İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

UNITY OYUN MOTORU İLE SİMÜLATÖR TASARIMI

YÜKSEK LİSANS TEZİ

Cihad DOĞAN

Kontrol ve Otomasyon Mühendisliği Anabilim Dalı

Kontrol ve Otomasyon Mühendisliği Programı

MAYIS 2019

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

UNITY OYUN MOTORU İLE SİMÜLATÖR TASARIMI

YÜKSEK LİSANS TEZİ

Cihad DOĞAN
(504161108)

Kontrol ve Otomasyon Mühendisliği Anabilim Dalı

Kontrol ve Otomasyon Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Leyla GÖREN SÜMER

MAYIS 2019

İTÜ, Fen Bilimleri Enstitüsü'nün 504161108 numaralı Yüksek Lisans öğrencisi Cihad DOĞAN, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı "UNITY OYUN MOTORU İLE SİMÜLATÖR TASARIMI" başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Prof. Dr. Leyla GÖREN SÜMER** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Dr. Ali Fuat ERGENÇ** İstanbul Teknik Üniversitesi	.....
	Doç. Dr. Akın DELİBAŞ Yıldız Teknik Üniversitesi	.....

Teslim Tarihi : 30 Nisan 2019
Savunma Tarihi : 30 Mayıs 2019

Aileme ve arkadaşlarıma,

ÖNSÖZ

Tez çalışmam boyunca bana zaman ayıran, yol gösteren ve hiçbir yardımı esirgemeyen değerli hocam Prof. Dr. Leyla GÖREN SÜMER'e teşekkürlerimi sunarım. Ayrıca eğitim hayatımda bu noktaya gelmemde, maddi ve manevi desteklerini hiçbir şekilde benden esirgemeyen annem, babam ve kardeşlerime şükranlarımı sunuyorum. Son olarak çalıştığım şirket olan DESİ Alarm ve Güvenlik firmasına yüksek lisans eğitimim boyunca verdikleri destek için teşekkür ederim.

Mayıs 2019

Cihad Doğan
Kontrol ve Otomasyon Müh.

İÇİNDEKİLER

Sayfa

KISALTMALAR

TCP **:** Transmission Control Protocol
UDP **:** User Datagram Protocol
UI **:** User Interface

SEMBOLLER

N	**:** Newton
L	**:** Lagrangian
g	**:** Yer Çekimi İvmesi
b	**:** Viskoz Sürtünme Katsayısı
J	**:** Eylemsizlik Katsayısı
w	**:** Açısal hız
v	**:** Çizgisel Hız
$\boldsymbol{E_k}$	**:** Kinetik Enerji
$\boldsymbol{E_p}$	**:** Potansiyel Enerji

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

UNITY OYUN MOTORU İLE SIMULATOR TASARIMI

ÖZET

Günümüzde bilgisayar simülasyonları birçok mühendislik projesinin yanında ekonomi, biyoloji, psikoloji ve tıp gibi alanlarda da aktif olarak kullanılmaktadır. Geliştirilen bilgisayar simülatörleri sayesinde birçok deney, test ve eğitim ucuz, hızlı ve herhangi bir insan hayatını tehlikeye atmadan gerçekleştirilmektedir. Bilgisayar simülatörleri çeşitli oyun motorları kullanılarak veya çok daha düşük seviye olan grafik ve fizik kütüphaneleri kullanılarak geliştirilebilir. Bu tez çalışmasında kontrol ve otomasyon mühendisliği eğitiminde kullanılan çeşitli deney setlerinin Unity oyun motoru kullanılarak simülatörlerinin oluşturulması amaçlanmıştır. Unity oyun motoru Unity Technologies firması tarafından geliştirilen bilgisayar, konsollar ve mobil platformlar için oyun ve simülatör geliştirilmesini sağlayan çapraz bir oyun motorudur. Unity oyun motoru kendi web adresinden ücretsiz olarak edinilebilir. Yazılım dili olarak C# kullanan motor Windows ortamında çalışabilmektedir.

Çalışmanın ilk aşamasında Unity oyun motorunun yapısı ve kullanımı incelenmiştir. Motorun içinde kullanılan fizik, grafik gibi bileşenler tek tek incelenerek motorun çalışma mantığı öğrenilmiştir. Daha sonra oyun motorunun gerçek dünyadaki fizik kanunlarına olan yakınlığını ölçmek için çeşitli fizik deneyleri yapılmıştır. Bu deneyler ile yer çekimi, eylemsizlik, Newton kanunları ve sürtünme gibi en temel fizik kanunlarının Unity oyun motorunda doğru şekilde uygulandığı görülmüştür.

Çalışmanın ikinci aşamasında geliştirilecek olan simülatörlerin Simulink programından kontrolünün sağlanması amaçlanmıştır. Bu amaçla TCP ve UDP haberleşme protokolleri irdelenmiştir. Bu karşılaştırmanın ve yapılan testlerin sonucunda daha hafif ve daha hızlı olması sebebiyle UDP protokolünün kullanılmasına karar verilmiştir. Bu aşamanın sonunda hem Simulink hemde Unity oyun motoru tarafına gerekli yazılımlar yazılarak geliştirilecek olan simülatörlerin UDP protokolü ile Simulink ortamından kontrolleri sağlanmıştır. Yazılan yazılım çeşitli hız testlerine tabi tutulmuş ve isterlerin karşılandığına emin olunmuştur.

Çalışmanın üçüncü ve son aşamasında ise top ve plaka sistemi, ters sarkaç sistemi , mobil robot sistemi ve örnek bir vinç sisteminin Unity oyun motoru kullanılarak simülatörü tasarlanması amaçlanmıştır. Tüm simülatörlerin tasarımında benzer yazılım mimarisi, grafik arayüzü ve test yöntemleri kullanılmıştır. Öncelikle sistemlerin gerçek deney setleri incelenmiş ve sistemler alt mekanik bileşenlerine ayrılmıştır. Bu alt bileşenler tek tek Unity oyun motorunda gerçeklenerek sistemlerin bilgisayar simülatörü oluşturulmuştur. Daha sonra sistemlerin doğrusal olmayan modelleri Euler-Lagrange yöntemiyle elde edilmiştir. Elde edilen doğrusal olmayan modelin Simulink programında simülasyon modeli oluşturulmuştur. Son olarak doğrusal olmayan model ve geliştirilen simülatörlerin aynı sistem girişleri için sistem çıkışları karşılaştırılmıştır. Bu karşılaştırmalar sonucunda geliştirilen simülatörlerin gerçekçiliği irdelenmiş ve çeşitli yorumlar yapılmıştır.

SIMULATOR DESIGN WITH UNITY GAME ENGINE

SUMMARY

Nowadays, computer simulations are actively used in many engineering projects as well as in economics, biology, psychology and medicine. Thanks to computer simulators, many experiments, tests and training are carried out cheaply, quickly and without endangering any human life. Computer simulations can be developed using game engines or directly using low level graphics and physics libraries. This thesis study is intended to create simulators using Unity game engine of various experiment sets used in control and automation engineering training.

Unity is a cross-platform game engine released in June 2005 by Unity Technologies to develop video games and simulations for PCs, consoles, mobile devices. Unity enables the development of both 2D and 3D video games and simulations with the libraries and tools it contains. It is used in many game and simulation projects thanks to its easy usage and cross-platform structure. Various examples of simulators developed with Unity can be found in [2], [3] and [4]. Unity uses the high level Microsoft C # language as script language. As a graphics engine, Unity uses the DirectX library developed by Microsoft for Windows and Xbox platforms, and OpenGL open source graphics library for mobile devices. Nvidia Physx is used as a physics engine. Unity can be downloaded for free from its Internet address and can be used with Windows, MacOS and Linux operating systems.

In the second stage of the study, the main graphics and phsyics libraries of Unity game engine has been examined. Physx is a physics engine that was developed in 2004 by the company NovodeX at the ETH Zurich University. The physics engine was purchased in 2008 by NVIDIA, one of the leading graphic processor companies. It is currently being developed as an open source under NVIDIA. The most current version is Physx 4.0, which was released in December 2018, but is currently using version 3.4.1 in the Unity game engine. DirectX is a collection of APIs released by Microsoft in 1994 for multimedia processing on their platforms. DirectX also includes Direct2D, Direct3D and DirectWrite. Through these libraries, complex vector operations, 2D/3D dimensional graphics operations can be performed and computer graphics are obtained as a result of these operations. The most recent version is DirectX 12 released in October 2018. In the Unity game engine DirectX 12 or DirectX 11 can be used. OpenGL (Open Graphics Library) is an open-source graphics library published in 1992. OenGL is supported by different hardware manufacturers and platforms like Windows, Linux, MacOS. The most current version of OpenGL is released on 4.6 July 2017. In the Unity game engine, any version from OpenGL 3.5 to OpenGL 4.5 can be used.

In the third stage, it’s aimed that developing a method to provide control of simulators that developed in Unity from. Simulink is a model-based design environment that frequently used in the education and application of many engineering disciplines. Because of this intense usage, an intermediate

communication software has been written to control the simulation in Unity environment by Simulink. For this purpose, the TCP and UDP communication protocols are examined. TCP is a communication protocol thar created to send data in advanced computer networks without loss. It is ensured that the data sent in the TCP protocol arrives client at the other end. With this feature, TCP protocol is preferred for systems that need to communicate with high accuracy and reliability. UDP is a protocol that created for fast and efficient data communication in computer networks. There is no flow or error control in the UDP protocol. Therefore, it is not guaranteed that the sent data will reach the other end. However, in addition to this disadvantage, the UDP protocol can be implemented and operated faster than the TCP protocol. As a result of this comparison it was decided to use the UDP protocol which lighter and faster. At the end of this stage necessary softwares are written on Simulink and Unity to provide a communication and control between Simulink and Unity. with UDP protocol. Communication software has been subjected to various speed tests and it has been assured that the requirements have been met.

In the fourth and final stage, developing computer simulators for different dynamical systems is aimed. For this purpose Ball and Plate system is selected first. Ball and Plate simulator system is developed. Ball and Plate system is frequently used system in control research. The main purpose of this system is to control the position of the ball that placed on an angularly movable plate or to follow a desired path on the plate. Because, it is used frequently in control applications, this system is implemented in physics engine and it is aimed to compare the performance of physics engine with real world physics and create an experiment environment. At first, system is divided into sub-mechanical components. Than these sub-components are created Unity game engine one by one to create complete system in engine. For testing accuracy of simulator, nonlinear models of the systems are obtained by Euler-Lagrange method. Simulation of nonlinear model mathematical model is created in Simulin. The system outputs for system same inputs are compared. As a result of these comparisons, the realism of the simulators is discussed and various interpretations were made. Inverse pendulum system was developed as second simulator system. The inverse pendulum system is one of the most classical control problem. The main purpose of this system is to control the angle of the pendulum connected to a vehicle moving linearly in a single axis.Like ball and plate system, inverse pendulum system is also selected because of it is wide in control applications. Similar methodologies are used for the developing and testing simulator of inverse pendulum system with ball and plate system. As a result of tests and comparions, the realism of the inverse pendulum simulator is discussed and various interpretations were made.A gantry crane system is developed as third simulator. Cranes are defined as construction machines used for lifting and transporting large and heavy objects where manpower is not sufficient. Generally speaking, although the basic mechanics of the cranes are the same, the structures may vary according to the areas where they are used. In this section, a simulator and experimental set of a traveling bridge crane system has been developed in Unity environment. Similar methodologies are used for the developing and testing simulator of gantry crane simulator system with ball and plate and inverse pendulum systems. As a result of tests and comparions, the realism of the simulator is discussed and various interpretations were made. Lastly a mobile robot simulator is developed. Mobile robots are used in many robotics and control projects. Especially autonomous mobile robots moving as swarm is one of most common topic nowadays. In this simulator, it

is aimed to create a simulator environment in which one or more mobile robots are controlled.

As a result of this study, 4 different simulators of 4 different systems were developed by using Unity game engine. In order to use these simulators with other tools and programs like Simulink, a communication interface software have been written with UDP communication protocol. Nonlinear model of each system is obtained and simulations of these models are created in Simulink program. Then, for same system inputs, the outputs of nonlinear models and simulators were compared. As a result of these comparisons, it was investigated whether the simulators act in accordance with the phsyic laws in the real world. Simulators and nonlinear model simulations were found to give similar outputs for same inputs.So we can say that simulators In future, the first thing to do is to test the simulators with different system values.aAter obtaining sufficient data, the simulators should be tested with the actual models of the systems.In future studies, simulators of more complex systems like cars, drones, trains,and aircrafts can be developed in Unity game engine. Simulators of such systems can be made. In this way, it is possible to use these systems which are expensive and dangerous in real world experiment sets in academic studies.

1. GİRİŞ

Simülasyon, gerçek dünyadaki bir sistemin veya bir sürecin yaklaşık imitasyonu olarak tanımlanmaktadır [1]. Bir sürecin simülasyonunu yapmak için öncelikle sistemin davranışını ve karakteristiğini içeren bir model elde edilmesi gerekmektedir. Elde edilen bu modelin işletilmesi ise simülasyon olarak adlandırılmaktadır. Bilgisayar simülasyonu ise elde edilen modellerin bilgisayar ortamında işletilmesidir. Günümüzde bilgisayar simülasyonları inşaat, kimya, havacılık, uzay ve otomotiv gibi birçok mühendislik alanının yanında ekonomi, biyoloji, psikoloji ve tıp gibi alanlarda da aktif olarak kullanılmaktadırlar (Şekil 1.1). Bu yaygın kullanımı sağlayan avantajlar şu şekilde sıralanabilir:

- Gerçek dünyada gerçekleştirilmesi tehlikeli olan deneyler ve testler simülasyon ortamında güvenli bir şekilde gerçekleştirilebilir. Örnek olarak pilotların eğitimi için geliştirilen uçuş simülatörleri gösterilebilir. Bu sayede pilotlar eğitimlerini herhangi bir insan hayatını tehlikeye atmadan tamamlarlar.
- Gerçek dünyada gerçekleştirilmesi pahalı olan deneyler ve testler çok daha düşük maliyetle gerçekleştirilebilir. Günümüzde birçok firma uzay programları kapsamında ürettikleri robotların ilk testlerini simülatörler üzerinden yapmaktadır. Bu şekilde robotu uzaya göndermeden bir çok testini dünya üzerinde çok daha ucuza mal etmektedirler.
- Bilgisayar simülasyonları zamandan ve mekandan bağımsızdır. Gerçek dünyada test edilmesi çok uzun sürecek sistemler bilgisayar simülasyonları ile çok daha kısa sürede test edilebilirler.
- Bilgisayar simülasyonları çok geniş bir şekilde özelleştirilebilir. Simülasyonun birçok parametresi değiştirilebilir. Bu sayede test edilen sistemin çok daha detaylı bir analizi yapılabilir ve gerçek sistemde oluşabilecek belirsizliklere karşı daha dayanıklı mühendislik sistemleri üretilebilir.

Bilgisayar simülasyonları, simülasyonun kullanılacağı alana göre Unreal Engine, Unity gibi oyun motorlarıyla geliştirilebildiği gibi daha düşük seviye olan PhsyX, Havok, ODE, Bullet ve Simbody gibi fizik kütüphanelerinin doğrudan kullanılmasıyla da geliştirilebilir.

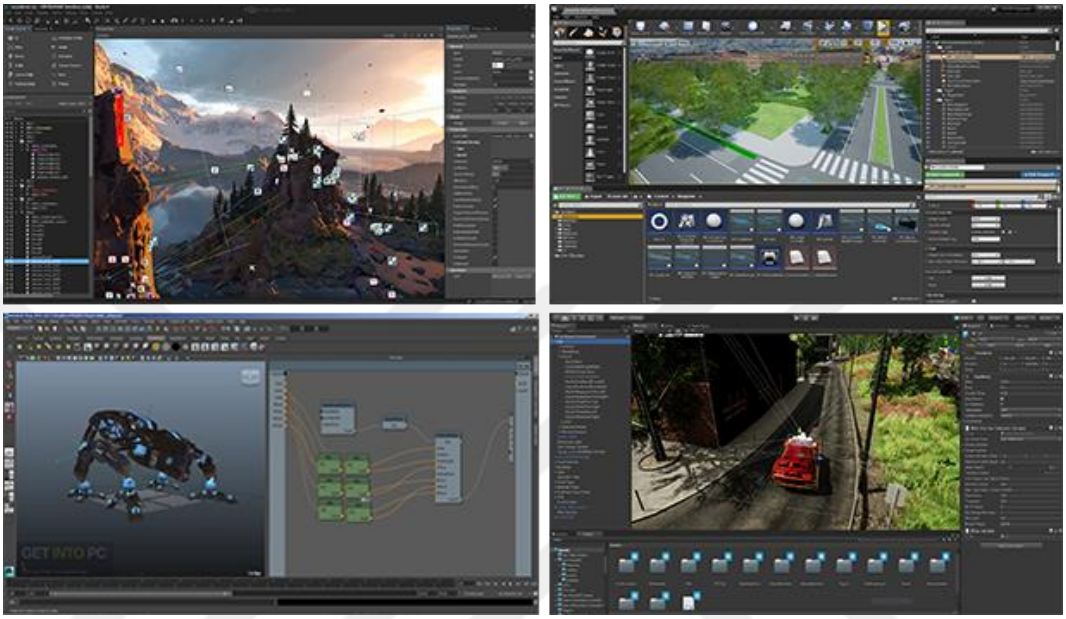
Şekil 1.1 : Boeing firmasına ait uçuş simülatörleri.

Bu bitirme projesinde Unity oyun motoru kullanılarak çeşitli fiziksel sistemlere ait gerçek zamanlı eğitim simülatörleri geliştirilmesi amaçlanmıştır. Bu sistemler oyun motoru ortamında oluşturulduktan sonra, sistemlere ait doğrusal olmayan modeller elde edilmiş ve oyun motorundaki oluşturulmuş simülasyonlar ile doğrusal olmayan modellerin Simulink ortamındaki simülasyonu karşılaştırılmıştır. Bu karşılaştırma sonucu yapılan simülatörlerin doğruluğu irdelenmiştir. Aynı zamanda bu sistemler üzerinde deneylerin yapılabilmesi için öğrenciler tarafından sıklıkla kullanılan Simulink yazılımı ile Unity ortamındaki simülatörlerin gerçek zamanlı olarak kontrol edilebilmesi için haberleşme arayüz yazılımları yazılmıştır.

1.1 Oyun Motorları ve Unity Oyun Motoru

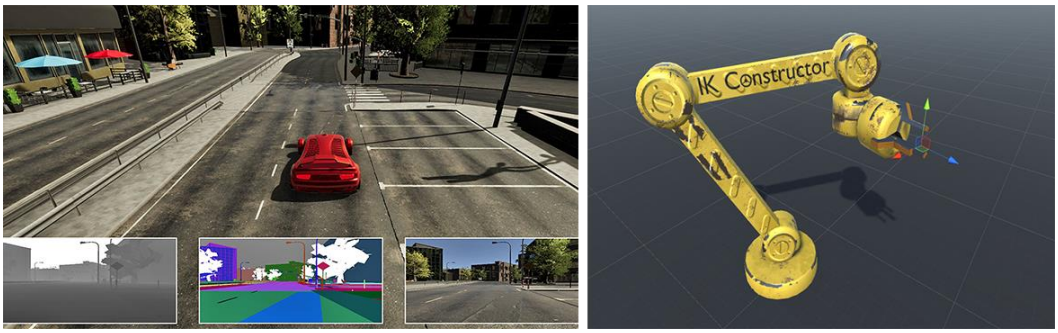
Oyun motorları hızlı ve etkili şekilde video oyunları ve simülasyon yazılımlarını geliştirmek için bünyesinde grafik, ses, fizik motoru, networking gibi birçok farklı

konuda kütüphaneyi bulunduran yazılımlara verilen genel isimdir. 1980'li yıllarda yapılan her oyun ve simülasyon yazılımları sıfırdan kod yazılarak geliştirilmekte idi. 1990'lı yıllara gelindiğinde ise geliştiriciler tekrar tekrar kullanılan kütüphanelere bir framework çatısı altında birleştirerek günümüzdeki oyun ve simülasyon motorlarının temelini oluşturdular.Farklı oyun motorlarına ait ekran görüntüleri şekil 1.2'de görülebilir.



Şekil 1.2 : Çeşitli oyun motorlarının geliştirme ortamları.

Unity pc, konsol, mobil cihazlar için video oyunları ve simülasyonları geliştirmek için Unity Technologies firması tarafından Haziran 2005 tarihinde ilk sürümü yayınlanmış çapraz platform bir oyun motorudur. Unity içerdiği kütüphaneler ve araçlar sayesinde hem 2D hem 3D video oyunu ve simulasyonu geliştirilmesine olanak sağlamaktadır. Kolay kullanımı ve çapraz platform olması sayesinde bir çok oyun ve simülasyon projesinde kullanılmaktadır (Şekil 1.3). Unity ile geliştirilen çeşitli simülatör örnekleri [2], [3] ve [4]'te görülebilir. Unity script dili olarak yüksek seviyeli olan Microsoft' C# dilini kullanmaktadır. Grafik motoru olarak ise Windows ve Xbox platformları için yine Microsoft'un geliştirdiği DirectX kütüphanesini, Mobil cihazlar için ise OpenGL açık kaynaklı grafik kütüphanesini kullanmaktadır. Fizik motoru olarak ise Nvidia Physx kullanmaktadır. Unity ücretsiz olarak kendi internet adresinden indirilebilir Windows, MacOS ve Linux işletim sistemleri ile kullanılabilir.



Şekil 1.3 : Unity ile geliştirilmiş Microsoft Airsim ve IK Constructor simülatörü.

1.1.1 Nvidia Physx fizik motoru

Physx, 2004 yılında ETH Zurich üniversitesi bünyesinde olan NovodeX adlı firma tarafından geliştirilmeye başlanmış bir fizik motorudur. Fizik motoru, 2008 yılında grafik işlemci sektörünün önde gelen firmalarından olan NVIDIA firması tarafından satın alınmıştır. Şu an açık kaynaklı olarak NVIDIA çatısı altında geliştirilmeye devam edilmektedir. En güncel sürümü Aralık 2018'de yayınlanan Physx 4.0'dır fakat Unity oyun motorunda şu anda 3.4.1 sürümü kullanılmaktadır.

1.1.2 DirectX ve OpenGL grafik kütüphaneleri

DirectX Microsoft firması tarafından kendi platformlarında multimedia işleyen yazılımlarda kullanılmak üzere ilk versiyonu 1994 yılında yayınlanmış bir API topluluğudur. DirectX içeresin de Direct2D, Direct3D ve DirectWrite gibi bir çok alt kütüphane barındırmaktadır. Bu kütüphaneler aracılığıyla karmaşık vektör işlemleri, 2D/3D boyutlu grafik işlemleri yapılabilir ve bu işlemlerin sonucunda bilgisayar grafikleri elde edilir. En güncel sürümü Ekim 2018'de yayınlanan DirectX 12'dir ve şuan Unity oyun motorunda DirectX 12 veya DirectX 11 sürümleri kullanılabilmektedir.

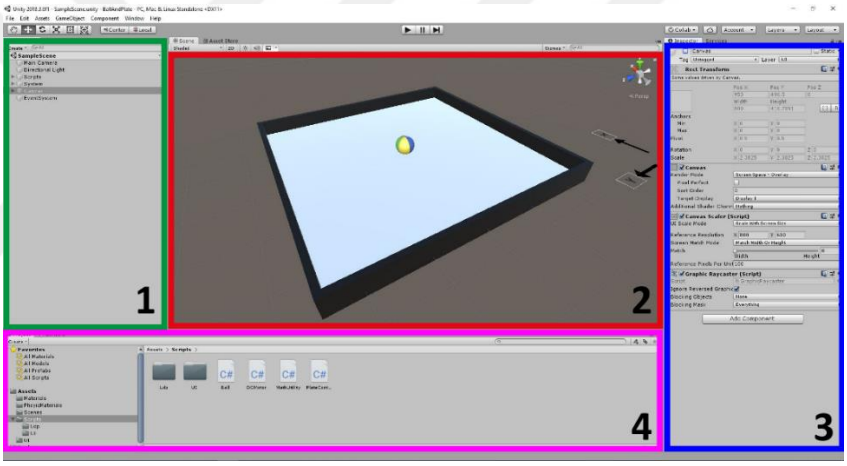
OpenGL (Open Graphics Library) ilk versiyonu 1992 yılında yayınlanmış açık kaynaklı bir grafik kütüphanesidir. Birçok donanım üreticisi ve Windows, Linux, MacOS farklı platformlar tarafından desteklenmektedir. Şuan en güncel sürümü olan OpenGL 4.6 Temmuz 2017 tarihinde yayınlanmıştır. Unity oyun motorun da ise kullanılan platformun donanımına bağlı olarak OpenGL 3.5'ten OpenGL 4.5'e kadar olan sürümlerden herhangi biri kullanılabilir.

2. UNITY OYUN MOTORUNUN TEMEL BİLEŞENLERİ

Unity oyun motoru birçok farklı kütüphaneden bir çok farklı bileşeni (komponenti) birleştirmektedir. Bu bileşenler bir kullanılarak farklı türlerde oyun ve simülasyon yazılımları geliştirilebilir.

2.1 Ana Menüler

Unity oyun motorunun ana ekranı şekil 2.1'te görüldüğü gibidir.



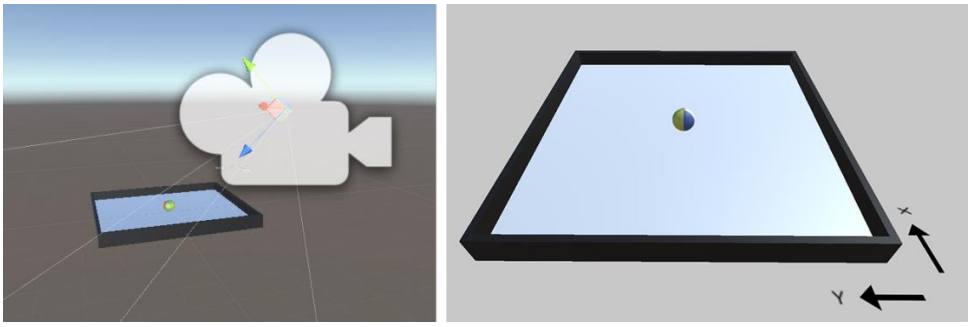
Şekil 2.1 : Unity oyun motorunun ana ekranı.

Unity projesi şekil 2.1'de görülen bu 4 panel üzerinden geliştirilmektedir. 1 numaralı panel simülasyon içindeki objeler listelendiği Hierarchy paneli, 2 numaralı panel üç boyutlu simülasyon ortamının görüntülendiği Scene paneli, 3 numaralı panel Hierarchy panelinde seçilmiş olan objenin bileşenlerinin yönetildiği Inspector paneli ve 4 numaralı panelde proje dosyalarının yönetildiği Project panelidir.

2.2 Sahne ve Objeler

Her Unity projesi en az bir sahneden (Scene) oluşmaktadır yapılan simülasyonun tüm objeleri bu sahnelerde içerisinde yer almaktadır. Sahne içerisinde yer alan her şey obje (Object) olarak adlandırılır. Her sahnede en az bir kamera (Camera) objesi yer

almaktadır. Kullanıcı kamera objesi aracılığıyla simülasyonun içini görür. Yani kamera, kullanıcının simülasyon dünyasına açılan penceredir (Şekil 2.2).



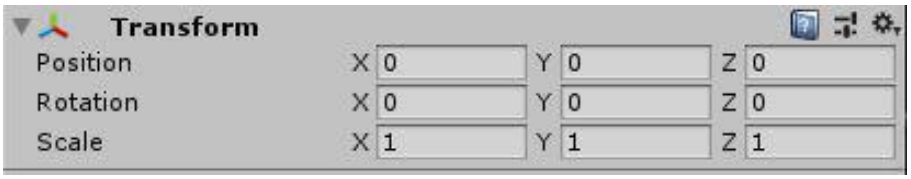
Şekil 2.2 : Solda kamera objesi ve görüş açısı, simülasyonda görülen alan.

2.3 Bileşenler (Komponentler)

Unity ortamında oluşturulan her objeye bir çok farklı bileşen eklenebilir veya çıkarılabilir. Objeler bu bileşenler sayesinde 3D-2D grafik, ses, kütle, hacim gibi bir çok özellik kazanabilir.

2.3.1 Transform Bileşeni

Transform bileşeni sahne içerisindeki objenin konumunu, duruş açılarını ve ölçeğinin değiştirilmesini sağlar (Şekil 2.3). Transform bileşeninde gösterilen değerler metre ve derece cinsindendir.



Şekil 2.3 : Transform bileşeninin Inspector panelindeki görüntüsü.

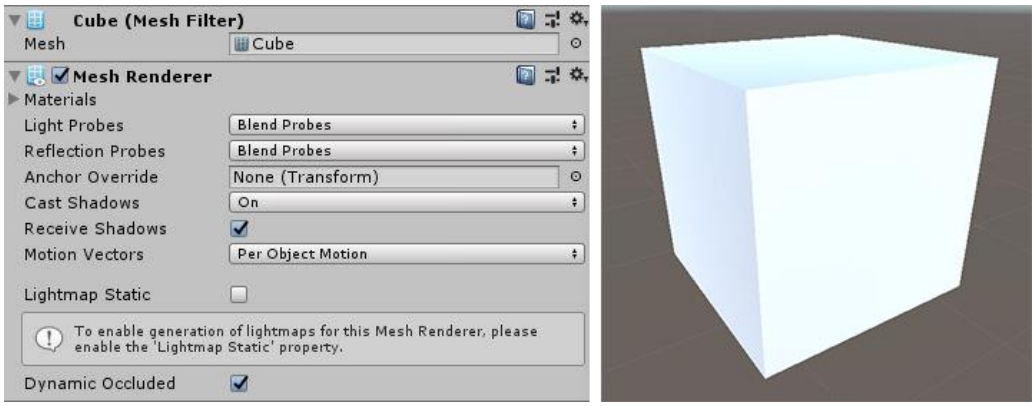
Sahneye eklenen her obje içine Transform bileşeni otomatik olarak atanır ve bu bileşen objelerden silinemez.

2.3.2 Üç ve iki boyutlu grafik bileşenleri

2.3.2.1 Mesh Renderer ve Mesh Filter

Renderer bileşenleri bir objenin ekranda gözükmesini sağlar. Örneğik AutoCAD'de modellenmiş bir .obj grafik dosyasının simulasyon ekranına aktarılmasında

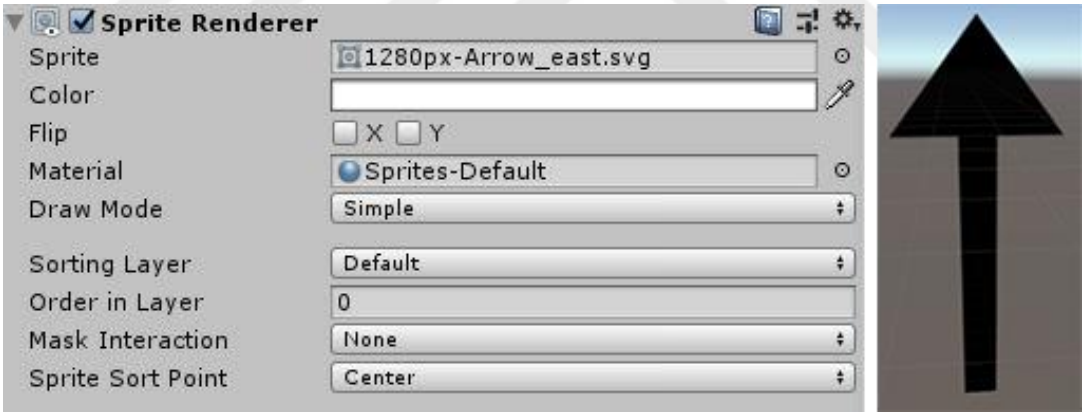
kullanılır. Çizilen obje şekil 2.4'teki Mesh Filter bileşenindeki "Mesh" ayarına eklenir.



Şekil 2.4 : Renderer ve Filter bileşenlerinin simulasyon ortamındaki görüntüsü.

2.3.2.2 Sprite Renderer

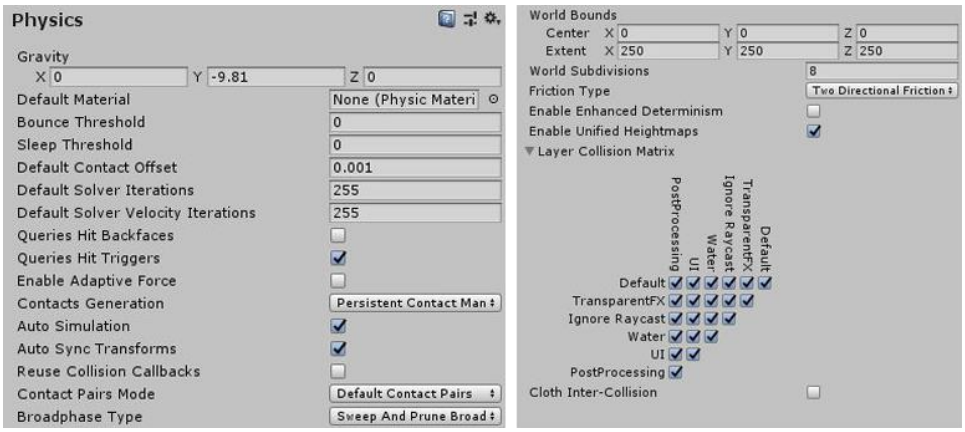
Sprite Renderer bileşeni üç boyutlu cisimler için kullanılan Mesh Renderer dan farklı olarak iki boyutlu grafiklerin simülasyon sahnesine aktarılmasını sağlamaktadır (Şekil 2.5). Çeşitli formatlardaki imaj dosyaları Sprite Renderer bileşeni sayesinde simülasyon ortamına taşınabilir.



Şekil 2.5 : Sprite Renderer bileşeni ve simulasyon ortamındaki görüntüsü.

2.3.3 Fizik yöneticisi ve fizik bileşenleri

Fizik yöneticisi yardımıyla Unity içerisindeki fizik motorunun çalışma şekline müdahale edilebilir (Şekil 2.6). Fizik yöneticisi menüsünden örnekleme süresi, çözücülerin iterasyon sayıları ve hızları, sürtünme hesaplamasının, enerji korunumu, yer çekimi kuvveti, çarpışma algılıyıcılarının hassasiyetleri gibi fizik motorunun temeline ait ayarlar yapılabilir. Bu sayede fizik motorunun daha gerçekçi davranışlar ile performans arasındaki dengesi ayarlanabilir.

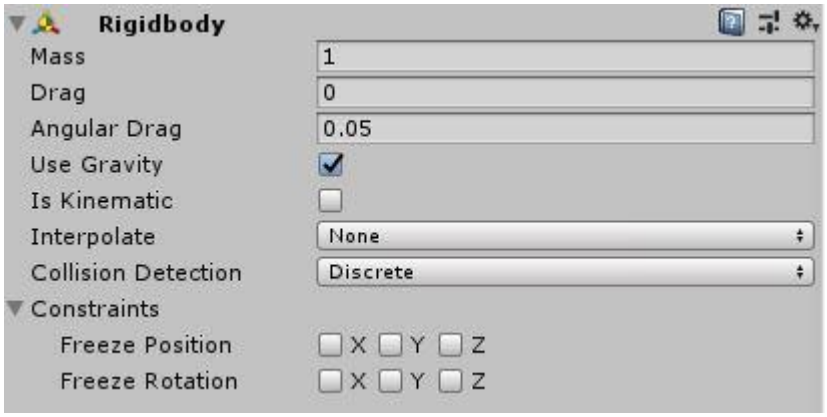


Şekil 2.6 : Fizik yöneticisi menüsü.

Burada oluşturulan ayarlar fizik bileşenleri kullanan objelerin hareketlerinin hesaplanmasında kullanılmaktadır. Fizik bileşenleri sahnede oluşturulan objelere gerçek dünyada olan kütle, hacim, eylemsizlik gibi özelliklerin eklenmesi sağlar. Yapılan fizik simülasyonlarında sistemlerin gerçek dünyadaki gibi hareket edebilmeleri bu bileşenler yardımıyla sağlanmıştır. Unity içerisinde Rigidbody, Collider, Joints, Physic Material gibi birçok fizik bileşen mevcuttur.

2.3.3.1 Rigidbody

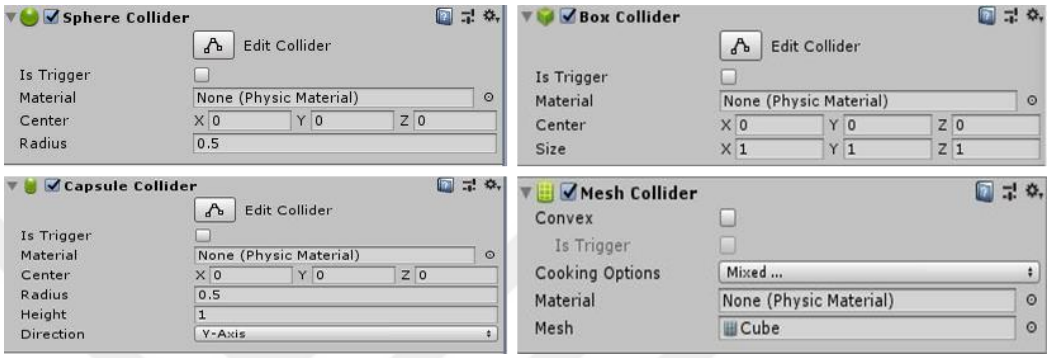
Rigidbody atandığı objenin fizik motoru tarafından kontrol edilmesini sağlayan temel bileşendir (Şekil 2.7). Rigidbody atanan objeler bir kütleye sahip olurlar ve yerçekimi kuvvetinden etkilenir hale gelirler. Objelerin eylemsizlik momentleri fizik motoru tarafından otomatik olarak hesaplanır ve kütlesi olan her cisim gibi eylemsizliğe sahip olurlar. Bu objelere ayrıca kuvvet uygulanabilir. Fakat Rigidbody bileşeninin atandığı objelerin diğer objeler ile temasa geçebilmeleri (Çarpışma mekaniği) için Collider bileşeni de gereklidir.



Şekil 2.7 : Rigidbody bileşeni.

2.3.3.2 Collider

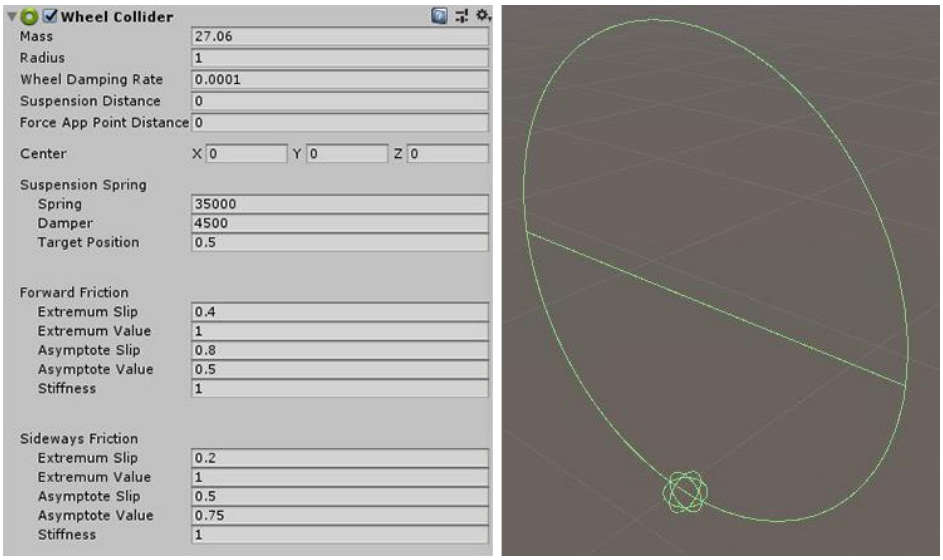
Collider bileşeni objelerin diğer objeler ile fiziksel olarak etkileşime geçebilmelerini sağlar. Gerçekçi davranış için collider bileşeninin şeklinin objenin şekli ile aynı olması gerekmektedir. Unity içerisinde farklı şekillerdeki cisimler için Box Collider, Sphere Collider, Capsule Collider, Mesh Collider ve Wheel Collider olmak üzere 5 farklı collider bileşeni mevcuttur. Her collider bileşeninin kendine özgü ayarları vardır (Şekil 2.8).



Şekil 2.8 : 4 farkli Collider bileşeni ve çeşitli ayarları.

2.3.3.3 Wheel Collider

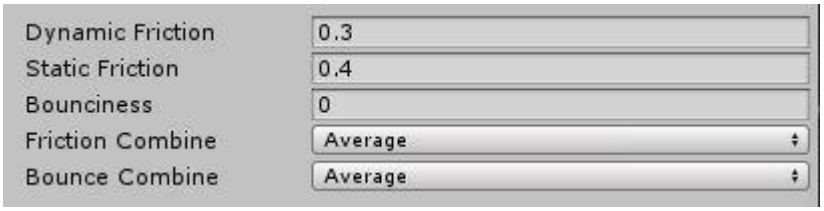
Wheel Collider bileşeni tekerlek simülasyonu yapmak için kullanılmaktadır. Bu komponentin fizik motoru tarafında hesaplanması diğer tüm komponentlerden farklı olarak gerçekleşmektedir (Şekil 2.9).



Şekil 2.9 : heel Collider ayarları ve simülasyon ekranındaki görüntüsü.

2.3.3.4 Fizik Materyali

Yüzeylerin sürtünmesi veya cismin bir noktaya çarpınca sekmesi "Collider" komponentine atanan "Physic Material" adlı komponent ile sağlanır (Şekil 2.10).



Şekil 2.10 : Fizik materyali ayarları.

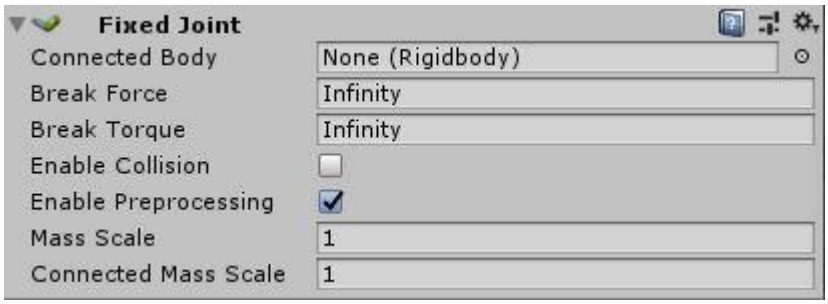
Phsyx fizik sürtünme işlemini simüle etmek için Coulomb sürtünme modelini kullanır. Bu modelde sürtünme statik ve dinamik katsayıları kullanılarak hesaplanır. Statik sürtünme katsayısı cisimler arasında kayma yok iken, dinamik sürtünme katsayısı ise cisimler arasında kayma oluştuğu andan itibaren kullanılır. Physic Material bileşeninde statik ve dinamik sürtünme katsayıları ayarlanabilir. Burada dikkat edilmesi gereken mesele Unity motorunun "Fizik Yöneticisi" menüsünden yapılacak "Friction Type" ayarıdır. Gerçek dünyadaki sürtünme modeline en yakın olan "Two Directional Friction Type" modeli seçilmelidir. Aksi halde fizik materyalin üzerinde ayarlanan katsayılar gerçek dünyanın yarısı kadar etki edecektir. Cismin sekmesi ise Fizik materyali üzerindeki "bounciness" katsayısı ile ayarlanır. Bu katsayının 1 olması cismin enerji kaybetmeden yüzeyden sekmesine neden olurken 0 olması cismin hiç sekmemesini sağlar.

2.3.3.5 Joints

Joint bileşenleri Unity ortamında eklem işlevi görerek Rigidbody bileşenine sahip objelerin birbirlerine bağlanmalarını veya belirli serbestlik derecelerinde beraber hareket etmelerini sağlar.

Fixed joint

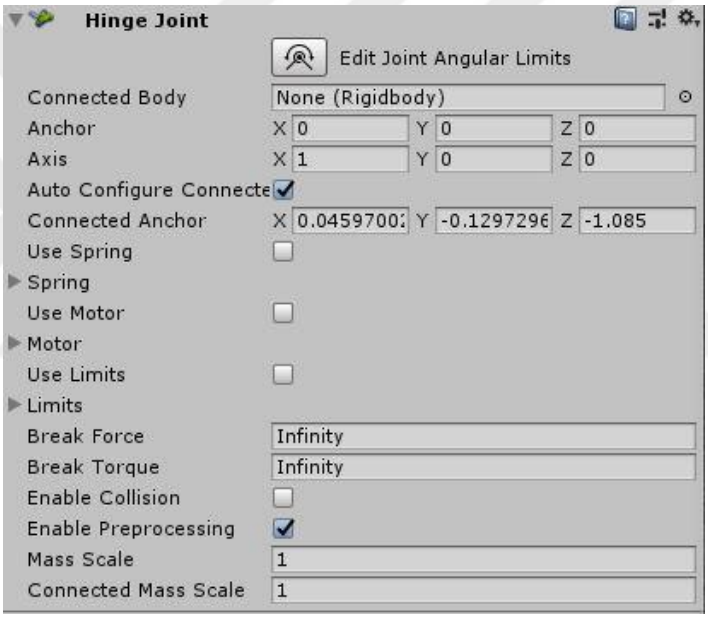
Fixed joint bileşeni bir objenin başka bir obje ile katı bir şekilde bağlanmasını sağlar. Yani objeler farklı kütlelere hacimlere sahip olmalarına rağmen sanki birbirlerine yapıştırılmış gibi davranırlar (Şekil 2.11).



Şekil 2.11 : Fixed Joint bileşeninin ayarları.

Hinge joint

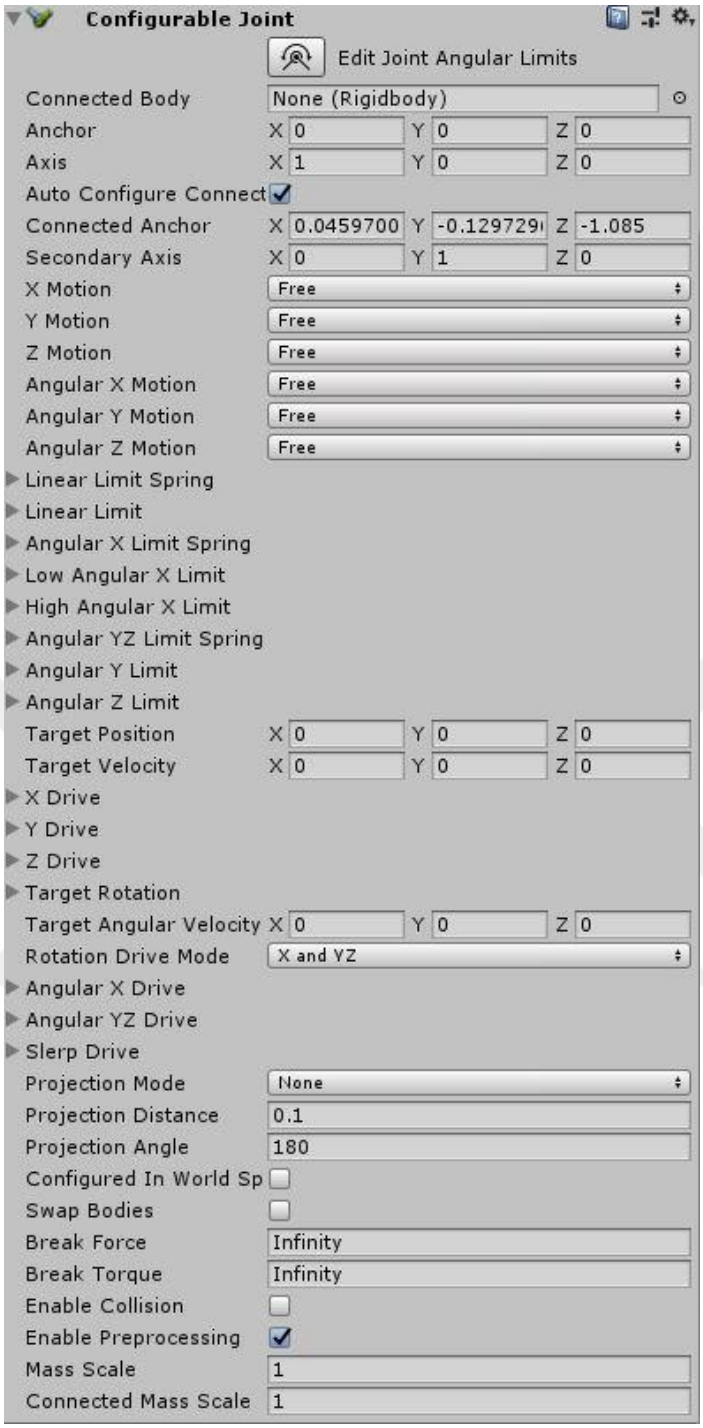
Hinge joint bileşeni 2 Rigidbody'nin birbirlerine bir menteşe ile bağlıymış gibi hareket etmelerini sağlar. Bu bileşen kullanılarak kapı, sarkaç, zincir gibi sistemler fizik motoru ortamında gerçeklenebilirler (Şekil 2.12).



Şekil 2.12 : Hinge joint bileşeni ayarları.

Configurable joint

Configurable Joint diğer tüm joint tiplerine çevrilebilecek şekilde ayarlanabilen bir joint tipidir (Şekil 2.13). Örneğin bazı mobil robotlarda bulunan her eksende dönebilen yuvarlak tekerlekler Configurable Joint bileşeni kullanılarak gerçekleştirilebilir. Bu bitirme projesinde birden fazla eksende rotasyonel olarak hareket etmesi gereken cisimlerin bağlantısı için kullanılmıştır.



Şekil 2.13 Configurable Joint bileşeni ayarları.

2.3.4 UI Bileşenleri

Unity ortamında UI bileşenleri kullanılarak kullanıcı arayüzleri oluşturulabilir. Bu ara yüz aracılığıyla kullanıcının simülatör ile etkileşime geçmesi , yönetmesi, çeşitli bilgileri ekran üzerinden okuyabilmesi sağlanabilir. Simulasyon ekranındaki test sisteminin verilerinin gösterildiği alanlar UI bileşenleri kullanılarak yapılmıştır.

2.3.4.1 Canvas

Unity ortamındaki tüm kullanıcı arayüzü bileşenleri Canvas bileşeninin alt objesi olarak simülasyon ortamına eklenirler. Canvas, simülasyon bilgisayarının ekran çözünürlüğüne bağlı olarak çeşitli çözünürlüklerde ayarlanabilir.

2.3.4.2 Image

Çeşitli formatlardaki imaj dosyalarının UI ekranında gözükmesini sağlar.

2.3.4.3 Button

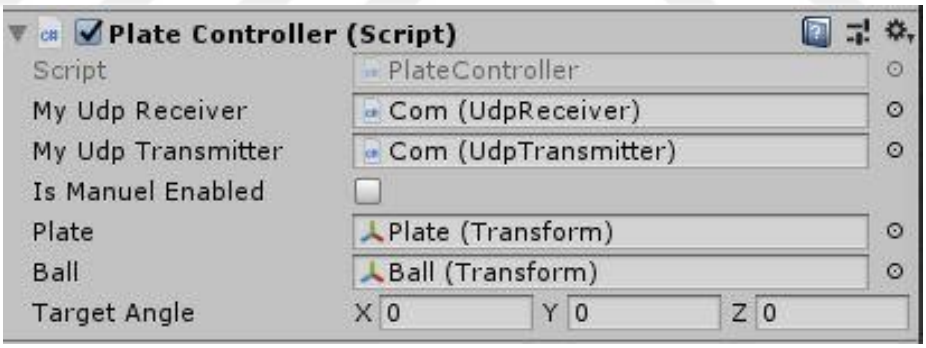
Kullanıcın tıklayarak işlem yapabileceği standart buton bileşenidir.

2.3.4.4 Text

Belirli bir font'a sahip yazıların UI ekranında gösterilmesini sağlar.

2.3.5 Script Bileşeni

Unity oyun motorunda C# dilinde .Net ortamında yazılmış scriptler sayesinde simulasyon kontrol edilir (Şekil 2.14). Bu scriptler aracılığıyla simülasyonun ve simülasyon içerisindeki objelerin davranışları kontrol edilebilir.



Şekil 2.14 : Top ve Plaka sistemi için yazılmış C# scriptinin ayarları.

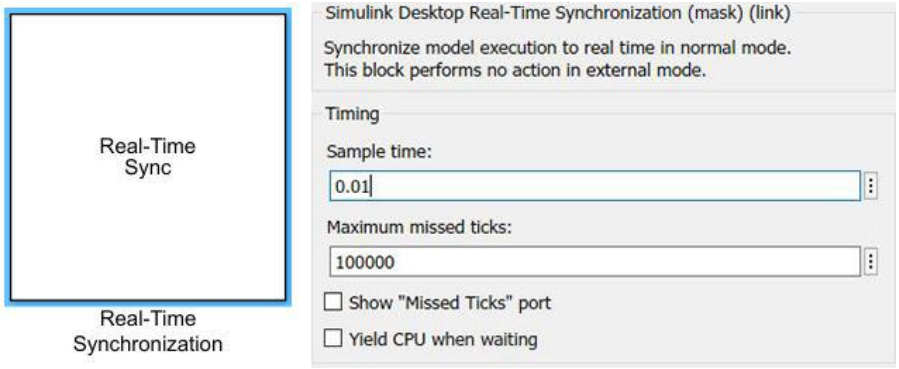
Tezin daha sonraki başlıklarında gösterilecek olan Simulink Unity arası yazılan haberleşme yazılımları bu Scriptler kullanılarak yazılmıştır. Bu sayede Simulink'ten gelen veriler sahnedeki objelere iletilmiş ve objelerin sistem girişleri doğrultusunda hareket etmesi sağlanmıştır.

3. SIMULINK VE UNITY HABERLEŞME ARAYÜZ YAZILIMI

Simulink birçok mühendislik dalının eğitiminde ve uygulamasında sıklıkla kullanılan model tabanlı bir tasarım ortamıdır. Bu yoğun kullanımı nedeniyle Unity ortamındaki simulasyonun Simulink tarafından kontrol edilebilmesi için bir ara haberleşme yazılımı yazılmıştır.

3.1 Simulink Realtime

Unity ortamında gerçekleştirilen simülasyonlar gerçek zamanlı olarak çalışmaktadır. Dolayısıyla Simulink ortamında kurulacak olan kontrol şemalarında gerçek zamanlı olarak çalışması gerekmektedir. Bu amaçla Simulink "Real-Time Synchronization" bloğu kullanılmıştır (Şekil 3.1).



Şekil 3.1 : Realtime bloğu ve ayarları.

"Real-Time Synchronization" bloğu şekil 3.1'de belirtilen örnekleme zamanına göre sistemi gerçek zamanlı olarak çalışacak şekilde senkronize eder. Örneğin simülasyon süresi 10s olarak ayarlandıysa simülasyonun tamamlanması gerçek dünyadaki 10s'lik zaman diliminde gerçekleşir. Simulink realtime bloğu ayrık olarak çalışmaktadır. Dolayısıyla simülasyon için bir örnekleme zamanı belirlenmelidir. Bu bitirme çalışmasında yapılan simülasyonlarda örnekleme zamanı 0.01sn seçilmiştir.

3.2 TCP ve UDP Haberleşme Protokolleri

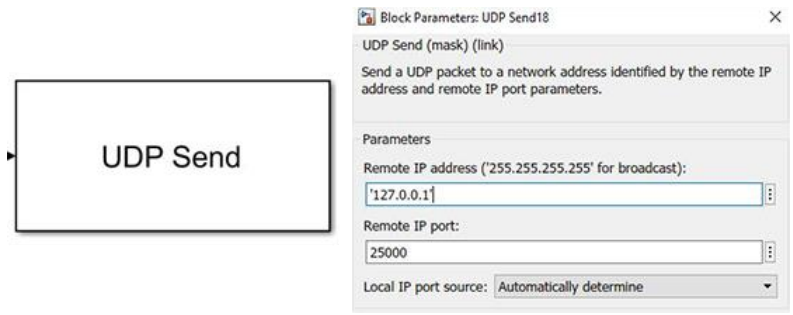
Unity ve Simulink arasındaki veri haberleşmesinin sağlanması için yapılan testlerde TCP (Transmission Control Protoco) ve UDP (User Datagram Protocol) haberleşme protokol öne çıkmıştır.

TCP gelişmiş bilgisayar ağlarında kayıpsız bir şekilde veri göndermek amacıyla oluşturulmuş bir iletişim protokolüdür. TCP protokolünde gönderilen verinin karşı tarafa gönderilen sırada ulaştığından emin olunur. Bu özelliği sayesinde yüksek doğrulukta ve güvenilirlikte haberleşmesi gereken sistemler için TCP protokolü tercih edilir.

UDP bilgisayar ağlarında hızlı ve etkili veri haberleşmesi için oluşturulmuş bir protokoldür. UDP protokolünde herhangi bir akış veya hata kontrolü yoktur. Bu nedenle gönderilen verinin karşıya tarafa ulaşması veya birden fazla verinin gönderildiği sırada ulaşması garanti edilmez. Fakat bu dezavantajının yanında UDP protokolü TCP protokolüne göre daha hızlı bir şekilde yapılabilir. Bu nedenle Gerçek-Zamanlı iletişimin daha önemli olduğu uygulamalarda UDP protokolü tercih edilmektedir. Unity ve Simulink arasındaki haberleşmenin gerçek zamanlı olarak gerçekleşmesi önemli olduğu için UDP protokolü tercih edilmiştir.

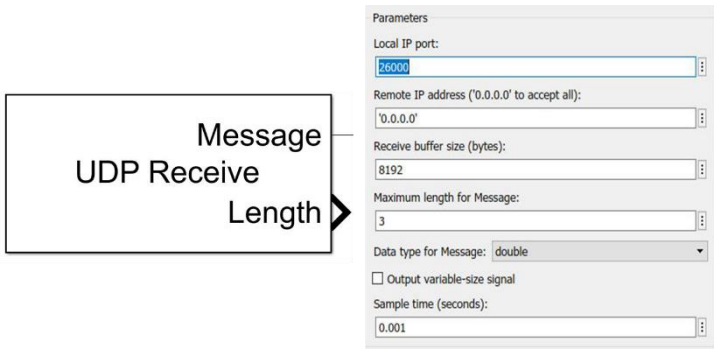
3.3 UDP Protokolü Arayüz Yazılımı

Her iki geliştirme ortamındada UDP protokolünün çalışabilmesi için gerekenler ayrı şekilde yapılmıştır. Öncelikle, simulink ortamında “UDP Send” ve “UDP Receive“ blokları kullanılmıştır (Şekil 3.2).



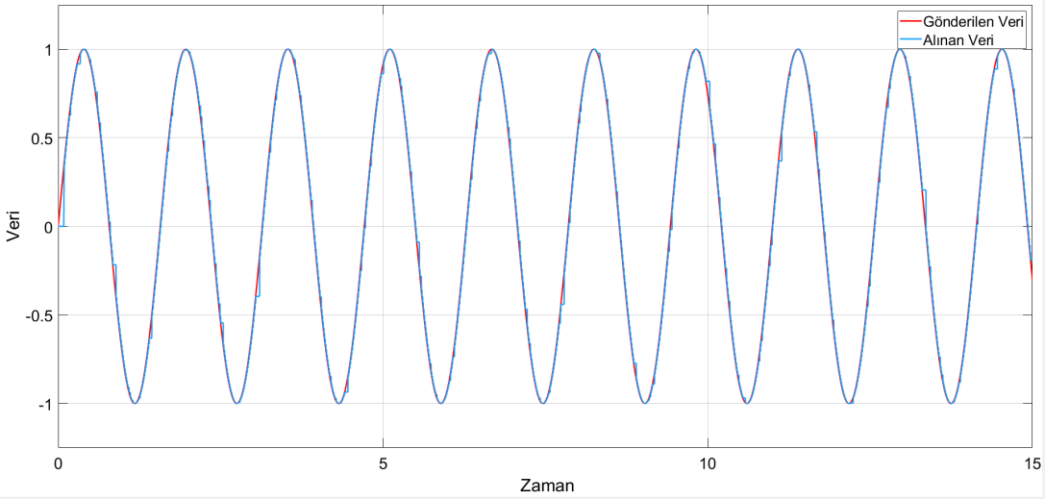
Şekil 3.2 : UDP Send bloğu ve ayarları.

Unity ortamında yazılan tüm simülasyonların alıcı portu 25000 verici portu ise 26000 olarak ayarlanmıştır.



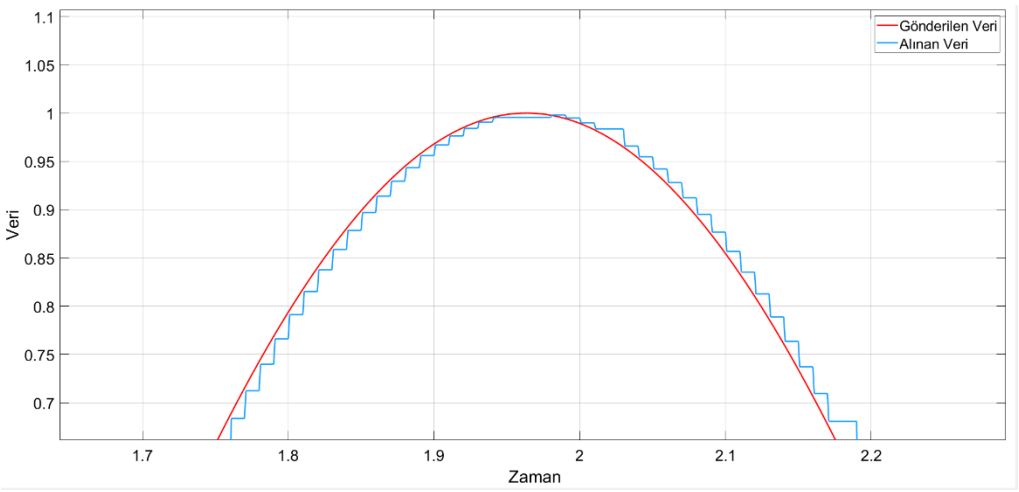
Şekil 3.3 : UDP Receive bloğu ve ayarları.

Unity ortamında ise daha önceden bahsedildiği gibi C# yazılım dili kullanılarak . NET geliştirme ortamında UDP veri haberleşmesi için gerekli olan yazılım yazılmıştır. İki geliştirme ortamında ayarlandıktan sonra veri haberleşmesinin testi aşamasına geçilmiştir. Bu amaçla çeşitli giriş değerleri UDP üzerinden Unity ortamına gönderilmiş Unity ortamından da gelen verinin tekrar Simulink ortamına geri gönderilmesi sağlanmıştır. Bu şekilde arada oluşan veri kayıpları ve gecikmeleri hakkında fikir edinmek istenmiştir.



Şekil 3.4 : Sinün sinyali olarak gönderilen ve alınan verinin karşılaştırılması.

Şekil 3.4'de görülebileceği üzere 15s boyunca uygulanan sinüs sinyalinin geri dönüşü yeterli hızda ve doğrulukta gerçekleşmektedir. Sinyalin başındaki olan hafif gecikme Simulink ortamında UDP send ve Receive bloklarının başlama gecikmesinden kaynaklanmaktadır.



Şekil 3.5 : Şekil 3. 4'teki grafiğin yakınlaştırılmış görüntüsü.

Şekil 3.5'de görülebileceği üzere gönderilen ve alınan sinyal arasındaki fark 0.01sn yani Simulink Realtime bloğu ile ayarladığımız örnekleme süresinden kaynaklanmaktadır.

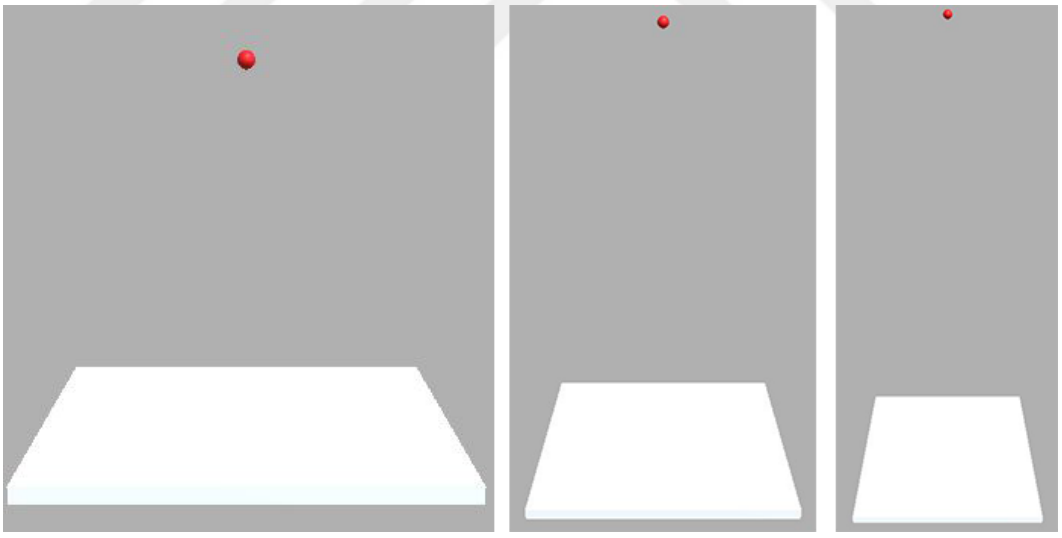
4. SİMÜLASYON TASARIMI VE UYGULAMALARI

4.1 Temel Fizik Testleri

Fizik motorunun çalışmasını daha iyi kavrayabilmek ve motorun temel fizik kurallarına ne seviyede uyduğunun kavranabilmesi amacıyla çeşitli testler yapılmıştır.

4.1.1 Yer Çekimi

Rigidbody başlığında anlatıldığı üzere, Rigidbody bileşeni eklenen cisimlere yer çekimi kuvveti uygulanmaktadır. Simülasyonda yüksekten bırakılan cisim yerçekimi ivmesiyle hızlanmaktadır. Bu ivmelenmenin doğruluğunu test etmek için simülasyondaki belirli yüksekliklere küpler yerleştirilmiştir (Şekil 4.1).



Şekil 4.1 : Farklı yüksekliklerden bırakılan toplar için hazırlanan sahneler.

Bu küpler bulundukları yükseklikten serbest bırakılmış ve yere varma süreleri ölçülmüştür. Alınan sonuçlar gerçek dünyada sürtünmesiz ortamda alınması gereken sonuç ile karşılaştırılmıştır. Çizelge 4.1'de simülatördeki sonuçlar ve gerçek dünyadaki değerlerin karşılaştırılması görülebilir.

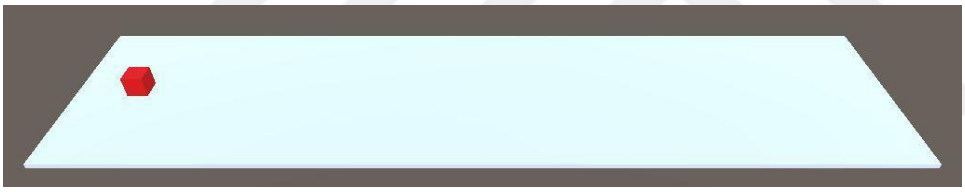
Çizelge 4.1 : Simülatörde alınan değerler ile gerçek değerlerin karşılaştırılması.

Yükseklik	Unity	Gerçek Değer
20m	2.0204 sn	2.01927 sn
40m	2.8645 sn	2.8556 sn
60m	3.5000 sn	3.4974 sn
120m	4.9500 sn	4.9461 sn

Çizelge 4.1'de ki sonuçlarda görülebileceği üzere ölçülen ile olması gereken arasında kümülatif olmayan çok küçük bir fark çıkmıştır. Bu hata muhtemelen fizik motorunun bir sonraki çevriminde değeri göstermesinden kaynaklanmaktadır.

4.1.2 Newton'un hareket kanunu ve sürtünme

Newton'un ikinci yasası cisimlerin hareketlerini ifade eden en temel fizik yasalarından birisidir. Oyun motorunun bu yasalara uygun olarak hareket ettiğini doğrulamak için kullanılmıştır. Oluşturulan simülasyonda 2 farklı ağırlıklı obje için farkli kuvvet uygulanmıştır (Şekil 4.2).



Şekil 4.2 Kuvvet uygulanan kutu ve sürtünmeli yüzey ile oluşturulan sahne.

Ayrıca yüzeylere sürtünme katsayısıda (μ_s) 0.2 olarak eklenerek cisimlerin hareket etmeye başladıkları nokta ile durduğu mesafeler ile gerçek olması gereken değerler karşılaştırılmıştır (Çizelge 4.2).

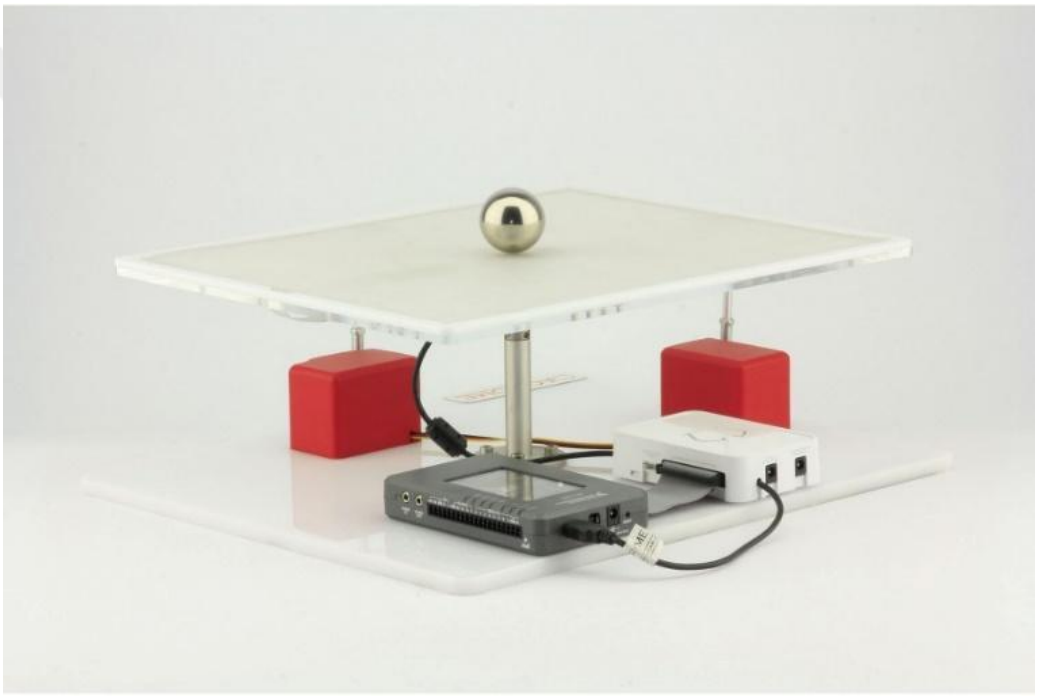
Çizelge 4.2 : Simülatörde alınan değerler ile gerçek değerlerin karşılaştırılması.

Ağırlık (kg)	Kuvvet (N)	Unity Mesafe (m)	Gerçek Değer (m)
1	10	25.4346	25.4845
1	50	636.8727	637.1049
1	100	2547.75	2548.4199
10	10	0.2497	0.2548
10	50	6.3464	6.3710
10	100	25.4338	25.4841

Çizelge 4.2’de ki sonuçlardan görülebileceği üzere ölçülen ile olması gereken arasında çok küçük bir fark çıkmıştır. Bu hatanında kümülatif olmadığı değerlerden görülebilmektedir. Dolayısıyla bu hatanın kaynağıyla ilgili daha önce varılan kanı bu test içinde geçerlidir.

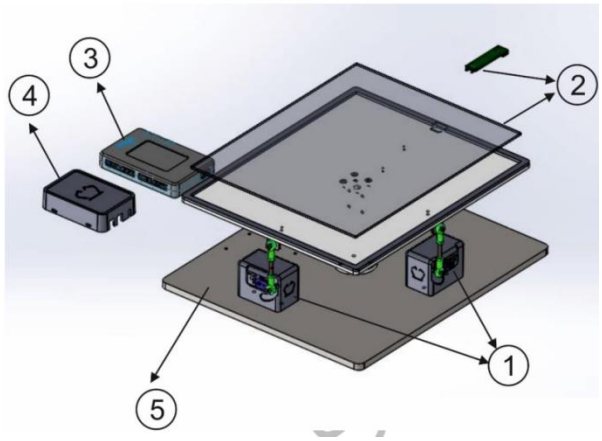
4.2 Top ve Plaka Sistemi

Top ve Plaka sistemi kontrol araştırmalarında sıkça kullanılan bir sistemdir (Şekil 4.4). Bu sistemde temel amaç açısal hareket edebilen bir plakanın üzerine konan topun plaka üzerindeki, konumunun kontrolü veya topun plaka üzerinde istenen bir yolu takip etmesidir.



Şekil 4.3 : Acrome firmasının Top ve Plaka deney düzeneği.

Okul bünyesinde bulunan ve tasarlanacak simülatöre örnek olarak alınan Acrome firmasının ürettiği Top ve Plaka deney setinin mekanik ve elektronik parçaları şekil 4.4’de ki gibidir.



Şekil 4.4 : Sistemin bileşenleri.

Kontrol uygulamalarında sıkça kullanılması sebebiyle bu sistemin fizik motorunda gerçeklenerek fizik motorunun performansının gerçek dünyadaki fizik ile kıyaslamasının yapılması amaçlanmış ve bir deney seti ortamı oluşturulması hedeflenmiştir.

4.2.1 Sistemin analizi ve modellenmesi

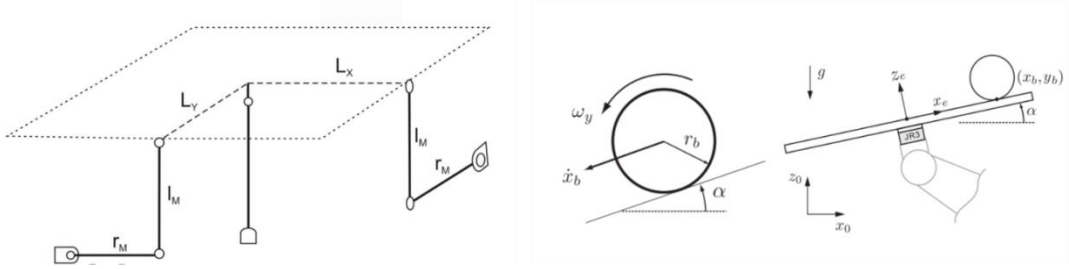
4.2.1.1 Kabuller

Sistem aşağıdaki kabuller yapılarak modellenmiştir.

1. Top kaymadan hareket etmektedir,
2. Top tam bir yuvarlaktır ve homojen yapıdadır,
3. Tüm sürtünmeler ihmal edilmiştir,
4. Top ve plaka sürekli olarak temastadır.

4.2.1.2 Matematiksel modelleme

Sistemin matematiksel modeli bulunurken şekil 4.5'de ki serbest cisim diyagramları kullanılmıştır [5].



Şekil 4.5 : Top ve plaka sisteminin diyagramları.

Sistemin matematiksel modellemesinde Euler-Lagrange yöntemi kullanılmıştır [5,6,7,8]. Bu amaçla bir L (Lagrangian) fonksiyonu tanımlanmıştır.

$$L = E_k - E_p \tag{4.1}$$

Bu denklemde E_k sistemin kinetik enerjisini E_p ise sistemin potansiyel enerjisini temsil etmektedir. Sistemin kinetik enerjisi topun kinetik enerjisi ve plakanın kinetik enerjisi olmak üzere 2 alt denkleme ayrılabilir.

$$E_k = E_{kt} + E_{kp} \tag{4.2}$$

Topun kinetik enerjisi (E_{kt}) lineer hızından(v_b) ve açısal hızından(w_b) gelmektedir. Bu bilgi doğrultusunda topun kinetik enerjisi hesaplanabilir.

$$E_{kt} = \frac{1}{2} m_b {v_b}^2 + \frac{1}{2} I_b {w_b}^2 = \frac{1}{2} m_b(\dot{x}_b^2 + \dot{y}_b^2) + \frac{1}{2} I_b \frac{(\dot{x}_b^2 + \dot{y}_b^2)}{r_b^2} \tag{4.3}$$

Plakanın kinetik enerjisi(E_{kp}) ise plakanın açısal hareketinden kaynaklanmaktadır.

$$E_{kp} = \frac{1}{2} (I_p + I_b)(\dot{\alpha}^2 + \dot{\beta}^2) + \frac{1}{2} m_b (x_b\dot{\alpha} + y_b\dot{\beta})^2 \tag{4.4}$$

Sistemin potansiyel enerjisi topun yüksekliğinin değişiminden kaynaklanmaktadır. Bu bilgi doğrultusunda sistemin potansiyel enerjisi hesaplanabilir.

$$E_p = m_b g h = m_b g (x_b sin\alpha + y_b sin\beta) \tag{4.5}$$

Bu aşamada L (Lagrangian) fonksiyonu artık yazılabilir.

$$L = (E_{kt} + E_{kp}) - E_p \tag{4.6}$$

Sistemdeki değişkenlerimiz topun x ekseninde ve y eksenindeki konumudur. Lagrian fonksiyonları aşağıdaki gibi tanımlanır.

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{x}_b}\right) - \frac{\partial L}{\partial x_b} = 0 \tag{4.7}$$

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{y}_b}\right) - \frac{\partial L}{\partial y_b} = 0 \tag{4.8}$$

4.7 numaralı denklemin çözümü Lagrangian'ın iki ayrı alt parçası ayrı olarak hesaplanmıştır.

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{x}_b}\right) = \frac{d}{dt}\left(\left(m_b + \frac{I_b}{r_b^2}\right)\dot{x}_b\right) = \left(m_b + \frac{I_b}{r_b^2}\right)\ddot{x}_b \tag{4.9}$$

$$\frac{\partial L}{\partial x_b} = m_b\dot{\alpha}\left(x_b\dot{\alpha} + y_b\dot{\beta}\right) - m_b g sin\alpha \tag{4.10}$$

Denklem 4.9 ve 4.10'yi birleştirirsek topun x(x_b) eksenindeki hareketinin modeli;

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{x}_b}\right) - \frac{\partial L}{\partial x_b} = \left(m_b + \frac{I_b}{r_b^2}\right)\ddot{x}_b - m_b\dot{\alpha}\left(x_b\dot{\alpha} + y_b\dot{\beta}\right) - m_b g sin\alpha = 0 \tag{4.11}$$

4.8 numaralı denklemin çözümü aynı şekilde için Lagrangian'ın iki ayrı alt parçası ayrı olarak hesaplanmıştır.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{y_b}}\right) = \frac{d}{dt}\left(\left(m_b + \frac{I_b}{r_b^2}\right)\dot{y_b}\right) = \left(m_b + \frac{I_b}{r_b^2}\right)\ddot{y_b} \tag{4.12}$$

$$\frac{\partial L}{\partial y_b} = m_b\dot{\beta}\left(x_b\dot{\alpha} + y_b\dot{\beta}\right) - m_b g sin\beta \tag{4.13}$$

Denklem 4.12 ve 4.13'i birleştirirsek topun y(y_b) eksenindeki hareketinin modeli;

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{y_b}}\right) - \frac{\partial L}{\partial y_b} = \left(m_b + \frac{I_b}{r_b^2}\right)\ddot{y_b} - m_b\dot{\beta}\left(x_b\dot{\alpha} + y_b\dot{\beta}\right) - m_b g sin\beta = 0 \tag{4.14}$$

Bu noktada sistemin doğrusal olmayan modeli denklem 4.11 ve denklem 4.14 olarak elde edilmiş bulunmaktadır.

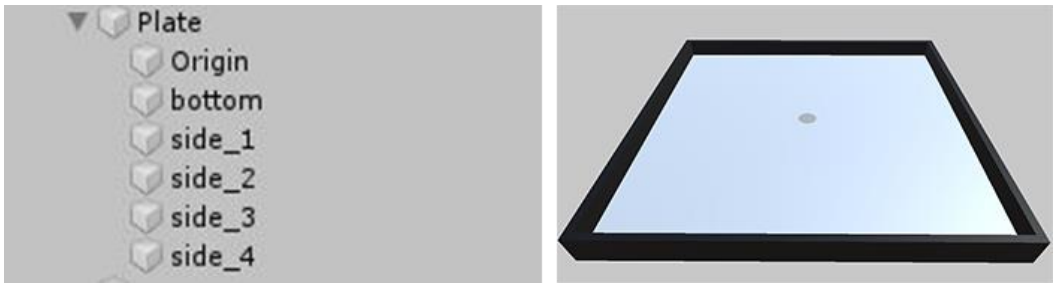
4.2.2 Sistemin Unity ortamına taşınması

Sistemin değerleri çizelge 4.3'te ki gibi seçilmiştir.

Çizelge 4.3 : Sistem değerleri.

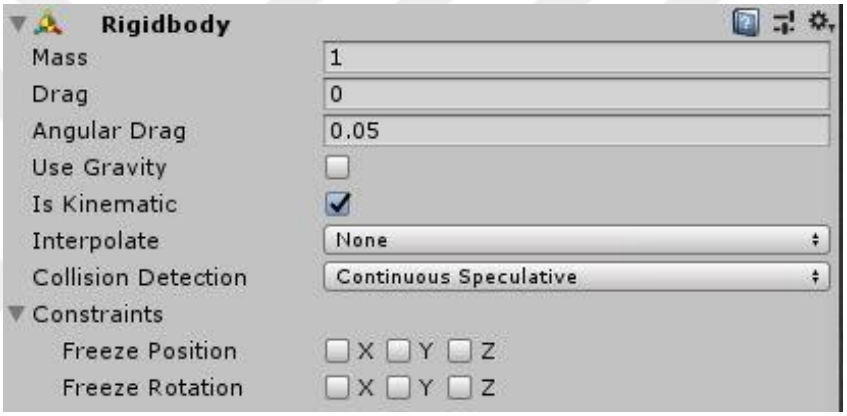
Sembol	Tanım	Değer
m_b	Topun ağırlığı	0.26 $[kg]$
m_P	Plakanın ağırlığı	1.0 $[kg]$
J_b	Topun açısal eylemsizliği	0.0000416 $[kg\ m^2]$
r_b	Topun yarıçapı	0.02 $[m]$
g	Yer çekimi ivmesi	9.81 $[m/s^2]$
L_x	Plakanın x uzunluğu	0.5 $[m]$
L_y	Plakanın y uzunluğu	0.5 $[m]$

Sistemin Unity ortamında gerçeklenmesine öncelikle plakanın modeli ile başlanmıştır. Bu amaçla plaka (bottom) ve plakanın çevresinde topun düşmesini engelleyen yükseltiler (side) için 5 obje oluşturulmuştur ve bu objeler aynı ana objenin(Plate) altında toplanmışlardır. Bu objelere Mesh Renderer ve Mesh Filter bileşenleri atanarak plakanın 3 boyutlu modeli Unity ortamında gerçeklenmiştir. Ayrıca plakanın orta noktasının görsel olarak anlaşılabilmesi için plaka objesinin tam ortasında Sprite bileşenine sahip bir obje (Origin) yerleştirilmiştir (Şekil 4.6).



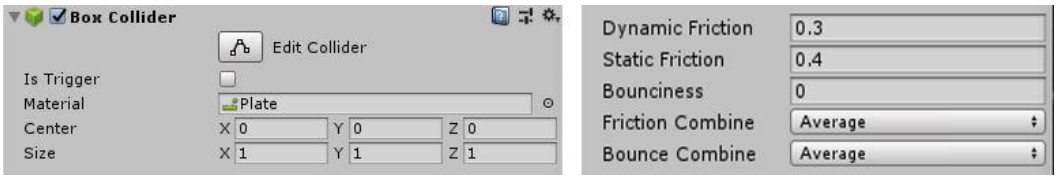
Şekil 4.6 : Solda oluşturulan objeler, sağda ise plakanın 3 boyutlu modeli.

Şu ana kadar yapılan işlemler plakanın sadece görsel olarak Unity ortamında gerçeklenmesini sağlamıştır. Plakaya fiziksel özellikler eklemek için daha önce anlatıldığı gibi Fizik Bileşenleri kullanılmıştır. Bu amaçla oluşturulan plaka ana (Plate) objesine Rigidbody bileşeni eklenmiştir (Şekil 4.7).



Şekil 4.7 : Plaka (Plate) ana objesi üzerindeki Rigidbody bileşeni.

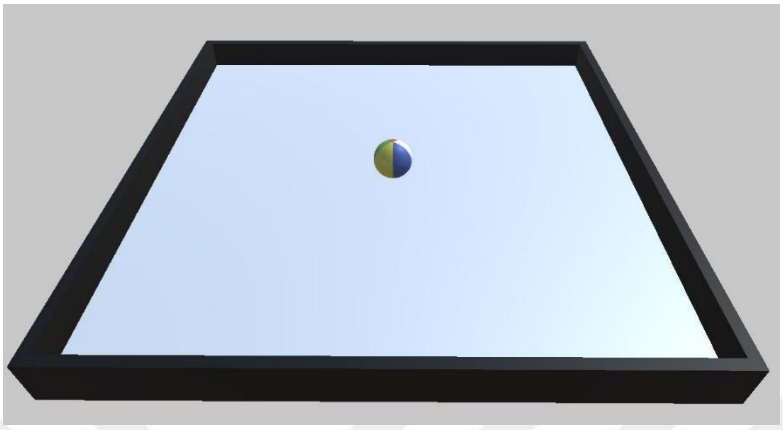
Daha sonra plaka objesinin alt objelerine Collider bileşeni eklenerek sonradan eklenecek top ile dinamik olarak temas etmesi ve çarpışma fiziğinin uygulanması sağlanmıştır (Şekil 4.8).



Şekil 4.8 : Alt objelere eklenen Collider bileşeni ve Fizik materyali.

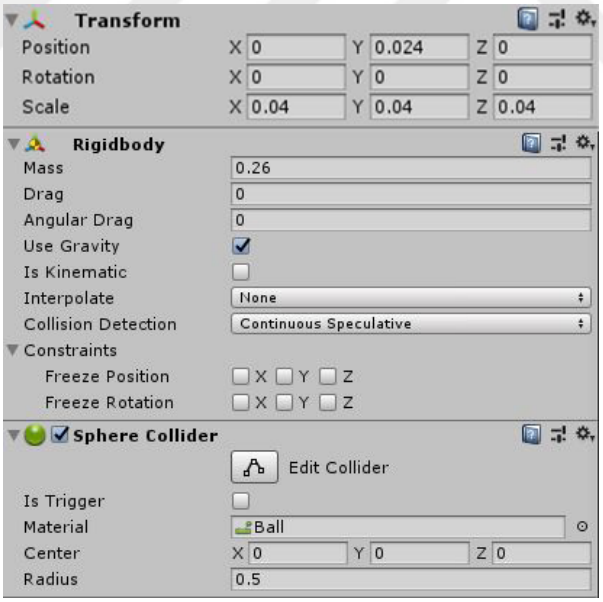
Top ve plaka arasındaki sürtünme katsayısı fizik materyali bileşeni kullanılarak sağlanmıştır. Dinamik ve statik sürtünme katsayısı belirlenirken topun metal ve plakanın plastik yapıda olduğu varsayılmıştır. Gerçek dünyadaki plastik ve metal arasındaki sürtünme katsayıları kinetik için 0.3, statik için ise 0.4 tür.

Plakadan sonra topun Unity ortamında oluşturulmasına geçilmiştir. Bu amaçla öncelikle topun 3 boyutlu modeli Mesh Renderer ve Mesh Filter bileşenleri kullanılarak oluşturulmuştur (Şekil 4.9).



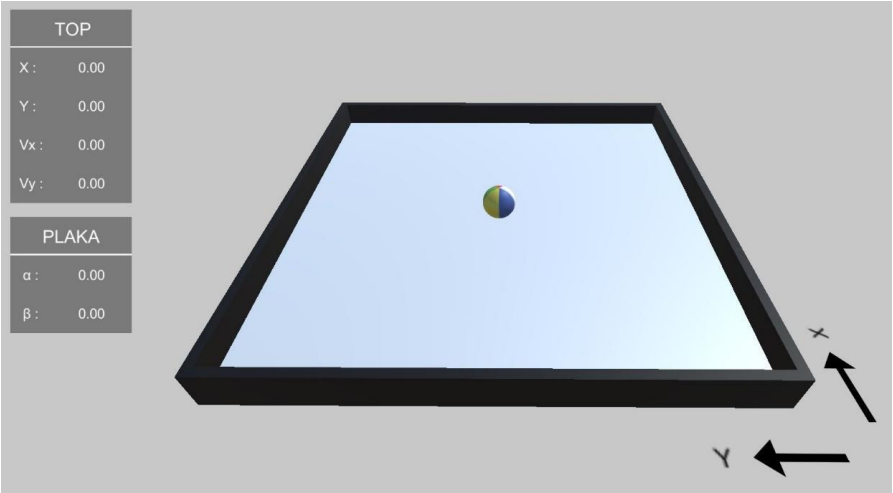
Şekil 4.9 : Daha önce oluşturulan plakanın üzerindeki top objesi.

Topun dinamik olarak hareket için Rigidbody bileşeni topun objesine eklenmiştir. Ayrıca topun çarpışma fiziğine sahip olabilmesi için Sphere Collider bileşeni de top objesine eklenmiştir (Şekil 4.10).



Şekil 4.10 : Top (Ball) objesindeki Rigidbody ve Sphere Collider bileşenleri.

Simulasyon ekranının sol üst köşesine UI bileşenleri kullanılarak basit bir kullanıcı arayüzü eklenmiştir. Plakanın açısı, topun çizgisel hızı topun açısal hızı ve topun konumu bilgileri bu ekrandan izlenebilir (Şekil 4.11).

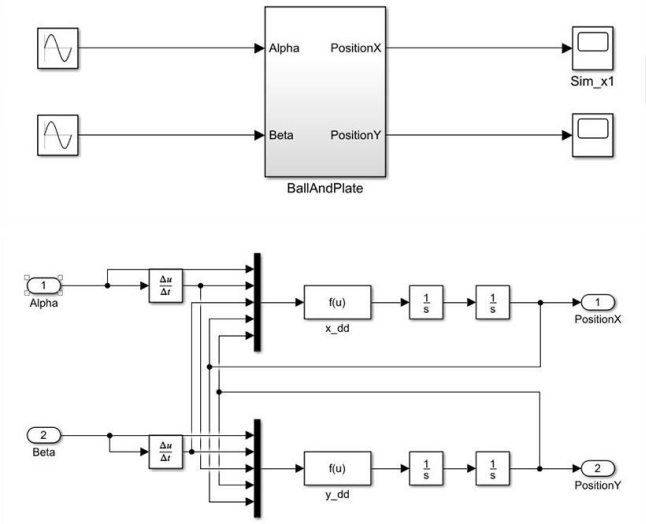


Şekil 4.11 : Geliştirilen simülatörün tüm bileşenleriyle birlikte görüntüsü.

4.2.3 Matematiksel modelin ve simülatörün karşılaştırılması

4.2.3.1 Matematiksel modelin simülasyonu

Elde edilen doğrusal olmayan model Simulink ortamına taşınmıştır (Şekil 4.12).



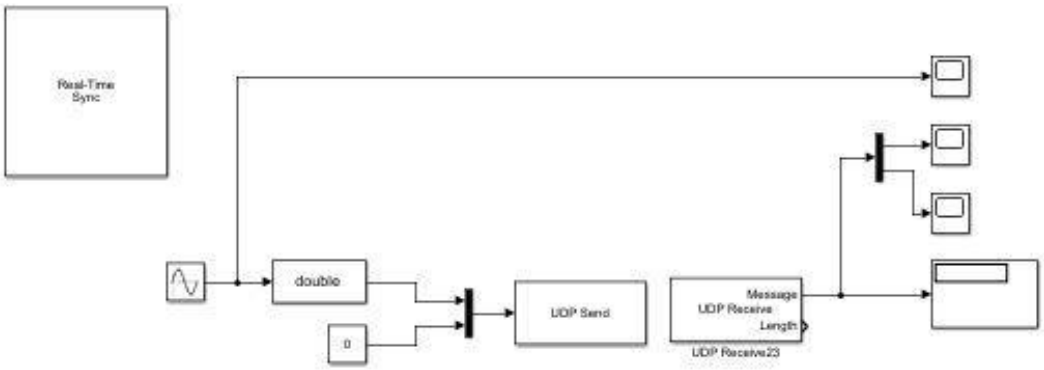
Şekil 4.12 : Doğrusal olmayan model simülasyonu.

Sisteme çeşitli giriş işaretleri uygulanmış ve sonuçlar kaydedilmiştir.

4.2.3.2 Simülatörün Simulink ortamından kontrolü

Unity modelinin Simulink ortamından kontrolü için daha önceki başlıklarda anlatılan UDP protokolü kullanılmıştır. Bu amaçla UDP Send bloğu sisteme giriş uygulamak için UDP Receive bloğu ise sistemin çıkışlarını almak için kullanılmıştır (Şekil 4.13).

Unity tarafında 25000 portu alıcı port 26000 portu ise verici portu olarak ayarlanmıştır.

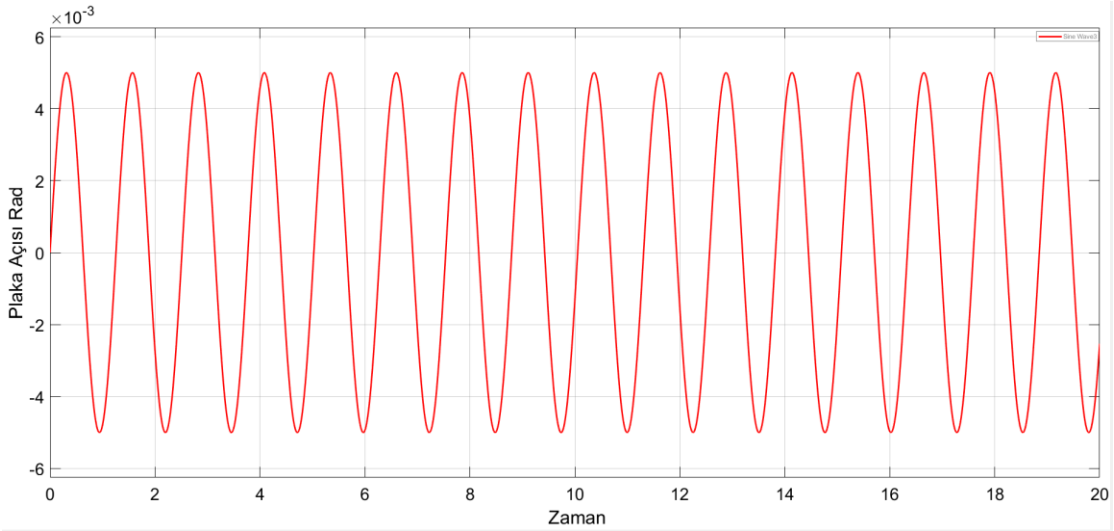


Şekil 4.13 : Simülatörün kontrolü için Simulink'te kurulan model.

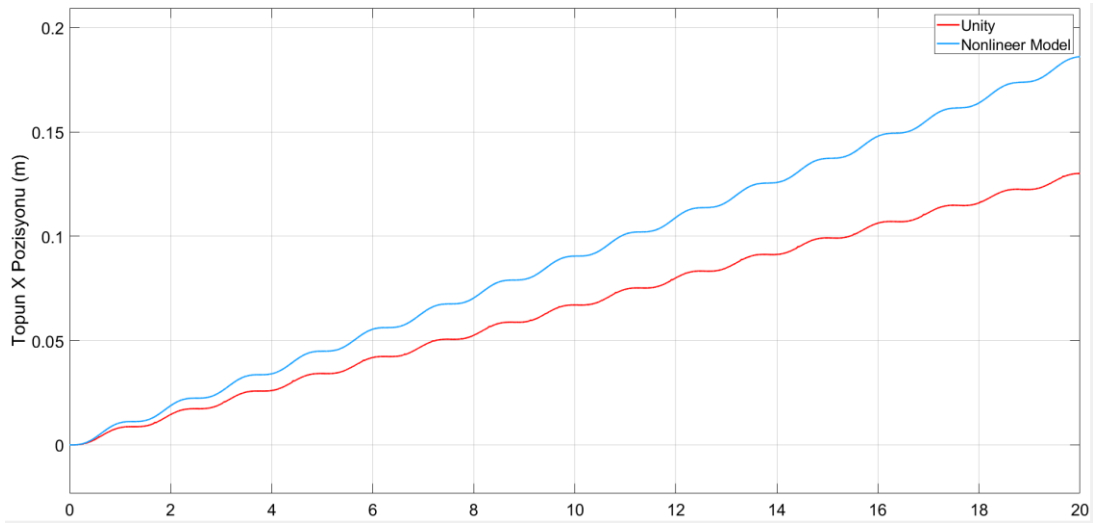
Haberleşme ayarlandıktan sonra sisteme çeşitli giriş işaretleri uygulanmış ve sonuçlar kaydedilmiştir.

4.2.3.3 Karşılaştırmalar

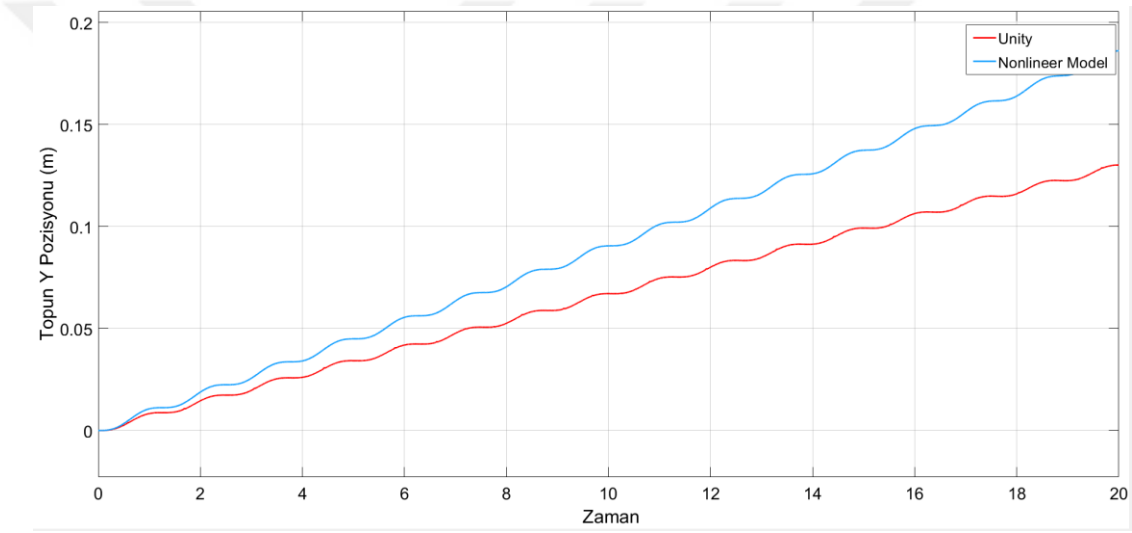
Bu noktada doğrusal olmayan modelin simulasyonu ve Unity ortamındaki sistemin girişlerine çeşitli sinüs işaretleri uygulayarak sonuçların karşılaştırılması hedeflenmiştir. Öncelikle sistemin iki girişinede düşük frekans ve genlikte eşit sinüs işaretleri uygulanmış ve sonuçlar kaydedilmiştir.



Şekil 4.14 : Sistemin iki girişine de uygulanan sinüs işareti.



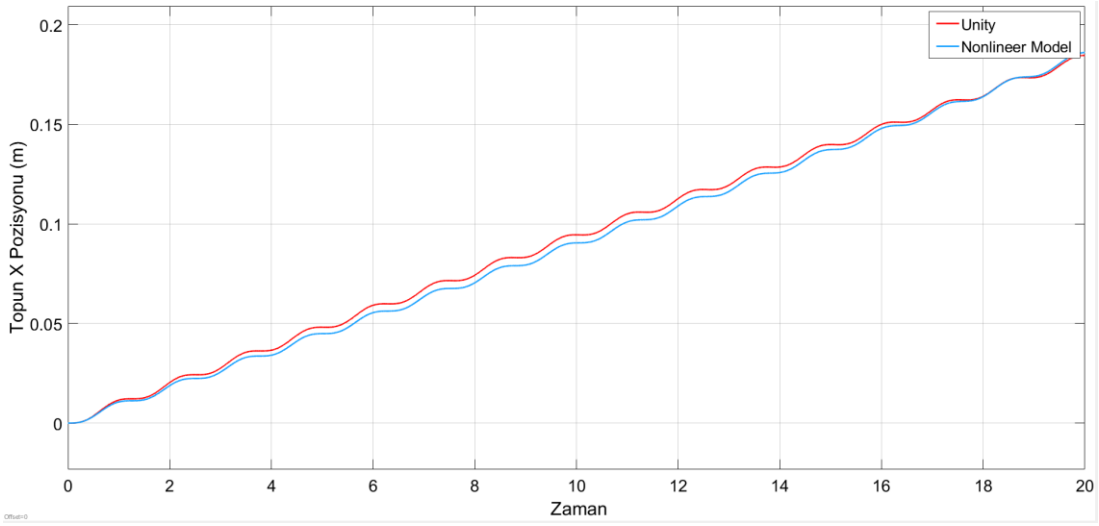
Şekil 4.15 : Topun X eksenindeki pozisyonu.



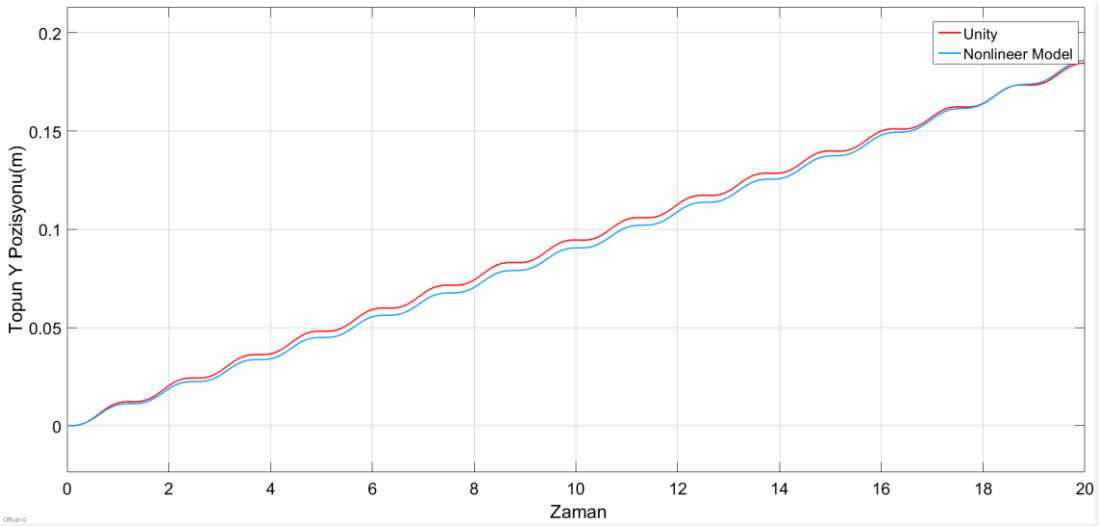
Şekil 4.16 : Topun Y eksenindeki pozisyonu.

Şekil 4.16'da görülebileceği gibi Unity ortamındaki simülatörün ve doğrusal olmayan modelin simülasyonu aynı giriş için aynı çıkış işaretlerini vermemektedir. Bu farkın nedenini anlamamız için doğrusal olmayan modele bakmamız gerekmektedir. Öncelikle doğrusal olmayan modelde sürtünme kuvveti hesaba katılmamıştır. Fakat Unity ortamında simülasyon hazırlanırken fizik materyali bileşeni kullanılarak sisteme sürtünme eklenmiştir. Ayrıca matematiksel modeli elde etmeden önce yaptığımız 1 ve 3 numaralı kabullerde tüm sürtünmelerin ihmal edildiği aynı zaman topun kaymadan plaka üzerinde ilerlediği varsayılmıştır. Gerçek dünyada sürtünmesiz ve eğimli bir plaka üzerine konan topun dönerek ilerlemesi mümkün değildir. Çünkü topun eğimli bir yüzeye konulduğunda dönmesine neden olan kuvvet sürtünme nedeniyle oluşmaktadır. Yani burada sistemin kolay

modellenmesi için yapılan kabuller doğrusal olmayan modelde hata oluşmasına ve Unity ortamındaki simülasyon ile farklı sonuç vermesine sebep olmaktadır. Bu farkın giderilmesi amacıyla Unity motorunda top objesinin fizik materyali bileşeni üzerindeki statik ve kinetik sürtünme katsayısı 0. 00002 seçilmiştir. Bu değerin seçilmesindeki amaç sürtünme değerinin ihmal edilebilecek kadar küçük seçmek ama aynı zamanda topun simülatör ortamında azda olsa dönerek hareket etmesini sağlamak böylece Unity ortamındaki emülasyonu sistemin doğrusal olmayan modeline yakınlaştırmaktır. Bu yeni değerlere göre aynı giriş işareti ile yapılan testin sonuçlar şekil 4.17 ve 4.18'de ki gibidir.



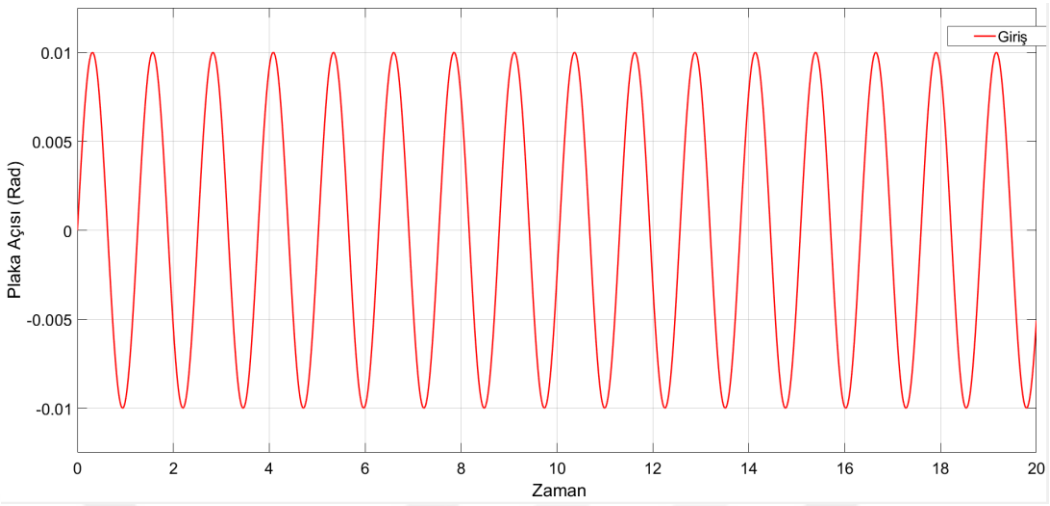
Şekil 4.17 : Topun X eksenindeki pozisyonu.



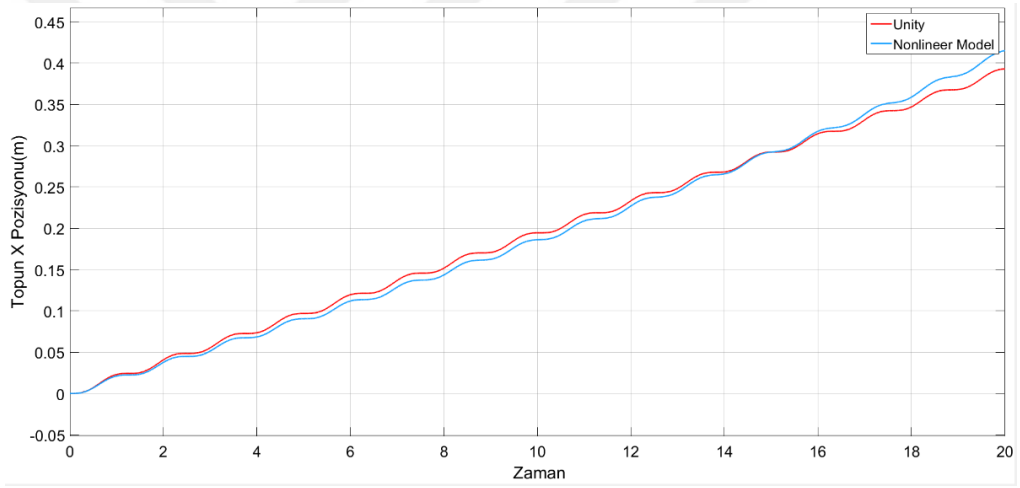
Şekil 4.18 : Topun Y eksenindeki pozisyonu.

Şekil 4.18 ve 4.17'de görülebileceği gibi düzenlemeden doğrusal olmayan model simülasyonu ve Unity motoru ortamında tasarlanan simülatör aynı girişler için yakın

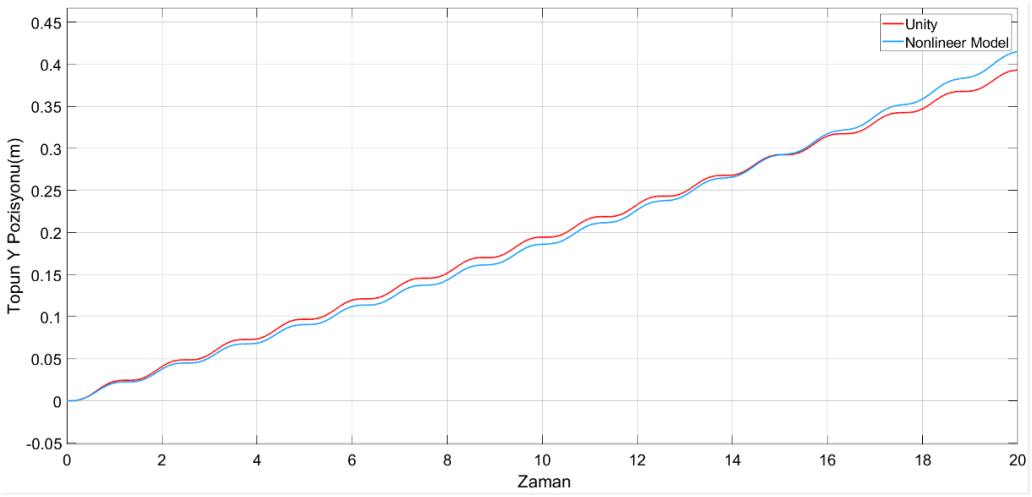
çıkış işaretleri vermiştir. Bu noktada sisteme farklı genlik ve frekansta giriş işaretleride uygulanarak daha detaylı bir analiz yapılmıştır.



Şekil 4.19 : Sistemin iki girişine de uygulanan sinüs işareti.

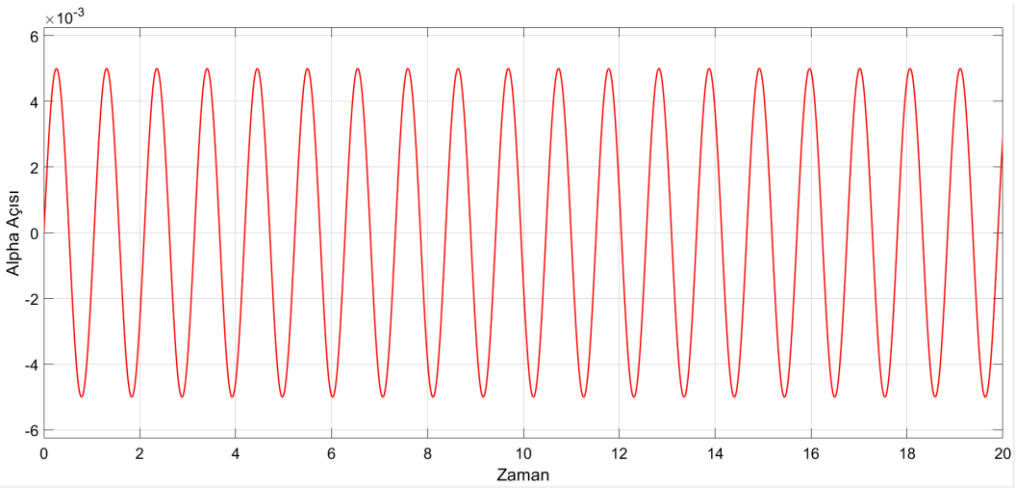


Şekil 4.20 : Topun X eksenindeki pozisyonu.

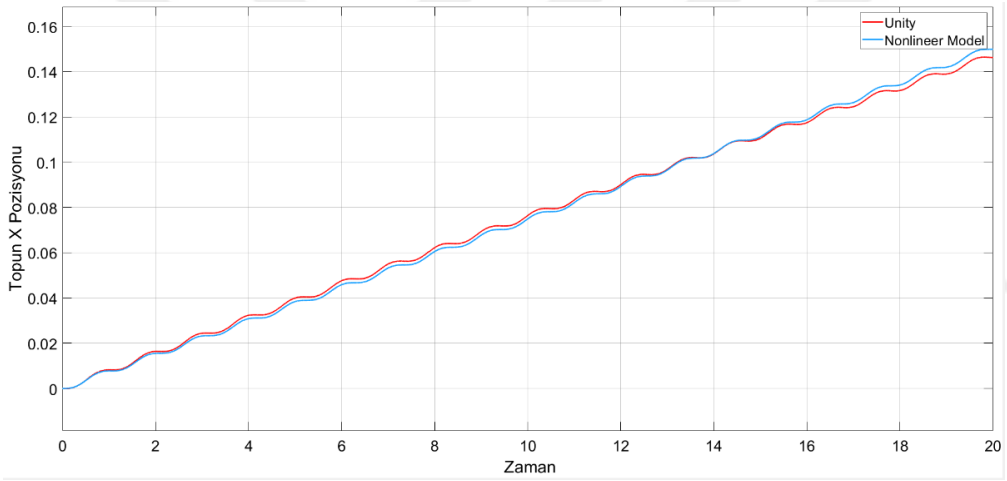


Şekil 4.21 : Topun Y eksenindeki pozisyonu.

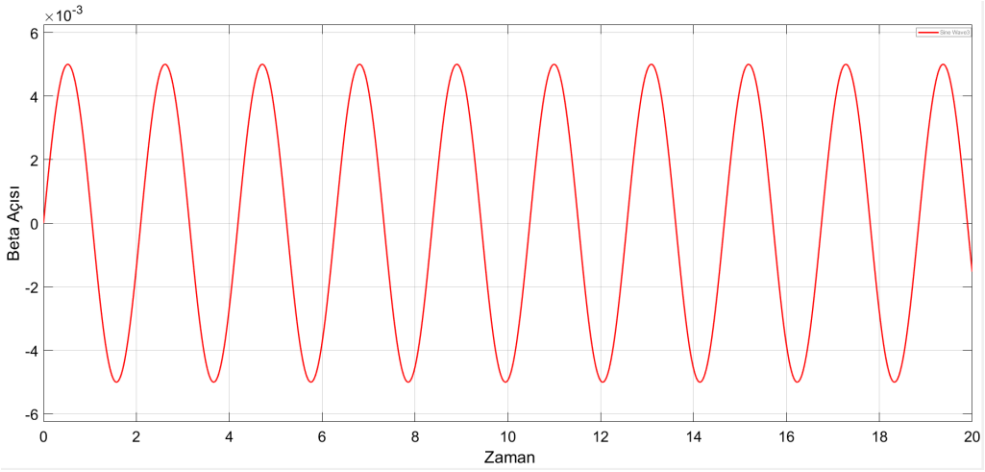
Sistemin iki girişine şekil 4.22 ve 4.24'te ki işaretler uygulanmış ve şekil 4.23 ve şekil 4.25'te ki sonuçlar elde edilmiştir.



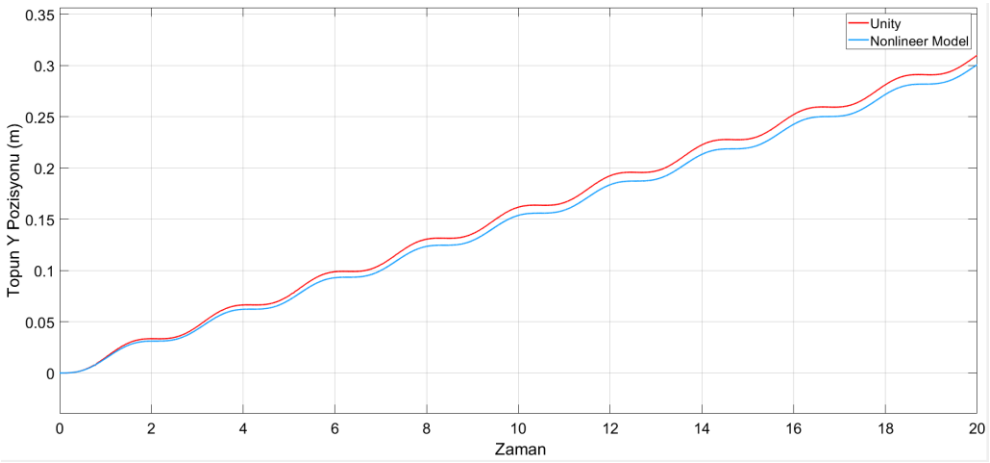
Şekil 4.22 : Sistemin alpha girişine uygulanan sinüs işareti.



Şekil 4.23 : Topun X eksenindeki pozisyonu.

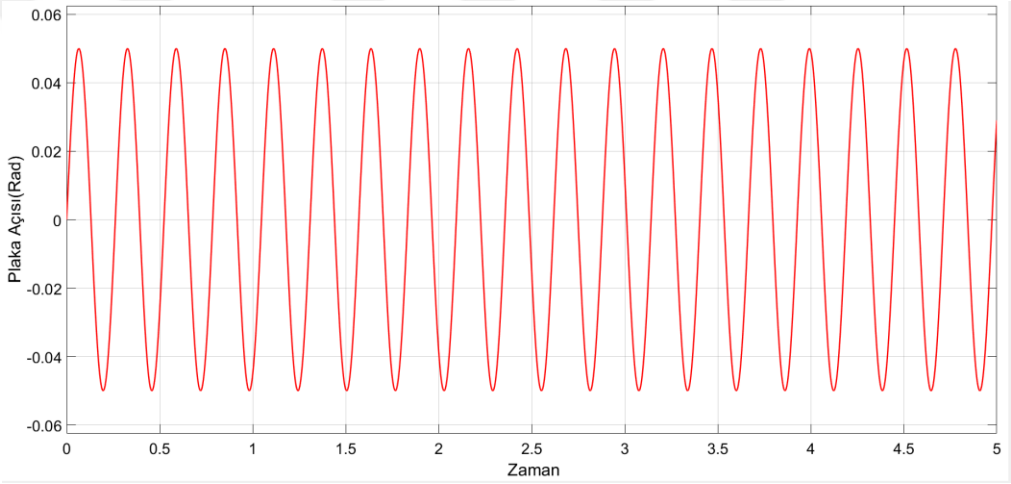


Şekil 4.24 : Sistemin alpha girişine uygulanan sinüs işareti.

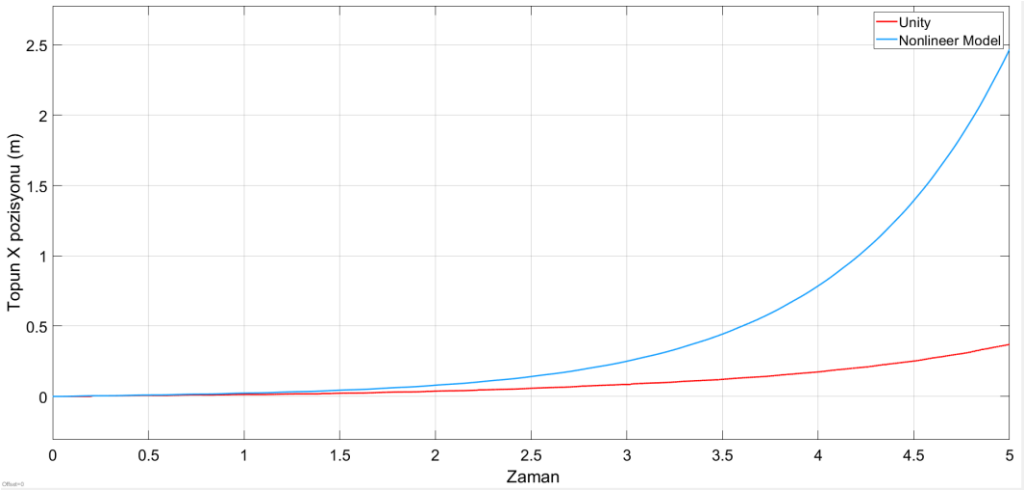


Şekil 4.25 : Topun Y eksenindeki pozisyonu.

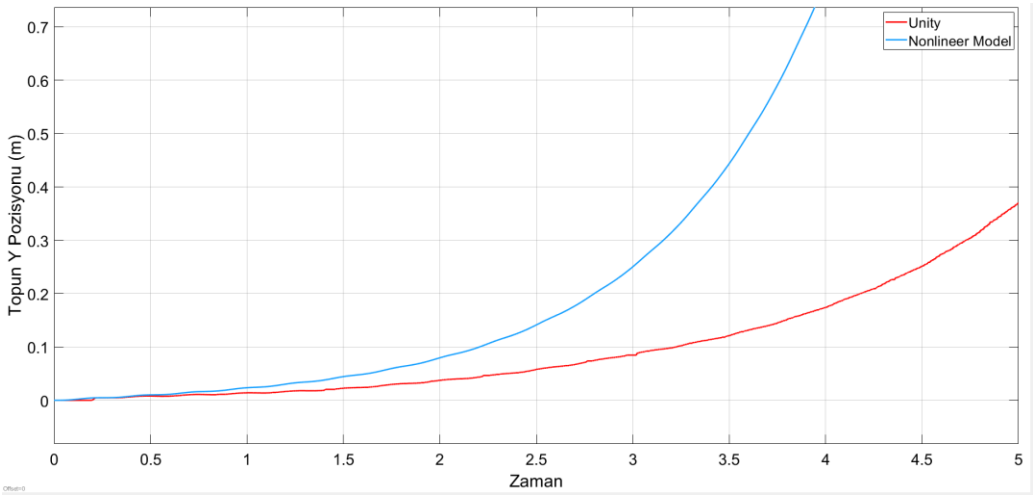
Uygulanan girişin frekansı ve genliği arttırıldığında simülatör ile doğrusal olmayan model arasındaki farklılıklar artmaktadır (Şekil 4.36, 4.27 ve 4.28).



Şekil 4.26 : Sistemin iki girişine de uygulanan sinüs işareti.



Şekil 4.27 : Topun X eksenindeki pozisyonu.

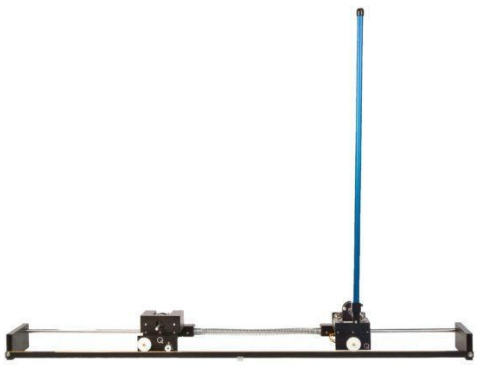


Şekil 4.28 : Topun Y eksenindeki pozisyonu.

Bu artan farkın sebebi doğrusal olmayan modelleme yaparken yapılan 4 numaralı varsayımdır. Çünkü yüksek frekansta giriş uygulandığında simülatör deki topun plaka ile olan teması kesilmektedir. Bu nedenle topun hareket grafikleri benzer yapıda olsa bile simülatör ile doğrusal olmayan model arasında fark oluşmaktadır. Yapılan tüm testlerin sonucunda varılabilir ki Unity oyun motorunun kullandığı Phsyx fizik motoru çeşitli ayarlar yapıldığında sistemin matematiksel modeline yakın işaretler vemektedir. Buradan Phsyx motorunun doğru bir altyapı ile çalıştığı sonucuna varılabilir.

4.3 Ters Sarkaç Sistemi

Ters sarkaç (Inverted Pendulum) sistemi en klasik kontrol problemlerinden biridir. Bu sistemde temel amaç tek eksende çizgisel olarak hareketen eden bir araca bağlı olan sarkacın açısının kontrol edilmesidir (Şekil 4.29).



Şekil 4.29 : Quanser firmasının Ters Sarkaç deney seti.

Kontrol uygulamalarında sıkça kullanılan sebebiyle bu sistemin fizik motorunda gerçeklenerek fizik motorunun performansının gerçek dünyadaki fizik ile kıyaslamasının yapılması amaçlanmış ve bir deney seti ortamı oluşturulması hedeflenmiştir.

4.3.1 Sistemin analizi ve modellenmesi

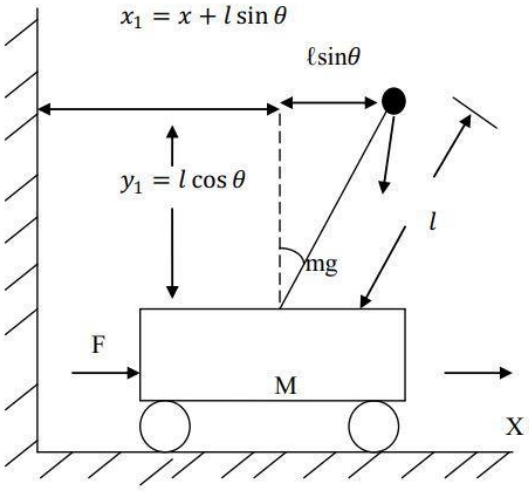
4.3.1.1 Kabuller

Sistem aşağıdaki kabuller yapılarak modellenmiştir.

1. Çubuk homojen yapıdadır ve kütle merkezi geometrik merkezdedir,
2. Araç ile ray arasında sadece sıvı sürtünmesi mevcuttur.

4.3.1.2 Matematiksel modelleme

Sistemin matematiksel modeli bulunurken şekil 4.30'da ki serbest cisim diyagramı kullanılmıştır [9].



Şekil 4.30 : Sistemin serbest cisim diyagramı.

Sistemin matematiksel modellemesinde Euler-Lagrange yöntemi kullanılmıştır [9]. Bu amaçla bir L (Lagrangian) fonksiyonu tanımlanmıştır.

$$L = E_k - E_p \tag{4.15}$$

Sistemin kinetik enerjisi aracın kinetik enerjisi ve sarkacın kinetik enerjisi olmak üzere 2 alt denkleme ayrılabilir.

$$E_k = E_{ka} + E_{ks} \tag{4.16}$$

Aracın kinetik enerjisi(E_{ka}) araca uygulanan F kuvveti ile oluşan çizgisel hızdan kaynaklanmaktadır.

(4.17)

$$E_{ka} = \frac{1}{2} M\dot{x}^2$$

Sarkacın kinetik enerjisi (E_{ks}) ise cizgisel (V) ve açısal (w) hızından kaynaklanmaktadır.

$$V = \dot{x_1}^2 + \dot{y_1}^2, \;\; x_1 = x + lsin(\theta), \;\; y_1 = lcos(\theta) \tag{4.18}$$

$$E_{ks} = \frac{1}{2} m\dot{V}^2 + \frac{1}{2} Iw^2 = \frac{1}{2} m\left(\dot{x}^2 + 2l\dot{x}\dot{\theta}\, cos\, cos\, (\theta) \; + l^2\dot{\theta}^2\right) + \frac{1}{2} I\dot{\theta}^2 \tag{4.19}$$

Sistemin potansiyel enerjisi(E_p) sadece sarkacin potansiyel enerjisinden(E_{ps}) kaynaklanmaktadır.

$$E_p = E_{ps} = mglcos(\theta) \tag{4.20}$$

Bu aşamada L(Lagrangian) fonksiyonu artık yazılabilir.

$$L = (E_{ka} + E_{ks}) \; - E_{ps} \tag{4.21}$$

Sistemdeki değişkenlerimiz aracın konumu ve sarkacın açısıdır. Lagrian fonksiyonları aşağıdaki gibi tanımlanır.

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} + b\dot{x} = F \tag{4.22}$$

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} + d\dot{\theta} = 0 \tag{4.23}$$

4.22 numaralı denklemi çözümü için Lagrangian'ın iki ayrı alt parçası ayrı olarak hesaplanmıştır.

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{x}}\right) = (M + m)\ddot{x} + ml\ddot{\theta}\, cos\, cos\, (\theta) \; - ml\dot{\theta}^2 sin(\theta) \tag{4.24}$$

$$\frac{\partial L}{\partial x} = 0 \tag{4.25}$$

Denklem 4.24 ve 4.25'yi birleştirirsek topun x(x_b) eksenindeki hareketinin modeli;

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} + b\dot{x} = (M + m)\ddot{x} + mlcos(\theta)\ddot{\theta} - mlsin(\theta)\dot{\theta}^2 + b\dot{x} = F \tag{4.26}$$

4.23 numaralı denklemi çözümü için Lagrangian'ın iki ayrı alt parçası ayrı olarak hesaplanmıştır.

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{\theta}}\right) = -ml\dot{x}\dot{\theta}\; sin\, sin\, (\theta) \; + ml\ddot{x}\; cos\, cos\, (\theta) \; + ml^2\ddot{\theta} + I\ddot{\theta} \tag{4.27}$$

$$\frac{\partial L}{\partial \theta} = -ml\dot{x}\dot{\theta}\; sin\, sin\, (\theta) \; + mglsin(\theta) \tag{4.28}$$

Denklem 4.27 ve 4.28'u birleştirirsek topun x(x_b) eksenindeki hareketinin modeli;

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} + d\dot{\theta} = (I + ml^2)\ddot{\theta} + mlcos(\theta)\ddot{x} + mglsin(\theta) + d\dot{\theta} = 0 \tag{4.29}$$

Bu noktada sistemin doğrusal olmayan modeli denklem 4.26 ve denklem 4.29 olarak elde edilmiş bulunmaktadır.

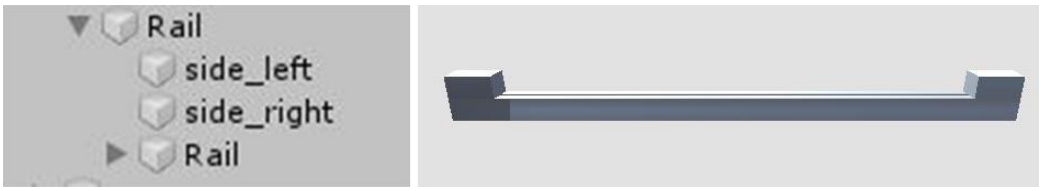
4.3.2 Sistemin Unity ortamına taşınması

Sistemin değerleri çizelge 4.4'te ki gibi seçilmiştir.

Çizelge 4.4 : Sistem değerleri.

Sembol	Tanım	Değer
M	Aracın ağırlığı	0.5
m	Sarkacın ağırlığı	0.2
b	Sürtünme katsayısı	0.1
g	Yer çekimi ivmesi	9.81 $[m/s^2]$
l	Sarkacın uzunluğu	0.6 $[m]$

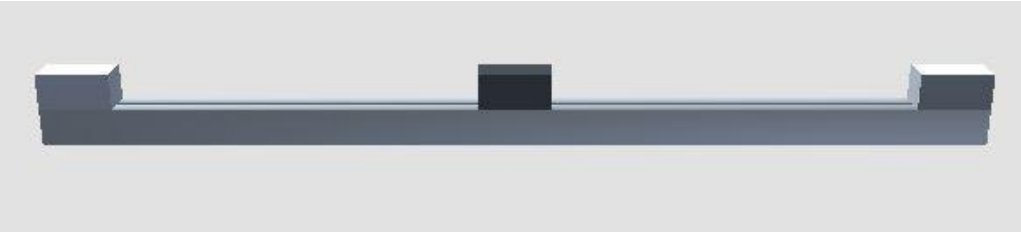
Sistemin Unity ortamında gerçeklenmesine öncelikle aracın üzerinde hareket edeceği rayın modeli ile başlanmıştır. Bu amaçla ray (Rail) ve rayın iki ucuna aracın rayın dışına çıkmasını engelleyecek plaka objeleri (side_left ve side_right) yerleştirilmiştir. Bu objelere Mesh Renderer ve Mesh Filter bileşenleri atanarak rayın 3 boyutlu modeli Unity ortamında gerçeklenmiştir (Şekil 4.31).



Şekil 4.31 : Oluşturulan objelerin listesi ve rayın 3 boyutlu modeli.

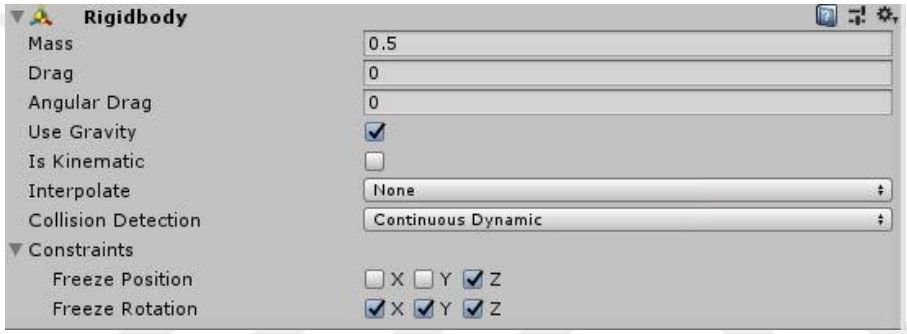
Rayın pozisyonu veya rotasyonu değişmeyeceği yani simülasyon boyunca ray sabit kalacağı için Rigidbody bileşeni raya eklenmemiştir. Fakat ayın çarpışma fiziğine sahip olabilmesi için uçlardaki engellere (side_left ve side_right) ve rayın kendisine Collider bileşeni eklenmiştir.

Daha sonra oluşturulan rayın üzerinde hareket edecek aracın Unity ortamında gerçeklenmesine geçilmiştir. Bu amaçla bir araç (Car) objesi oluşturulmuş ve bu objenin 3 boyutlu modeli Mesh Renderer ve Mesh Filter bileşenleri ile oluşturulmuştur (Şekil 4.32).



Şekil 4.32 : Ray ve aracın görünümü.

Aracın kütle dinamiğine sahip olabilmesi için Rigidbody bileşeni eklenmiştir. Burada dikkat edilmesi gereken nokta Rigidbody üzerinden aracın açısal hareketi ve z eksenindeki hareketi engellenmiştir. Buradaki amaç aracın sadece tek eksende hareket edebilmesidir. Aksi halde yüksek hızlı dönen sarkaç aracın dengesini bozup raydan düşürebilir (Şekil 4.33).



Şekil 4.33 : Aracın Rigidbody bileşeni ve uygulanan kısıtlar (Constraints).

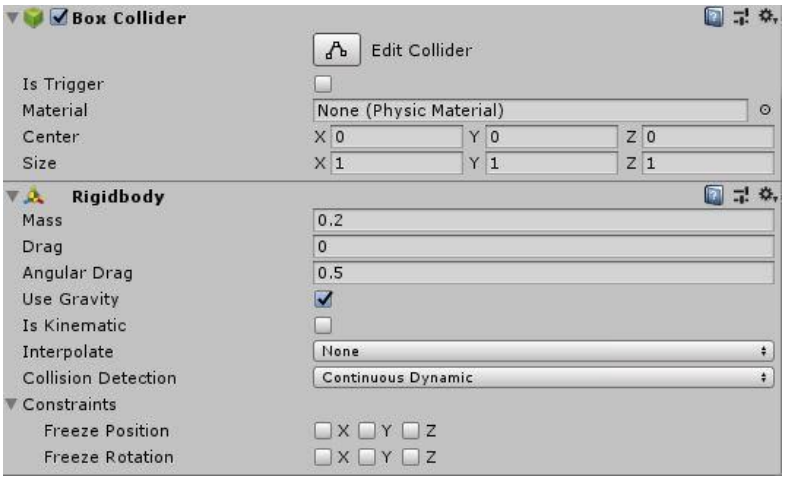
Araca Box Collider bileşeni de eklenerek ray ile dinamik olarak temas etmesi ve çarpışma fiziğinin uygulanması sağlanmıştır.

Unity ortamındaki Fizik Materyali bileşeni Coulomb sürtünme modelini kullandığı için sadece kuru yüzeylerin sürtünmesini simüle etmektedir. Fakat doğrusal olmayan modelde Araç ile ray arasındaki sürtünme sıvı sürtünmesi olarak ifade edilmiştir. Unity motorunda sıvı sürtünmesini hesaplayan ve simüle eden ayrı bir bileşen bulunmamaktadır. Bu nedenle sıvı sürtünmesini hesaplayacak ve eklendiği objeye etkisini uygulayacak bir sıvı sürtünmesi (Viscosity) bileşeni C# scripti ile yazılmıştır (Şekil 4.34).

Şekil 4.34 : C# ile yazılan viscosity bileşeni.

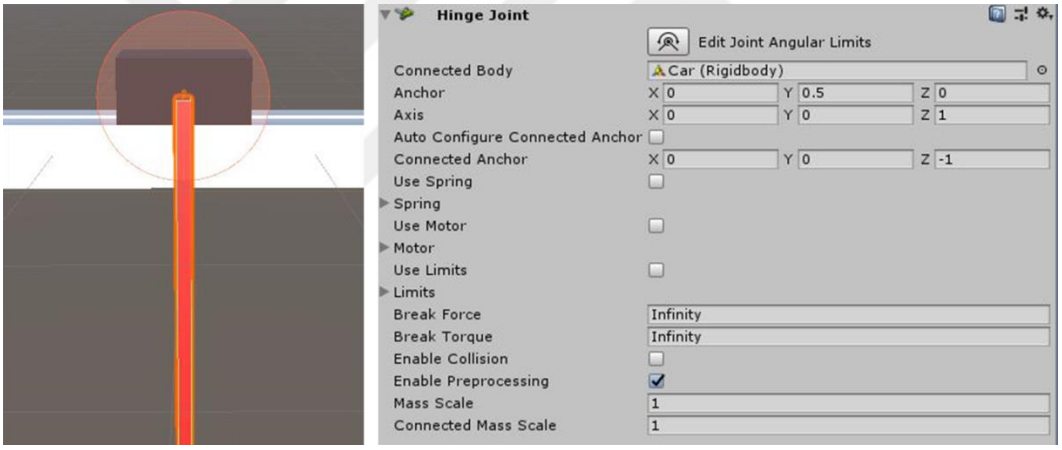
Son olarak sisteme sarkacın eklenmesi amacıyla bir sarkaç (Pendulum) objesi simülasyon içerisine eklenmiştir. Objenin 3 boyutlu modeli Mesh Renderer ve Mesh

Filter bileşenleri ile oluşturulmuştur. Sarkaca kütle dinamiği eklemek için Rigidbody ve çarpışma dinamiği için Collider bileşeni eklenmiştir (Şekil 4.35).



Şekil 4.35 : Sarkaç objesine eklenen Rigidbody ve Box Collider bileşeni.

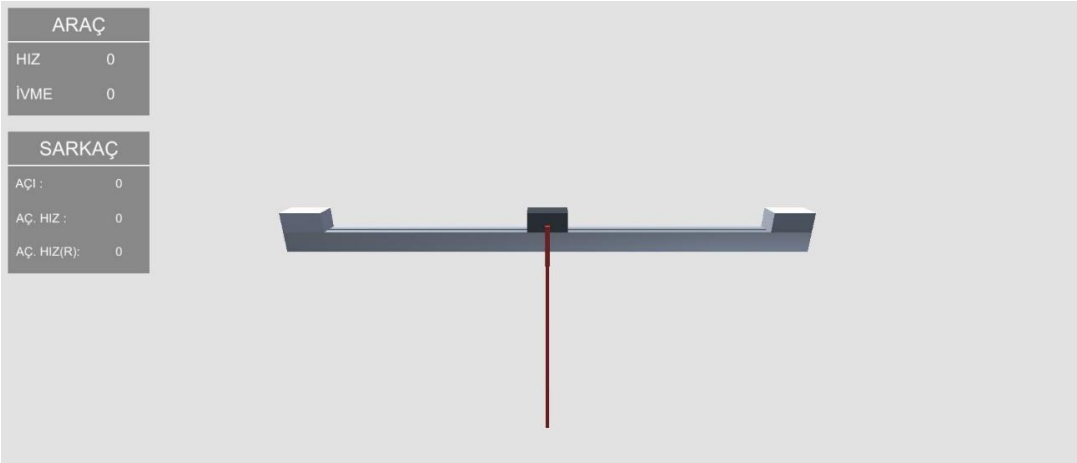
Araç ile sarkaç arasındaki eklem dinamiği Hidge Joint bileşeni kullanılarak sağlanmıştır. Sarkac uc kısmından Hidge Joint ile araca bağlanması sağlanmıştır.



Şekil 4.36 : Eklenen Hinge Joint bileşeni ve ayarları.

Hinge Joint bileşeni üzerindeki Anchor ve Connected Anchor değerleri ayarlanarak sarkacın uç noktasından dik araca dikey şekilde bağlanması sağlanmıştır. Şekil 4.36'da sol tarafta kırmızı ile gösterilen daire sarkacın çevresinde açısal olarak hareket edebileceği ekseni göstermektedir.

Simulasyon ekranının sol üst köşesine UI bileşenleri kullanılarak basit bir kullanıcı arayüzü eklenmiştir. Aracın hızı ve konumu, sarkacın açısı ve açısal hızı bilgileri bu ekrandan izlenebilir (Şekil 4.37).

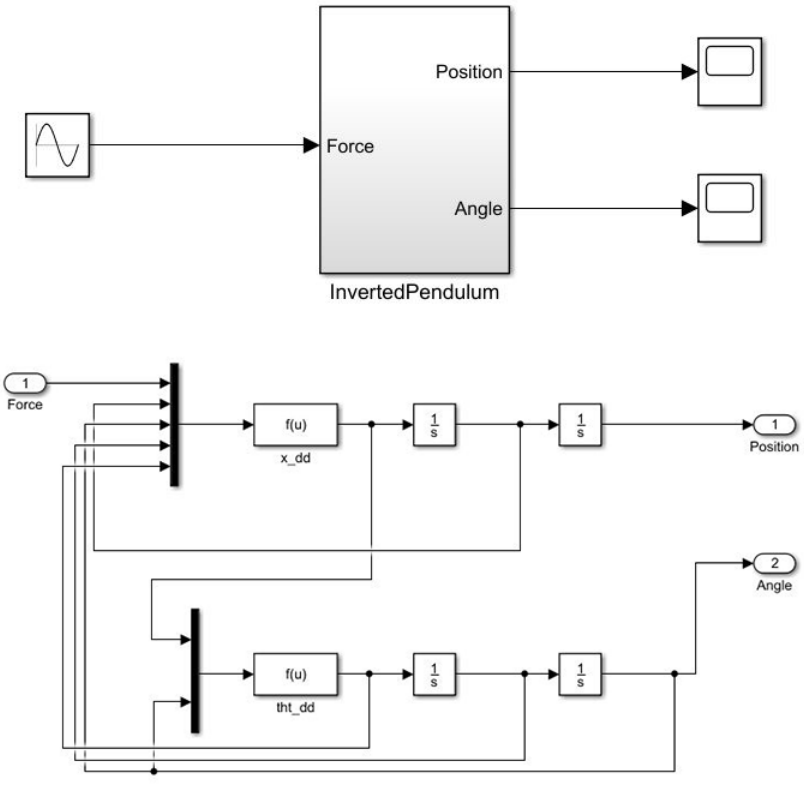


Şekil 4.37 : Geliştirilen simülatörün tüm bileşenleriyle birlikte görüntüsü.

4.3.3 Matematiksel modelin ve simülatörün karşılaştırılması

4.3.3.1 Matematiksel modelin simülasyonu

Elde edilen doğrusal olmayan model Simulink ortamına taşınmıştır (Şekil 4.38).

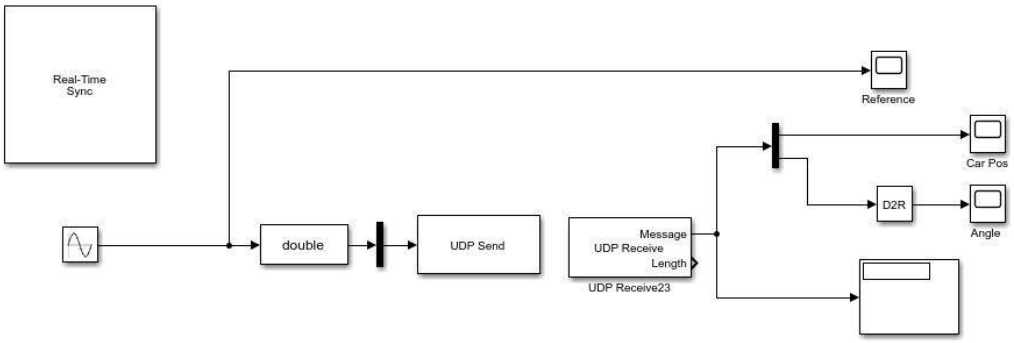


Şekil 4.38 : Doğrusal olmayan model simülasyonu.

Sisteme çeşitli giriş işaretleri uygulanmış ve sonuçlar kaydedilmiştir.

4.3.3.2 Simülatörün Simulink ortamından kontrolü

Unity modelinin Simulink ortamından kontrolü için top ve plaka sistemindeki düzeneğin aynısı kullanılmıştır. Unity tarafında 25000 portu alıcı port 26000 portu ise verici portu olarak ayarlanmıştır.

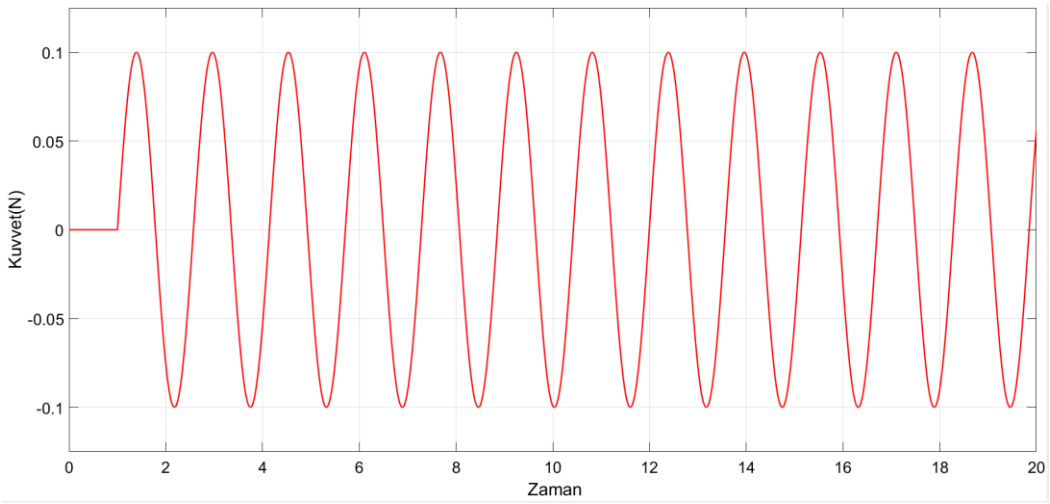


Şekil 4.39 : Simülatörün kontrolü için Simulink'te kurulan model.

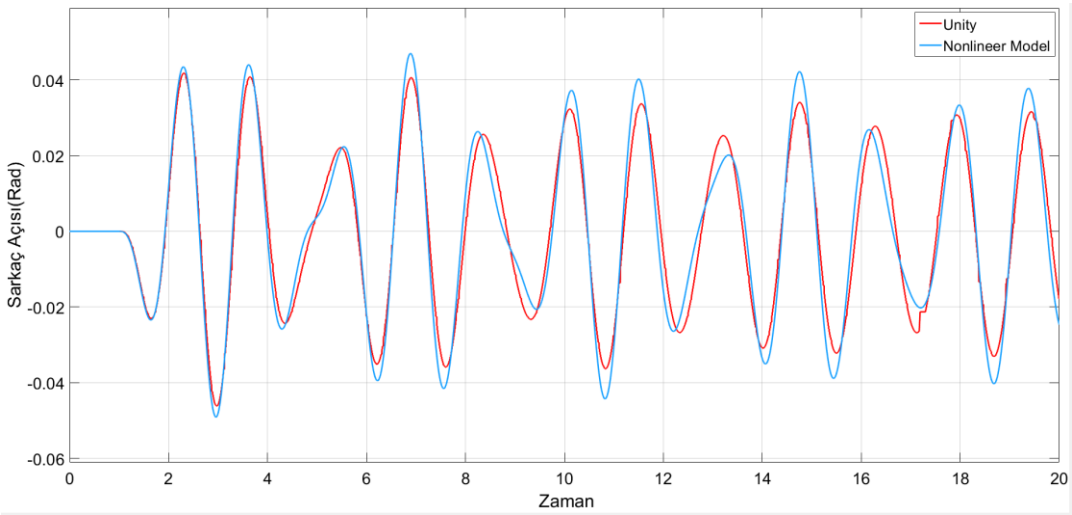
Haberleşme ayarlandıktan sonra sisteme çeşitli giriş işaretleri uygulanmış ve sonuçlar kaydedilmiştir.

4.3.3.3 Karşılaştırmalar

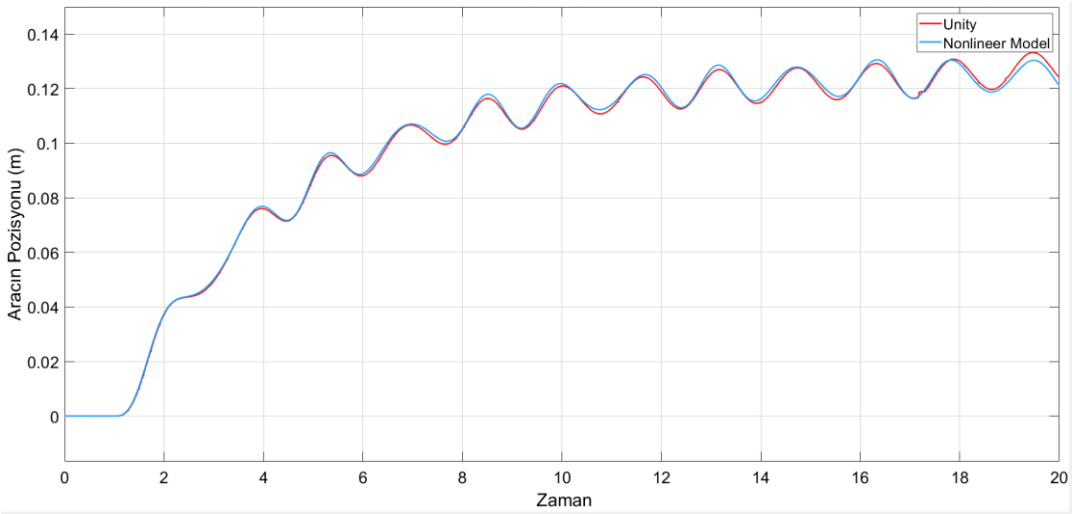
Doğrusal olmayan modelin simülasyonu ve Unity ortamındaki simülatörün farklı girişler için sonuçları şekil 4.40, 4.41 ve 4.42'de görülmektedir. Giriş sarkaç 0 derece açı başlangıç durumundayken uygulanmıştır yani sarkaç yere bakar durumdadır.



Şekil 4.40 : Sistemin girişine uygulanan sinüs işareti.



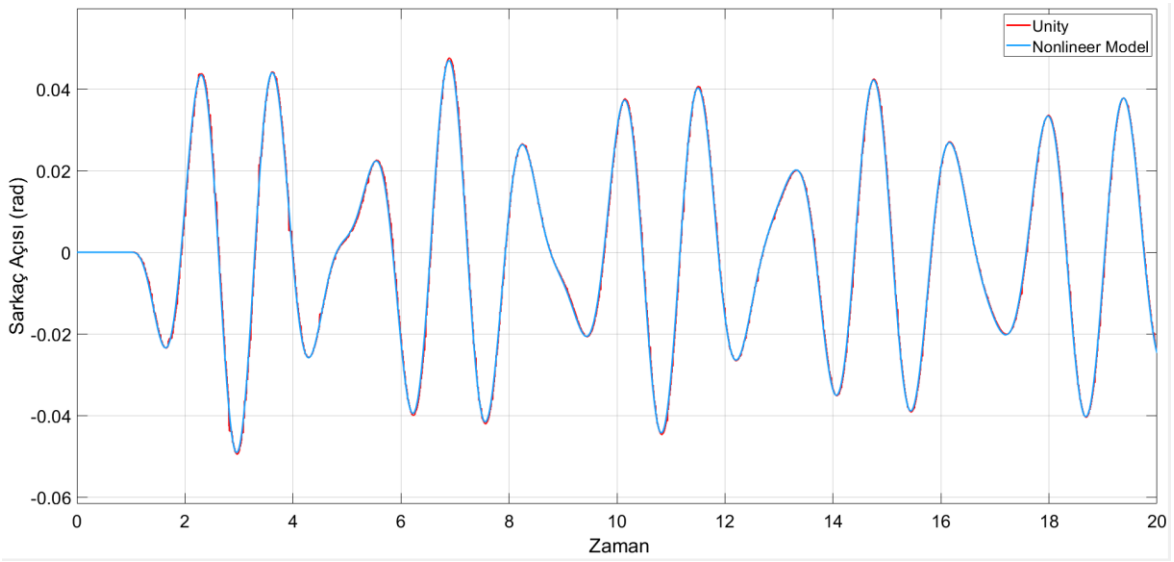
Şekil 4.41 : Uygulanan giriş ile sarkacın açısının değişimi.



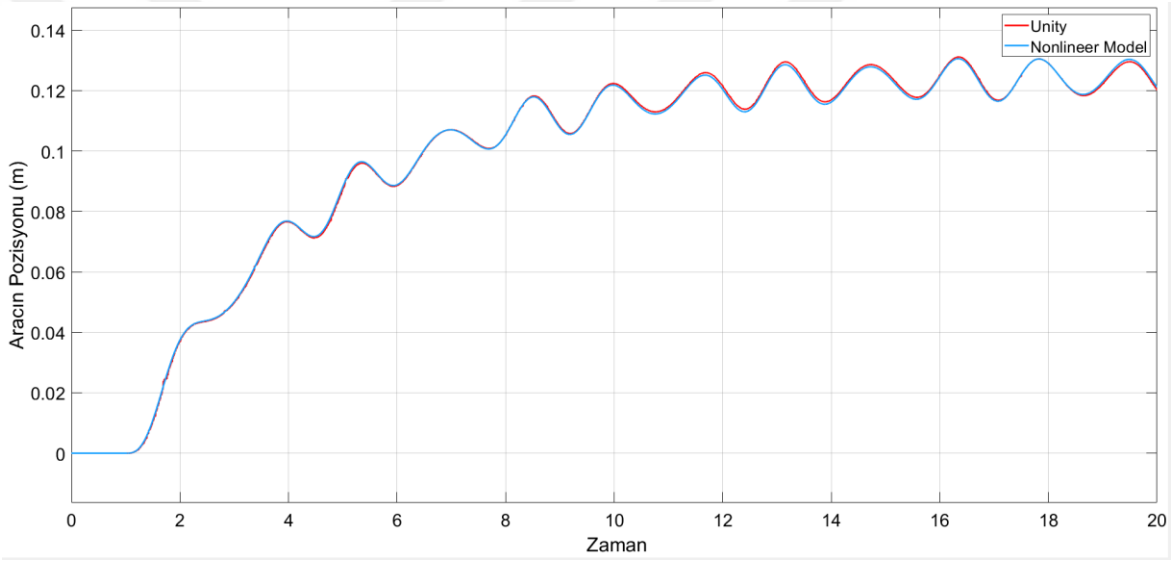
Şekil 4.42 : Uygulanan giriş ile aracın pozisyonunun değişimi.

Şekil 4.40, 4.41 ve 4.42' de görülebileceği gibi Unity ortamındaki simülatörün ve doğrusal olmayan modelin simülasyonu aynı giriş için oldukça yakın fakat kısmen farklı sonuçlar vermektedirler. Bu fark top ve plaka sisteminde olduğu gibi doğrusal olmayan modeldeki eksiklerden kaynaklanmaktadır. Gerçek dünyada sürtünmelerden dolayı sistemde oluşan enerji kayıpları doğrusal olmayan modele eklenmemiştir. Fakat Unity ortamında oluşturulan emülasyonda varsayılan olarak bu sönüm (damping) değeri sisteme eklenmektedir.

Bu farkın giderilmesi amacıyla Unity motorunda sarkacın Rigidbody bileşenindeki angular drag değeri 0' olarak girilmiş ve test tekrarlanmıştır. Bu yeni güncellemeyle aynı giriş sinyali ile yapılan testlerin sonuçlar şekil 4.43 ve 4.44'de ki gibidir.

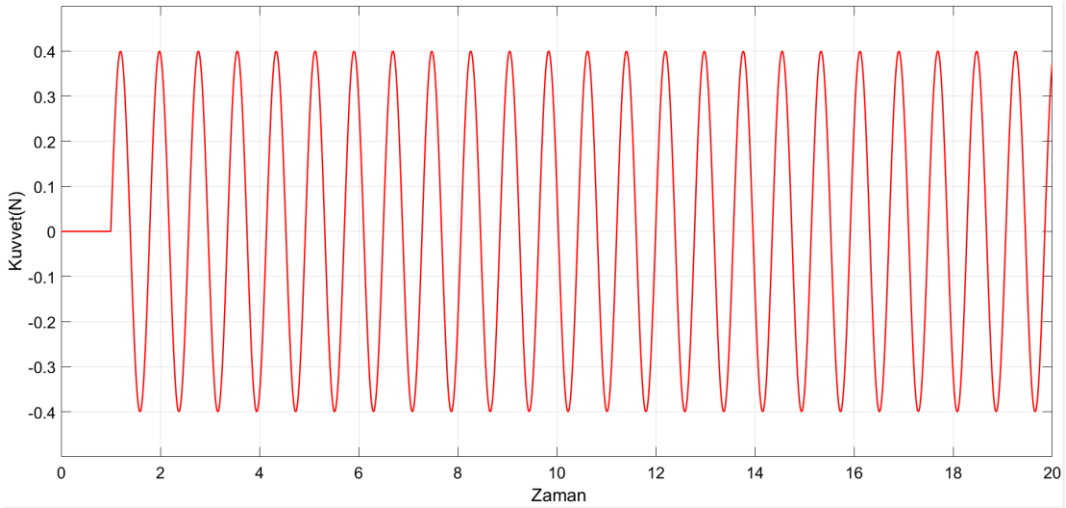


Şekil 4.43 : Uygulanan giriş ile sarkacın açısının değişimi.

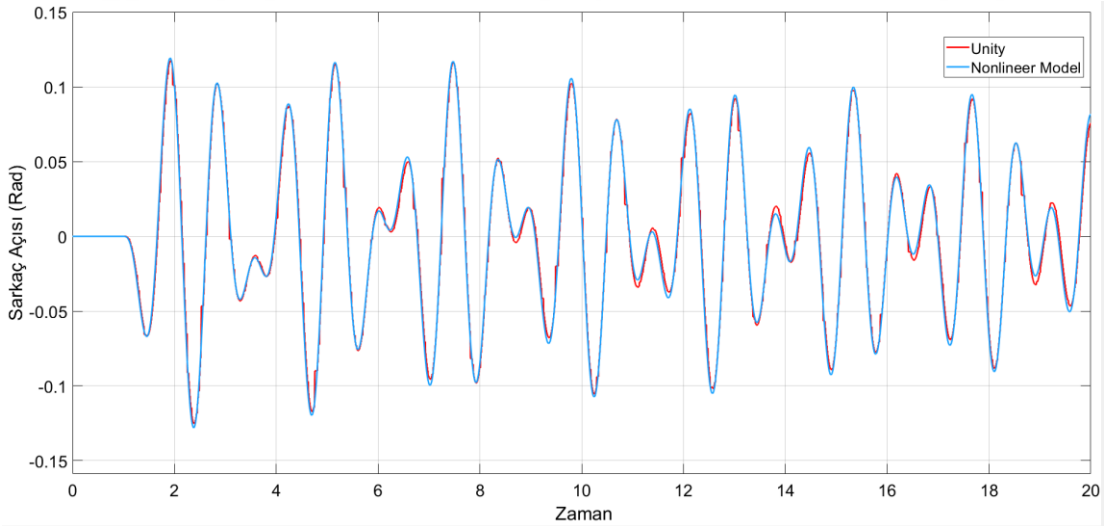


Şekil 4.44 : Uygulanan giriş ile aracın pozisyonunun değişimi.

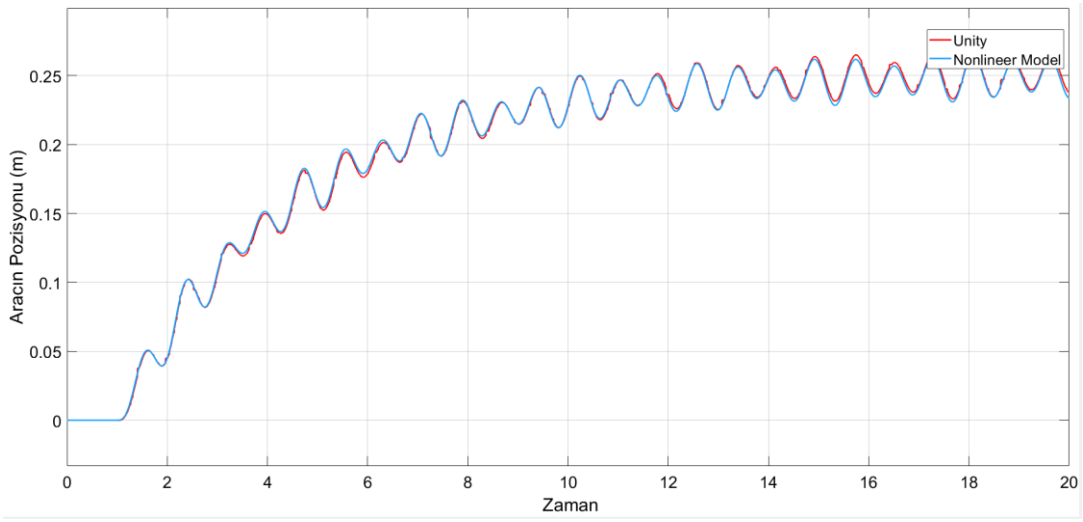
Şekil 4.43 ve 4.44'de görülebileceği gibi yapılan düzenlemeden sonra doğrusal olmayan modelin simülasyonu ve Unity motoru ortamında tasarlanan simülasyon ile aynı girişler için neredeyse aynı çıkış işaretleri vermiştir. Bu noktada sisteme farklı genlik ve frekansta giriş işaretleride uygulanarak daha detaylı bir analiz yapılmıştır.



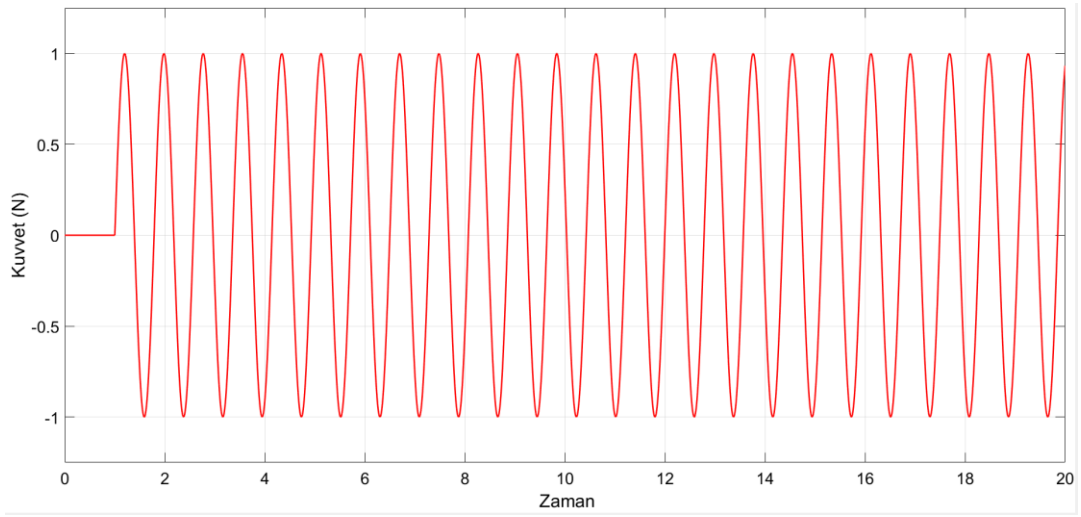
Şekil 4.45 : Sistemin girişine uygulanan sinüs işareti.



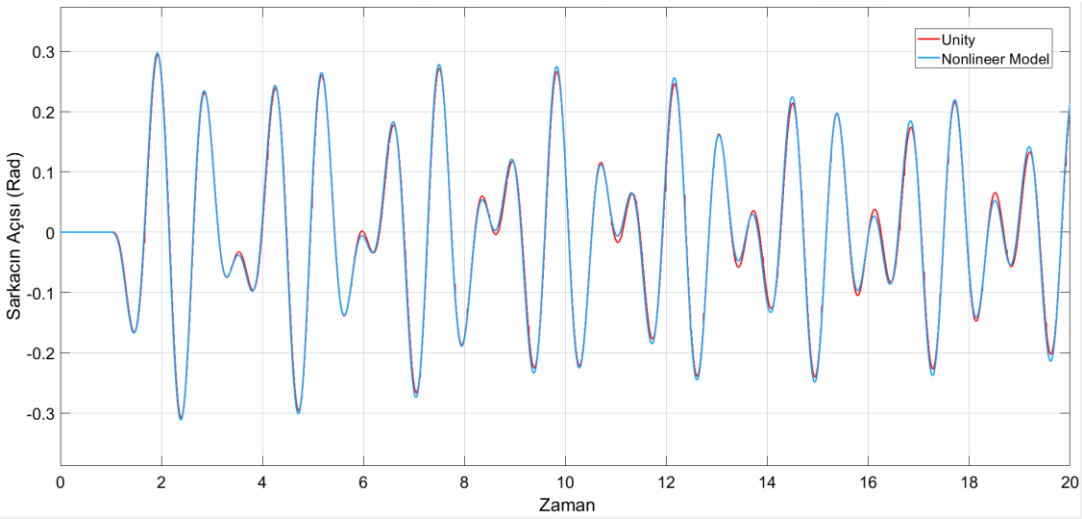
Şekil 4.46 : Uygulanan giriş ile sarkacın açısının değişimi.



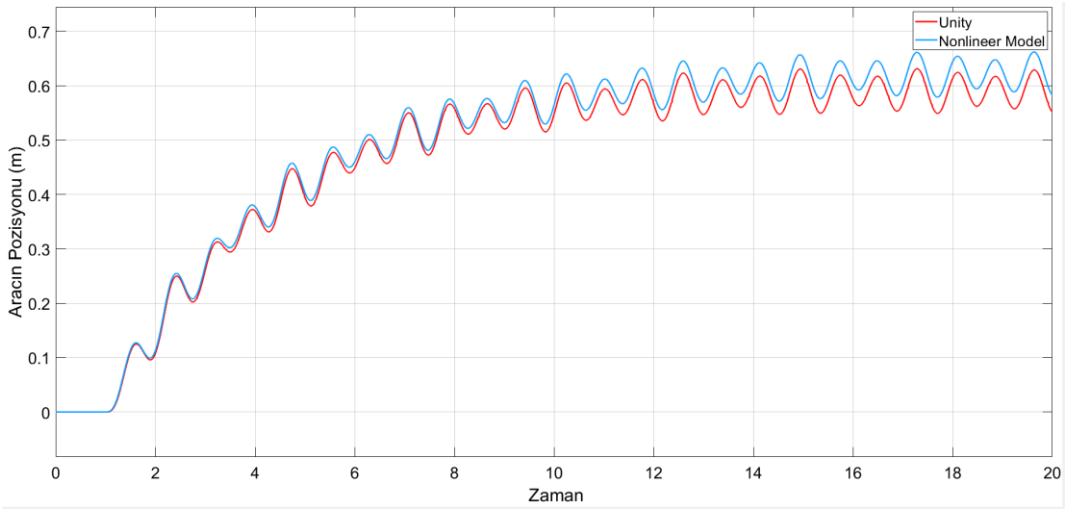
Şekil 4.47 : Uygulanan giriş ile aracın pozisyonunun değişimi.



Şekil 4.48 : Sistemin girişine uygulanan sinüs işareti.

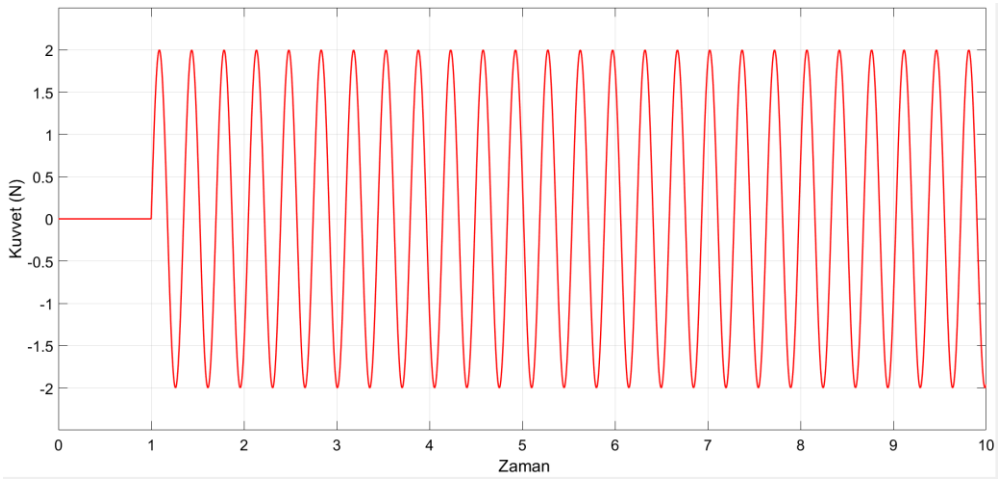


Şekil 4.49 : Uygulanan giriş ile sarkacın açısının değişimi.

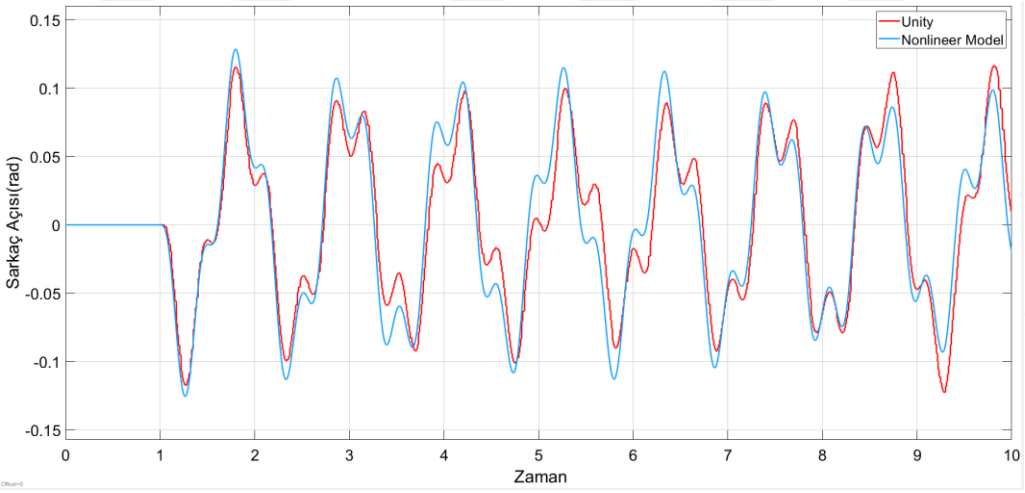


Şekil 4.50 : Uygulanan giriş ile aracın pozisyonunun değişimi.

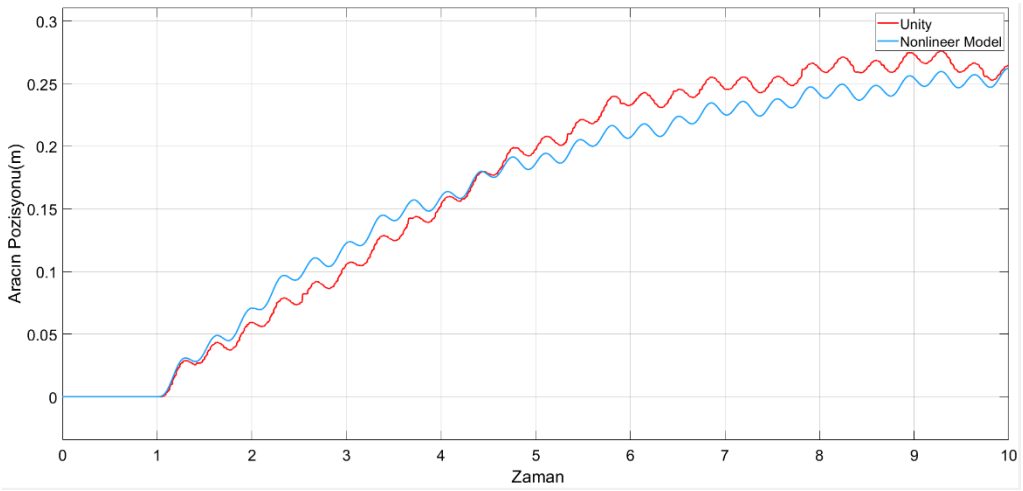
Uygulanan girişin frekansı ve genliği arttırıldığında simülatör ile doğrusal olmayan model arasındaki farklılıklar artmaktadır.



Şekil 4.51 : Sistemin girişine uygulanan sinüs işareti.



Şekil 4.52 : Uygulanan giriş ile sarkacın açısının değişimi.



Şekil 4.53 : Uygulanan giriş ile aracın pozisyonunun değişimi.

Bu artan farkın sebebi top ve plaka sistemindeki gibi doğrusal olmayan modelleme sırasında yapılan hatalardır. Hatalara rağmen simülatörün genel davranışı doğrusal olmayan modele oldukça benzemektedir. Bu karşılaştırmalar ışığında varılabilir ki geliştirilen simülatör sistemi ile benzer dinamikleri taşımaktadır.

4.4 Vinç Sistemi

Vinçler insan gücünün yeterli olmadığı büyük ve ağır cisimlerin kaldırılmasında ve taşınmasında kullanılan iş makineleri olarak tanımlanırlar. Genel olarak bakıldığında vinçlerin temel mekanikleri aynı olsada, kullanıldıkları alanlara göre yapıları farklılık gösterebilir (Şekil 4.54).

Şekil 4.54 : Gezer köprülü vinçler.

Vincin doğrusal hareketi taşımakta olduğu yük üzerinde salınımına yol açar. Oluşan bu salınım taşıma verimliliğini düşürmektedir. Bu problemin çözümüne ilişkin literatürde birçok çalışma mevcuttur. Bu çalışmalar birçok farklı firmanın deney setleri üzerinde denenmektedirler. Bu bölümde bu amaca yönelik gezer köprülü bir vinç sisteminin Unity ortamında simülatörü ve deney seti oluşturulmuştur. Yapılan simülatör, sistemin doğrusal olmayan modeli ile karşılaştırılmış ve fizik motorunun performansı test edilmiştir.

4.4.1 Sistemin analizi ve modellenmesi

4.4.1.1 Kabuller

Sistem aşağıdaki kabuller yapılarak modellenmiştir.

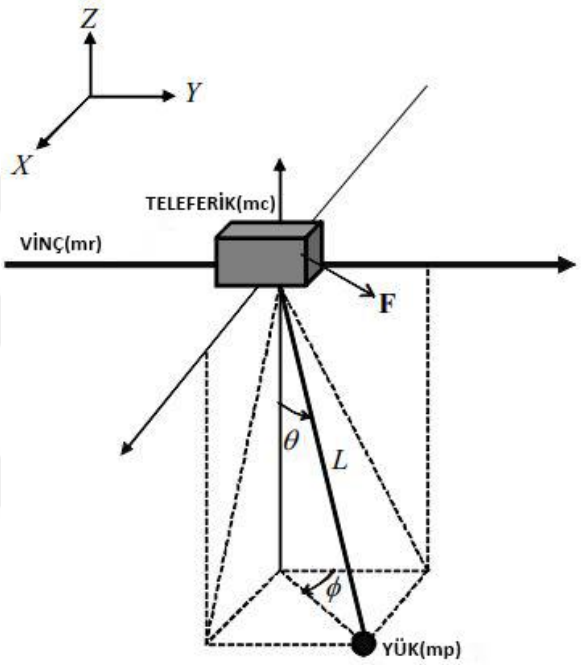
1. Teleferik ve ray arasındaki sürtünme ihmal edilmiştir.

2. Teleferik ve yükün ağırlıkları geometrik merkezde ve noktasal olarak kabul edilmiştir.

3. Yük ile teleferik arasındaki ip katı bir çubuk olarak düşünülmüş, ipin uzaması ve katlanması ihmal edilmiştir.

4.4.1.2 Matematiksel modelleme

Sistemin matematiksel modeli bulunurken şekil 4.55'de ki serbest cisim diyagramı kullanılmıştır [10].



Şekil 4.55 : Sistemin serbest cisim diyagramı.

Sistemin matematiksel modellemesinde Euler-Lagrange yöntemi kullanılmıştır [10,11,12,13]. Bu amaçla bir L (Lagrangian) fonksiyonu tanımlanmıştır.

$$L = E_k - E_p \tag{4.30}$$

Sistemin kinetik enerjisi vincin, teleferiğin ve yükün kinetik enerjisi olmak üzere 3 alt denkleme ayrılabilir.

$$E_k = E_{kp} + E_{kr} + E_{kc} \tag{4.31}$$

Vincin kinetik enerjisi sadece x yönündeki çizgisel hızdan kaynaklanmaktadır.

$$E_{kr} = \frac{1}{2} m_r \dot{x}^2 \tag{4.32}$$

Teleferiğin kinetik enerjisi x ve y yönündeki olan çizgisel hızdan kaynaklanmaktadır.

$$E_{kc} = \frac{1}{2} m_c \dot{x}^2 + \frac{1}{2} m_c \dot{y}^2 \tag{4.33}$$

Yükün kinetik enerjisi ise yükün x ve y yönündeki çizgisel hareketi ile θ ve ϕ açısal hareketinden kaynaklanmaktadır.

$$E_{kp} = \frac{1}{2}m_p\dot{x}^2 + \frac{1}{2}m_p\dot{y}^2 + +m_pLcos\theta sin\phi\dot{x}\dot{\theta} + m_pLsin\theta cos\phi\dot{x}\dot{\phi} + m_pLcos\theta cos\phi\dot{y}\dot{\theta}$$
$$- m_pLsin\theta sin\phi\dot{y}\dot{\phi} + \frac{1}{2}\left(m_pL^2 + J\right) + \frac{1}{2}\left(m_pL^2 + sin^2\theta + J\right)\dot{\phi}^2 \quad (4.34)$$

Sistemin potansiyel enerjisi yükün yüksekliğinin değişimi sonucu oluşmaktadır.

$$E_p = E_{pp} = -m_pgLcos\theta \quad (4.35)$$

Bu aşamada L fonksiyonu artık yazılabilir.

$$L = (E_{kp} + E_{kr} + E_{kc}) - E_{pp} \quad (4.36)$$

Sistemde teleferiğin x ve y eksenlerindeki pozisyonu, yükünde θ ve ϕ açısı olmak üzere 4 değişken mevcuttur. Bu değişkenler için Lagrangian fonksiyonları aşağıdaki gibi tanımlanır.

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = f_x \qquad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{y}}\right) - \frac{\partial L}{\partial y} = f_y$$
$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = 0 \qquad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\phi}}\right) - \frac{\partial L}{\partial \phi} = 0 \quad (4.37)$$

Denklem 4.7'daki Lagrangian fonksiyonlarının sonuçlarının bulunmasıyla sistemin modeli matris formuyla denklem 4.39 ve 4.40'daki gibi yazılabilir.

$$F = \begin{bmatrix} f_x & f_y & 0 & 0 \end{bmatrix}^T \quad q = [x \quad f_y \quad 0 \quad 0]^T \quad (4.38)$$

$$m_{11} = m_p + m_r + m_c \quad m_{13} = m_pLcos\theta sin\phi \quad m_{14} = m_pLsin\theta cos\phi$$
$$m_{22} = m_p + m_c \quad m_{23} = m_pLcos\theta cos\phi \quad m_{24} = -m_pLsin\theta sin\phi$$
$$m_{31} = m_pLcos\theta sin\phi \quad m_{32} = m_pLcos\theta cos\phi \quad m_{33} = m_pL^2 + J$$
$$m_{41} = m_pLsin\theta cos\phi \quad m_{42} = -m_pLsin\theta sin\phi \quad m_{44} = m_pL^2sin^2\theta + J$$

$$M(q) = \begin{bmatrix} m_{11} & 0 & m_{13} & m_{14} \\ 0 & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & 0 \\ m_{41} & m_{42} & 0 & m_{44} \end{bmatrix} \quad (4.39)$$

$$c_{13} = -m_pLsin\theta sin\phi\dot{\theta} + m_pLcos\theta cos\phi\dot{\phi} \quad c_{14} = m_pLcos\theta sin\phi\dot{\theta} - m_pLsin\theta sin\phi\dot{\phi}$$
$$c_{23} = -m_pLsin\theta cos\phi\dot{\theta} + m_pLcos\theta sin\phi\dot{\phi} \quad c_{24} = -m_pLcos\theta sin\phi\dot{\theta} - m_pLsin\theta cos\phi\dot{\phi}$$
$$c_{34} = -m_pL^2sin\theta cos\theta\dot{\phi}$$
$$c_{43} = m_pL^2sin\theta cos\theta\dot{\phi} \quad c_{44} = m_pL^2sin\theta cos\theta\dot{\theta}$$

$$C(q,\dot{q}) = \begin{bmatrix} 0 & 0 & c_{13} & c_{14} \\ 0 & 0 & c_{23} & c_{24} \\ 0 & 0 & 0 & c_{34} \\ 0 & 0 & c_{43} & c_{44} \end{bmatrix} \quad (4.40)$$

$$G(q) = [0 \quad 0 \quad m_p g L sin\theta \quad 0] \quad (4.41)$$

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = F \quad (4.42)$$

Bu noktada sistemin doğrusal olmayan modeli denklem 4.42 olarak elde edilmiş bulunmaktadır. Kancanın XZ ve YZ düzlemindeki açıları ise denklem 4.45 ve 4.46 daki gibi elde edilir.

$$\theta_{xz} = tan^{-1}(tan\theta sin\phi) \quad (4.43)$$

$$\theta_{yz} = sin^{-1}(sin\theta cos\phi) \quad (4.44)$$

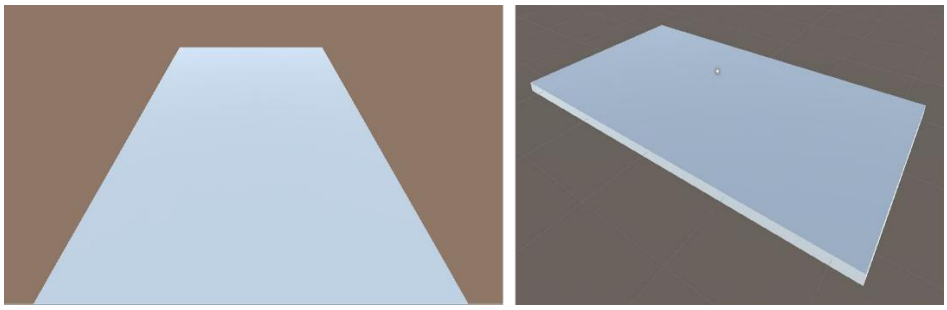
4.4.2 Sistemin Unity ortamına taşınması

Sistemin değerleri çizelge 4.5'te ki gibi seçilmiştir.

Çizelge 4.5 : Sistem değerleri.

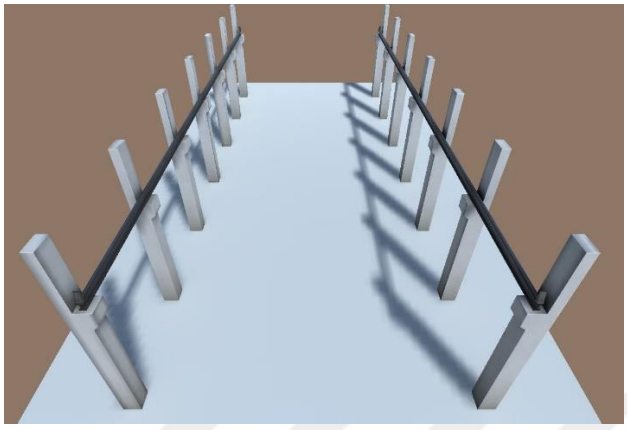
Sembol	Tanım	Değer
m_p	Yükün ağırlığı	1.0$[kg]$
m_r	Vincin ağırlığı	1.0$[kg]$
m_c	Teleferiğin ağırlığı	1.0$[kg]$
g	Yer çekimi ivmesi	9.81 $[m/s^2]$
L	Yükün teleferiğe uzaklığı	2.0 $[m]$
J	Yükün eylemsizliği	1.0 $[kg\ m^2]$

Sistemin Unity ortamında gerçeklenmesine öncelikle tüm sistemin üzerine duracağı yer modelinin oluşturulması ile başlanmıştır. Bu amaçla yer (Ground) objesi oluşturulmuştur (Şekil 4.56). Daha sonra yer objesine Collider bileşeni eklenmiştir.



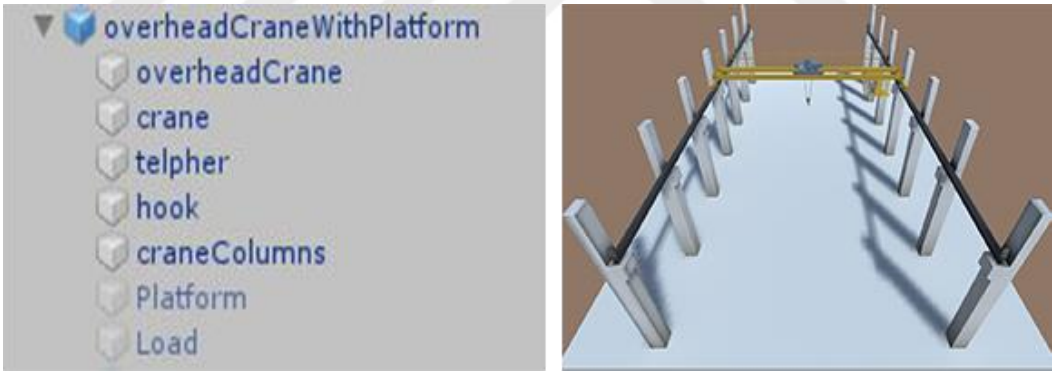
Şekil 4.56 : Yer objesinin farklı açılardan görünüşü.

Daha sonra kullanılacak olan 3D vinç modelinin Unity ortamına eklenmesine geçilmiştir. Vinç modeli kompleks olduğu için gereken model parçaları satın alınmıştır. Bu parçalar ile öncelikle vincin x ekseninde hareket edeceği ray sisteminin 3 boyutlu modeli oluşturulmuştur (Şekil 4.57).



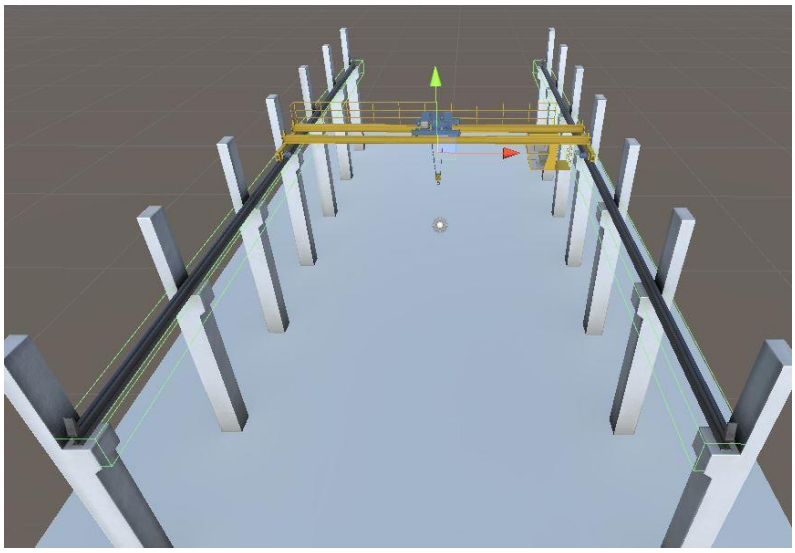
Şekil 4.57 : Ray objesinin yer objesi üzerine yerleştirilmiş hali.

Ray objesinden sonra bu obje üzerinde hareket edecek vinç objesi (crane),vincin üzerinde hareket edecek teleferik objesi (telpher) ve bu teleferiğe bağlı olan kanca(hook) sisteminin 3 boyutlu modelleri oluşturulmuştur (Şekil 4.58).



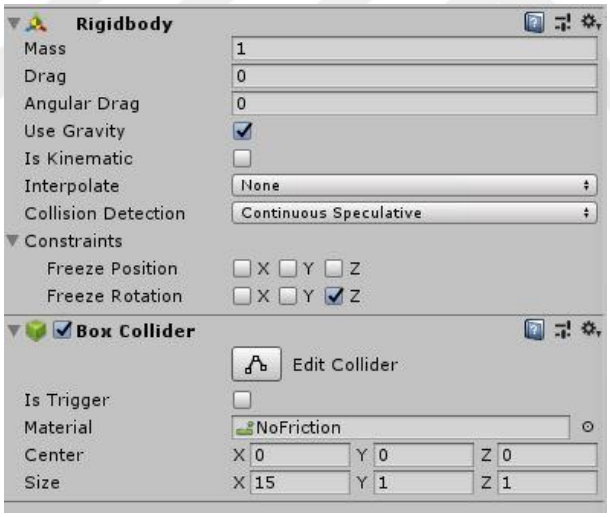
Şekil 4.58 : Oluşturulan objelerin listesi ve vincin 3 boyutlu modeli.

Bu aşamada sistem üzerinde iki eksende çizgisel hareket sağlayan vinç ve teleferik araçları ile teleferiğe bağlı olan bir kanca parçalarının görsel olarak oluşturulması tamamlanmıştır. Daha sonra bu üç parçanın fizik bileşenlerinin ayarlama işlemine geçilmiştir. Fiziksel bileşenlerin eklenmesine öncelikle vincin üzerinde hareket edeceği ray ile başlanmıştır. Rayın çarpışma fiziğine sahip olabilmesi için iki tarafına da collider bileşeni eklenmiştir (Şekil 4.59). Bu sayede rayın üzerine konulacak vincin ray üzerinde durabilmesi sağlanmıştır.



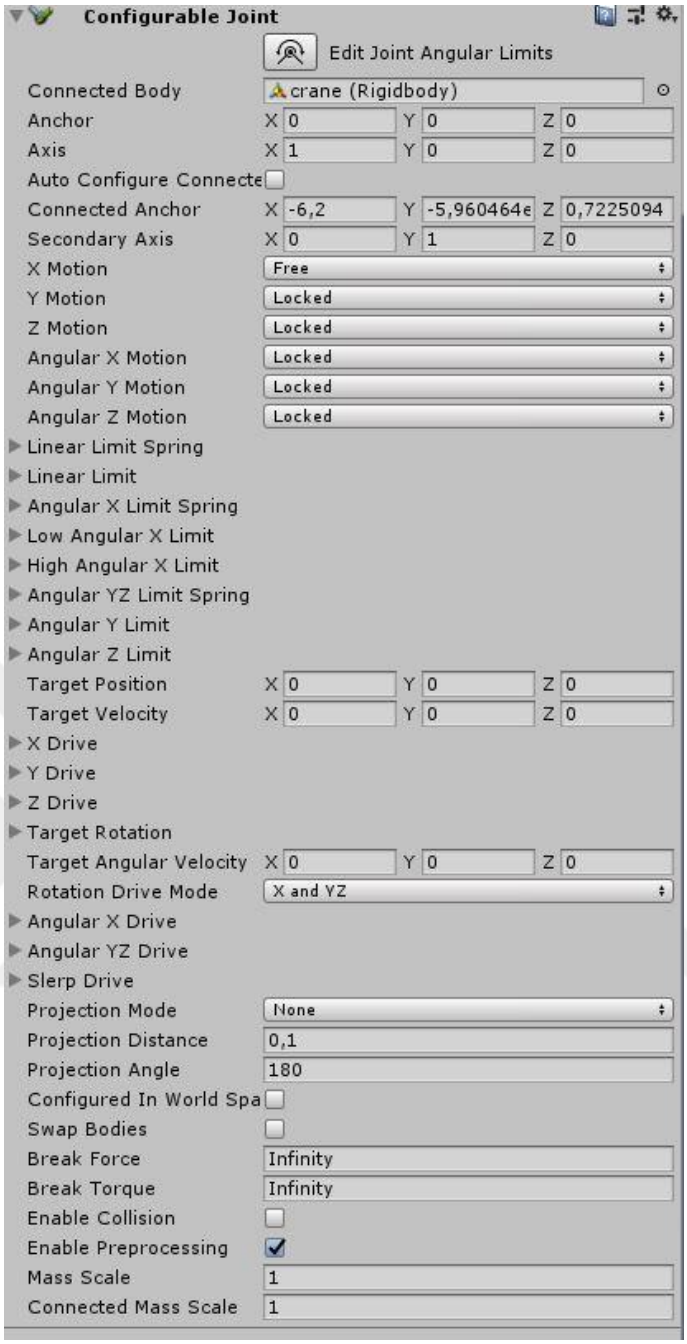
Şekil 4.59 : Rayın üzerinde yerleştirilen collider bileşeni.

Daha sonra vinç objesinin fiziğine geçilmiştir. Vincin kütle dinamiğine sahip olabilmesi için Rigidbody bileşeni, çarpışma dinamiğine sahip olabilmesi için Collider bileşeni eklenmiştir. Vincin açısal olarak hareket etmeyeceği için Rigidbody bileşeni üzerinde açısal hareket kısıtlanmıştır (Şekil 4.60).



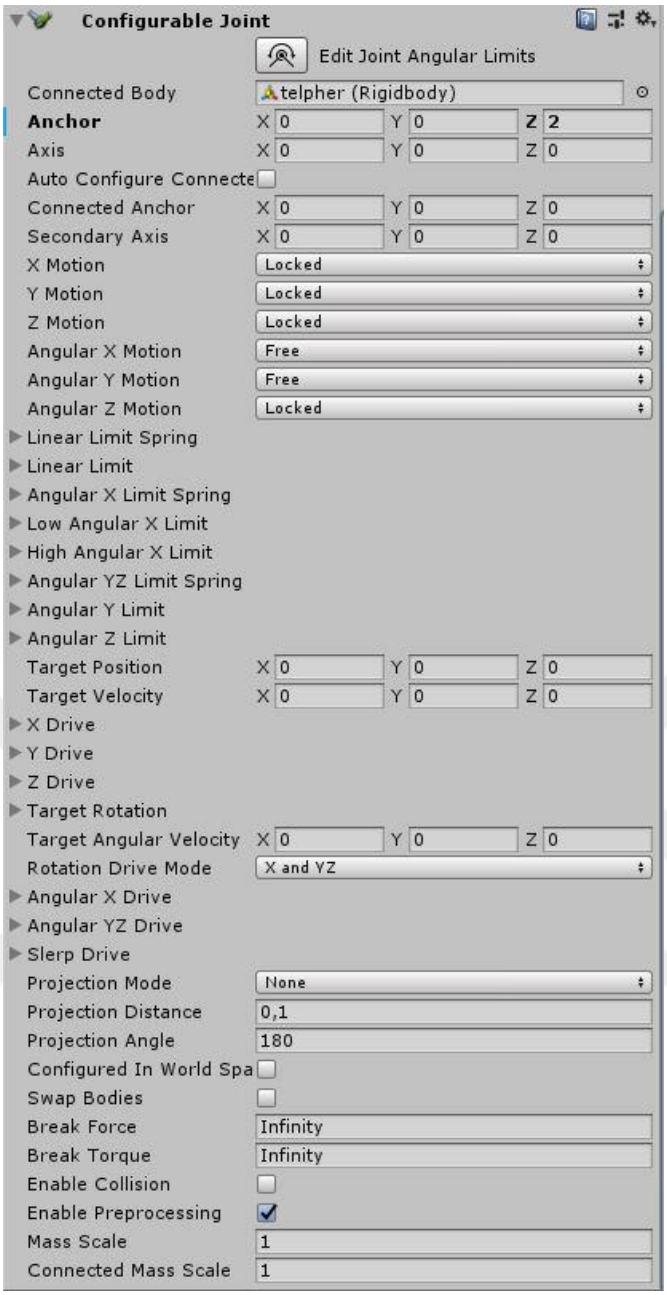
Şekil 4.60 : Vinç üzerindeki Rigidbody ve Collider bileşenleri.

Vinç üzerindeki teleferik objesinede öncelikle Rigidbody bileşeni eklenmiştir. Ayrıca teleferik vinç objesine bağlı olarak hareket edeceğinden Configurable Joint bileşeni kullanılarak teleferiğin vinç sistemine entegre şekilde hareket etmesi sağlanmıştır (Şekil 4.61). Burada dikkat edilmesi gereken teleferik sadece tek eksen üzerinde (y ekseni) serbestçe hareket edebilecek olmasıdır. Diğer eksenlerde ise vincin hareketine bağlı olarak hareket edecektir. Bu amaçla Configurable Joint bileşeni üzerinde gerekli düzenlemeler yapılmıştır.



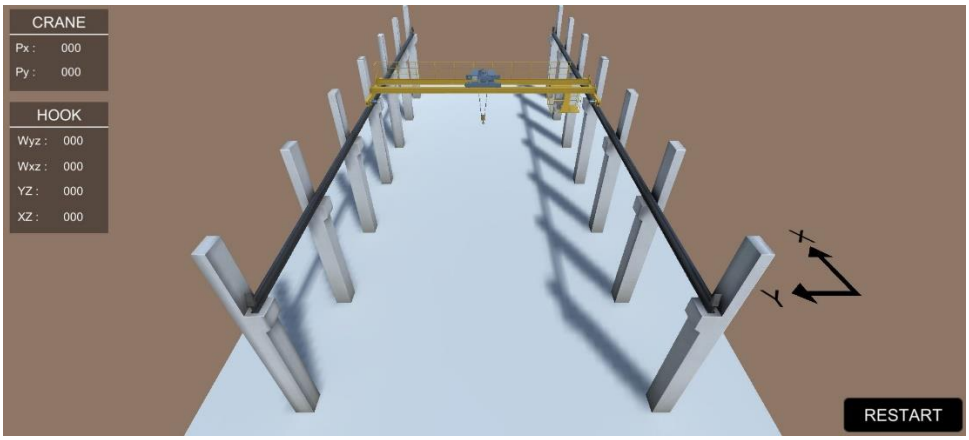
Şekil 4.61 : Teleferik üzerindeki Configurable Joint bileşeni ve ayarları.

Son olarak teleferiğe bağlı olan kanca objesinin fiziğine geçilmiştir. Bu aşamada kancaya öncelikle Rigidbody bileşeni eklenmiştir. Daha Configurable Joint bileşeni kancaya eklenerek kancanın 2 eksende açısal hareket yaparken diğer eksenlerde ise teleferiğe bağlı olarak hareket etmesi sağlanmıştır (Şekil 4.62). Önceki sistemlerde yapılan damping ayarı bu sistemin bileşenlerine de ‘0’ olarak eklenmiştir.



Şekil 4.62 : Kanca üzerindeki Configurable Joint bileşeni ve ayarları.

Simulasyon ekranının sol üst köşesine UI bileşenleri kullanılarak basit bir kullanıcı arayüzü eklenmiştir. Vincin ve teleferiğin pozisyonu, kancanın açısal hızı ve açısı bilgileri bu ekrandan izlenebilir (Şekil 4.63).

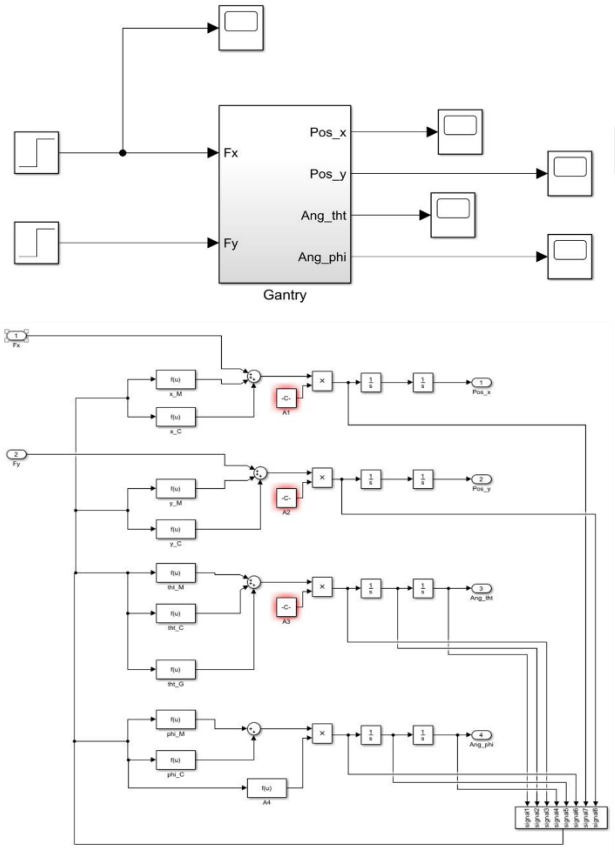


Şekil 4.63 : Geliştirilen simülatörün tüm bileşenleriyle birlikte görüntüsü.

4.4.3 Matematiksel modelin ve simülatörün karşılaştırılması

4.4.3.1 Matematiksel modelin simülasyonu

Elde edilen doğrusal olmayan model Simulink ortamına taşınmıştır (Şekil 4.64).

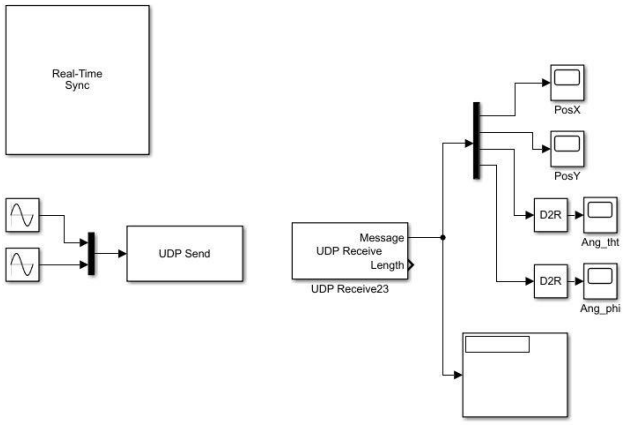


Şekil 4.64 : Doğrusal olmayan model simülasyonu.

Sisteme çeşitli giriş işaretleri uygulanmış ve sonuçlar kaydedilmiştir.

4.4.3.2 Simülatörün Simulink ortamından kontrolü

Unity modelinin Simulink ortamından kontrolü için daha önceki başlıklarda anlatılan UDP protokolü kullanılmıştır. Bu amaçla UDP Send bloğu sisteme giriş uygulamak için UDP Receive bloğu ise sistemin çıkışlarını almak için kullanılmıştır (Şekil 4.65). Unity tarafında 25000 portu alıcı port 26000 portu ise verici portu olarak ayarlanmıştır.

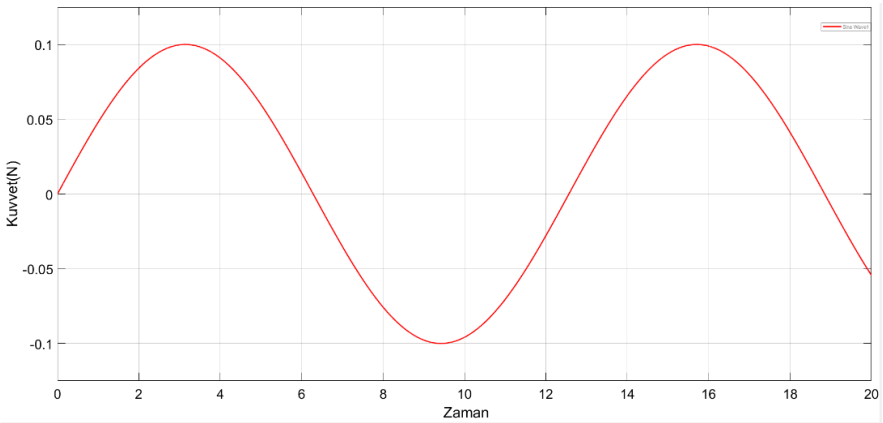


Şekil 4.65 : Simülatörün kontrolü için Simulink'te kurulan model.

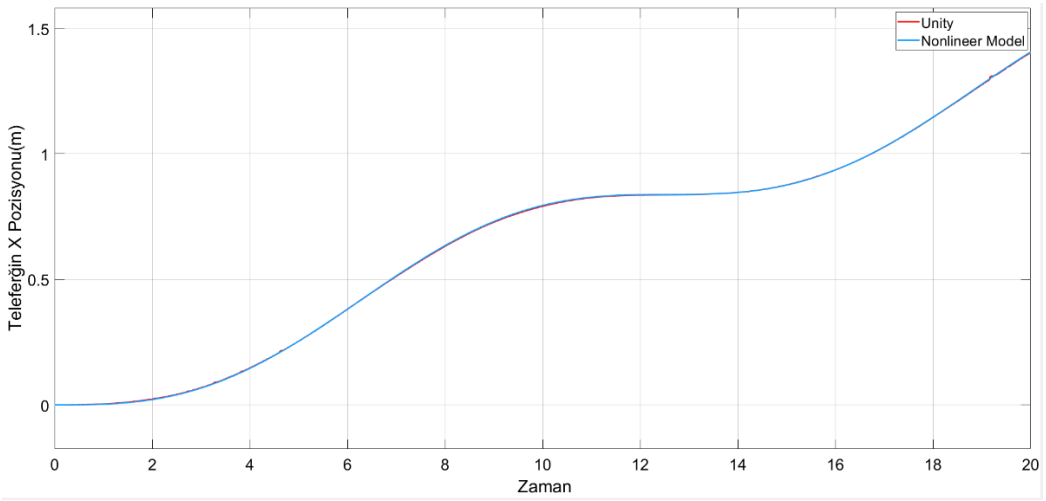
Haberleşme ayarlandıktan sonra sisteme çeşitli giriş işaretleri uygulanmış ve sonuçlar
kaydedilmiştir.

4.4.3.3 Karşılaştırmalar

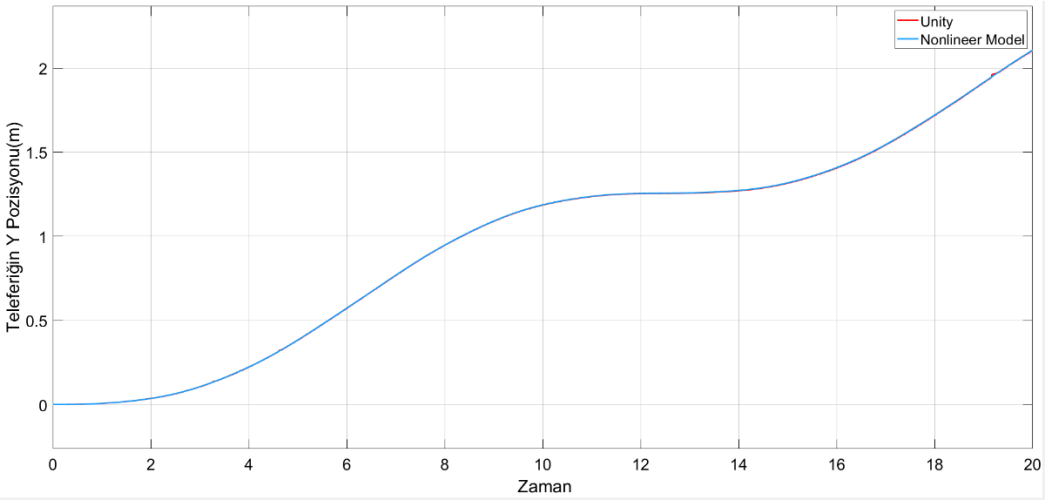
Doğrusal olmayan modelin simülasyonu ve Unity ortamındaki simülatöre farklı girişler için sonuçları şekil 4.67, 4.68, 4.69 ve 4.70'te görülmektedir. Başlangıçta teleferik (0, 0) pozisyonunda, XZ ve YZ düzlemdeki açıları ise 0 derecedir.



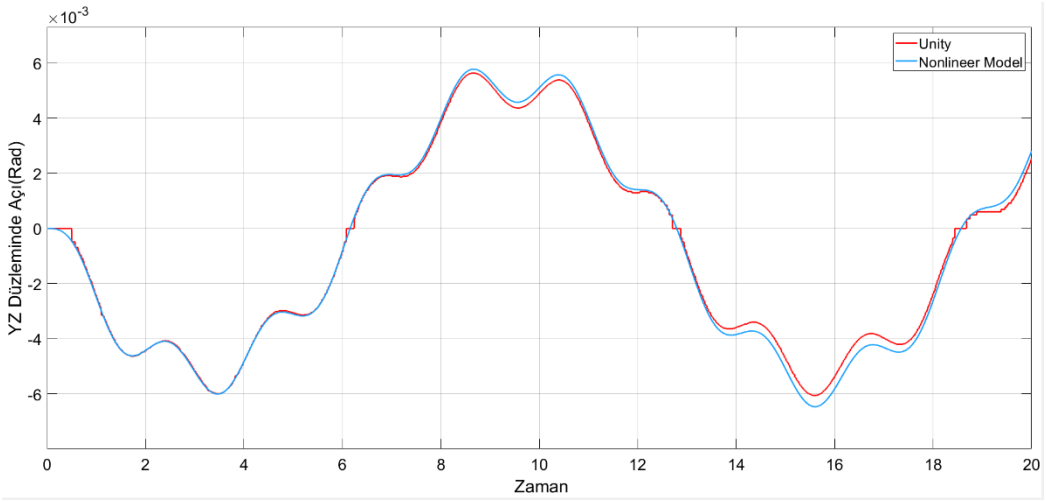
Şekil 4.66 : Sistemin iki girişine de uygulanan sinüs işareti.



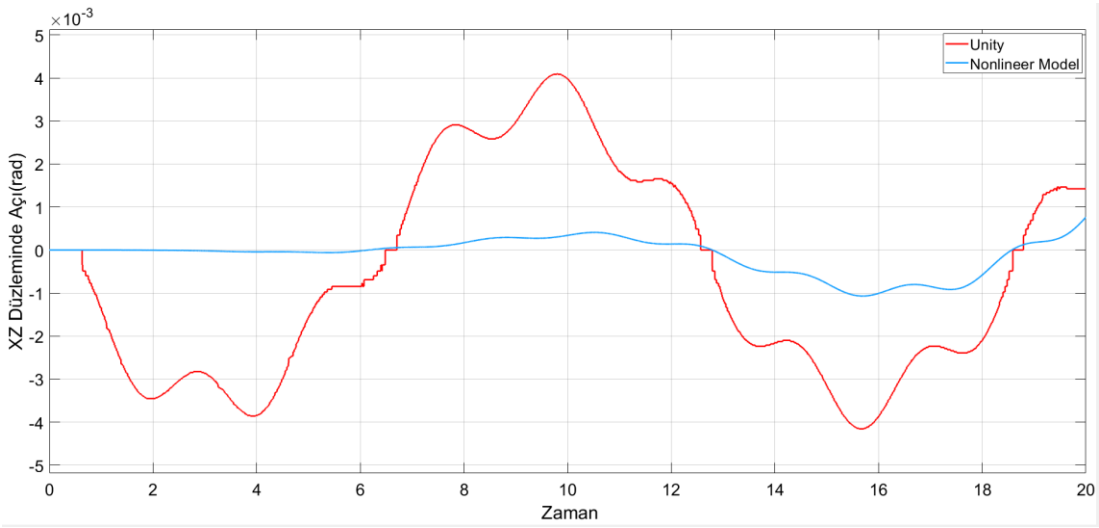
Şekil 4.67 : Teleferiğin X eksenindeki pozisyonu.



Şekil 4.68 : Teleferiğin Y eksenindeki pozisyonu.

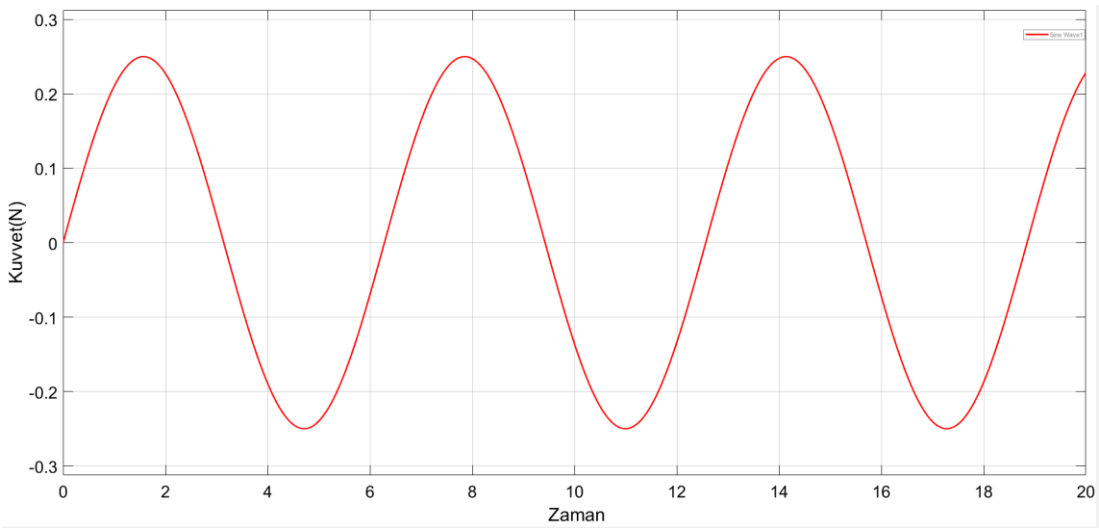


Şekil 4.69 : Kancanın YZ düzlemindeki açısı.

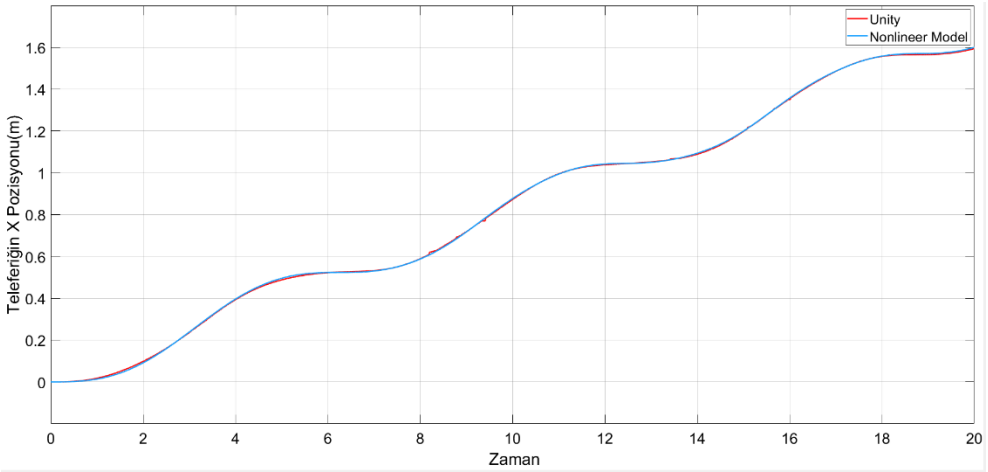


Şekil 4.70 : Kancanın XZ düzlemindeki açısı.

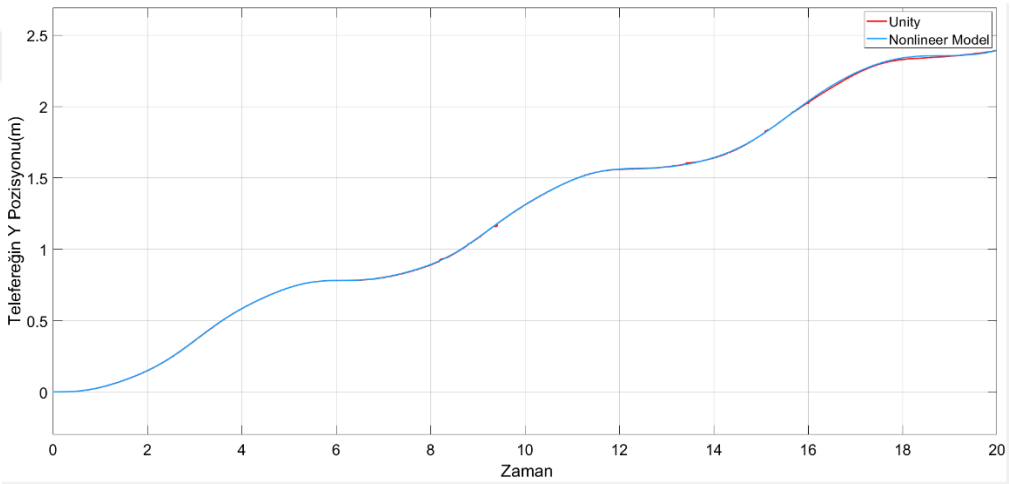
Şekil 4.67 ve 4.68'de görülebileceği üzere teleferiğin pozisyonu doğrusal olmayan model ile yüksek oranda aynı sonucu vermektedir. Şekil 4.69'da görülebileceği üzere kancanın YZ düzlemindeki açısıda sistemin doğrusal olmayan modeli ile oldukça benzer sonuçlar vermektedir. Fakat XZ düzlemindeki açı ise doğrusal olmayan model ile simülatör arasındaki fark oldukça fazla çıkmıştır. Bu noktada sistem girişlerine ilk girişten 2 kat daha yüksek frekans ve genlikte kuvvet uygulanmıştır (Şekil 4.71).



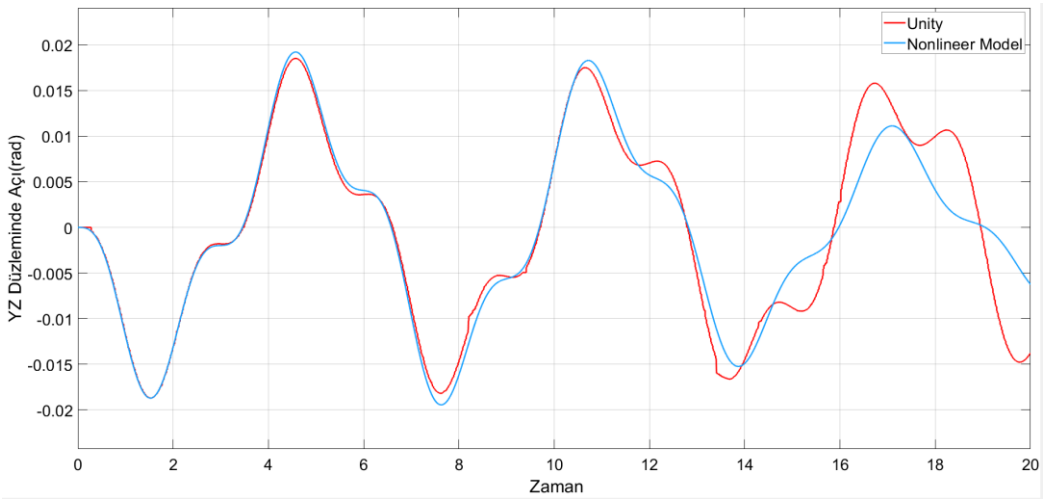
Şekil 4.71 : Sistemin iki girişine de uygulanan sinüs işareti.



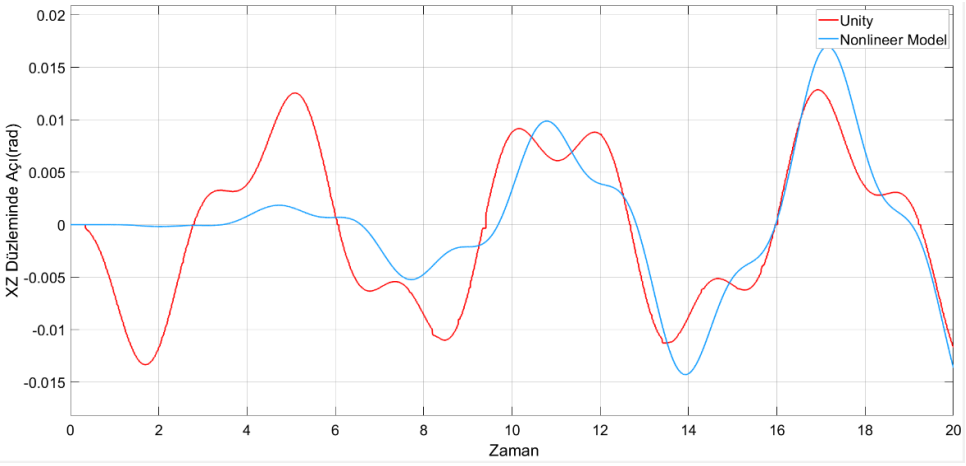
Şekil 4.72 : Teleferiğin X eksenindeki pozisyonu.



Şekil 4.73 : Teleferiğin Y eksenindeki pozisyonu.

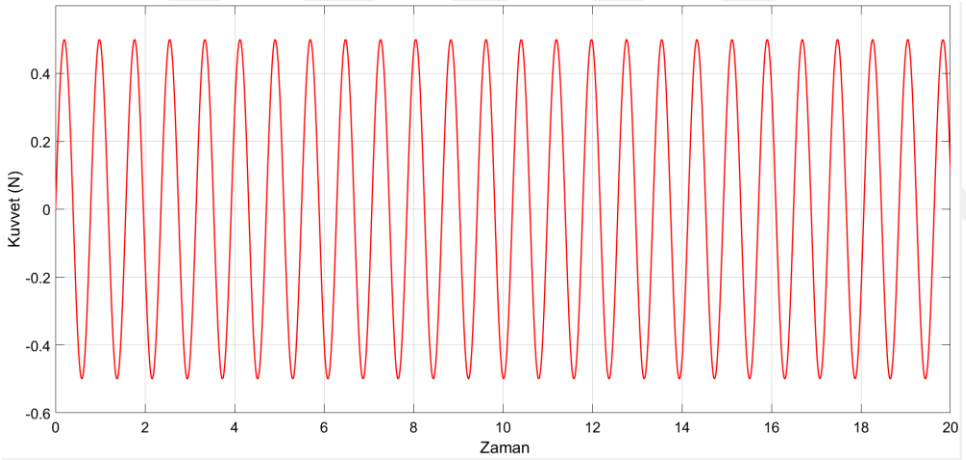


Şekil 4.74 : Kancanın YZ düzlemindeki açısı.

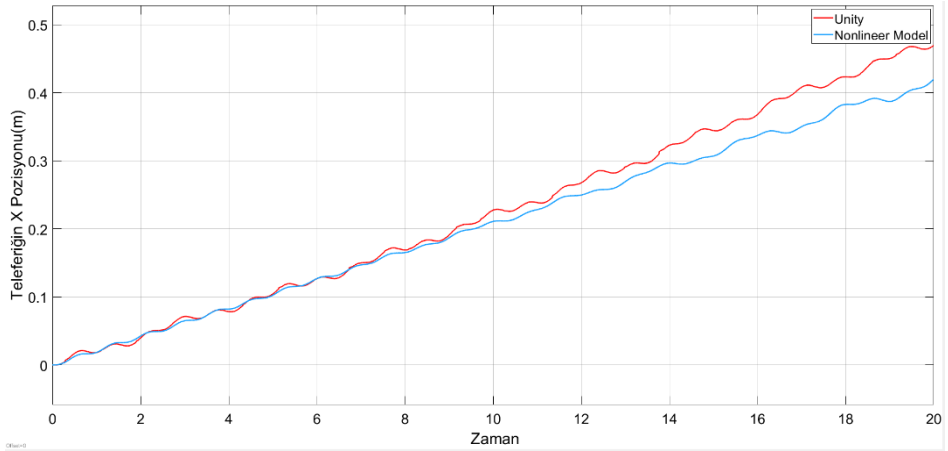


Şekil 4.75 : Kancanın XZ düzlemindeki açısı.

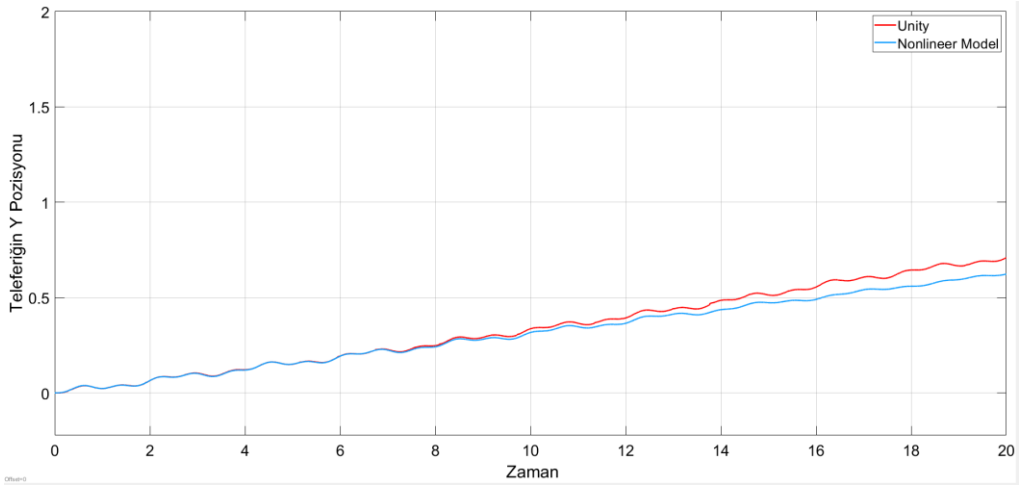
Uygulanan girişin frekansı ve genliği arttırıldığında simülatör ile doğrusal olmayan model arasındaki fark şekil 4.72, 4.73, 4.74 ve 4.75'te görülebileceği üzere artmaktadır. Son olarak sisteme şekil 4.76'da ki yüksek frekanslı giriş uygulanmıştır.



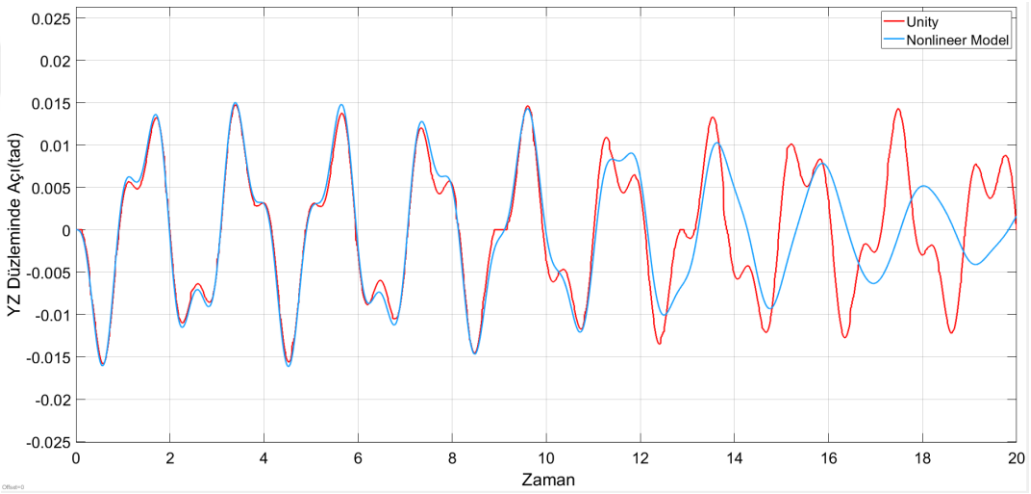
Şekil 4.76 : Sistemin iki girişine de uygulanan sinüs işareti.



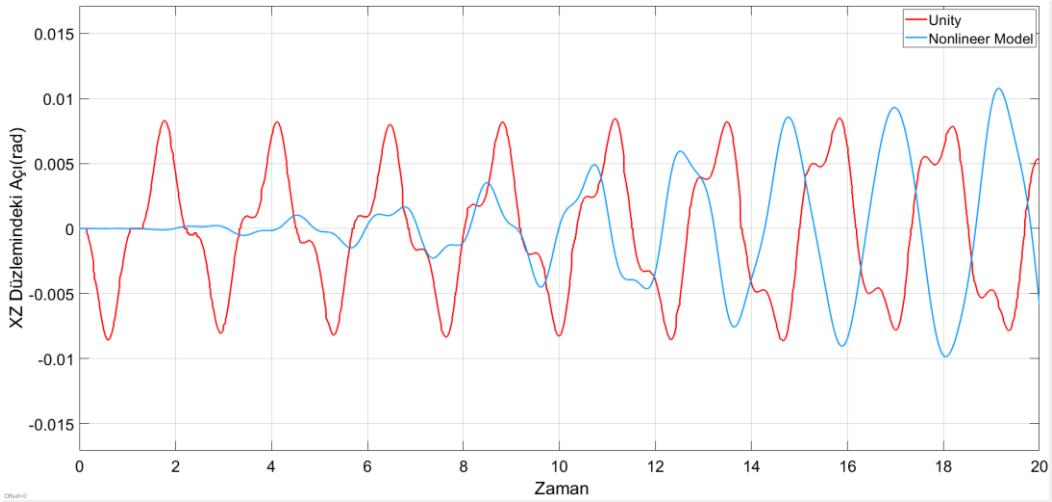
Şekil 4.77 : Teleferiğin X eksenindeki pozisyonu.



Şekil 4.78 : Teleferiğin Y eksenindeki pozisyonu.



Şekil 4.79 : Kancanın YZ düzlemindeki açısı.



Şekil 4.80 : Kancanın XZ düzlemindeki açısı.

Şekil 4.77, 4.78, 4.79 ve 4.80'de görülebilen bu artan farkın sebebi daha önceki sistemlerde olduğu gibi doğrusal olmayan modelleme sırasında yapılan hatalardır.

Hatalara rağmen simülatörün genel davranışı doğrusal olmayan modele oldukça benzemektedir. Bu karşılaştırmalar ışığında varılabilir ki geliştirilen simülatör sistemi ile benzer dinamikleri taşımaktadır.

4.5 Mobil Robot Sistemi

Mobil robotlar birçok robotik ve kontrol projesinde kullanılmaktadırlar (Şekil 4.81). Özellikle otonom mobil robotlar ve sürü halinde hareket eden mobil robotlar günümüzde sıkça çalışılan konular arasındadır. Tezin bu bölümünde bir veya birden fazla mobil robotun kontrol edildiği bir simülatör ortamı oluşturulması amaçlanmıştır.

Şekil 4.81 : Mühendislik eğitimi için üretilmiş ActivityBot mobil robotu.

Simülatör en basit mobil robot şekillerinden biri olan çift tekerlekli mobil robot baz alınarak geliştirilmiştir.

4.5.1 Sistemin Analizi

Sistem temelde sabit plaka üzerine bağlanmış kontrol kartı, bataryalar, sensörler ve hareket etmeyi sağlayan çift tekerden oluşmaktadır. Robotun dengesini sağlamak için önde ve arkada sarhoş tekerlekler bulumaktadır.

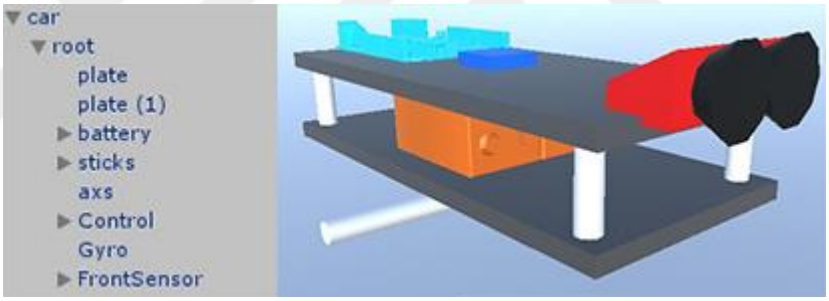
4.5.2 Sistemin Unity ortamına taşınması

Sistemin değerleri çizelge 4.6'daki gibi seçilmiştir.

Çizelge 4.6 : Sistem değerleri.

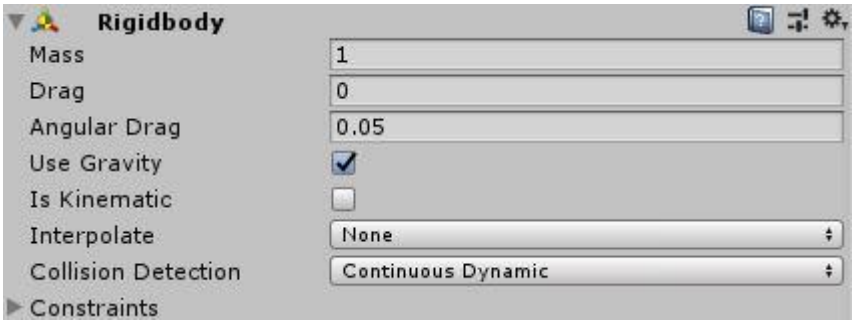
Sembol	Tanım	Değer
m	Sistemin Ağırlığı	1.0 $[kg]$
r_w	Tekerlek Yarıçapı	3.8 $[cm]$
m_w	Tekerlek Ağırlıkları	0.1 $[kg]$
L_x	Araç Uzunluğu	18 $[cm]$
L_y	Araç Genişliği	8 $[cm]$
L_z	Araç Yüksekliği	4.25 $[cm]$

Sistemin Unity ortamında gerçeklenmesine öncelikle tüm bileşenlerin bağlanacağı çift katlı plakanın oluşturulması ile başlanmıştır. Daha sonra plaka üzerine kontrol kartın, batarya ve ultrasonik sensör objeleri eklenmiştir (Şekil 4.82).



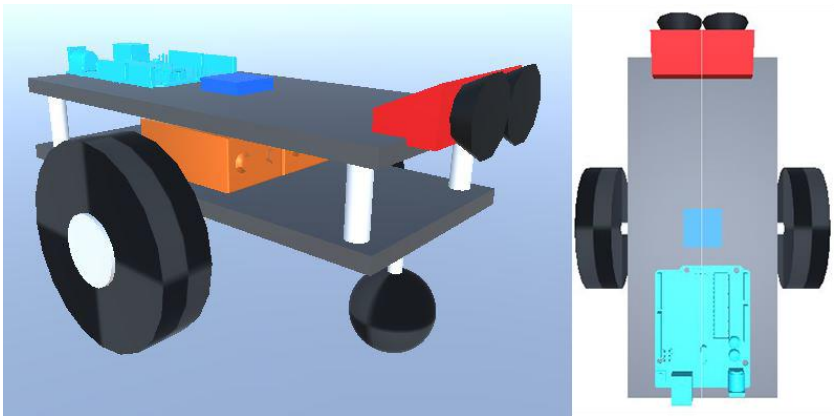
Şekil 4.82 : Solda oluşturulan objeler, sağda ise plakanın 3 boyutlu modeli.

Şu ana kadar yapılan işlemler plakanın sadece görsel olarak Unity ortamında gerçeklenmesini sağlamıştır. Araca fiziksel özellikler eklemek için daha önce anlatıldığı gibi Fizik Bileşenleri kullanılmıştır. Bu amaçla oluşturulan araç (car) ana objesine Rigidbody bileşeni eklenmiştir (Şekil 4.83).



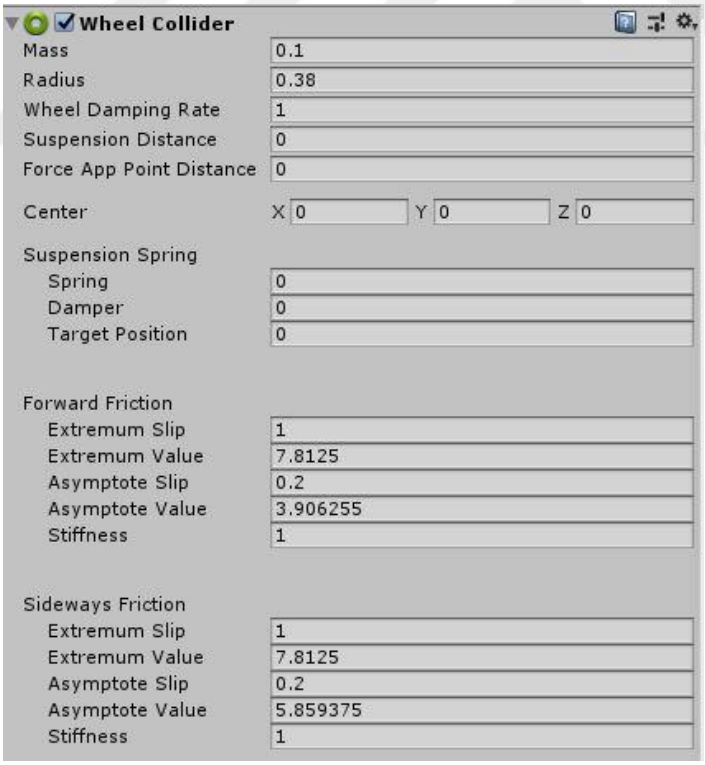
Şekil 4.83 : Araç (car) ana objesi üzerindeki Rigidbody bileşeni.

Daha sonra mobil araca tekerlek ekleme işlemine geçilmiştir. Öncelikle tekerleklerin 3 boyutlu modelleri Mesh Renderer ve Mesh Filter bileşenleri kullanılarak oluşturulmuştur (Şekil 4.84).



Şekil 4.84 : Mobil robotun 2 adet sarhoş ve 2 adet sabit teker eklenmiş modeli.

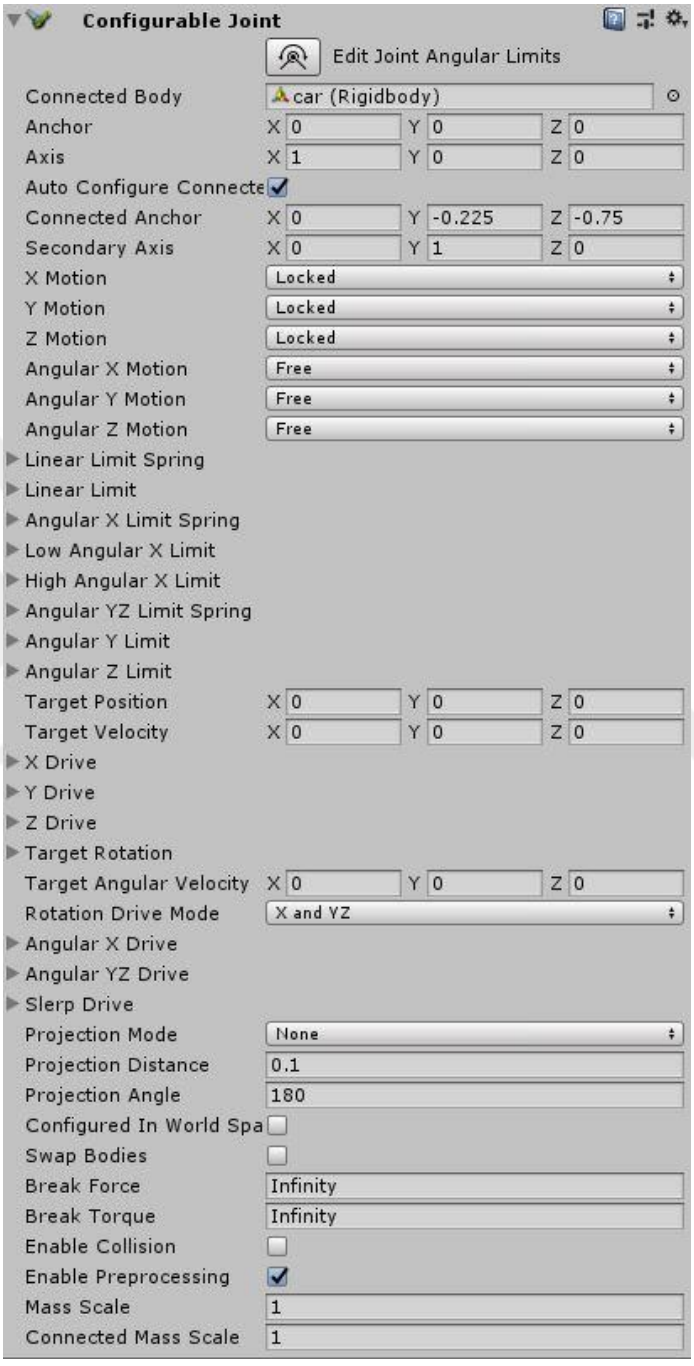
Sabit tekerleklerin fiziği Wheel Collider bileşeni kullanılarak sağlanmıştır. Bu amaçla 2 taraftaki sabit tekerleğe Wheel Collider eklenerek gerekli ayarlar bileşen üzerinde yapılmıştır (Şekil 4.85).



Şekil 4.85 : Sabit tekerleklere eklenen Wheel Collider bileşeni.

Sarhoş tekerleklerin fiziği ise Configurable Joint, Rigidbody ve Spehere Collider bileşenleri kullanarak sağlanmıştır (Şekil 4.86). Sarhoş tekerlekler tüm yönlere

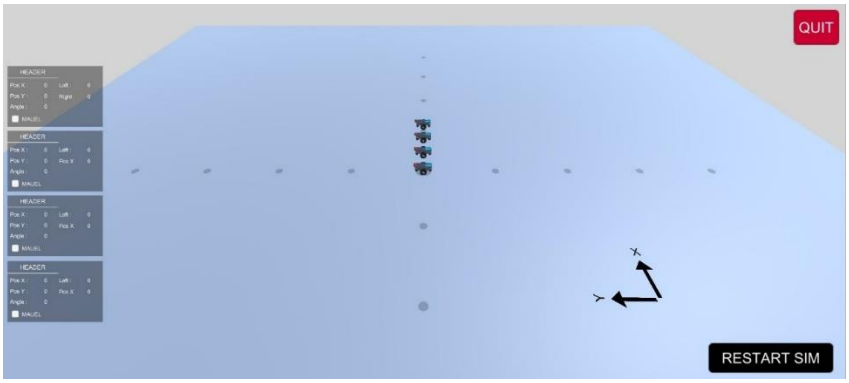
rotasyonel olarak hareket etmelerini sağlamak için gerekli ayarlar Configurable Joint bileşeni üzerinden yapılmıştır.



Şekil 4.86 : Sarhoş tekerleklere eklenen Configurable Joint bileşeni.

Şu ana kadar yapılan sistem tek bir mobil robotun fiziğidir. Oluşturulan mobil robot çoğaltılarak simülatör üzerinde aynı anda 8 taneye kadar mobil robotun kontrolü sağlanabilir. Simülasyon ekranına UI bileşenleri kullanılarak her robotun

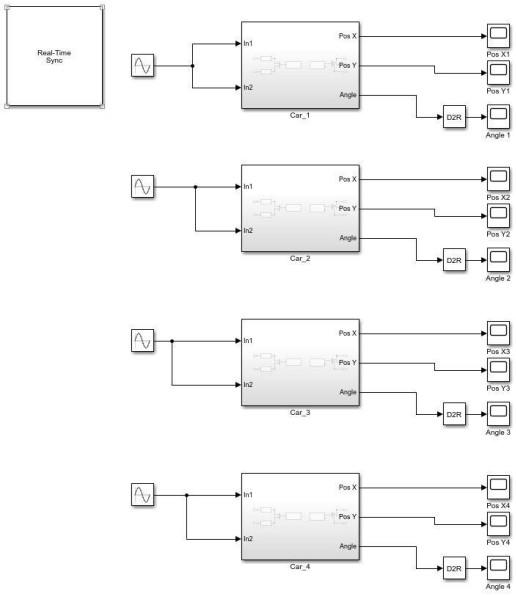
pozisyonunun ve hızının takip edilebileceği basit bir kullanıcı arayüzü eklenmiştir (Şekil 4.87). Ayrıca istenen robot Manuele alınarak klavyeden kullanıcı tarafından kontrol edilebilmektedir.



Şekil 4.87 : Geliştirilen simülatörün tüm bileşenleriyle birlikte görüntüsü.

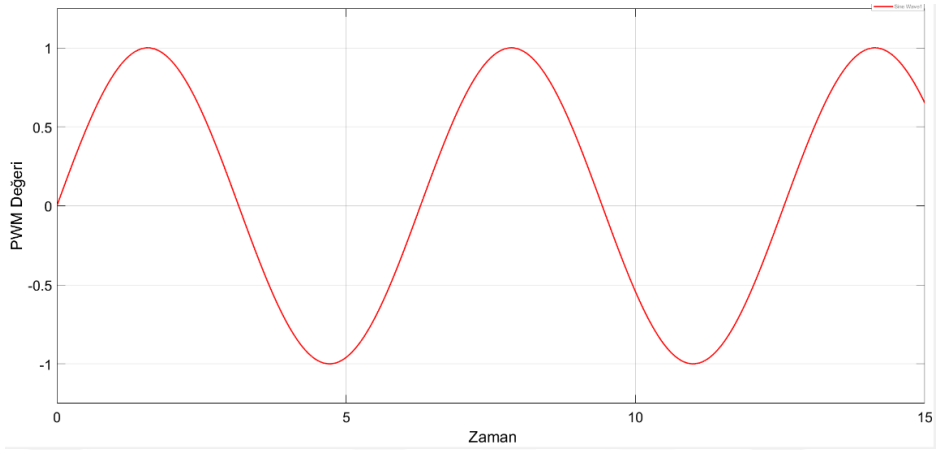
4.5.3 Simülatörün Simülink Ortamından Kontrolü

Unity modelinin Simulink ortamından kontrolü için daha önceki sistemlerde olduğu gibi UDP protokolü kullanılmıştır. Bu amaçla UDP Send bloğu sisteme giriş uygulamak için UDP Receive bloğu ise sistemin çıkışlarını almak için kullanılmıştır (Şekil 4.88). Simülatörde oluşturulan her robot için ayrı alıcı ve verici portu kullanılmıştır. Robotun numarasına göre verici portlar 25000'den alıcı portlar ise 26000'den başlamaktadır.

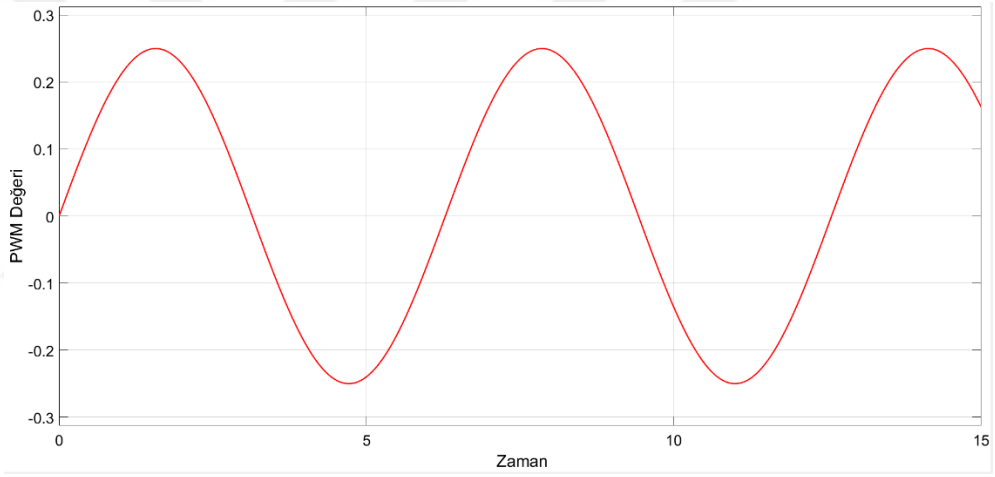


Şekil 4.88 : Simülatörün kontrolü için Simulink'te kurulan model.

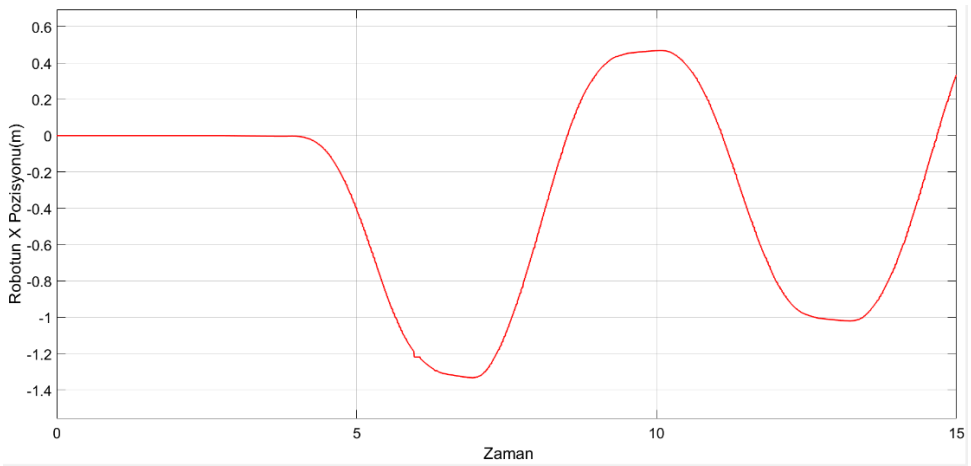
Sisteme giriş olarak mobil robotun sağ ve sol tekerleklerinin hızını belirleyen 0 ile 1 arasında double bir girilmektedir. Çıkış olarak ise mobil robotun pozisyonu ve duruş açısı alınmaktadır.



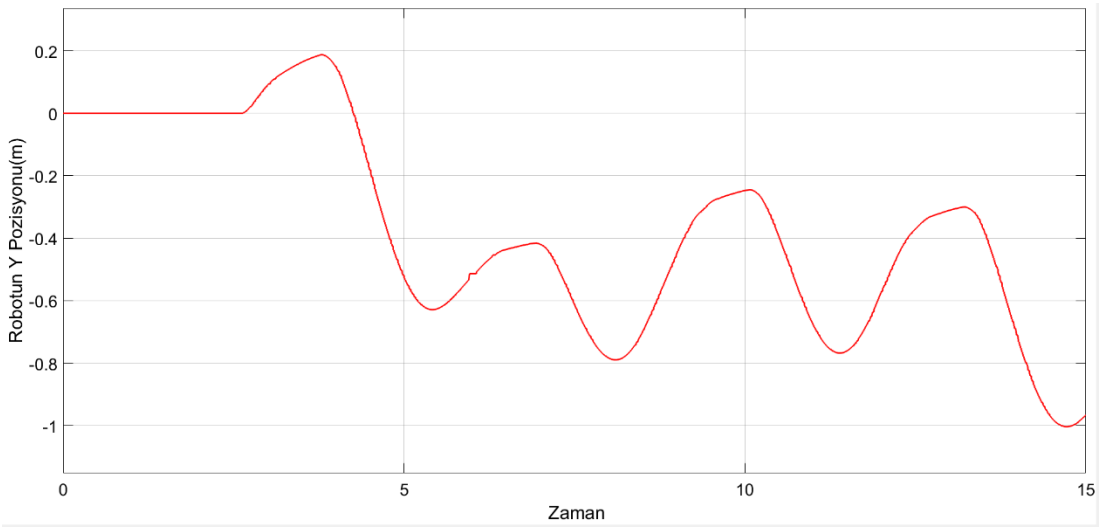
Şekil 4.89 : Sağ tekere uygulanan PWM işareti.



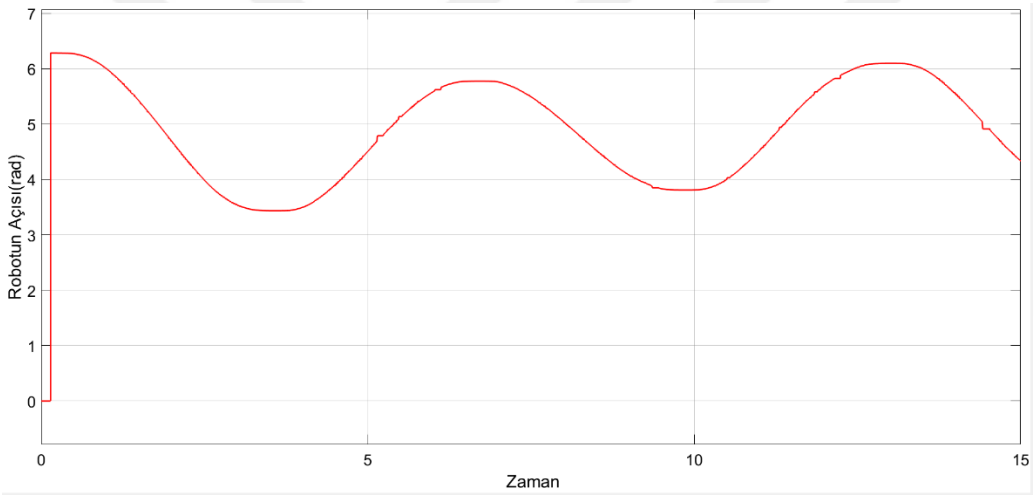
Şekil 4.90 : Sol tekere uygulanan PWM işareti.



Şekil 4.91 : Aracın X eksenindeki pozisyonu.



Şekil 4.92 : Aracın Y eksenindeki pozisyonu.



Şekil 4.93 : Robotun global eksene göre açısı.

Örnek olarak simülatörde 1 numaralı robotun sağ ve sol tekerleğine şekil 4.89 ve 4.90'da ki giriş işaretleri uygulanmıştır. Uygulanan işaretlere göre robotun konumu ve açısı şekil simülink ortamından gerçek zamanlı olarak okunmuş ve şekil 4.91, 4.92 ve 4.93 elde edilmiştir.

5. SONUÇLAR

Bu bitirme çalışmasında 4 farklı sistemin Unity oyun motoru kullanılarak simülatörleri geliştirilmiştir. Bu simülatörlerin başka araçlar ve programlar aracılığıyla kullanılabilmeleri için UDP protokolleri ile haberleşme arayüz yazılımı yazılmış ve simülatörlerin Simülink platformundan kontrol edilebilmesi sağlanmıştır. Simülatörü geliştirilen her sistemin doğrusal olmayan modeli elde edilmiş ve Simulink programında bu modeller simülasyonları oluşturulmuştur. Daha sonra doğrusal olmayan modellerin simülasyonu ile geliştirilen simülatörlerin çeşitli giriş işaretleri için olan çıkış işaretleri karşılaştırılmıştır. Bu karşılaştırmalar sonucunda simülatörlerin gerçek dünyada ki fizik kanunlarına ve gerçek sistemlerine uygun hareket edip etmediği irdelenmiştir. Simülatörlerin ve doğrusal olmayan model simülasyonlarının benzer çıkışlar verdiği görülmüştür. Sonuç olarak 4 farklı sistem için bir deney seti ortamı oluşturulmuştur. Bu aşamadan sonra öncelikli olarak yapılması gereken şey simülatörlerin daha farklı sistem değerleri ile test edilmesi olacaktır. Yeterli veriler elde edildikten sonra son olarak simülatörler sistemlerin gerçek modelleriyle test edilmelidirler. Bu amaçla daha önceki başlıklarda bahsedilen firmaların deney setleri kullanılabilir. Gelecek çalışmalarda ise çok daha kompleks olan otomobil, drone, tren, uçak vb. gibi sistemlerin simülatörleri yapılabilir. Bu sayede gerçek dünyada deney setleri pahalı ve tehlikeli olan bu sistemlerin akademik çalışmalarda kullanılması sağlanabilir.

KAYNAKLAR

[1] Critchley, L. (2017). *Computer Simulations in Engineering*, Retrieved from http://www.statisticsviews.com/details/feature/10482673/ComputerSimulations-in-Engineering. html

[2] Gruszkiewicz, S. P. & Baumgart, E. (2012). Three-Dimensional Engine Simulators with Unity3D Game Software, *The 13th Annual General Assembly of the JAMU*, Newfoundland, Labrador, Canada : October 15-17.

[3] Yang, K & Jie, J. (2011). The Designing of Training Simulation System Based on Unity3D (Ed.), *Fourth International Conference on Intelligent Computation Technology and Automation*, (pp.976-978). Shenzhen, Guangdong, China : March 28-29.

[4] Biurrun, Carlos, Serrano, Luis & Olaverri Monreal, C. (2017). Microscopic Driver-Centric Simulator: Linking Unity3D and SUMO, *Advances in Intelligent Systems and Computing*, Vol. 1, 851-860 .

[5] Nokhbeh, M. & Khashabi, D. (2011). Modelling and Control of Ball-Plate System**,**Retrieved:http://www.cis.upenn.edu/~danielkh/files/2011_LinearControl/16.pdf

[6] Andrews, G., Colasuonno, C & Herrmann, A. (2004). Ball on Plate Balancing System, USA : Rensselaer Polytechnic Institute-Control Systems Design, Retrieved : http://cats-fs. rpi.edu/~wenj/ ECSE4962S04/final/ final/team2finalreport. pdf

[7] Awtar, S., Bernard, C, Boklund, N, Master, A, Ueda, D, Craig, Kevin. (2002). Mechatronic design of a ball-on-plate balancing system, *Mechatronics*, Vol. 12, 217-228.

[8] Börü, E. (2008). *Modelling and Controlling of Ball and Plate System(Master Thesis)*,Retrieved from : https://polen.itu.edu.tr

[9] Kumar, P., Chakraborty, K., Mukherjee, R. & Mukherjee, S. (2019). Modelling and Controller Design of Inverted Pendulum, *International Journal of Advanced Research in Computer Engineering & Technology*, Vol. 2, 2278–1323.

[10] Ismail, R. M. T. R., Ahmad, M. A., Ramli, M. S. & Rashidi, F. R. M.. (2009). Nonlinear Dynamic Modelling and Analysis of a 3-D Overhead Gantry Crane System with Payload Variation, *Third UK Sim European Symposium on Computer Modeling and Simulation*, (pp.350-354). Athens, Greece : November 25-27.

[11] Chen, H., Gao, B. & Zhang, X.. (2011). Dynamical modelling and nonlinear control of a 3D crane, *International Conference on Control and Automation*, (pp.1085-1090). Singapore : July 30-31.

[12] Han, S., Hasan, S., Al-Hussein, M., Gökçe, K. U & Bouferguene, A. (2012). Simulation of Mobile Crane Operations in 3D Space, *Proceedings of the 2012 Winter Simulation Conference*, (pp.1-12). Berlin, Germany : December 9-12.

[13] Renuka, V. S. & Abraham, T. M. (2000). Precise Modelling of a Gantry Crane System Including Friction, 3D Angular Swing and Hoisting Cable Flexibility, *International Journal on Theoretical and Applied Research in Mechanical Engineering*, Vol. 14, 2319–3182.

ÖZGEÇMİŞ

Ad-Soyad **:** Cihad DOĞAN

Doğum Tarihi ve Yeri **:** İstanbul, 22 Temmuz 1993

E-posta **:** cihaddogan1@gmail.com

ÖĞRENİM DURUMU:

- **Lisans** **:** 2011-2016, İstanbul Teknik Üniversitesi, Elektrik-Elektronik Fakültesi, Kontrol ve Otomasyon müh.

MESLEKİ DENEYİM :

06. 06. 2014 - 14. 08. 2014 :

Desi Alarm ve Güvenlik, İstanbul, Stajer

07. 06. 2015 - 07. 08. 2015 :

Baykar Makina, İstanbul, Stajer

07. 08. 2015 – Halen :

Desi Alarm ve Güvenlik, İstanbul, Yazılım Müh.