wiki/Delta3D** (2010).

37. Yüksel, İ., “Otomatik kontrol”, ***Vipaş A.Ş.***, Bursa, 1-10, 191-210 (2001).

38. İnternet: Wikipedia “Control System”, **http://en.wikipedia.org/wiki/Control_system** (2010).

39. Kuo, B.,C., ”Otomatik kontrol sistemleri”, Çeviri: Bir, A., ***Literatür Yayınları***, Ankara 687-738 (1999).

40. Elmas, Ç., “Yapay zeka uygulamaları”, ***Seçkin Yayıncılık***, Ankara, 289-290 (2007).

41. Elmas, Ç., “Bulanık mantık denetleyiciler”, ***Seçkin Yayıncılık***, Ankara, 24-39 (2003).

42. Lee, C. C., “Fuzzy logic in control system: fuzzy logic controller - part 1”, ***IEEE Transaction On Systems, Man, And Cybernetics,*** 20 (2): 404-418 (1990).

43. Lee, C. C., “Fuzzy logic in control system: fuzzy logic controller - part 2”, ***IEEE Transaction On Systems, Man, And Cybernetics***, 20 (2): 419-435 (1990).

44. “PCI-1710 Series 12/16 bit multifunction card user’s manual”, ***Advantech***, Taiwan, (2001).

45. İnternet: Wikipedia “PID Controller”, **http://en.wikipedia.org/wiki/PID_controller** (2010).

46. İnternet: Delta3d, “GM utorials Parts 1 and 2”, **http://www.delta3d.org/article.php?story=20060620123144266&topic=tutorials** (2010).

47. İnternet: Tutorial using map generated from stage, **http://sourceforge.net/apps/mediawiki/delta3d/index.php?title=TutorialUsingMapGeneratedFromStage** (2010).

ÖZGEÇMİŞ

Ferhat ATASOY 1984 yılında Ankara'da doğdu. İlk, orta ve yüksek öğrenimini aynı şehirde tamamladı. Türk Telekom Anadolu Meslek Lisesi Telekomünikasyon Bölümünden 2002, Gazi Üniversitesi Elektronik ve Bilgisayar Eğitimi Bölümünden 2008 yılında mezun oldu. 2008 yılında Karabük Üniversitesi Bilgisayar Mühendisliği Bölümü'nde araştırma görevlisi olarak göreve başladı ve halen aynı yerde çalışmaktadır. Aynı yıl Gazi Üniversitesi Bilişim Enstitüsü Elektronik-Bilgisayar Eğitimi Bölümü'nde başladığı yüksek lisans eğitimini 2010 yılında Karabük Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Bölümünde tamamladı.

ADRES BİLGİLERİ

Adres : Karabük Üniversitesi
Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü
Balıklarkayası Mevkii / KARABÜK

Tel : (555) 520 97 57
E-posta : ferhatatasoy@karabuk.edu.tr

EK AÇIKLAMALAR A
PROGRAM KODU

```
/* Headers listed below are included in functions.h
#include "Test_Car.h"
#include "carfollower.h"

#include <dtABC/labelactor.h>
#include <dtAudio/audiomanager.h>
#include <dtAudio/sound.h>
#include <dtAudio/soundeffectbinder.h>
#include <dtCore/deltawin.h>
#include <dtCore/base.h>
#include <dtCore/scene.h>
#include <dtCore/system.h>
#include <dtCore/object.h>
#include <dtCore/particlesystem.h>
#include <dtCore/effectmanager.h>
#include <dtCore/odecontroller.h>
#include <dtCore/odebodywrap.h>
#include <dtCore/odespacewrap.h>
#include <dtCore/globals.h>
#include <dtCore/environment.h>
#include <dtCore/cloudplane.h>
#include <dtCore/skydome.h>
#include <dtCore/skybox.h>
#include <dtUtil/log.h>
#include <osg/Geode>
#include <osg/ShapeDrawable>
#include <osg/Texture2D>
#include <osg/StateSet>
#include <osg/Switch>
#include <osgDB/ReadFile>
#include <osgDB/FileUtils>
#include <cassert>

#include <stdio.h> */

// Program Code
#include "functions.h"

bool TestCar::KeyPressed(const dtCore::Keyboard* keyboard, int ki)
{
   bool handled(false);
   switch(ki)
   {
      case osgGA::GUIEventAdapter::KEY_Escape:
       close();
       Quit();
         handled = true;

         break;
      case osgGA::GUIEventAdapter::KEY_F1:
         if(!dtCore::System::GetInstance().GetPause())
         {

        if (menu)
        {
          menu=0;
         mLabel->SetActive(mLabel->GetActive());
        }
        else menu=1;
```

```
         }
         break;
      case osgGA::GUIEventAdapter::KEY_Up:
         if(!dtCore::System::GetInstance().GetPause())
         {
            mControlInput.mJoystickY = -1.0;
            if(mPitchDegree > SOUND_CAR_ENGINE_PITCH_MIN)
            {
               mPitchDegree -= SOUND_CAR_ENGINE_PITCH_STEP;
               ChangeSoundPitch(SOUND_CAR_ENGINE,
mSoundInput[SOUND_CAR_ENGINE].mPitch +
abs(cos(osg::DegreesToRadians(mPitchDegree))));
            }
         }
         break;
      case osgGA::GUIEventAdapter::KEY_Down:
         if(!dtCore::System::GetInstance().GetPause())
         {
            mControlInput.mJoystickY = 1.0;
            if(mPitchDegree < SOUND_CAR_ENGINE_PITCH_MAX)
            {
               mPitchDegree += SOUND_CAR_ENGINE_PITCH_STEP;
               ChangeSoundPitch(SOUND_CAR_ENGINE,
mSoundInput[SOUND_CAR_ENGINE].mPitch +
abs(cos(osg::DegreesToRadians(mPitchDegree))));
            }
         }
         break;
      case osgGA::GUIEventAdapter::KEY_Left:
         if(!dtCore::System::GetInstance().GetPause())
         {
            mControlInput.mJoystickX = -1.0;
         }
         break;
      case osgGA::GUIEventAdapter::KEY_Right:
         if(!dtCore::System::GetInstance().GetPause())
         {
            mControlInput.mJoystickX = 1.0;
         }
         break;
      case osgGA::GUIEventAdapter::KEY_Space:
         if(!dtCore::System::GetInstance().GetPause())
         {
            // Get new camera view from mCarFollwer's updated
positions
            mCarFollower->UpdatePosition(CAMERA_TIME_STEP * 2.0,
true);
            mEyeVector = mCarFollower->followPosition;
            mCenterVector = mCarFollower->carPosition;
            mUpVector = osg::Z_AXIS;

            // Adjust the Camera position
            mTransform.Set(mEyeVector, mCenterVector, mUpVector);
            GetCamera()->SetTransform(mTransform);

         }
         break;
      case 'C':
      case 'c':
```

```
         if(!dtCore::System::GetInstance().GetPause())
         {
            mCameraUseDirection = !mCameraUseDirection;
            mCameraUseDirection ? mCarFollower->followDistance =
CAMERA_DISTANCE_1 : mCarFollower->followDistance =
CAMERA_DISTANCE_2;
            mCameraUseDirection ? mCarFollower->followHeight =
CAMERA_HEIGHT_1 : mCarFollower->followHeight = CAMERA_HEIGHT_2;

         }
         break;

      case 'P':
      case 'p':

dtCore::System::GetInstance().SetPause(!dtCore::System::GetInstance(
).GetPause()(;
         break;
      case 'G':
      case 'g':
         if(!dtCore::System::GetInstance().GetPause())
         {
            DrawCollisionGeometry();

         }
         break;
      case 'S':
      case 's':
         if(!dtCore::System::GetInstance().GetPause())
         {
            SetNextStatisticsType();
         }
         break;
      case 'T':
      case 't':
         if(!dtCore::System::GetInstance().GetPause())
         {
            mControlInput.mJoystickY *= 10.0;
            PlaySound(SOUND_CAR_TURBO);
         }
         break;

      case 'F':
      case 'f':
         if(!dtCore::System::GetInstance().GetPause())
         {
            dBodyAddForce(mCar[CAR_BODY]->GetBodyID(), 0, 0,
(CAR_BODY_MASS * CAR_WHEEL_MASS) * 2.0);
            dBodyAddRelTorque(mCar[CAR_BODY]->GetBodyID(), 0,
(CAR_BODY_MASS * CAR_WHEEL_MASS) * 2.0, 0);
            PlaySound(SOUND_CAR_UPRIGHT);
         }
         break;
      // Make cases for other keys
      default:
         break;
   }

   DriveCar(-mControlInput.mJoystickY, mControlInput.mJoystickX);
```

```
   return handled;
}


bool TestCar::KeyReleased(const dtCore::Keyboard* keyboard, int kc)
{
   bool handled(false);
   switch(kc)
   {
      case osgGA::GUIEventAdapter::KEY_Up:
         mControlInput.mJoystickY = 0.0;
         break;
      case osgGA::GUIEventAdapter::KEY_Down:
         mControlInput.mJoystickY = 0.0;
         break;
      case osgGA::GUIEventAdapter::KEY_Left:
         mControlInput.mJoystickX = 0.0;
         break;
      case osgGA::GUIEventAdapter::KEY_Right:
         mControlInput.mJoystickX = 0.0;
         break;
      // Make cases for other keys
      default:
         break;
   }

   DriveCar(-mControlInput.mJoystickY, mControlInput.mJoystickX);

   return handled;
}

void TestCar::PreFrame(const double deltaFrameTime)
{
   // Called prior to rendering of frame, do you scene updates here
}


void TestCar::PostFrame(const double deltaFrameTime)
{
   // Called after frame has been rendering, collect information
about results from scene interaction here
   UpdateCamera();

   // It works when KeyReleased event occur
   if(mControlInput.mJoystickY == 0.0)
   {
       mPitchDegree < SOUND_CAR_ENGINE_PITCH_IDLE ? mPitchDegree +=
SOUND_CAR_ENGINE_PITCH_STEP : NULL;
       mPitchDegree > SOUND_CAR_ENGINE_PITCH_IDLE ? mPitchDegree -=
SOUND_CAR_ENGINE_PITCH_STEP : NULL;

       if(mPitchDegree != SOUND_CAR_ENGINE_PITCH_IDLE)
       {
          ChangeSoundPitch(SOUND_CAR_ENGINE,
mSoundInput[SOUND_CAR_ENGINE].mPitch +
abs(cos(osg::DegreesToRadians(mPitchDegree))));
       }
       else if(mParticleSystem->GetActive())
       {
          mParticleDegree -= SOUND_CAR_ENGINE_PITCH_STEP;
```

```
            ChangeParticleLifeTime();
        }
    }
}



void TestCar::UpdateCamera()
{
    osg::Quat mRotation;
    osg::Vec3 mDirection;

    // Update car parameters in mCarFollower
    mCar[CAR_BODY]->GetTransform(mTransform);
    mCarFollower->carPosition = mTransform.GetTranslation();
    mCarFollower->carVelocity = mCar[CAR_BODY]->GetBodyWrapper()-
>GetLinearVelocity();
    mTransform.GetRotation(mRotation);
    mDirection = mRotation * osg::Y_AXIS;
    mCarFollower->carDirection = osg::Vec2(mDirection.x(),
mDirection.y());

    if(mCarFollower->carDirection.length2() < 0.8)
    {
        mDirection = mRotation * (osg::Y_AXIS * -1);
        mCarFollower->carDirection = osg::Vec2(mDirection.x(),
mDirection.y());
    }
    mCarFollower->carDirection /= mCarFollower-
>carDirection.length();

    // Get new camera view from mCarFollwer's updated positions
    mCarFollower->UpdatePosition(CAMERA_TIME_STEP,
mCameraUseDirection);
    mEyeVector = mCarFollower->followPosition;

     mCenterVector = mCarFollower->carPosition;
     mUpVector = osg::Z_AXIS;

    mCar[CAR_FRONT_LEFT_WHEEL]->GetTransform(mTransform);
    mCarFollower->carPosition = mTransform.GetTranslation();

    piston_height[0]=(mCarFollower-
>carPosition.z())*scale_coefficent;

    mCar[CAR_FRONT_RIGHT_WHEEL]->GetTransform(mTransform);
    mCarFollower->carPosition = mTransform.GetTranslation();

    piston_height[1]=(mCarFollower-
>carPosition.z())*scale_coefficent;

    mCar[CAR_BACK_LEFT_WHEEL]->GetTransform(mTransform);
    mCarFollower->carPosition = mTransform.GetTranslation();

    piston_height[2]=(mCarFollower-
>carPosition.z())*scale_coefficent;

    mCar[CAR_BACK_RIGHT_WHEEL]->GetTransform(mTransform);
    mCarFollower->carPosition = mTransform.GetTranslation();
```

```
   piston_height[3]=(mCarFollower-
>carPosition.z())*scale_coefficent;


CreatePistonLabel(piston_height[0],piston_height[1],piston_height[2]
,piston_height[3]);


//Heights are calculated by reference
   reference
=(min(piston_height[0],(min(piston_height[1],(min(piston_height[2],p
iston_height[3])))))));

   for (int h=0;h<4;h++)
     piston_height[h]=piston_height[h]-reference;

   // Sending voltage values to command screen
   printf("Val_0:%f",piston_height[0]);
   printf("\tVal_1:%f",piston_height[1]);
   printf("\tVal_2:%f",piston_height[2]);
   printf("\tVal_3:%f\n",piston_height[3]);


   //Sending height data to cards
   set_piston_height(0,0,fabs(piston_height[0]));
   set_piston_height(0,1,fabs(piston_height[1]));
   set_piston_height(1,0,fabs(piston_height[2]));
   set_piston_height(1,1,fabs(piston_height[3]));

   // Adjust the Camera position
   mTransform.Set(mEyeVector, mCenterVector, mUpVector);
   GetCamera()->SetTransform(mTransform);
}

void TestCar::DriveCar(double velocity, double steering)
{
   static const dReal steeringRate = M_PI * 4 / 3;
   static const dReal steeringLimit = M_PI / 6;

   dReal wheelVelocity = 12 * M_PI * velocity;

   if(fabs(steering) < 0.1)
   {
      steering = 0.0;
   }

   for(short i = CAR_FRONT_LEFT_WHEEL; i < CAR_NUMBER; i++)
   {
      double desiredPosition = steering * (((i ==
CAR_BACK_LEFT_WHEEL) || (i == CAR_BACK_RIGHT_WHEEL)) ? 0 : 1);
      double actualPosition = dJointGetHinge2Angle1(mCarJoints[i]);
      double steeringVelocity = (desiredPosition - actualPosition) *
50;

      dJointSetHinge2Param(mCarJoints[i], dParamHiStop, (steering ==
0.0) ? 0.0 : steeringLimit);
      dJointSetHinge2Param(mCarJoints[i], dParamLoStop, (steering ==
0.0) ? 0.0 : -steeringLimit);
      dJointSetHinge2Param(mCarJoints[i], dParamVel,
steeringVelocity);
```

```
      dJointSetHinge2Param(mCarJoints[i], dParamVel2, ((i % 2) == 0)
? -wheelVelocity : wheelVelocity);

   }
}

void TestCar::DrawCollisionGeometry()
{
   mDrawCollisionGeometry = !mDrawCollisionGeometry;

   // RenderCollisionGeometry for mWall
   for(short i = 0; i < WALL_NUMBER; i++)
   {
      if(mWall[i].valid())
      {
         mWall[i]->RenderCollisionGeometry(mDrawCollisionGeometry);
      }
   }

   // RenderCollisionGeometry for mCar
   for(short i = 0; i < CAR_NUMBER; i++)
   {
      if(mCar[i].valid())
      {
         mCar[i]->RenderCollisionGeometry(mDrawCollisionGeometry);
      }
   }

}


int main()
{
   //set data search path to parent directory and delta3d/data
   dtCore::SetDataFilePathList(dtCore::GetDeltaDataPathList() + ";"
+ dtCore::GetDeltaRootPath() + "/examples/data" + ";" + "./data" +
";");

   //Instantiate the application and look for the config file
   app = new TestCar("config.xml");
   assert(app.valid());

   app->Config(); //configuring the application
   app->Run(); // running the simulation loop

       // Set zero all outputs
        set_piston_height(0,0,0.0);
        set_piston_height(0,1,0.0);
        set_piston_height(1,0,0.0);
        set_piston_height(1,1,0.0);
        Sleep(1000);

   return 0;
}
```