

Railpower

Table of Contents

Introduction

1. Icerik

- 1.1 Numerik Integrator
- 1.2 Araclar
- 1.3 Baglantilar
- 1.4 Simulasyon Birimi
- 1.5 Sonuclar ve Test Paketleri

2. Runtime Birimi (API)

- 2.1 API Fonksiyonları
- 2.2 Sınıflar

3. Yapılacaklar

4. Kaynakça

Railpower

DEVELOPMENT OF DYNAMIC ENGINE AND GRAPHICAL USER INTERFACE SOFTWARE FOR RAILWAY SYSTEMS

VERSİYON 1.0.1

Railpower projesi birincil kullanıcısı [TCDD](#) olacağı gözetilerek yapılmıştır.

Railpower kelimesi; projenin günümüz raylı sistemlerinin kontrol işlemlerine ek bir güç sağlayacağı düşüncesi ile isim olarak yer etmiştir.

Projenin danışman öğretim üyesi: **Yrd. Doç. Dr. Metin Özkan** olup Eskişehir Osmangazi Üniversitesi öğrencilerinden:

- Mahmut Bulut
- Abdurrahman Kolsuz

tarafından bitirme tezi olarak oluşturulmuştur.

Terimler:

Terim	Açıklama
coupling	Raylı sistem birimleri arasında bulunan bağlantı
car	Raylı sistem birimlerinden kuvvet üretmeyen birimler
freight car	Raylı sistem birimlerinden yük taşıyan birimler diğer adı ile <i>goods car</i>
passenger car	Raylı sistem birimlerinden insan taşıyan birimler

Birimler

Birimler Açıklamalar

Nm⁻¹ Bağlantı Yay Sabiti (Coupler Spring Constant)

Nsm⁻¹ Bağlantı Damping Sabiti (Coupler Damping Constant)

Icerik

İçerik

Bu bölümde sistemin bilimsel içeriği oluşturulacaktır. Sistemin çalışma prensiplerine deðinilecek ve çıktıları gözlenecektir.

Sistem birçok alt üiteden oluşur. Bu alt üniteler bu bölümde verilecektir, alt üniteler sistemin içinde oluşturulma sırasına göre aşağıdaki gibi sıralanabilir:

- Numerik Integratör
- Araçlar
- Baðlantılar
- Ana simülasyon

ve bunların dışında sisteme yapılan çağrıların bu katmanlara aktarıldığı

- Runtime Birimi

yer almaktadır.

Gereçler

Proje genel olarak C++'ta kodlanmıştır. Çaðrılar C çağrı olup `__stdcall` konvansiyonunu kullanır. `C+11` standardını kullanır. `Boost` kütüphanesinin içine sonradan dahil edilen `ODEINT` integratörü sistemin numerik integratörü olarak görev yapmaktadır.

Numerik Integratör

`Boost` kütüphanesine dahil edilen ordinary differential equation solver olarak tabir edilen `odeint` sistemin numerik integratördür.

Numerik integratör sistemde gelecek versiyonlarda özellikler eklenecek şekilde kodlanmıştır. Şu anda kullanılan integratör **adaptif integratördür**.

```
integrate_adaptive( stepper_controlled , *this , x , 0.0 , step_time , getDT() , observer );
```

Genel adaptif integratör dökümantasyonu için: [Boost / Integrate functions](#) kısmına bakılabilir. Adaptif geliştirme bu konunun kapsamına girmektedir.¹

Stepper

Baþlı başına integratörler hiçbir anlam ifade etmemekte olup integrasyonların alındığı aralıkları düzenleyen bu aralıklarda her hesaplamanın sonucunu bize geri döndüren bir yapıya ihtiyaç vardır, bu yapılar **stepper** olarak adlandırılırlar. Bu sistemde **controlled stepper** kullanılmıştır. Bu stepper birim zamanda hata aralığının içinde kalan her değer için integrasyon aralığını düşürür ve daha doğru sonuçlar vermeye çalışır. Buna rağmen adım sayısı verilen aralıkta değişmez, hata artarsa ve adımlar iterasyona yetmeyecek duruma gelmeye başlarsa adım genişliğini artırrı.

```
auto stepper_controlled = make_controlled( 1E-12 , 1E-12 , runge_kutta_dopri5< state_type >() );
```

Railpower projesinin sürümlerinde ilerleme kaydedildikçe diğer stepperların kullanımı da dahil edilecektir.

Stepper **Dormand-Prince** metodunu kullanmaktadır 5. dereceden çözüm üretmektedir.

Şu anda yukarıda verilen kod örneðinde görüldüğü üzere kontrollü stepper kullanılmıştır. Stepper'in ilk argümanı

absolute error(mutlak hata), ikinci argümanı **relative error(bağıl hata)** olarak belirlenmiştir ve değerleri **1E-12** dir.

Araçlar

Railpower şimdilik aşağıdaki tipleri desteklemektedir.

- Locomotive
- Car
- Freight Car
- Passenger Car

Her bir aracın kendine özgü özellikleri(yük, yolcu ve diğer spesifik özellikler) ve bu özelliklerinin sisteme etkisi ilerleyen versiyonlarında olacaktır.

Şu anda her sistemin kendine ait basit seviyede:

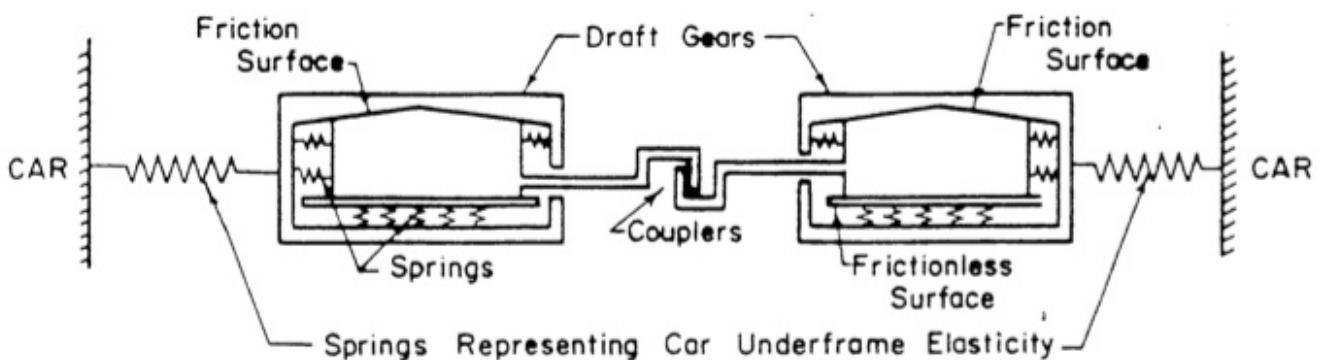
- kütle
- ivme
- hız
- pozisyon

değerleri bulunmaktadır.

Bir sonraki bölümde belirtilecek olan bağlantılar ile araç modellemeleri geçmiş araştırma konvansiyonları üzerine yapılmıştır. Sinamik sistemin modellenmesi ve çalışma biçimi zor bir konu olduğundan bu konu hakkındaki tüm implementasyon önceki araştırmalardan alınmıştır.⁵

Bağlantılar

Bağlantılar diğer adıyla **coupling** yapıları iki aracı birbirine bağlanmaya ve/veya araçlar arasında güç aktarmaya yarayan sistemlerdir. Bir coupling'in genel yapısı aşağıdaki şekildeki şeyledir:



Bağlantı sabitleri

- Damping sabiti (bağlantı yayının/yaylarının damping sabiti)
- Spring sabiti (bağnatı yayının sabiti)

Kuvvet

Bağlantının aktardığı kuvveti bulmak için çeşitli denklemler bulunmaktadır. Bu projede aşağıda belirtilen bağlantı

kuvveti hesaplama denklemi kullanılmıştır.³ $m_1\ddot{a}_1 + c_1(v_1 - v_2) + k_1(x_1 - x_2) = F_{t/db} - F_{r1} - F_{g1}$

m ve **a** sırasıyla kütle ve ivmeyi, **c** damping sabiti, **k** spring sabitidir. Denklemin sağ el tarafı net kuvvet olarak coupling kuvvetini bize vermektedir.

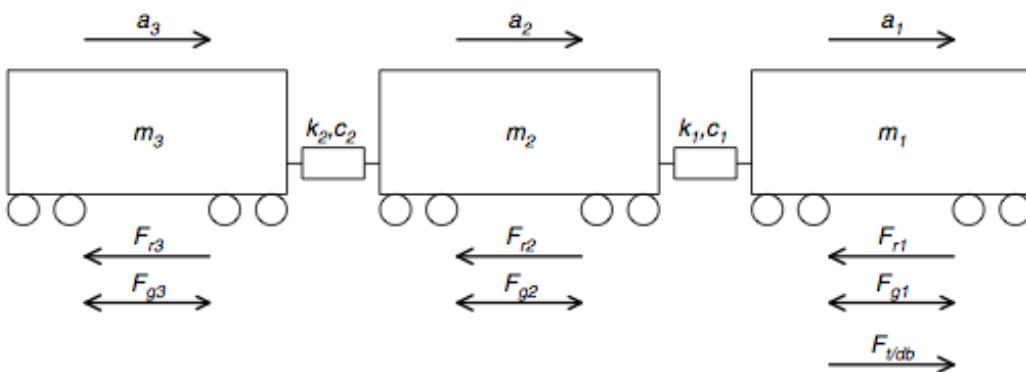


FIGURE 9.1 Three mass train model, where: a is vehicle acceleration, m/sec^2 ; c is damping constant, Nsec/m ; k is spring constant, N/m ; m is vehicle mass, kg ; v is vehicle velocity, m/sec ; x is vehicle displacement, m ; F_g is gravity force components due to track grade, N ; F_r is sum of retardation forces, N ; and $F_{t/db}$ is traction and dynamic brake forces from a locomotive unit, N .

Bağlantı kuvvetinin integrasyon sürecinde simülasyonun bir sonraki adımı kuvvetin hesaplama adımından her zaman bir Δt hesabı kadar önde olması gerekmektedir.⁴

Simülasyon Birimi

Simülasyon birimi `test.txt` dosyasına her iterasyon sonrası:

- İterasyon aralığının yazılıma ait iç iterasyon zamanını
- i. aracın ivmesini
- i. aracın hızını
- i. aracın pozisyonunu

yazmaktadır. İlerleyen versiyonlarda dosyanın adının da kullanıcıya bağlı olması sağlanacaktır.

Bu iterasyon sonrası veriler matlab'da görselleştirilebilir ve doğruluğu kontrol edilebilir. **Matlabda** sadece lokomotifin olduğu bir test görsellemesi aşağıdaki şekilde oluşturulabilir:

```
A = importdata('test.txt');
disp('DATA IMPORTED');

figure('name', 'ACCELERATION');
slice_acce = [A(:,1), A(:,2)];
plot(slice_acce(:,1), slice_acce(:,2), 'r');
grid on;
disp('ACCELERATION PLOT');

figure('name', 'VELOCITY');
slice_velo = [A(:,1), A(:,3)];
plot(slice_velo(:,1), slice_velo(:,2), 'm');
grid on;
disp('VELOCITY PLOT');

figure('name', 'POSITION');
slice_pos = [A(:,1), A(:,4)];
```

```

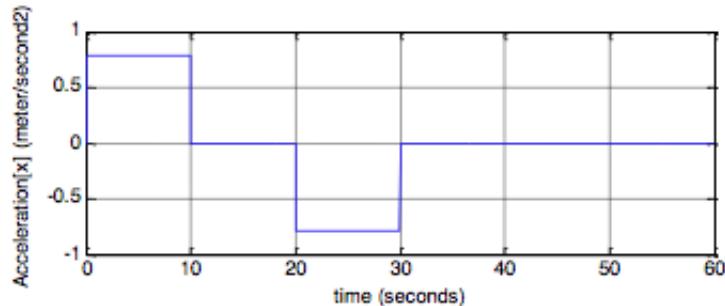
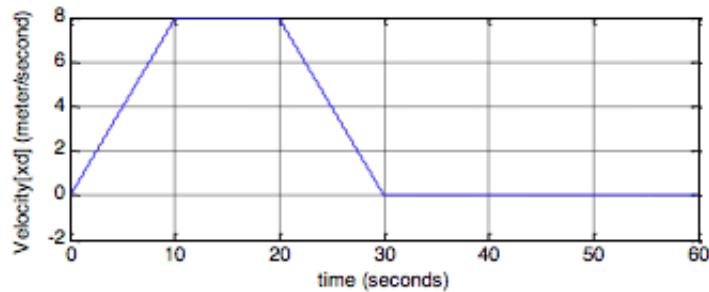
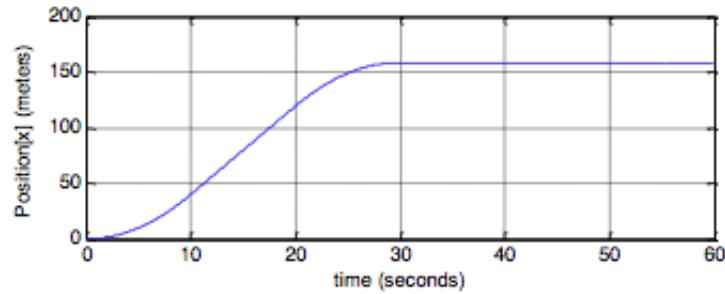
plot(slice_pos(:,1), slice_pos(:,2), 'b');
grid on;
disp('POSITION PLOT');

```

Sonuçlar ve Test Paketleri

Bölüm 1.4 te belirtildiği üzere script yeniden kullanılarak (her durum için farklı veya araç sayısı arttıkça durumları yönetip görselleştirmeyi yapan bir script) 3 test paketi aşağıdaki şekilde başarıyla ve hata oranı belirlenen şekilde geçmiştir.⁶

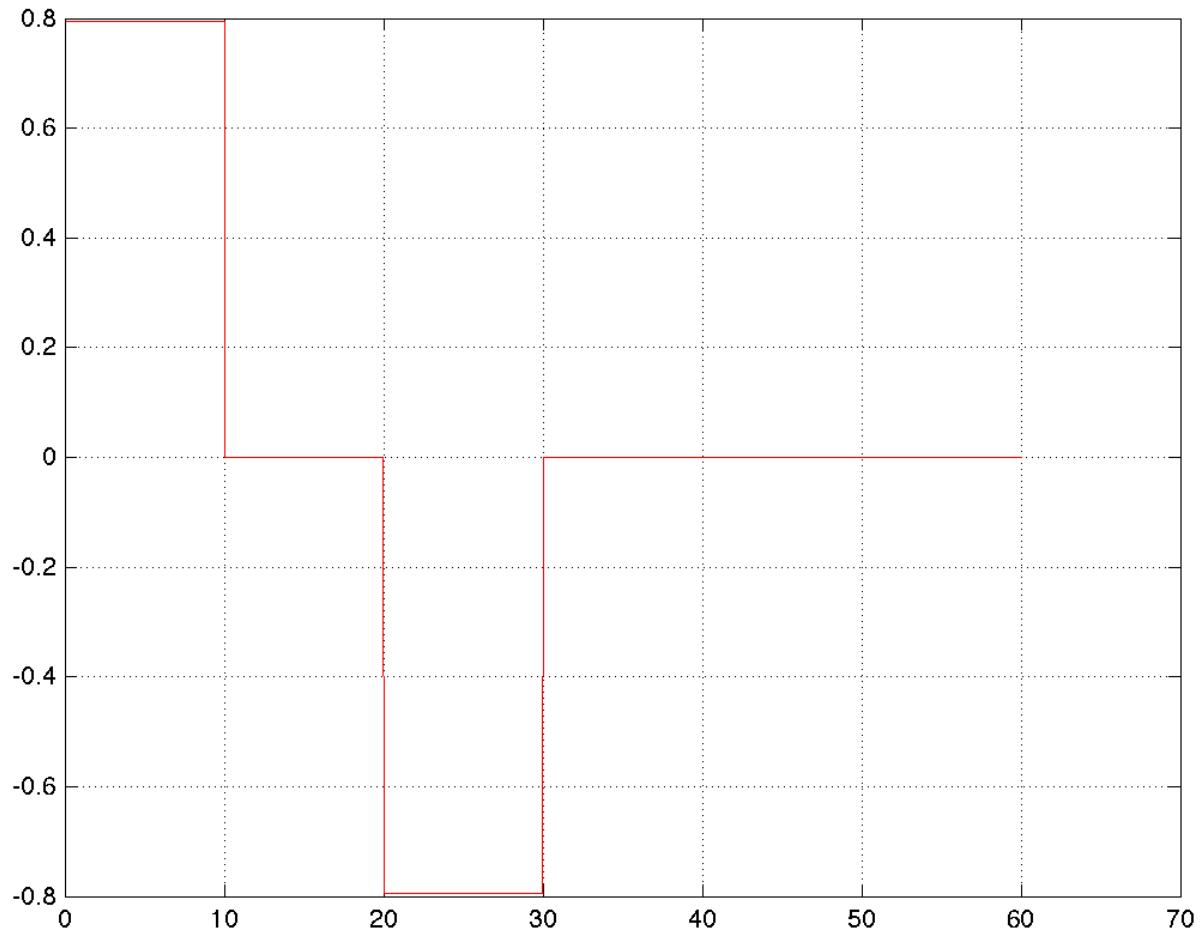
Test 1 Pack



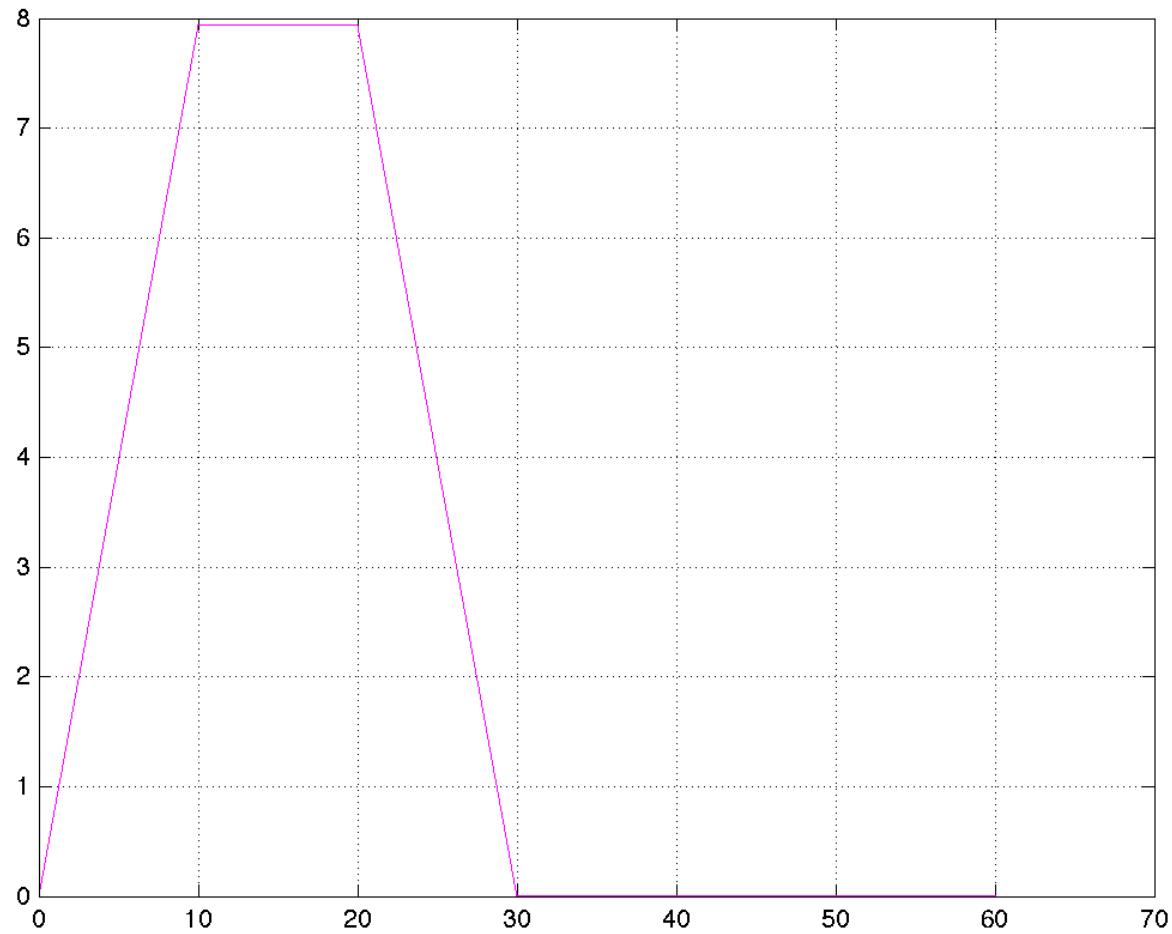
Test 1 için beklenen değerler:

Test 1'in Railpower değerleri:

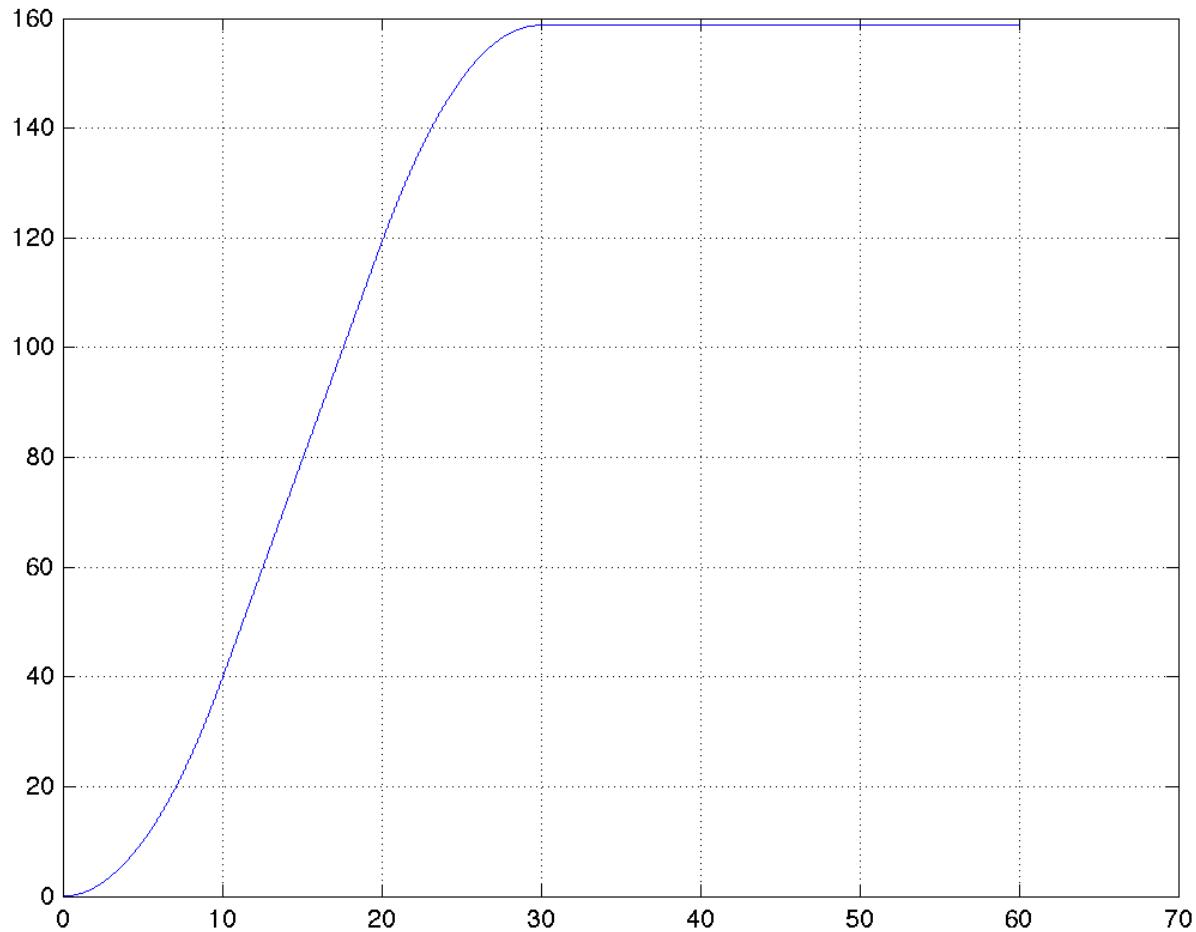
İvme



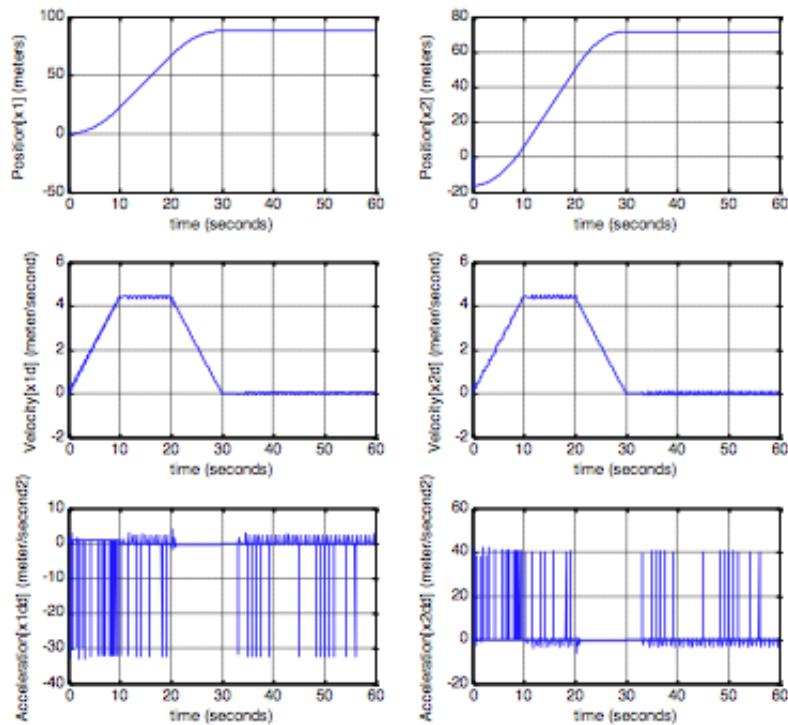
Hz



Pozisyon



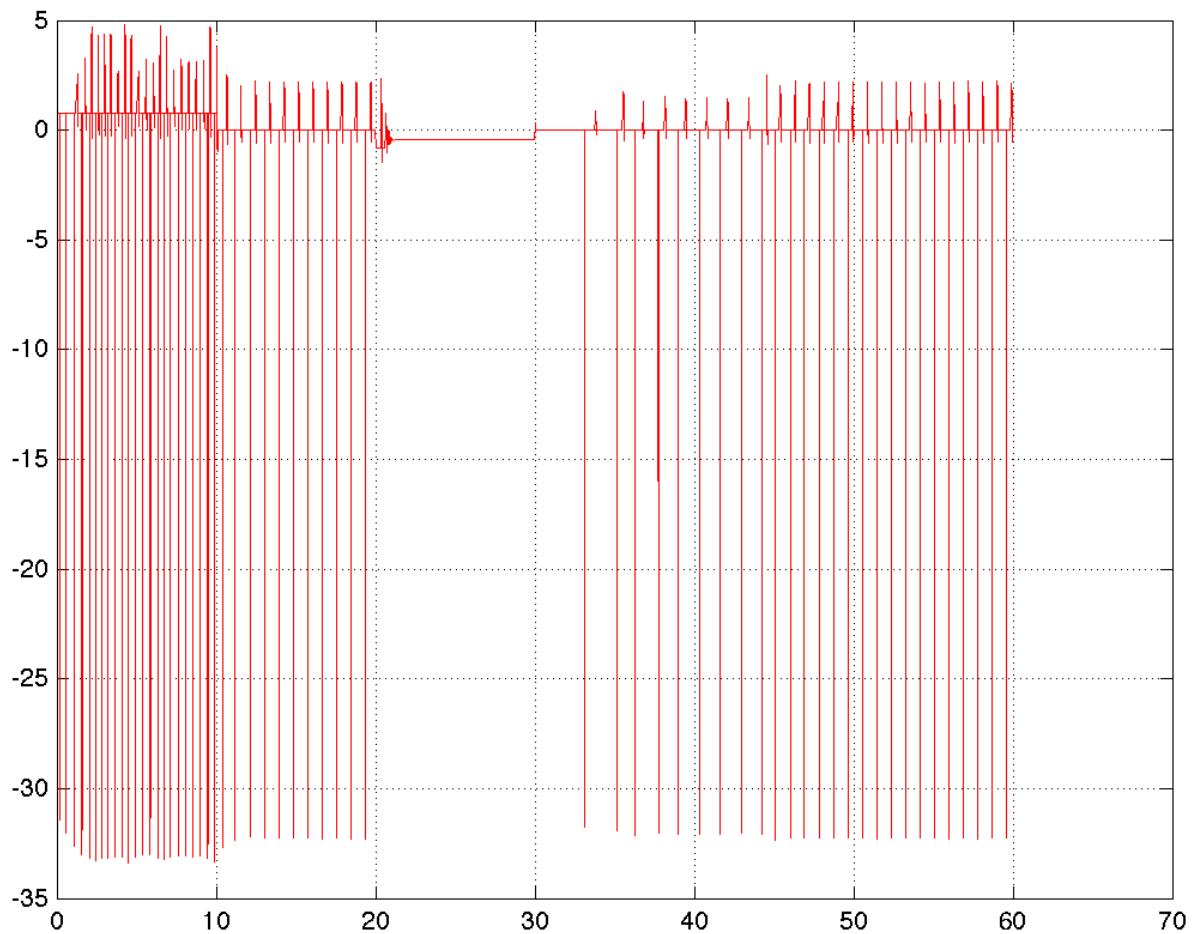
Test 2 Pack



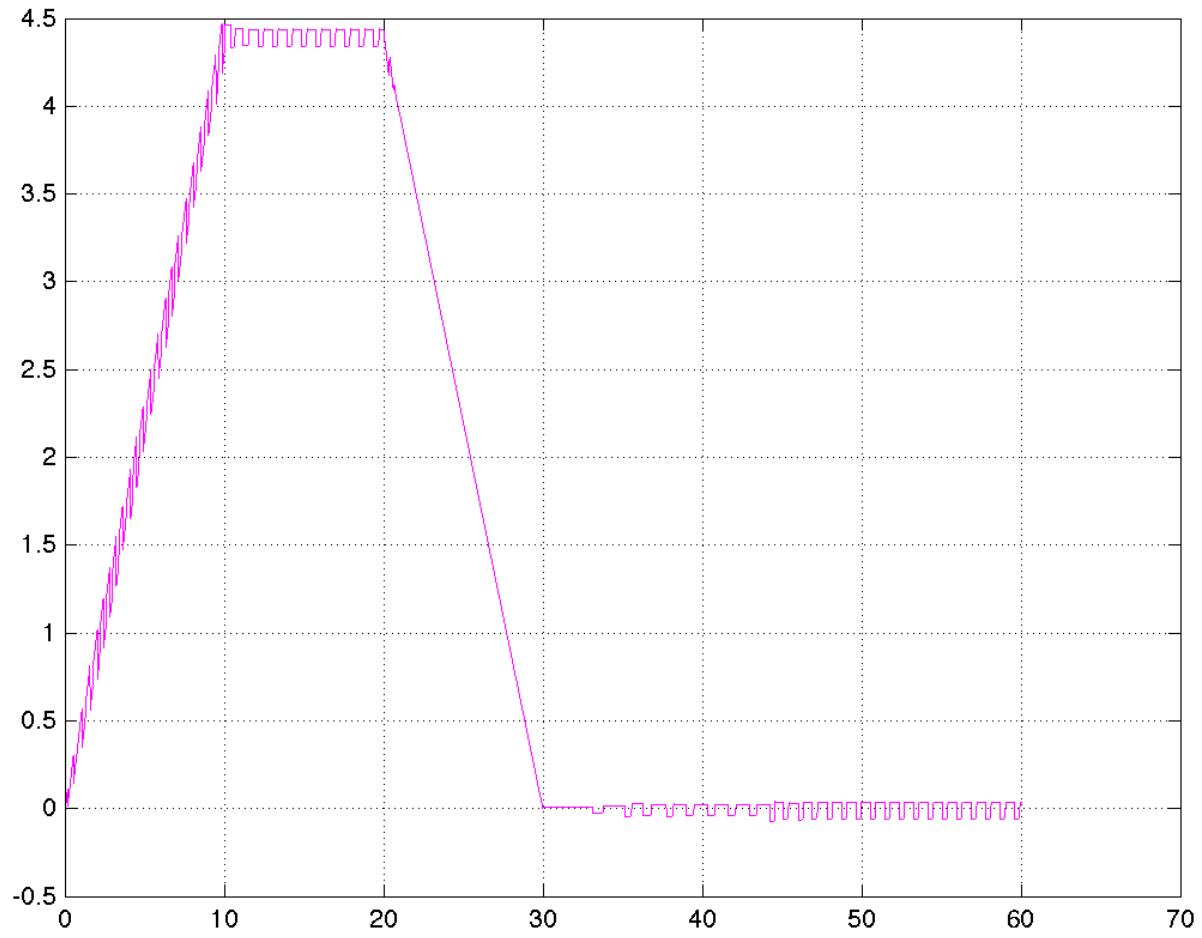
Test 2 için beklenen değerler:

Test 2'nin Railpower değerleri:

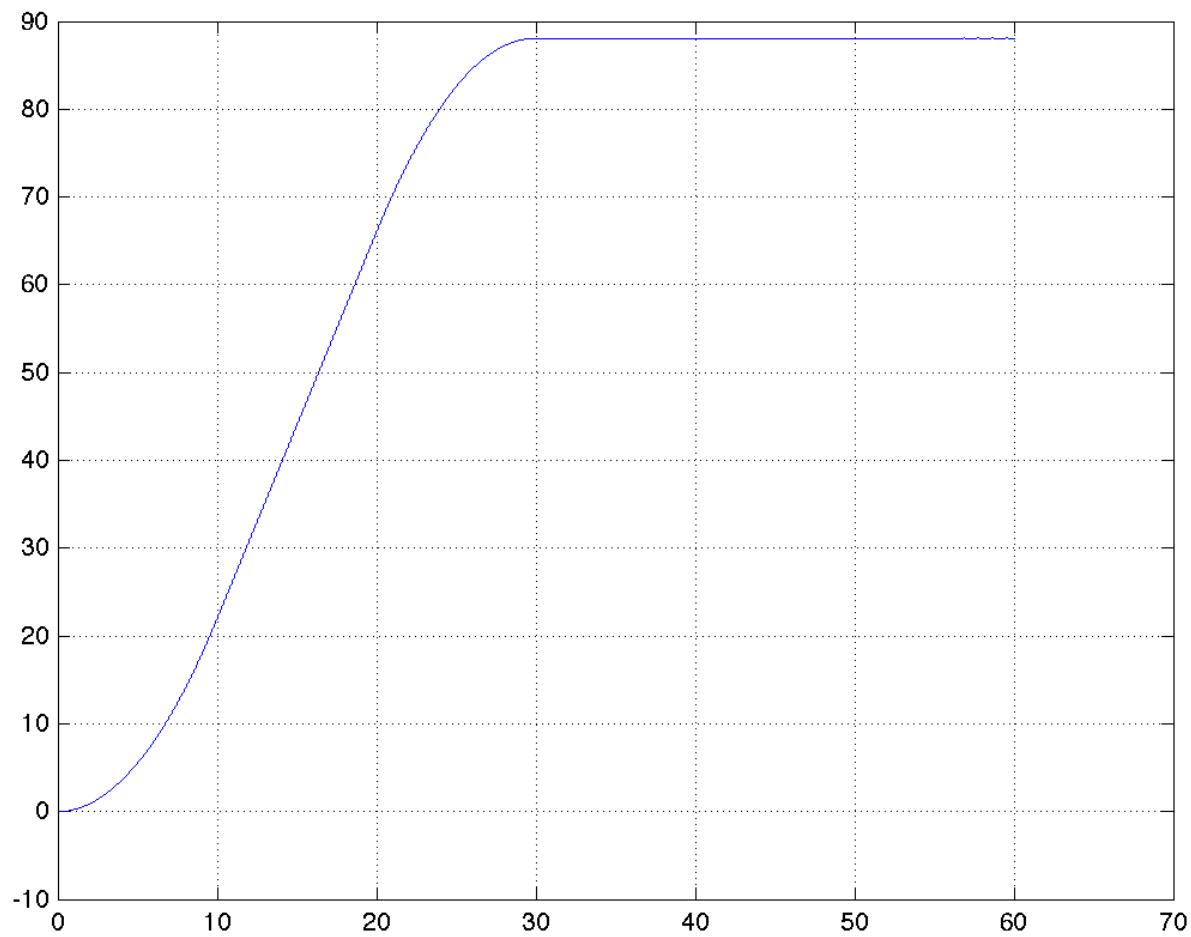
İvme 1. araç



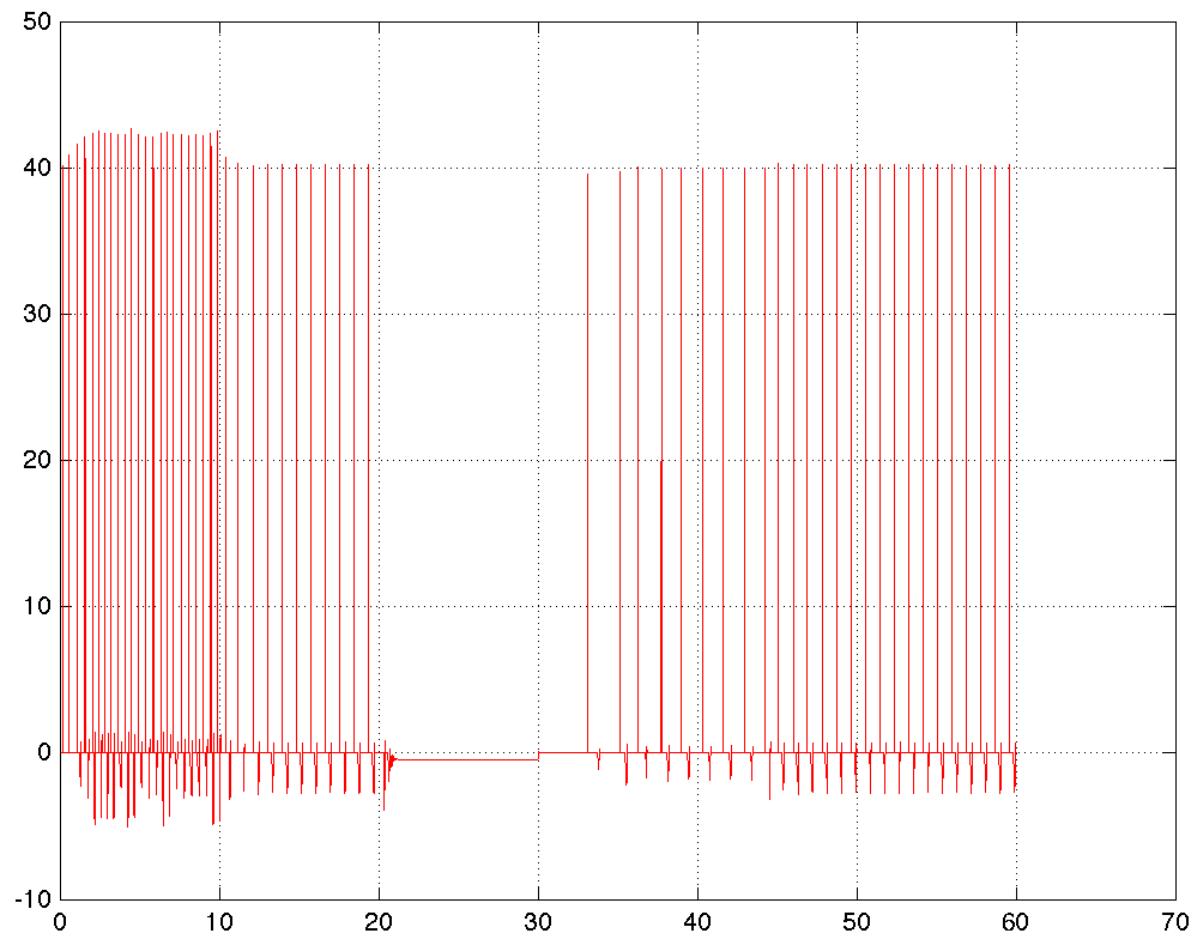
Hız 1. araç



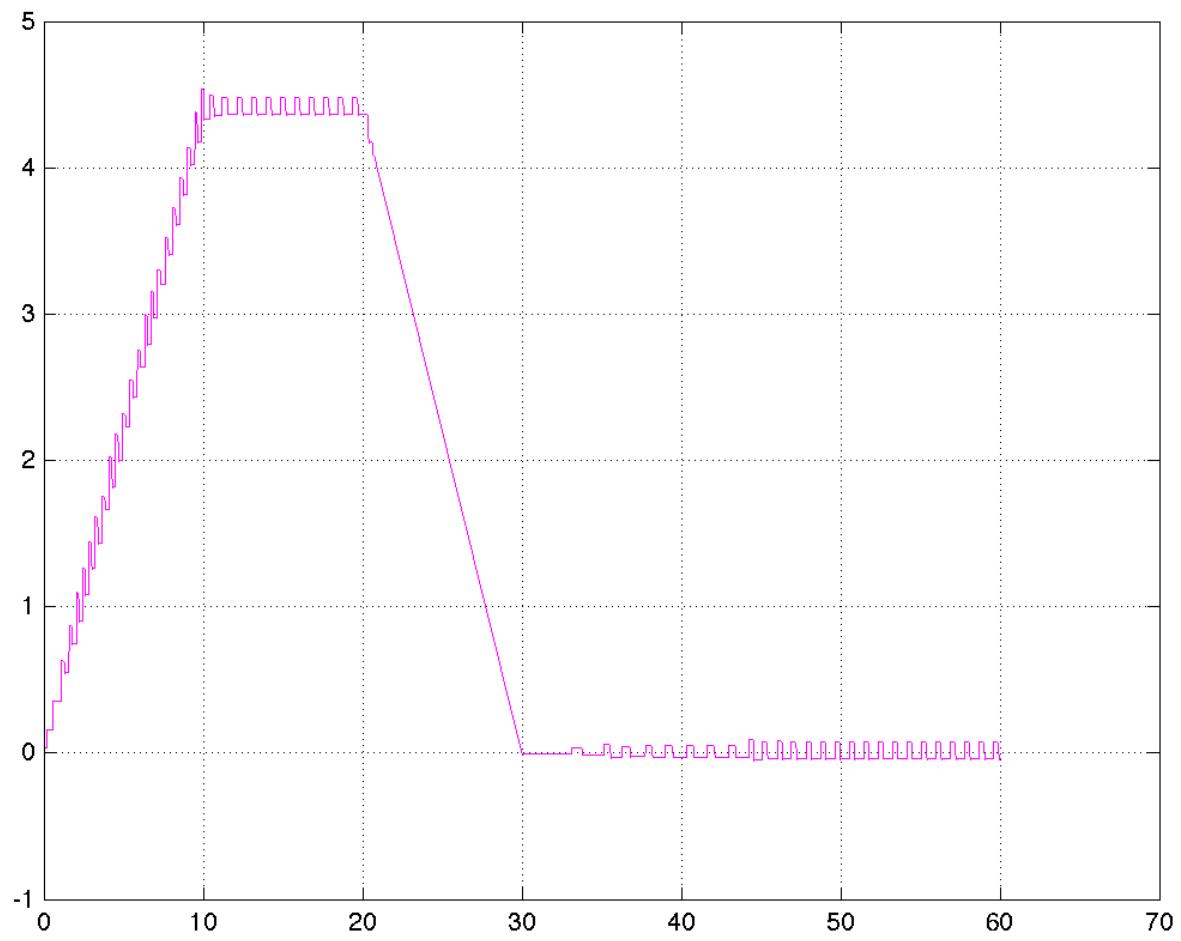
Pozisyon 1.araç



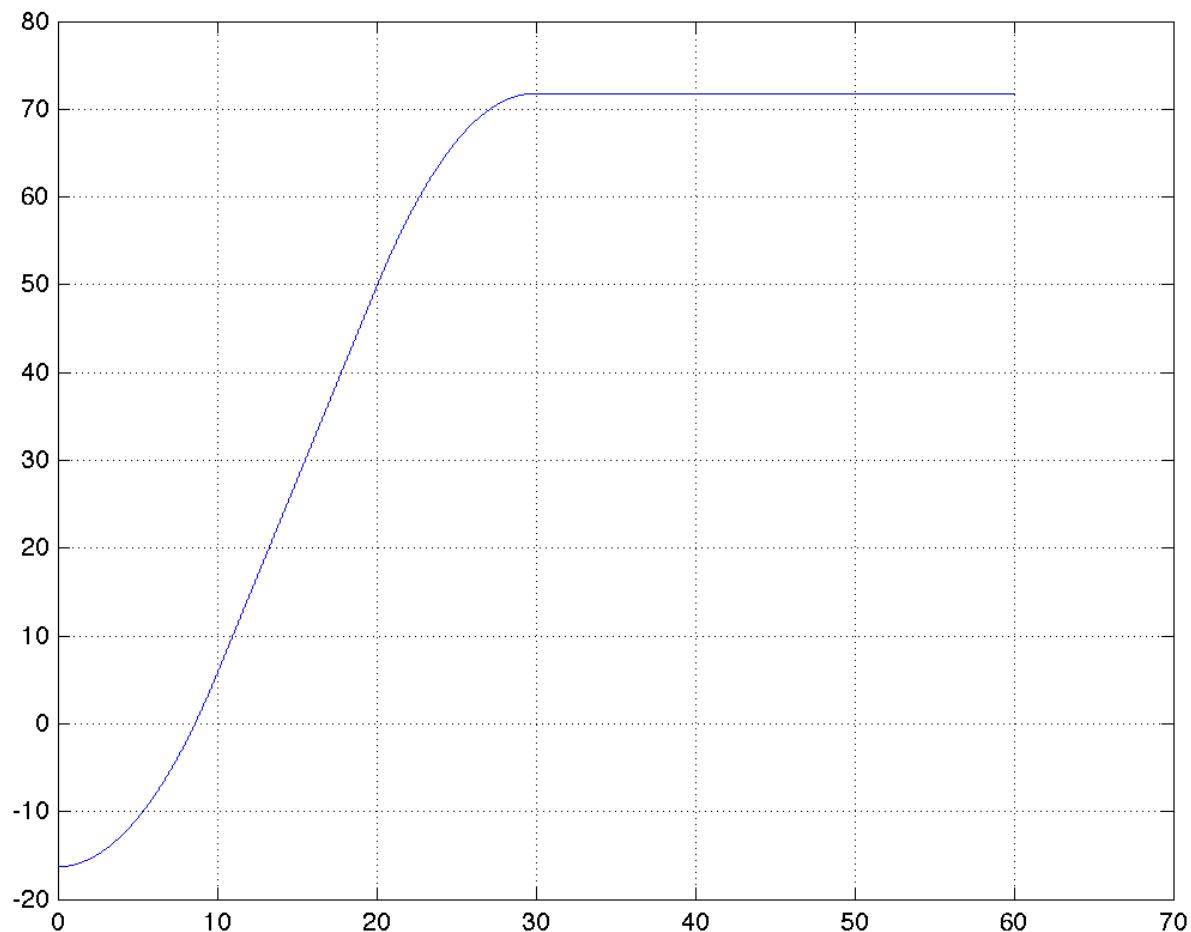
İvme 2. araç



Hız 2. araç

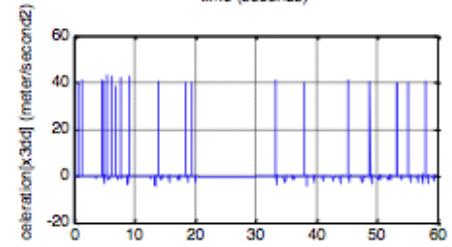
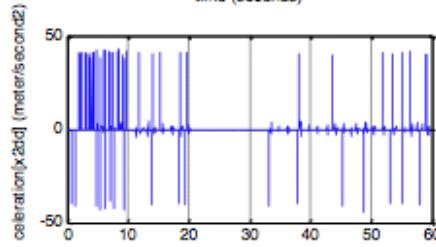
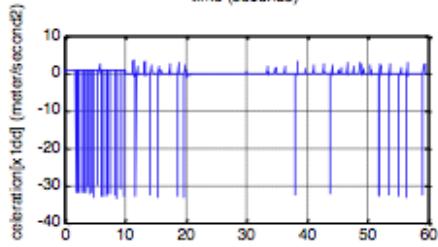
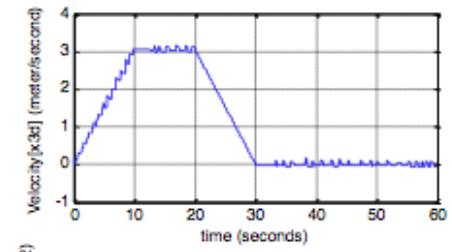
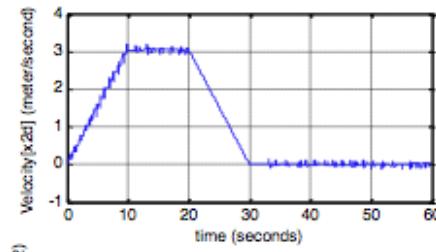
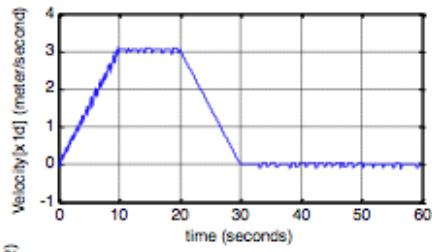
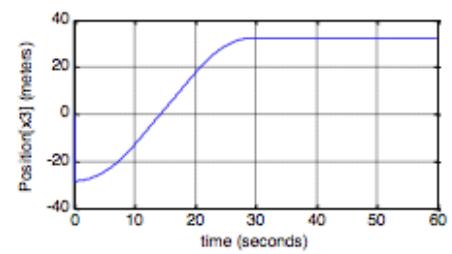
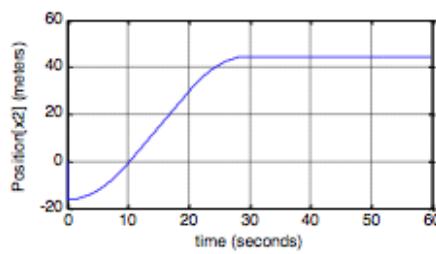
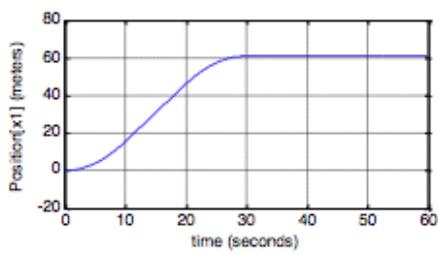


Pozisyon 2.araç



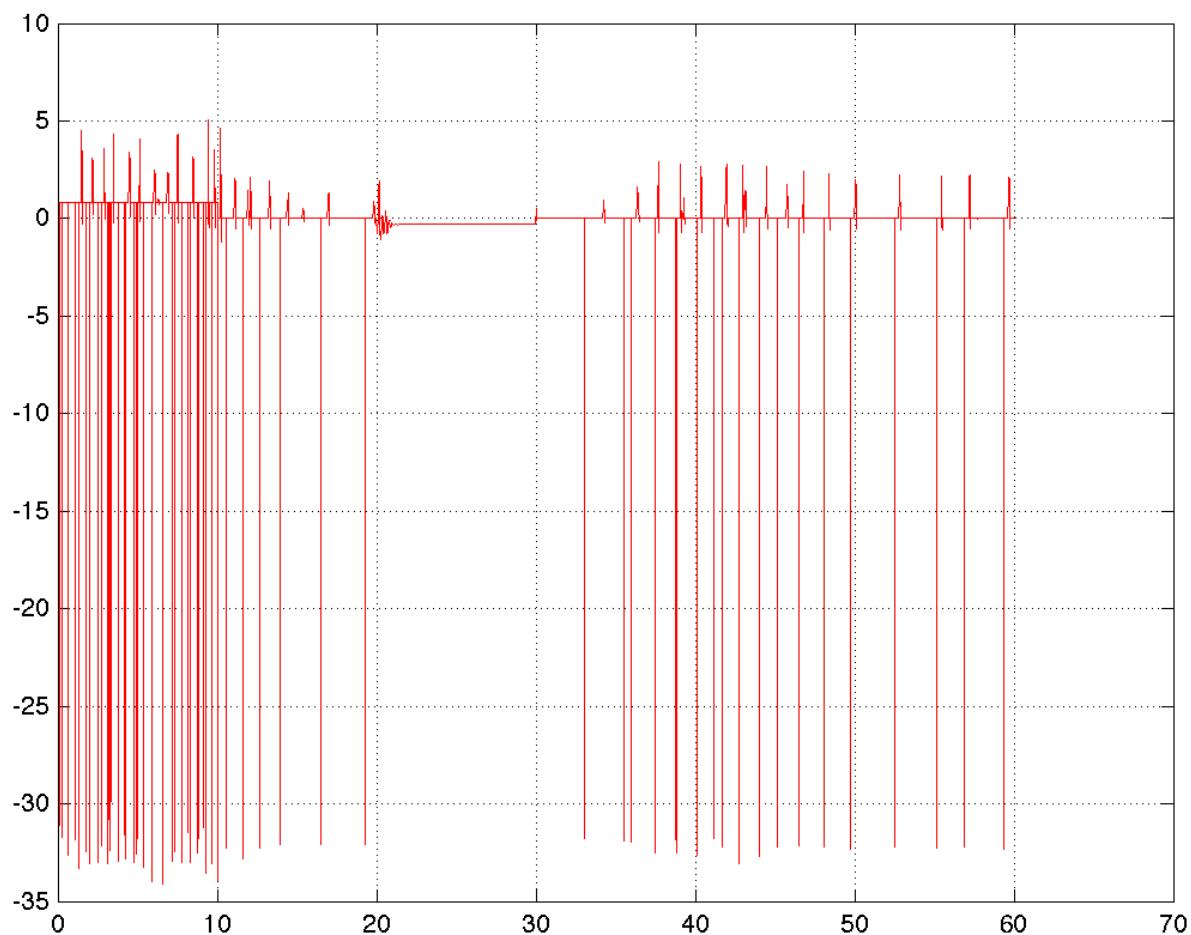
Test 3 Pack

Test 3 için beklenen değerler:

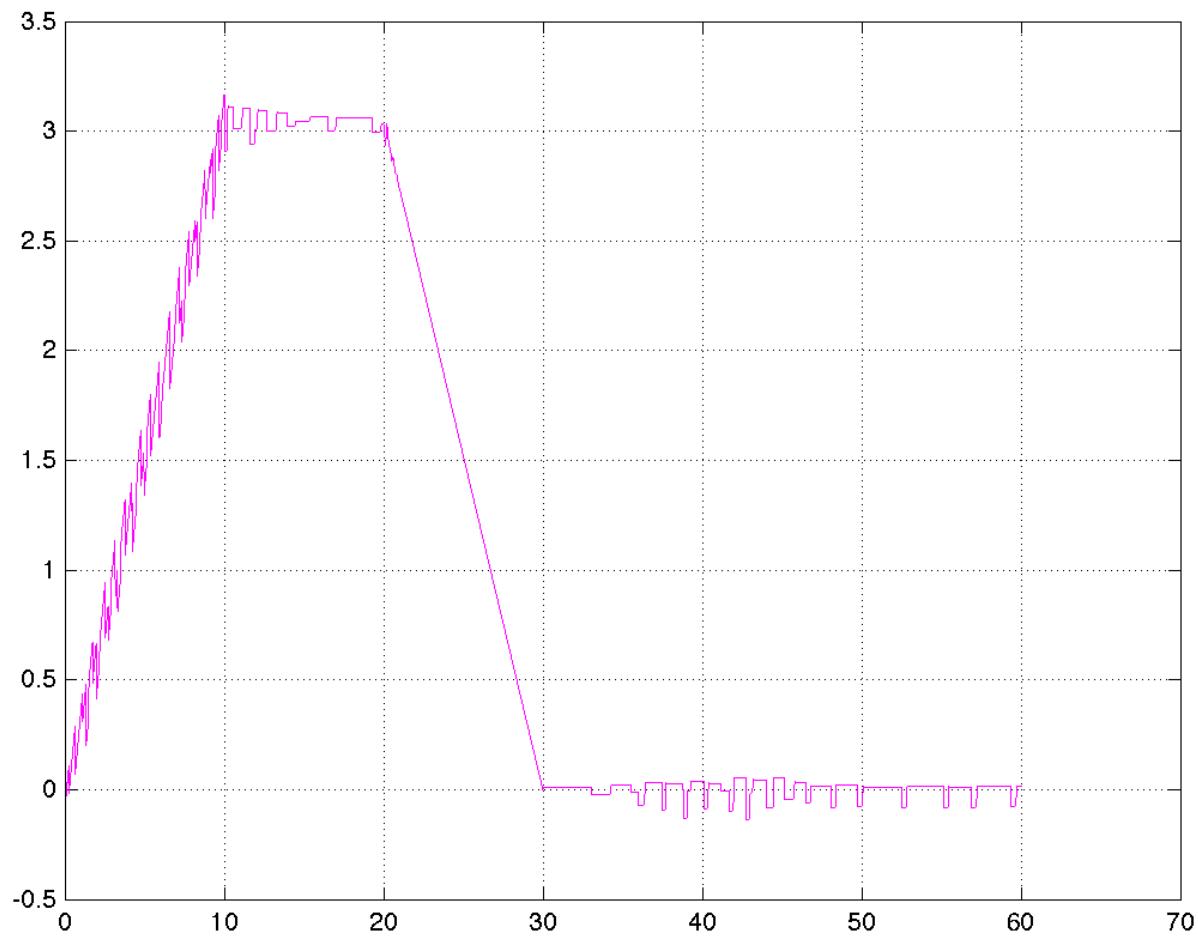


Test 3'nin Railpower değerleri:

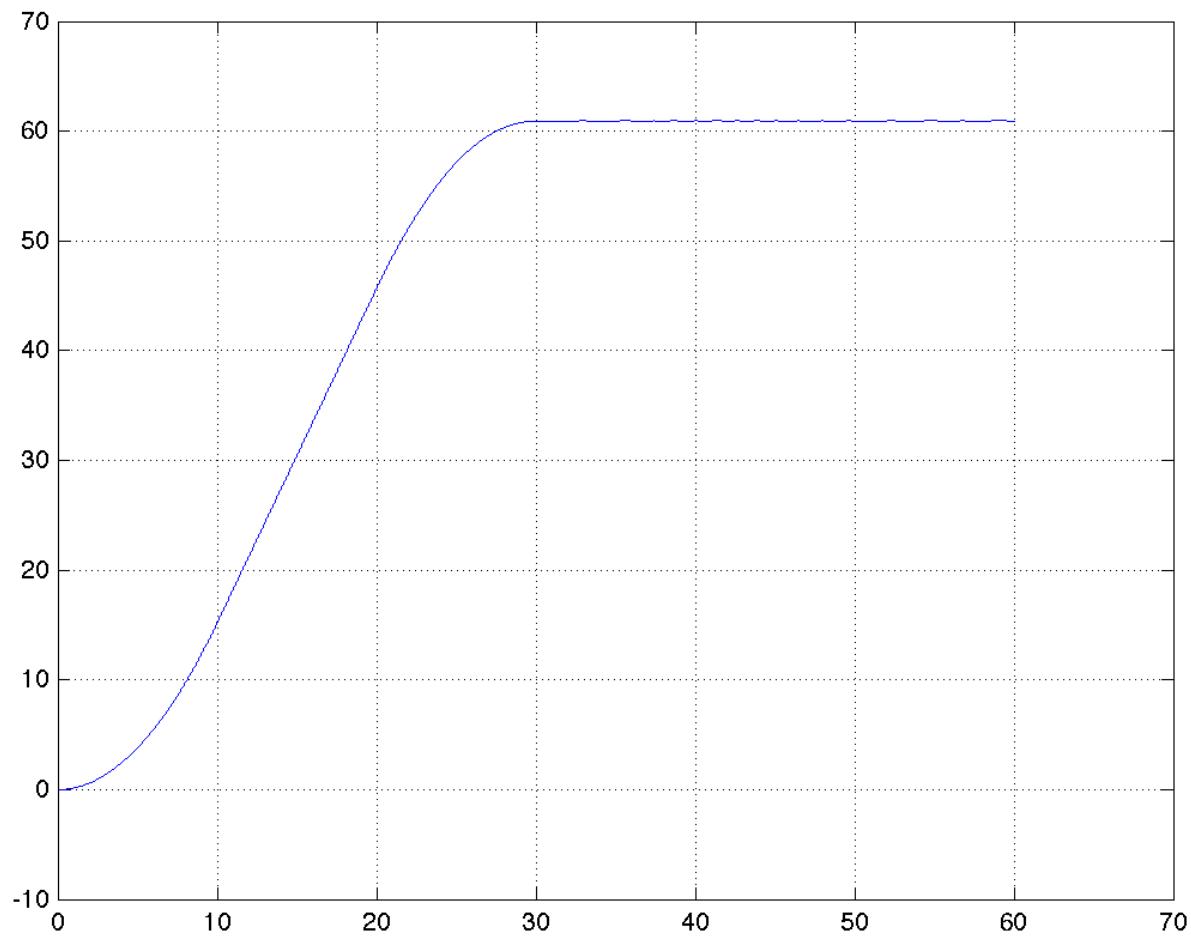
İvme 1. araç



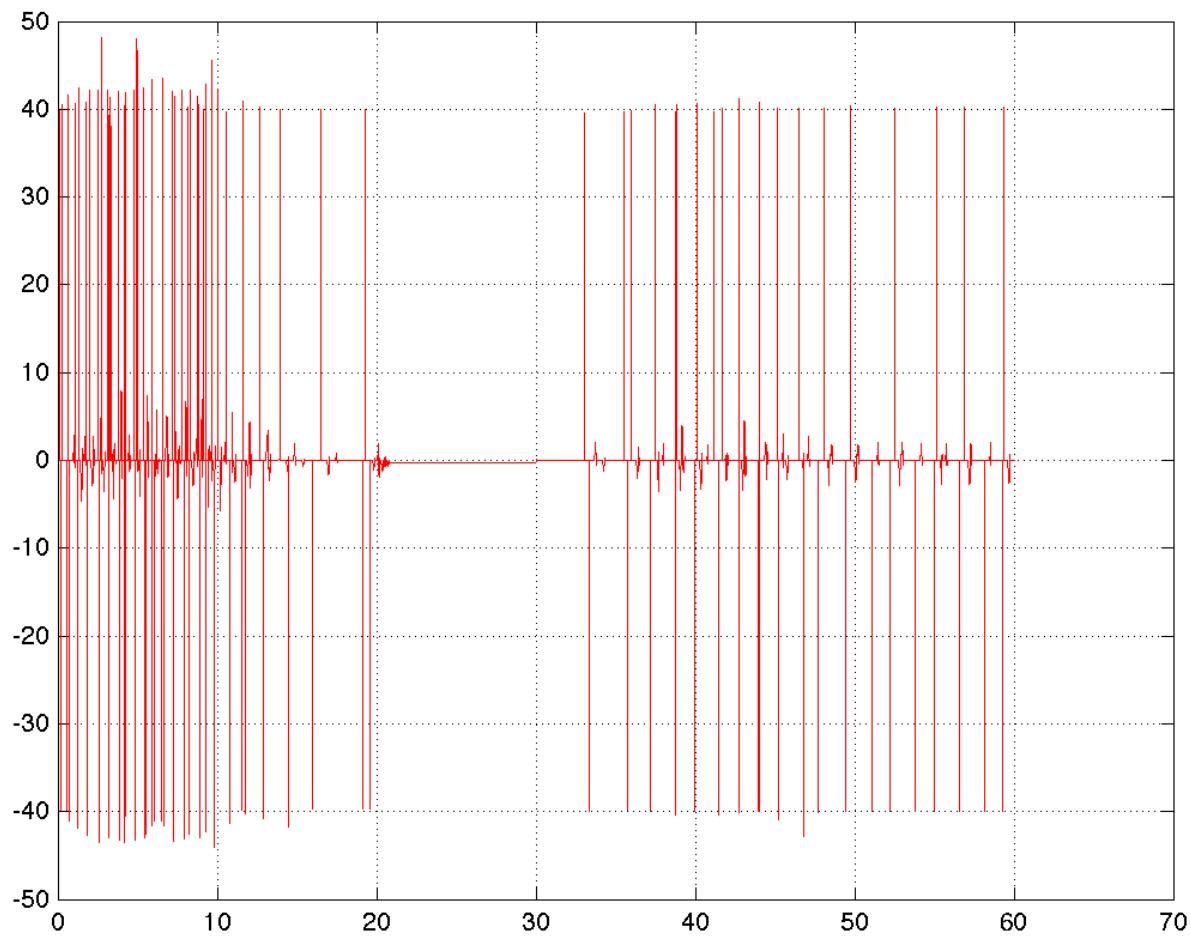
Hız 1. araç



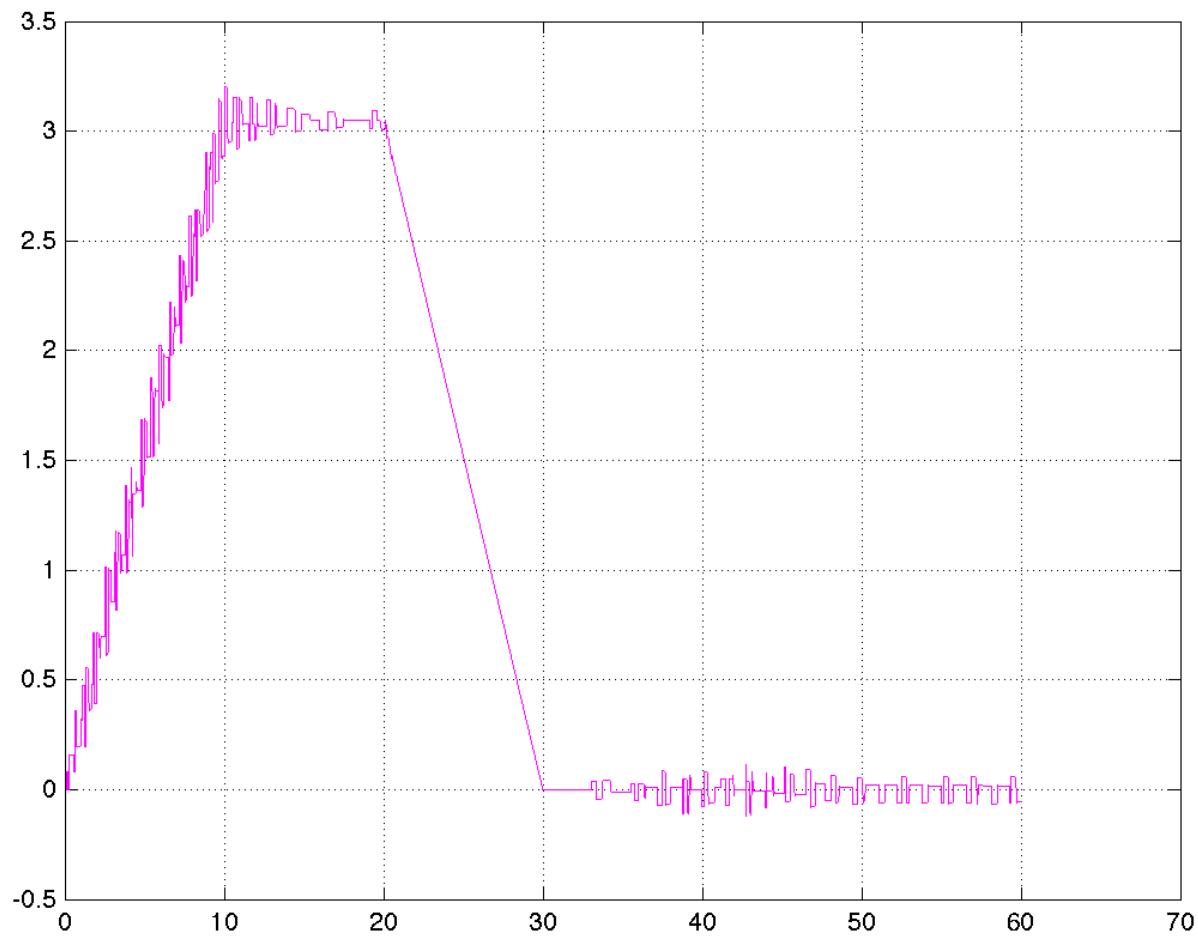
Pozisyon 1.araç



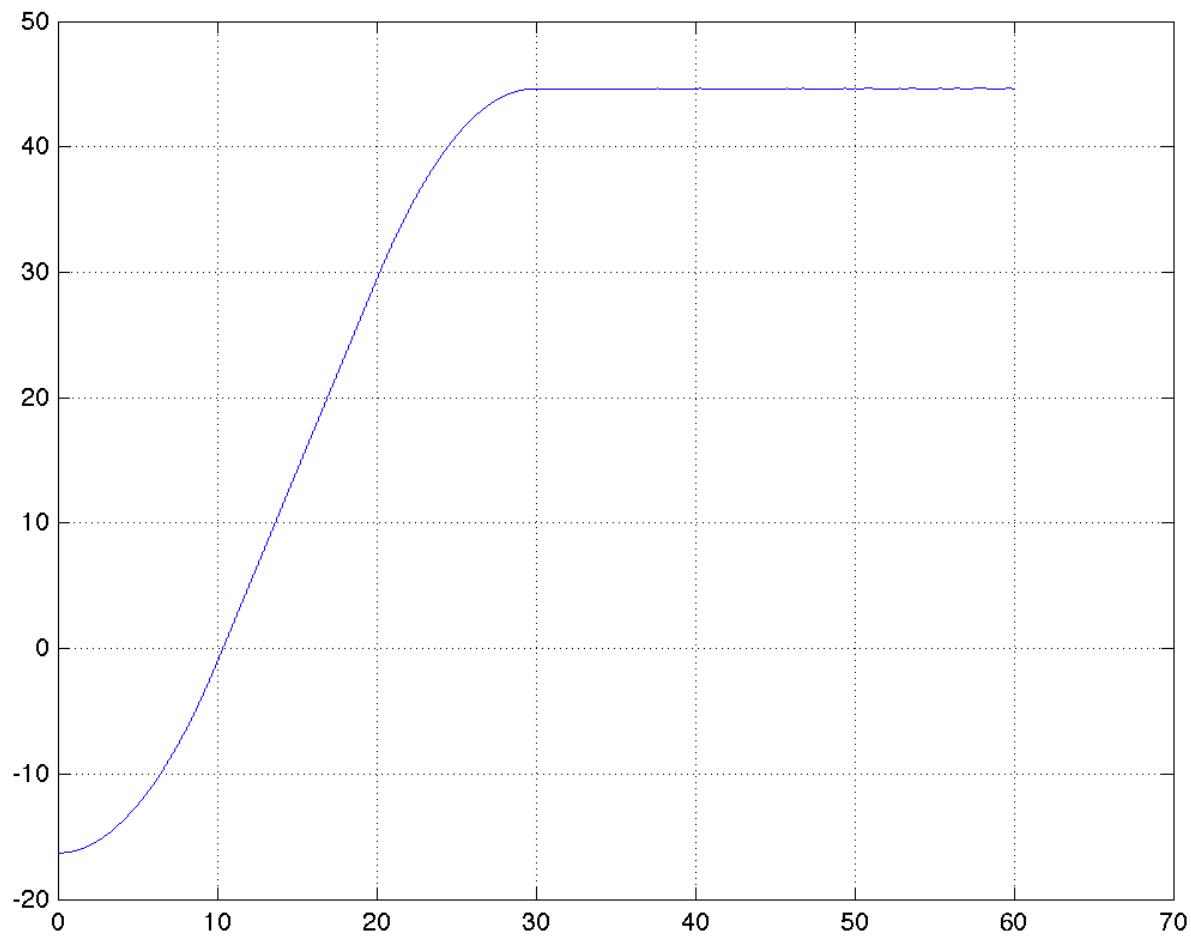
İvme 2. araç



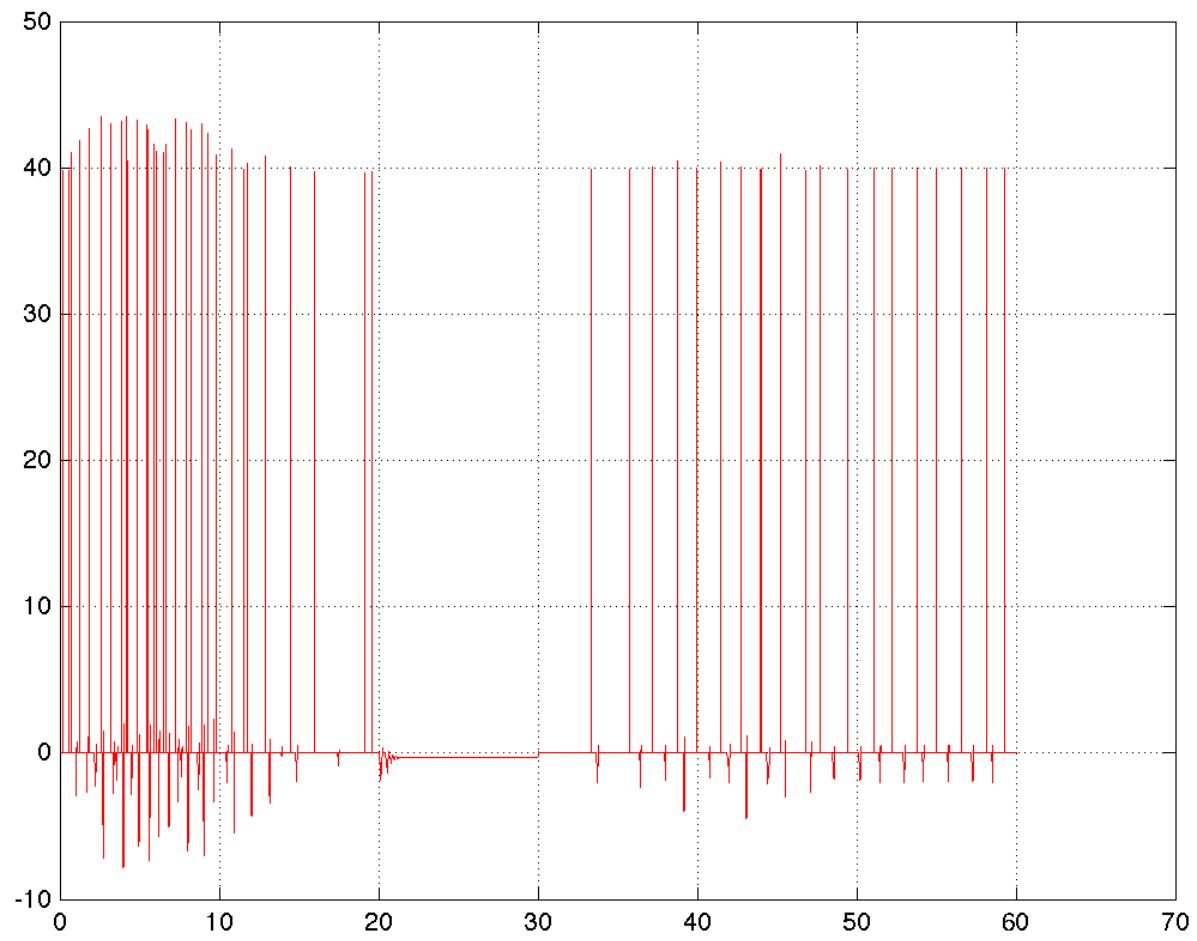
Hız 2. araç



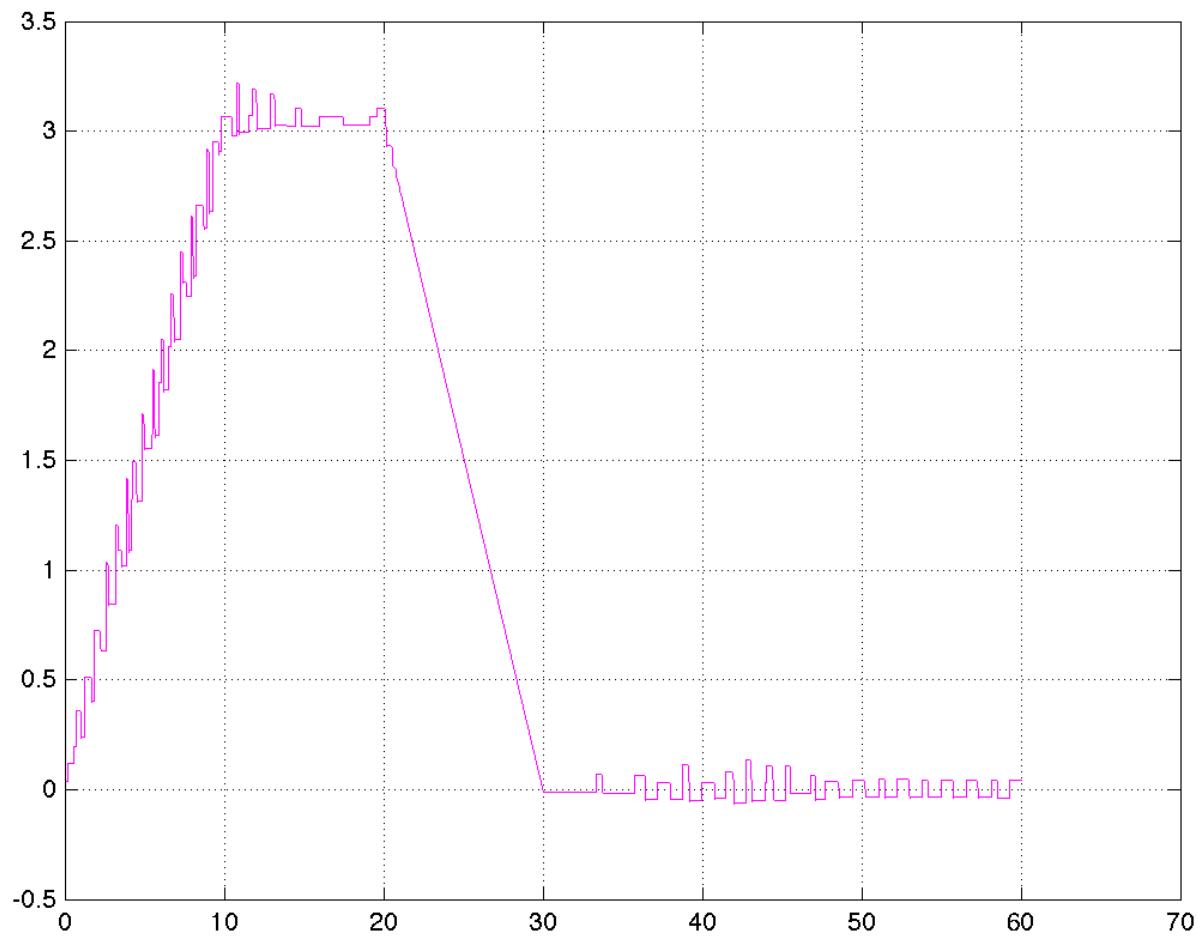
Pozisyon 2.araç



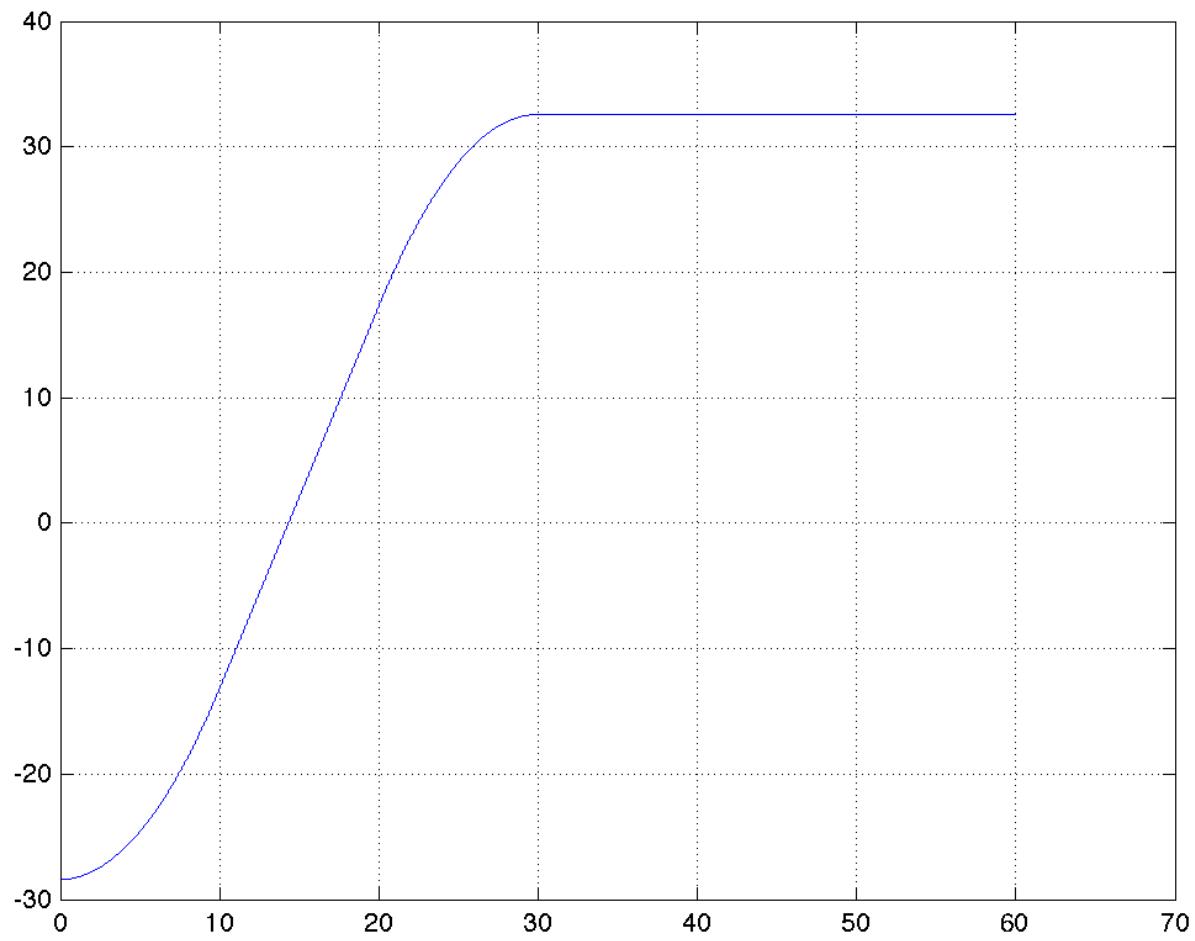
İvme 3. araç



Hız 3. araç



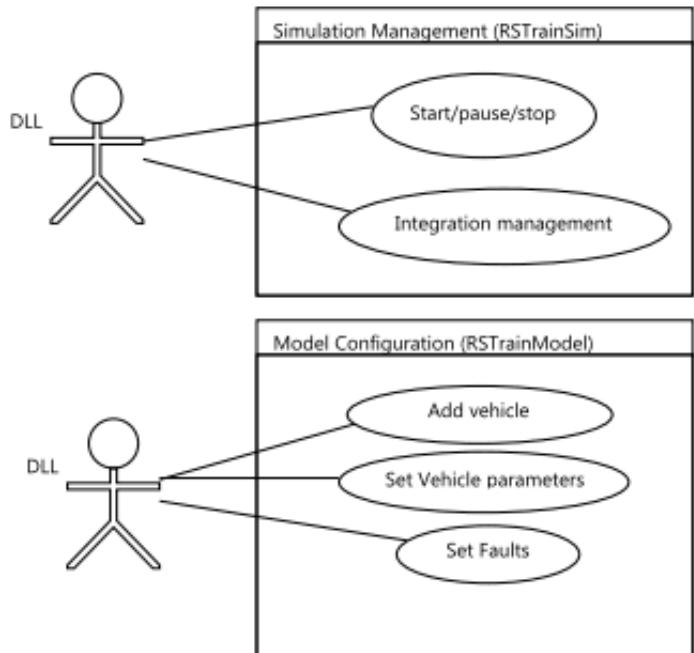
Pozisyon 3.araç



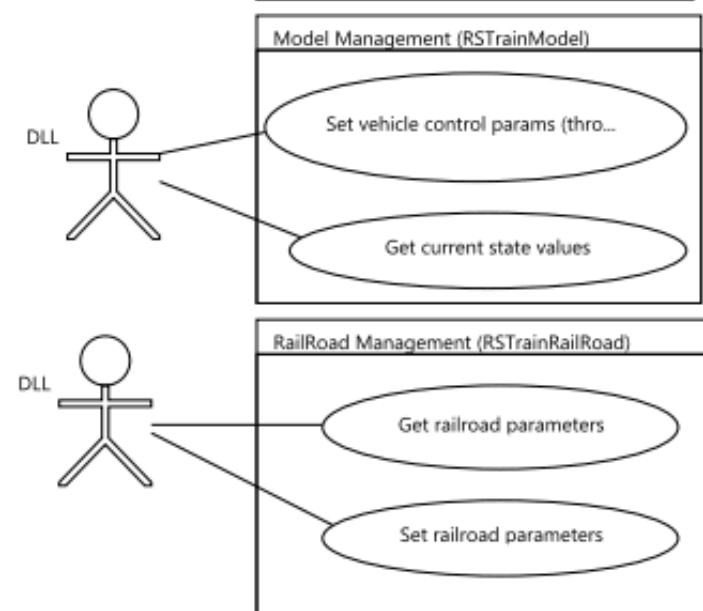
Testlerin validasyonu yapılmıştır.⁶

Runtime Birimi (API)

Runtime Birimi



Runtime birimi için use case diyagramı



API Fonksiyonları

```
int initCore();
```

Simülasyon süresince gerekli olacak olan nesnelerin hafızada yerini ayırır.

```
int mapPointers();
```

Tüm fiziksel nesne ekleme/çıkarma işlemlerinden sonra hafızadaki nesnelerin pointerlarını kullanılır hale getirir.

```
int addVehicleToSim(int ttype, double mass, double pos, double velo, double acce);
```

- *int ttype* : Araç tipini belirtir.
- *double mass* : Araç kütlesini belirtir. (**kg**)
- *double pos* : Aracın ilk iterasyona başlamadan önceki uzlamsal konumunu belirtir. (**m**)

- *double velo* : Aracın ilk iterasyona başlamadan önceki hızını belirtir. (**m/s**)
- *double acce* : Aracın ilk iterasyona başlamadan önceki ivmesini belirtir. (**m/s²**)

Sisteme araç ekler.

NOT: Bu fonksiyon çağrısından sonra tüm pointerlar **mapPointers** yardımıyla map edilmelidir.

```
int addCouplingToSim(double frontDistRefToCoupling, double rearDistRefToCoupling, double tensionDistance, bool nullify);
```

- *double frontDistRefToCoupling* : öndeği aracın orta noktasının couplinge olan uzaklığı
- *double rearDistRefToCoupling* : arkadaki aracın orta noktasının couplinge olan uzaklığı
- *double tensionDistance* : iki araç arasında kuvvet aktarımı olmayan en kısa uzaklık
- *bool nullify (opsiyonel)* : NULL PTR tipinde bir coupling ekler. Test durumlarının gözetimi içindir önerilmez.

Sisteme bağlantı ekler.

NOT: Bu fonksiyon çağrısından sonra tüm pointerlar **mapPointers** yardımıyla map edilmelidir.

```
int addForce(int vehIndex, double startTime, double stopTime, double forceValue);
```

- *int vehIndex* : kuvvetin uygulanacağı aracın eklenme sırasına göre(FILO) indeks numarasını belirtir.
- *double startTime* : kuvvetin integrasyon süresi boyunca hangi saniyeden itibaren uygulanacağını belirtir.(**s**)
- *double stopTime* : kuvvetin uygulanmasının integrasyon süresi boyunca hangi saniyede sonlanacağını belirtir.(**s**)
- *double forceValue* : kuvvetin değerini belirtir (**N**)

Sisteme kuvvet ekler.

NOT: Bu fonksiyon çağrısından sonra tüm pointerlar **mapPointers** yardımıyla map edilmelidir.

```
int setMainCouplerSpringConstant(int coupIndex, double sprConst);
```

- *int coupIndex* : spring sabitinin ekleneceği aracın indexini belirtir.
- *double sprConst* : spring sabitini belirtir.

Spring sabitini set eder.

Spring sabiti default olarak *1e8* dir.

```
int setCouplerDampingConstant(int coupIndex, double dmpConst);
```

- *int coupIndex* : damping sabitinin ekleneceği aracın indexini belirtir.
- *double dmpConst* : damping sabitini belirtir.

Damping sabitini set eder.

Damping sabiti default olarak *1e6* dir.

```
int setFrontCouplingToVehicle(int vehindex, int coupindex);
```

- *int vehindex* : öndeği bağlantının set edileceği aracın indexi
- *int coupindex* : bağlantının indeksi

İndeksi belirtilen araç nesnesinin önündeki bağlantının atamasını yapar.

```
int setRearCouplingToVehicle(int vehindex, int coupindex);
```

- *int vehindex* : arkadaki bağlantının set edileceği aracın indexi
- *int coupindex* : bağlantının indeksi

İndeksi belirtilen araç nesnesinin arkasındaki bağlantıların atamasını yapar.

```
int setFrontVehicleToCoupling(int coupindex, int vehindex);
```

- *int coupindex* : önündeki aracın set edileceği coupling'in indeksi
- *int vehindex* : set edilecek aracın indeksi

İndeksi belirtilen coupling nesnesinin önündeki aracın atamasını yapar.

```
int setRearVehicleToCoupling(int coupindex, int vehindex);
```

- *int coupindex* : arkasındaki aracın set edileceği coupling'in indeksi
- *int vehindex* : set edilecek aracın indeksi

İndeksi belirtilen coupling nesnesinin arkasındaki aracın atamasını yapar.

```
int linkVehiclesToCoupling(int coupIndex, int frontVehicleIndex, int rearVehicleIndex);
```

Yukarıda belirtilen `setFrontVehicleToCoupling` ve `setRearVehicleToCoupling` fonksiyonlarını birlikte işler.
Çift yönlü ilişkilendirme yapar.

```
int linkCouplingsToVehicle(int vehIndex, int frontCouplingIndex, int rearCouplingIndex);
```

Yukarıda belirtilen `setFrontCouplingToVehicle` ve `setRearCouplingToVehicle` fonksiyonlarını birlikte işler.
Çift yönlü ilişkilendirme yapar.

```
int setDt(double dt);
```

- *double dt* : integrasyon adım aralığının değeri

İntegrasyon adım aralığını set eder.

```
int prepareSim();
```

İntegratörün gereksinimi olan pointer koleksiyonlarını integratöre verir ve sistemi başlamaya hazır hale getirir.

```
int clearVehicles();
```

Tüm araçları sistemden siler.

```
int clearCouplings();
```

Tüm bağlantıları sistemden siler.

```
int clearForces();
```

Tüm kuvvetleri sistemden siler.

```
int clearSimulator();
```

Simülatörü sistemden siler. (Debug amaçlıdır yapılması tasvip edilmez.)

```
int runIntegration(double runningTime);
```

- *double runningTime* : integrasyonun koşulacağı süre

İntegrasyonu belirtilen süre boyunca koşar.

Default olarak `test.txt` dosyasına çıktıları oluşturmaktadır.

Sınıflar

Sınıf yapısı aşağıdaki şekilde görüldüğü gibidir:

RSForceSlice.cpp
RSForceSlice.h
RSRailRoad.cpp
RSRailRoad.h
RSTrainCar.cpp
RSTrainCar.h
RSTrainCoupling.cpp
RSTrainCoupling.h
RSTrainFreightCar.cpp
RSTrainFreightCar.h
RSTrainLocomotive.cpp
RSTrainLocomotive.h
RSTrainLogger.cpp
RSTrainLogger.h
RSTrainParser.cpp
RSTrainParser.h
RSTrainPassengerCar.cpp
RSTrainPassengerCar.h
RSTrainRailRoadParser.cpp
RSTrainRailRoadParser.h
RSTrainSim.cpp
RSTrainSim.h
RSTrainVehicle.cpp
RSTrainVehicle.h
RSTrainVehicleParser.cpp
RSTrainVehicleParser.h

Yapılacaklar

Yapılacaklar

- Fonksiyonlar ne işe yarar (doc)
- Bilimsel açıklama
- Değerlilik payı hakkında

Kütüphanede Yapılacaklar

- Exception mekanizması `catch (...) {` tarzı
- return değerleri gözden geçir
- thrust ekleme
- boost strip

Kaynakça

Kaynakça

1. Gauss-Lobatto integration with adaptive refinement, Bojan Nikolic: Numerical and Quantitative Methods in C++
2. Method of Analysis for Determining the Coupler forces and Longitudinal Motion of a Long Freight Train in Over-the-road Operation, G.C.Martin - W.W.Hay, University of Illinois
3. Longitudinal Train Dynamics, Colin Cole, p. 242
4. Mixed H_2/H_∞ cruise controlled design for high speed train, Ciann-Dong Yang & Yun-Ping Sun, International Journal of Control
5. Suboptimal Control Strategies for Multilocomotive Powered Trains, Peter Gruber, Mohamed M. Bayoumi IEEE Transactions on Automatic Control, Vol. AC-27, No. 3, JUNE 1982
6. Modelling and model validation of heavy-haul trains equipped with electronically controlled pneumatic brake systems, M. Chou, X. Xia, C. Kayser, ScienceDirect Control Engineering Practice 15 (2007) p. 501-509