Inverted pendulum with a mobile base

Segway

Authors :
Pierre-Henry ***Poret***
Julien ***Huynh***

Supervised by :
Mr. Sellami

May 29, 2020

Contents

Introduction

Segway and inverse pendulum models

The Segway is a two-wheeled self-balancing personal transporter.
This project, which has been started in 1994, was at the time seen as extremely promising. The Segway was potentially the personal vehicle of tomorrow, which everyone would use in their daily journeys. The founder of Apple Steve Jobs even told that it was "as big a deal as the PC".
The first units have been delivered in early 2002, and even if they did not have the expected success, we can today still find some in any part of the world, especially for tourism.

Figure 1: Segway used for tourism

Originally developed with a maximum speed between 10 and 13 km/h, they are now able to reach 40 km/h.

The principle of the dynamics governing the Segway is similar to an inverted pendulum:

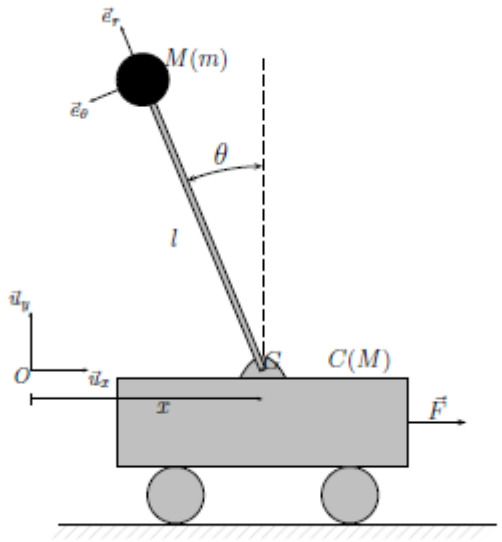


Figure 2: Inverted pendulum model

In a Segway, we are supposed to find electric motors in each wheel, powered by lithium batteries, as well as tilt sensors going along with gyroscopic sensors in order to input the direction orders.

Parameters

These are the parameters with which we are going to work on:

Data	Value
g	9.81 m/s^2
l	1 m
m	1 kg
M	15 kg

Table 1: Parameters

All codes and models will be made public on 1st June 2020 on our GitHub repository if you would like to test them :

`https://github.com/ECChew/SegwayModelMatlab`

Chapter 1

Non linear segway model

1.1 State space vector choice and determination

As we have two degrees of freedom with x and θ we will look at those variables aswell as their related time derivatives (velocities). Thus, our state space vector is :

$$X = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} \tag{1.1}$$

In order to get our non linear state space model, we will try to describe our system with $\dot{X} = f(X,\ t)$. This can be done easily as we have the expressions of $\ddot{x}$ and $\ddot{\theta}$.

$$\dot{X} = \begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \frac{-ml\dot{\theta}^2 \sin(\theta) + mg\cos(\theta)\sin(\theta) + \frac{T}{l}\cos(\theta) + F}{M+m-m\cos^2(\theta)} \\ \dot{\theta} \\ \frac{(M+m)gl\sin(\theta) - ml^2\dot{\theta}^2\cos(\theta)\sin(\theta) + \frac{M+m}{m}T + lF\cos(\theta)}{(M+m-m\cos^2(\theta))l^2} \end{bmatrix} \tag{1.2}$$

1.2 Simulation

We can now simulate the non linear system on Simulink. The model is thus :

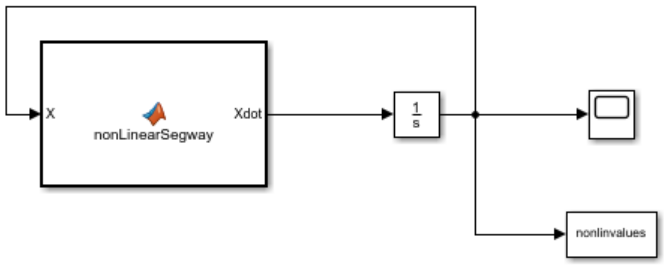


Figure 1.1: Non linear model - Simulink model

The function inserted in the Matlab function block is :

```
function Xdot = nonLinearSegway(X, m, M, g, l, T, F)
x = X(1); xd = X(2); theta = X(3); thetad = X(4);
xdd = (-m*l*thetad^2*sin(theta)+m*g*cos(theta)*sin(theta)+...
    T/l*cos(theta) + F)/(m+M-m*cos(theta)^2);
thetadd = ((M+m)*g*l*sin(theta)- m*m^2*thetad^2*cos(theta)*sin(theta)+...
    (M+m)/m * T+l*F*cos(theta))/((m+M-m*cos(theta)^2)*l^2);
Xdot = [xd; xdd; thetad; thetadd];
end
```

Then, the initial condition given to our integrator is :

Data	Value
x	0 m
$\dot{x}$	0 m/s
θ	$10° = \frac{\pi}{18}$ rad
$\dot{\theta}$	0 rad/s

Table 1.1: Initial condition - Non linear model

We can then plot the response of the system in this case. In this particular part we are taking a close look at the position x and the angle θ as we have a point of comparison with the source paper[1] where the authors did it with Maple.

[1] https://www.pobot.org/IMG/pdf/Pendule_inverse_en_robotique.pdf

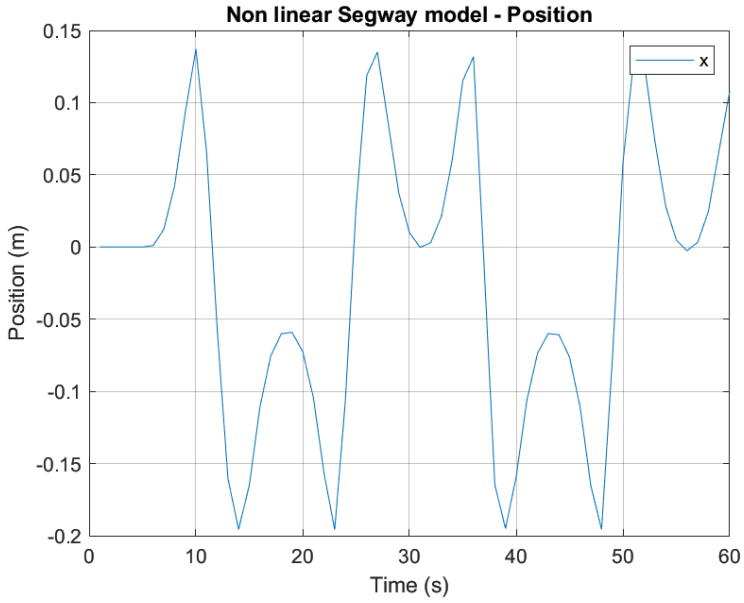


Figure 1.2: Non linear model - Position

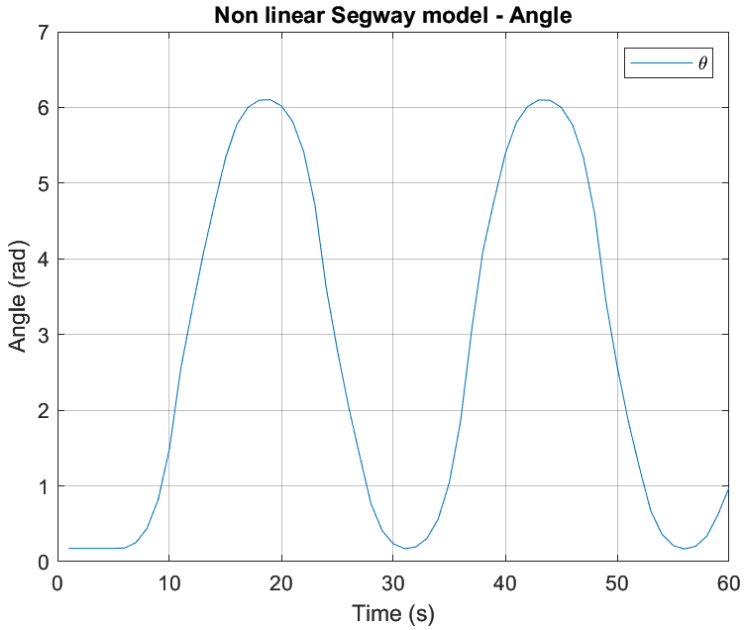


Figure 1.3: Non linear model - Angle

In order to get those graphs, we did set the torque and the force values at 0 N and 0 Nm.
Also we should note that our tests with non-zero values for the torque as well as for the force outputted the expected results which were an indefinitely increasing of the position for a non-zero force and an indefinitely increasing of the angle for a non-zero torque.

While the amplitude on the position seems to differ from what the authors of the source document had [2], the overall shape of the position over time is the same and the angle variation over time is the exact same as the one they had on Maple.

However, while we are getting the same shapes of curves, we would say that the result is not quite satisfactory.

As we can see on the first plot (position over time), the segway seems to oscillate for ever which is far from what we want. We started here from a 10° angle and the resulting position is varying between -20 **cm** and $+14$ **cm**. Needless to say that this is far from being a good result. This could either come from the segway itself or the model.

The angle plot isn't much better, it is oscillating between our initial angle and almost 2π which is first very bad since the segway rod would simply be just turning around the center of gravity without stopping and the angle should never get out of the $]-\frac{\pi}{2}, \frac{\pi}{2}[$ interval as it defines the "higher" and "free" plane since the lower one would be the ground and we for sure do not want to collide with it.

Later on, we will design a controller that will fix this issue and make this segway acceptable in terms of response and stability.

[2] `https://www.pobot.org/IMG/pdf/Pendule_inverse_en_robotique.pdf`

Chapter 2

Equilibrium points

As we could see in the previous chapter, our system was non linear. We wish to linearize it around one of the equilibrium points which first requires us to find those points.

The equilibria are given by solving $\dot{X} = 0$. When considering the same state vector as used in the previous chapter, this means :

$$\begin{bmatrix} \dot{x} \\ \frac{-ml\dot{\theta}^2\sin(\theta)+mg\cos(\theta)\sin(\theta)+\frac{T}{l}\cos(\theta)+F}{M+m-m\cos^2(\theta)} \\ \dot{\theta} \\ \frac{(M+m)gl\sin(\theta)-ml^2\dot{\theta}^2\cos(\theta)\sin(\theta)+\frac{M+m}{m}T+lF\cos(\theta)}{(M+m-m\cos^2(\theta))l^2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.1}$$

By supposing that $T = 0$ Nm and $F = 0$ N, the equation becomes :

$$\begin{bmatrix} \dot{x} \\ \frac{-ml\dot{\theta}^2\sin(\theta)+mg\cos(\theta)\sin(\theta)}{M+m-m\cos^2(\theta)} \\ \dot{\theta} \\ \frac{(M+m)gl\sin(\theta)-ml^2\dot{\theta}^2\cos(\theta)\sin(\theta)}{(M+m-m\cos^2(\theta))l^2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.2}$$

And as $\dot{x} = 0$ and $\dot{\theta} = 0$:

$$\begin{cases} mg\cos(\theta)\sin(\theta) & = 0 \\ (M+m)gl\sin(\theta) & = 0 \end{cases} \tag{2.3}$$

In order to have both of those equations being equal to 0, we need :

$$\sin(\theta) = 0 \Rightarrow \theta = 0 \textbf{ or } \theta = \pi \tag{2.4}$$

While we have two solutions, $\theta = \pi$ does not seem to be realistic as it would be that the segway would be rotated in a way that the user and the handling rod would both be under ground.

Chapter 3

Linear model

3.1 Linearizing around vertical equilibrium

We decided to linearize only around 0 even though π is also considered as an equilibrium as with an angle around π, the rod of the segway aswell as the user would be underground which is unrealistic.

Thus, for small values of θ, we have :

$$\begin{cases} \cos(\theta) & \approx 1 \\ \sin(\theta) & \approx \theta \end{cases}$$

And since we are close to an equilibrium point, $\dot{\theta}$ is close to 0 so we can say that $\dot{\theta}^2 \to 0$. We can then express our linearized model :

$$\begin{cases} \ddot{x} & = \frac{mg\theta + \frac{T}{l} + F}{M} \\ \ddot{\theta} & = \frac{(M+m)gl\theta + \frac{M+m}{m}T + lF}{ml^2} \end{cases} \tag{3.1}$$

And our expression of $\dot{X}$ becomes :

$$\dot{X} = \begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{(m+M)g}{Ml} & 0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}}_{X} + \underbrace{\begin{bmatrix} 0 & 0 \\ \frac{l}{m} & \frac{1}{m} \\ 0 & 0 \\ \frac{m+M}{mMl^2} & \frac{1}{Ml} \end{bmatrix}}_{B} \underbrace{\begin{bmatrix} T \\ F \end{bmatrix}}_{u} \tag{3.2}$$

If we were to define the C and D matrix for the output, those would be :

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.3}$$

$$D = 0 \tag{3.4}$$

3.2 Simulation and stability tests

We can then enter the calculated matrices in Matlab and use Simulink to take a closer look at the response. As for the previous test, we use a Matlab function in Simulink with an integrator.

The model is :

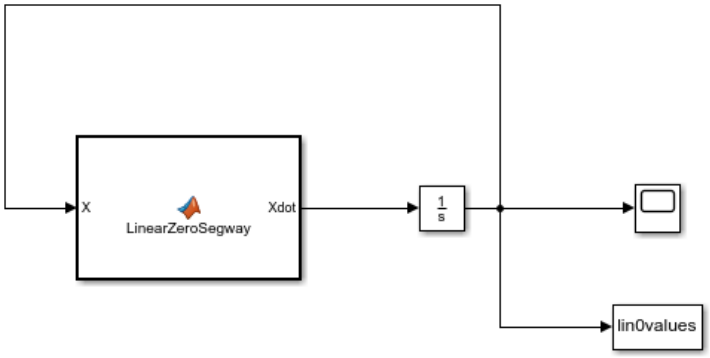


Figure 3.1: Linearized model - Simulink model

The function contained in the block is :

```
function Xdot = LinearZeroSegway(X, A, B, T, F)
u = [T; F];
Xdot = A * X + B * u;
end
```

3.2.1 First test

For this first test, the initial condition we chose is :

Data	Value
x	0 m
$\dot{x}$	0 m/s
θ	$5° = 5 \times \frac{\pi}{180}$ rad
$\dot{\theta}$	0 rad/s

Table 3.1: Initial condition - Linearized model around 0

The resulting response of the system is given by :

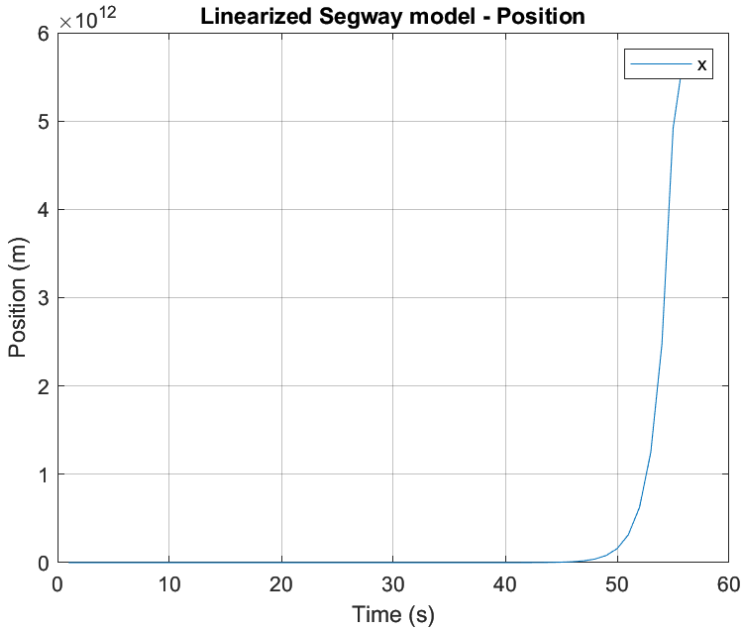


Figure 3.2: Linearized model - Position - 5 Deg

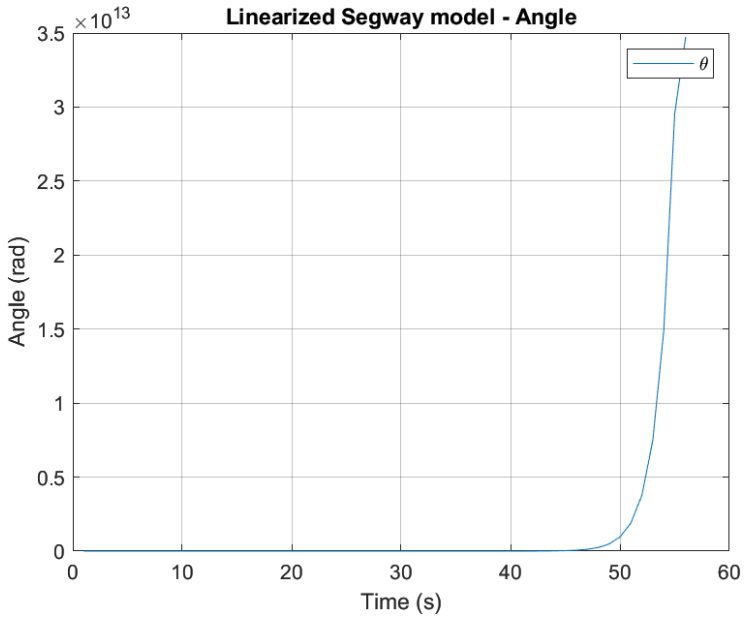


Figure 3.3: Linearized model - Angle - 5 Deg

As we can see on those curves, the system is highly unstable. This might be due to the fact that the system was already oscillatory and then with our linearization, there could the fact that the velocity becomes constant and different from 0 which means that we will move to infinity. We then need more tests and most likely a controller to fix this issue.

3.2.2 Second test - $\theta = 0°$

We saw previously that our system was diverging with an angle of 5° so before checking anything else, we wanted to make sure that the system would remain at the equilibrium if it started there.

Thus, we are setting here that $x = 0$ and $\theta = 0$. We would get the same kind of response (or more like absence of response from the system in this case) if we kept the θ to 0 and set x to another value. This was tested but not shown in this report as it has little interest.

With Simulink, we could indeed confirm that there was no response :

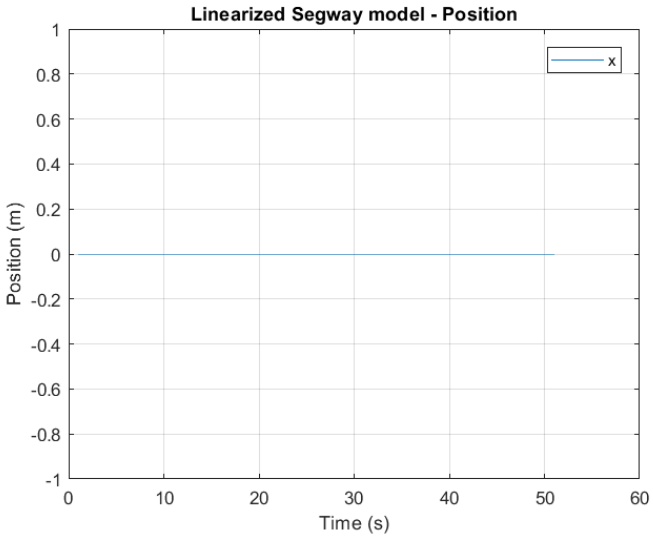


Figure 3.4: Linearized model - Position - 0 Deg

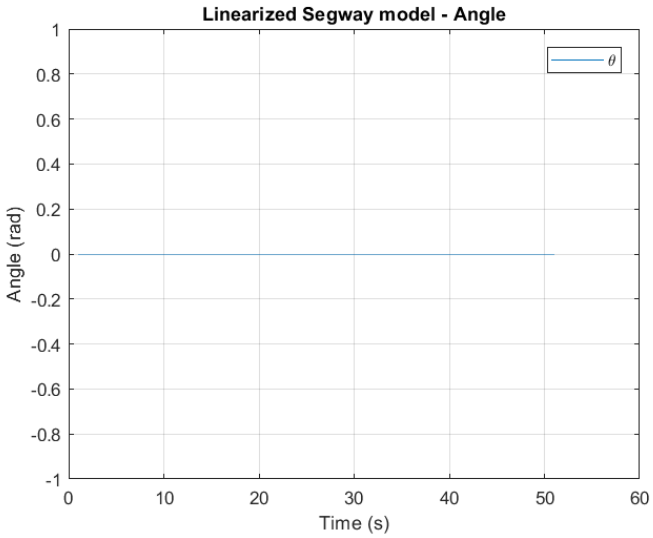


Figure 3.5: Linearized model - Angle - 0 Deg

As expected when the segway is already at the equilibrium point and has no input, it just stays at this equilibrium explaining that we see no movement at all. We will then check for other values of θ.

3.2.3 Third test - $\theta = 1°$

After checking and confirming that our system was indeed stable when at an equilibrium, we wanted to confirm that the diverging motion that we got as a result in our first test was not due to our initial condition being too far from the linearization point (which would then make our approximation irrelevant).

Thus we decided to test with an angle of $\theta = 1°$ and we kept the initial position at $x = 0$ m.

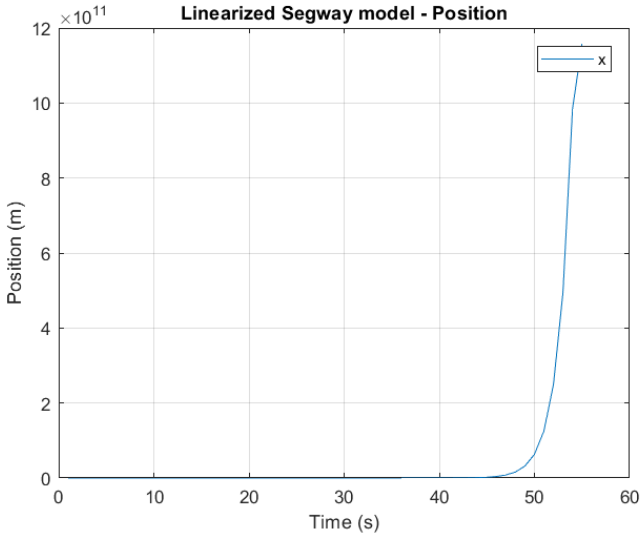


Figure 3.6: Linearized model - Position - 1 Deg

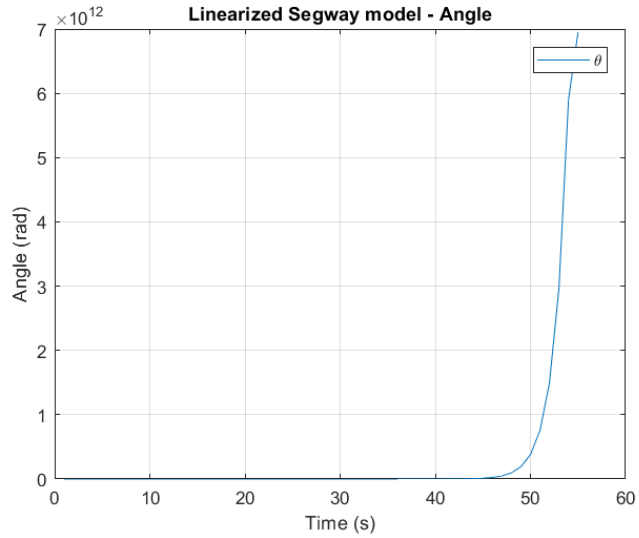


Figure 3.7: Linearized model - Angle - 1 Deg

We could tell by those two previous plots that even when we are very close to the equilibrium (very slight perturbation of 1°, the system behaves in a very unstable way, showcasing the same issue as the one we had in the first part.

We can also confirm this tendency with an initial condition further away from the equilibrium even though we do expect it to behave in this same diverging way.

3.2.4 Fourth test - $\theta = 10°$

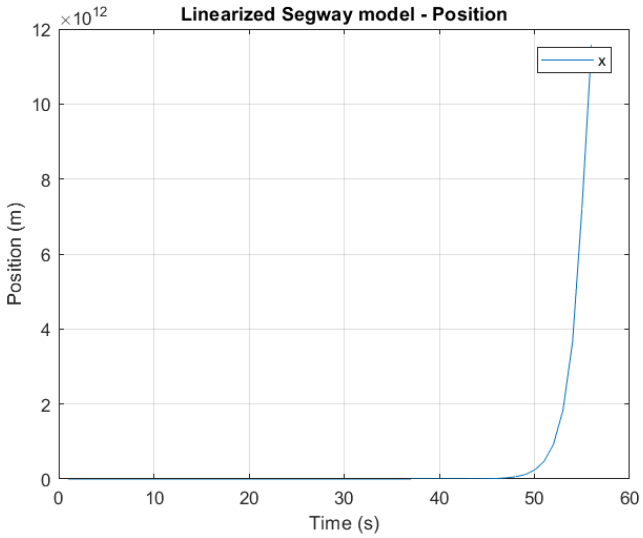


Figure 3.8: Linearized model - Position - 10 Deg

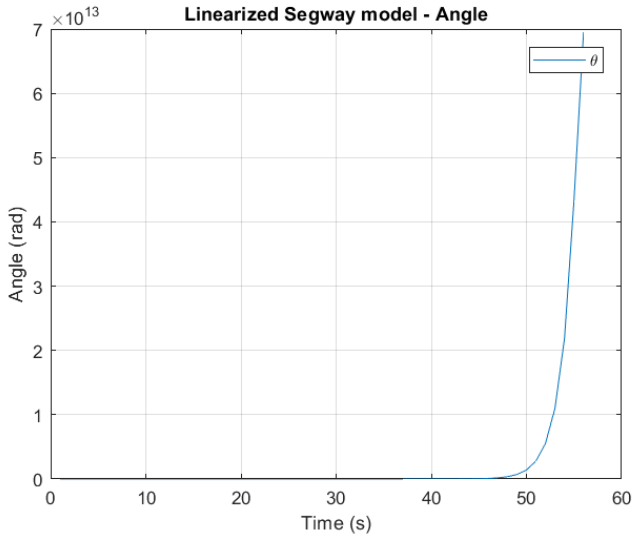


Figure 3.9: Linearized model - Angle - 10 Deg

Chapter 4

Controller design

4.1 Controllability

After noticing that our linearized system was unstable with the attempted initial conditions, we decided that it was time to move onto designing a controller to improve the system's stability. First we shall verify that the system is indeed controllable. This is done by using the controllability matrix that should be full rank :

$$Ctrb(sys) = \begin{bmatrix} B & AB & A^2B & A^3B \end{bmatrix} \tag{4.1}$$

With Matlab, we can check wheter this matrix is of full rank or not :

```
CtrSys = ctrb(A, B)
[n, m] = size(CtrSys)
if rank(CtrSys) == min(n, m)
    fprintf('Controllable')
end
```

The result is that it is indeed of full rank as it is of rank 4 for a 4×8 matrix. Thus, we can move onto the controller design with pole placement.

4.2 Pole placement

For the pole placement method, we need to have the following kind of model :

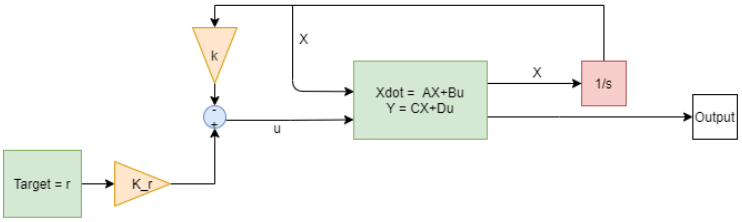


Figure 4.1: Pole placement - General model

The r target changes depending on the kind of task we want to accomplish. The K value is computed using *place* with some stable, non oscillatory desired poles. And the K_r gain is actually the inverse of the dcgain of the system. One important thing to note is that the 4 desired poles we will use are :

$$p = \begin{bmatrix} -1 \\ -2 \\ -3 \\ -4 \end{bmatrix} \tag{4.2}$$

4.3 From $(x \neq 0,\ \theta = 0)$ to $(x = 0,\ \theta = 0)$

First of all, we want the segway to get back to the $(x = 0,\ \theta = 0)$ point when starting from another x position and an initial angle of $\theta = 0$. The initial velocities are also set to 0. The targeted vector is $Target_{PP1} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T$.

We then design the following model : The model is :

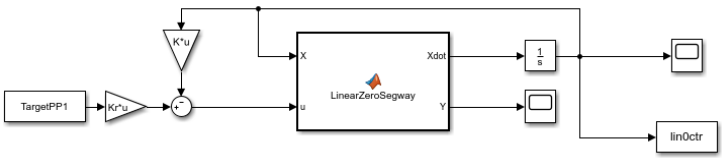


Figure 4.2: Controlled linearized model - Simulink model

The function contained in the block is :

```
function [Xdot, Y] = LinearZeroSegway(X, A, B, C, D, u)
Xdot = A * X + B * u;
Y = C*X + D*u;
end
```

The initial condition considered is :

Data	Value
x	1 m
$\dot{x}$	0 m/s
θ	0 rad
$\dot{\theta}$	0 rad/s

Table 4.1: Initial condition - Pole placement from a $x \neq 0$ to $x = 0$

The response is then :

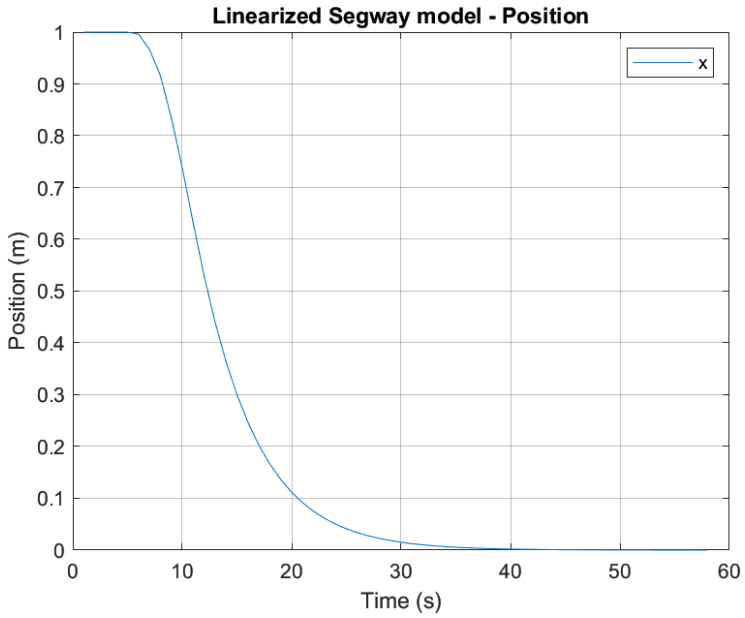


Figure 4.3: Controlled linearized model - Position

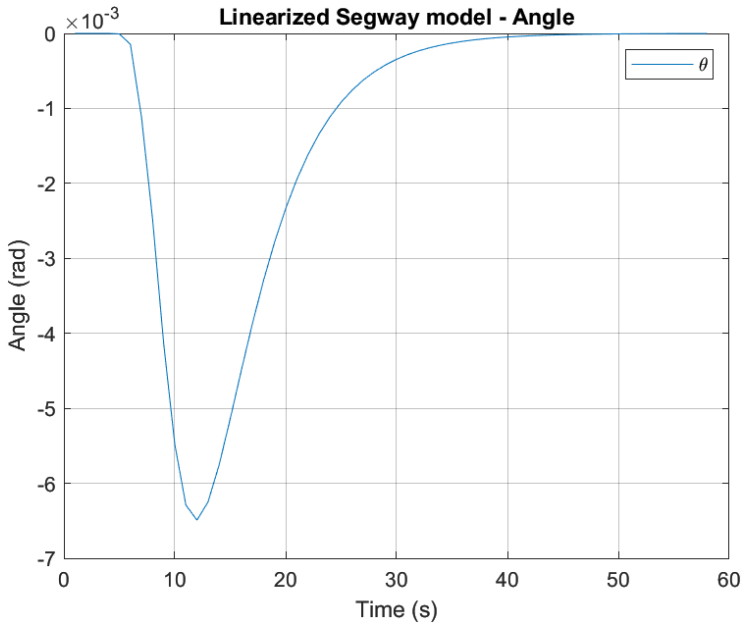


Figure 4.4: Controlled linearized model - Angle

As we can see on those previous figures, the response of the segway is very satisfactory as it moves back to $x = 0$ and the angle is rather small which is good since this means the inclination of the segway remains low and the user will not feel uncomfortable.

However, the velocity is rather small which means that the movement is smooth but we can improve the speed by just modifying the desired poles. For example, if we multiply all those negative poles by 5, the response becomes :

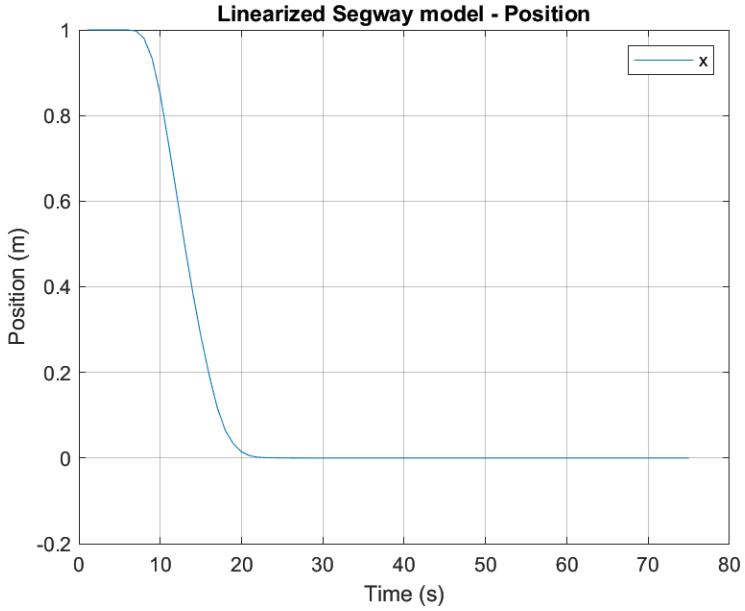


Figure 4.5: Controlled linearized model with modified poles - Position

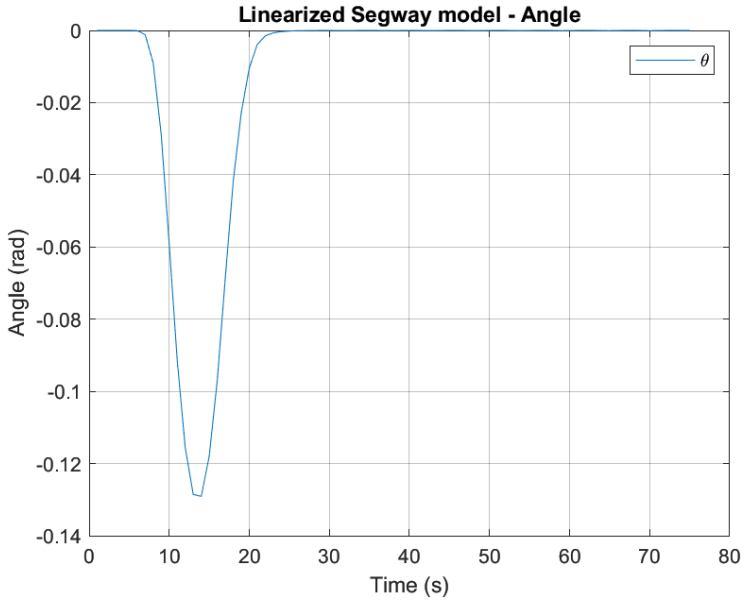


Figure 4.6: Controlled linearized model with modified poles - Angle

As we can see on those plots, the system moves faster towards our goal but we can't just change the poles randomly as our system might not be able to give this kind of response depending on the actuators we use. For the following simulations, we will keep the poles that we mentionned in the introduction of the controller design.

4.4 From $(x = 0,\ \theta = 0)$ to $(x \neq 0,\ \theta = 0)$

For this part, what we will change in the previous model is the target and the initial condition with the target being $Target_{PP2} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T$ as we want our segway to go from $x = 0$ to $x = 1$. The model is :

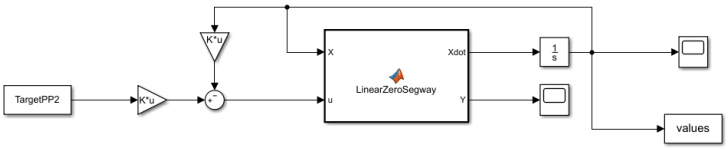


Figure 4.7: Controlled linearized model to a given position - Simulink model

And the initial condition is :

Data	Value
x	0 m
$\dot{x}$	0 m/s
θ	0 rad
$\dot{\theta}$	0 rad/s

Table 4.2: Initial condition - Pole placement from a $x = 0$ to $x \neq 0$

The reponse is then :

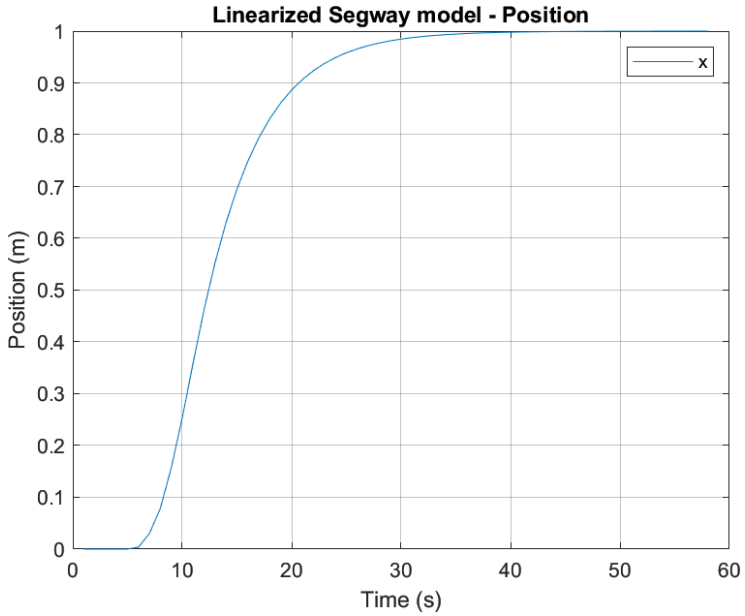


Figure 4.8: Controlled linearized model to a given position - Position

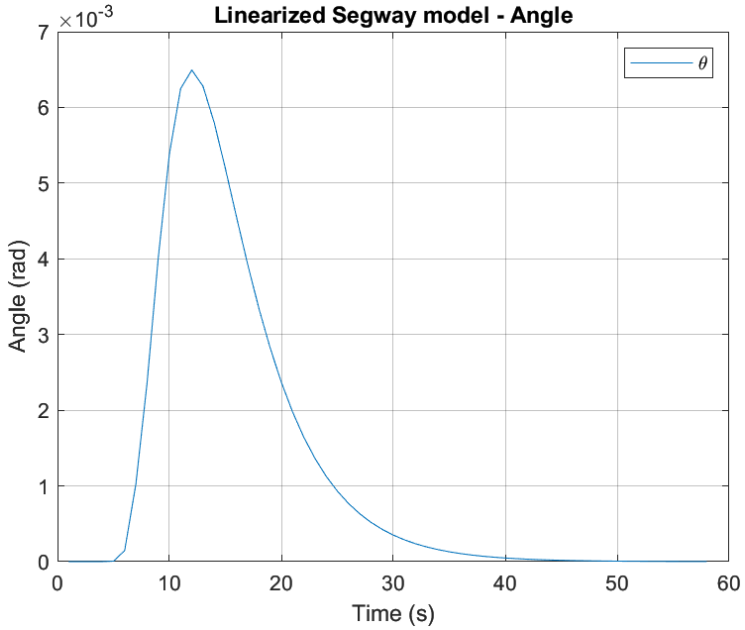


Figure 4.9: Controlled linearized model to a given position - Angle

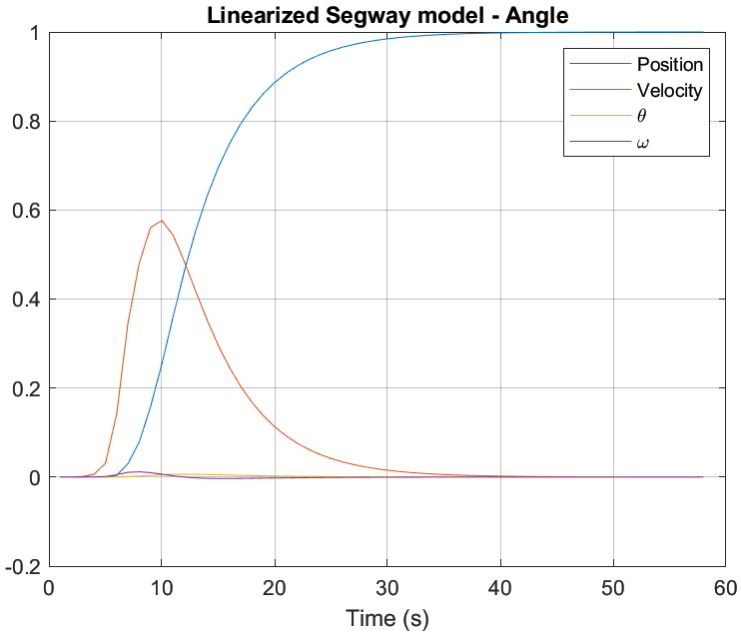


Figure 4.10: Controlled linearized model to a given position - All variables

The response is rather satisfactory as the segway does go to the required position and stabilizes there. However, we can still mention that it is rather slow at accomplishing this task.

4.5 Moving from ($x = 0,\ \theta = 0$) with a constant velocity (attempts)

We had several ideas to work with on this point. We first thought that if we wanted to control the velocity, we should change the C matrix to match the desired output. With our state vector being $X = \begin{bmatrix} x & \dot{x} & \theta & \dot{\theta} \end{bmatrix}^T$, the new C matrix becomes :

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.3}$$

But then we realized that our linear model was linearized around an equilibrium point which means that around it, it will try to move towards it with the target being $\dot{X} = 0$.

We then tried to do the linearization with the Jacobian matrix via Matlab with the linmod command but the response was very far from what we expected and thus, this part should be studied further.

Results of our attempts :

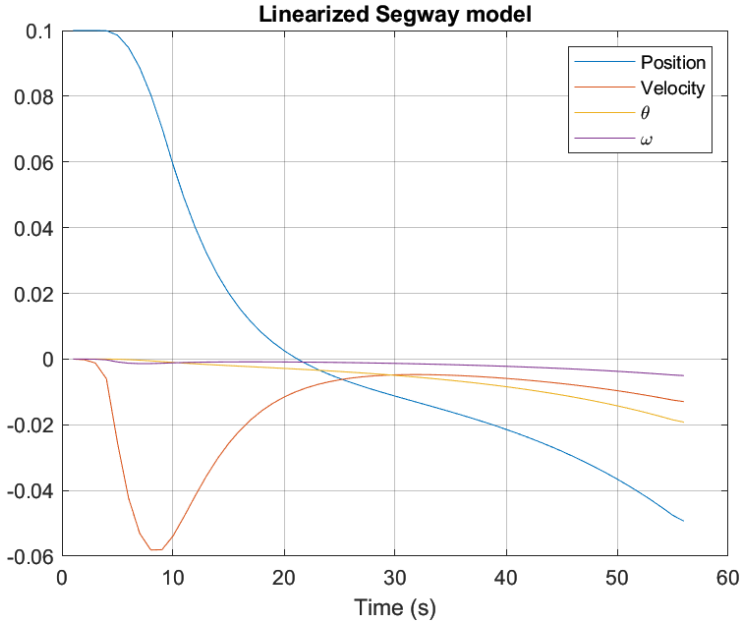


Figure 4.11: Attempt at moving from the origin with constant velocity

Conclusion

Through this work, we have focused on understanding better how the dynamics of the Segway was working.

Basically, we have worked on the *inverted pendulum*. Starting with the analyze of this on linear model, we simulated it and observed its responses.
Not really satisfied with what we obtained, we then worked on the finding of equilibrium points in order to, afterward, go through linearization.
Finally, we worked on a controller design which gave us very encouraging results.

What may seem paradoxical is that on one hand, for someone who has little or no knowledge of physics, the Segway can appear as a really complicated object, really hard to design. On the other hand, for someone learning engineering, the Segway may not seem so complex, the model being simply an inverted pendulum.

However it's when we start getting interested in the control part that we realize that there are still some difficulties but, with a little work, we always manage to get something very effective.

List of Figures

List of Tables

TP - Segway

Authors : Pierre-Henry Poret et Julien Huynh

```
clear vars; clc;
```

Parameters

```
g = 9.81;
l = 1;
m = 1;
M = 5;
T = 0;
F = 0;
```

Initial condition

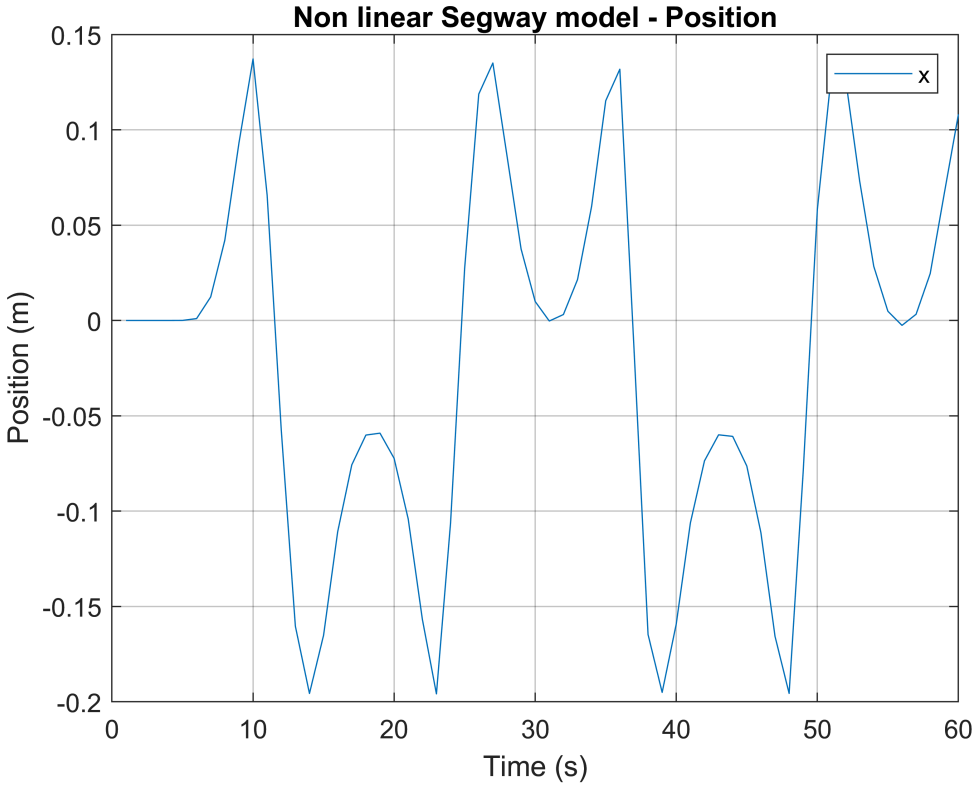
```
X0 = [0; 0; pi/18; 0]; % Non linearized model
X0lin0 = [0; 0; 5*pi/180; 0]; % Linearized around 0
X0To0 = [1; 0; 0; 0]; % First pole placement
X0ToPos = [0; 0; 0; 0]; % Second Pole placement
FromX0Spd = [0.1; 0; 0; 0]; % Third polce placement
```

Targets

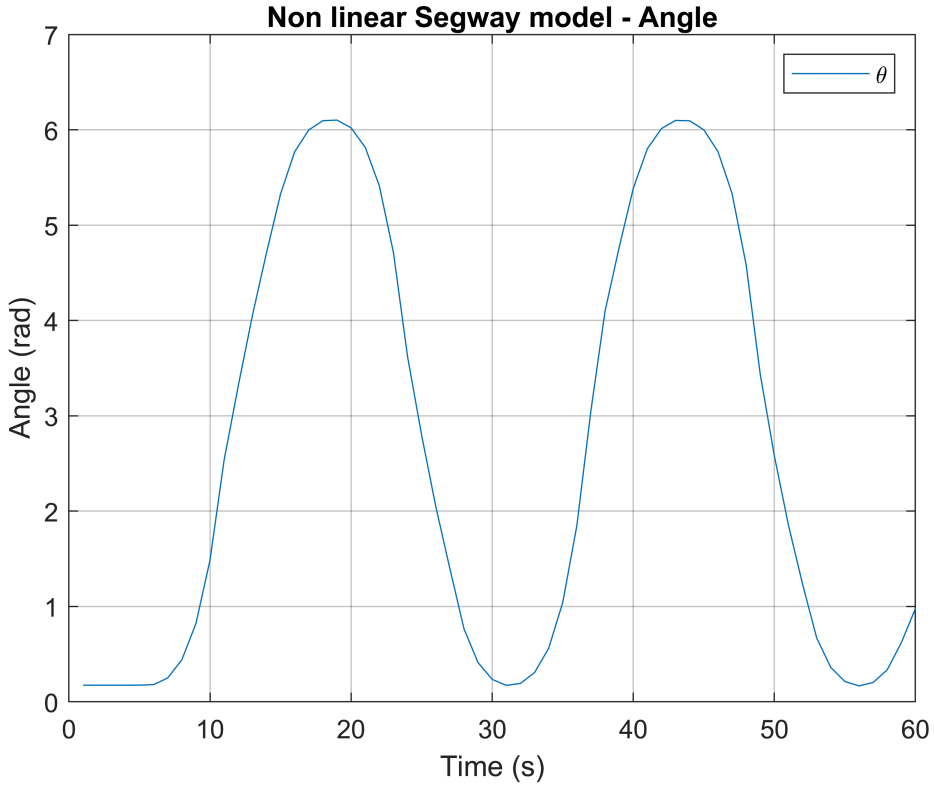
```
TargetPP1 = [0; 0; 0; 0]; % Pole placement 1, from a position to 0
TargetPP2 = [1; 0; 0; 0]; % Pole placement 2, from 0 to a position
TargetPP3 = [10; 100; 0; 0]; % Pole placement 3, from 0 to a position
```

Non linear segway

```
simu = sim("SegwayModel");
figure()
plot(simu.nonlinvalues(:, 1)); % Plot x
legend('x');
xlabel("Time (s)")
ylabel("Position (m)")
grid on;
title("Non linear Segway model - Position")
```




```
figure()
plot(simu.nonlinvalues(:, 3)); % Plot theta
xlabel("Time (s)")
ylabel("Angle (rad)")
legend('{\theta}');
grid on;
title("Non linear Segway model - Angle")
```



Linearized around $\theta = 0$

```
A = [0 1 0 0
     0 0 m*g/M 0
     0 0 0 1
     0 0 (M+m)*g/(M*l) 0]
```

```
A = 4×4
         0    1.0000         0         0
         0         0    1.9620         0
         0         0         0    1.0000
         0         0   11.7720         0
```

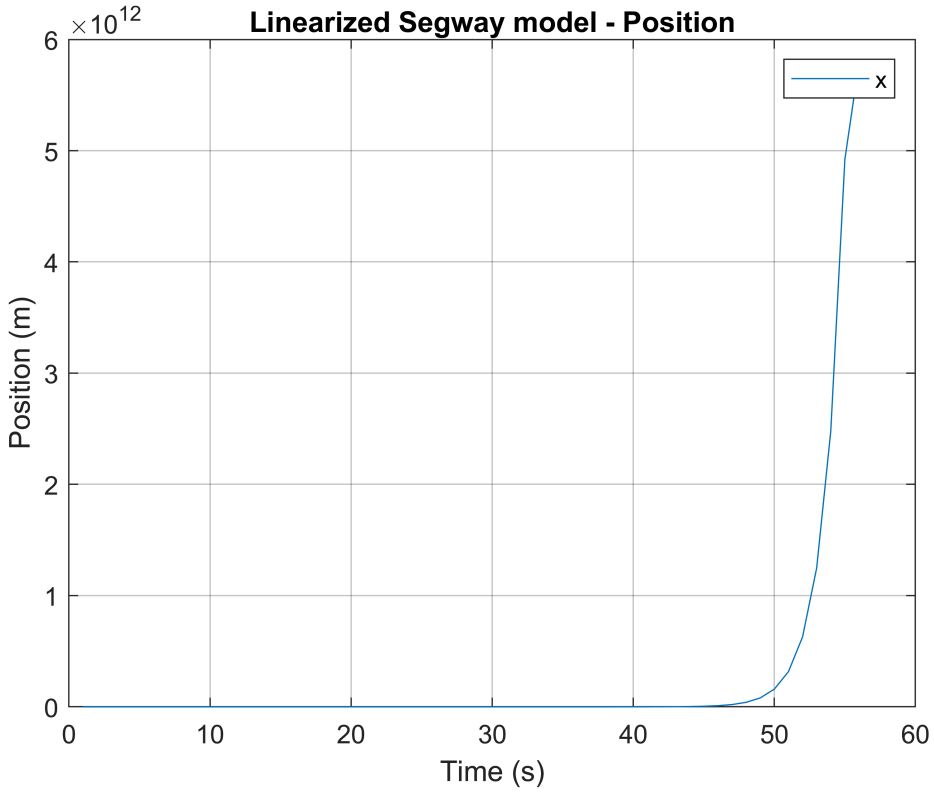
```
B = [0 0
     l/m 1/M
     0 0
     (m+M)/(m*M*l^2) 1/(M*l)]
```

```
B = 4×2
         0         0
    1.0000    0.2000
         0         0
    1.2000    0.2000
```

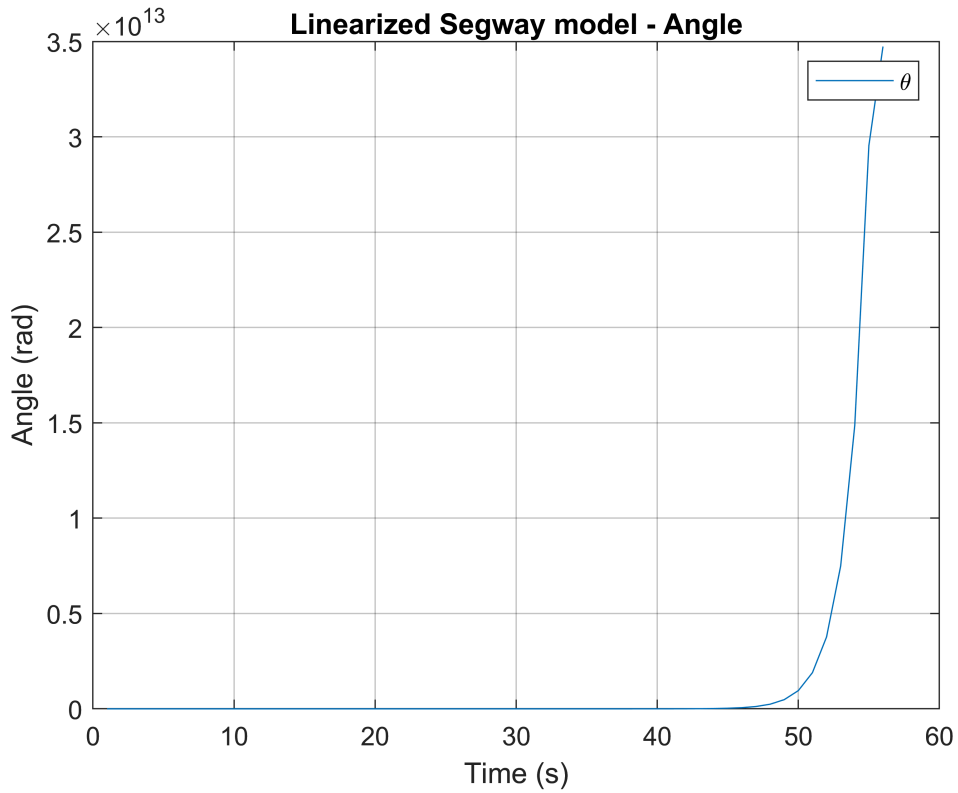
```
C = [1 0 0 0
     0 0 0 0
     0 0 1 0
```

```
     0 0 0 0];
D = [0 0
    0 0
    0 0
    0 0];

simuLin0 = sim("SegwayModelLinearized0");
figure()
plot(simuLin0.lin0values(:, 1)); % Plot x
legend('x');
xlabel("Time (s)")
ylabel("Position (m)")
grid on;
title("Linearized Segway model - Position")
```




```
figure()
plot(simuLin0.lin0values(:, 3)); % Plot theta
xlabel("Time (s)")
ylabel("Angle (rad)")
legend('{\theta}');
grid on;
title("Linearized Segway model - Angle")
```



Pole placement

```
CtrSys = ctrb(A, B)
```

```
CtrSys = 4×8
         0         0    1.0000    0.2000         0         0    2.3544    0.3924
    1.0000    0.2000         0         0    2.3544    0.3924         0         0
         0         0    1.2000    0.2000         0         0   14.1264    2.3544
    1.2000    0.2000         0         0   14.1264    2.3544         0         0
```

```
[n, m] = size(CtrSys)
```

```
n = 4
m = 8
```

```
if rank(CtrSys) == min(n, m)
    fprintf('Controllable')
end
```

```
Controllable
```

```
p = [-1, -2, -3, -4];
K = place(A, B, p)
```

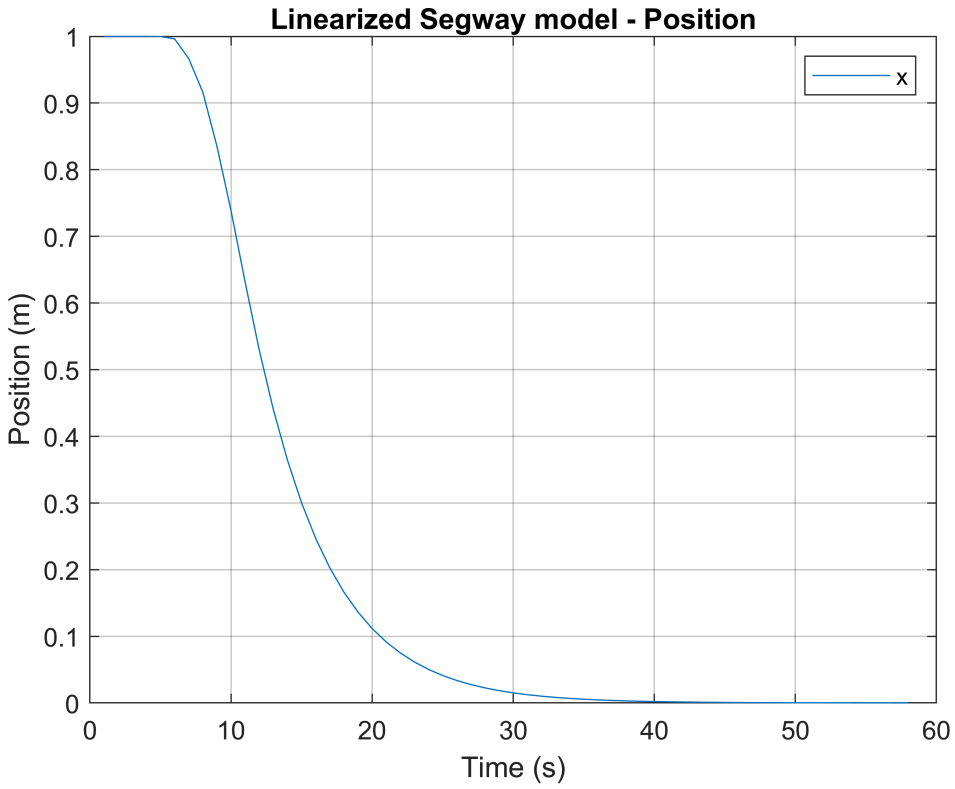
```
K = 2×4
  -14.4135  -19.7484   88.2390   29.6743
   87.0866  118.7500 -430.5934 -148.0540
```

```
Acl = A-B*K;
syscl = ss(Acl, B, C, D);
```

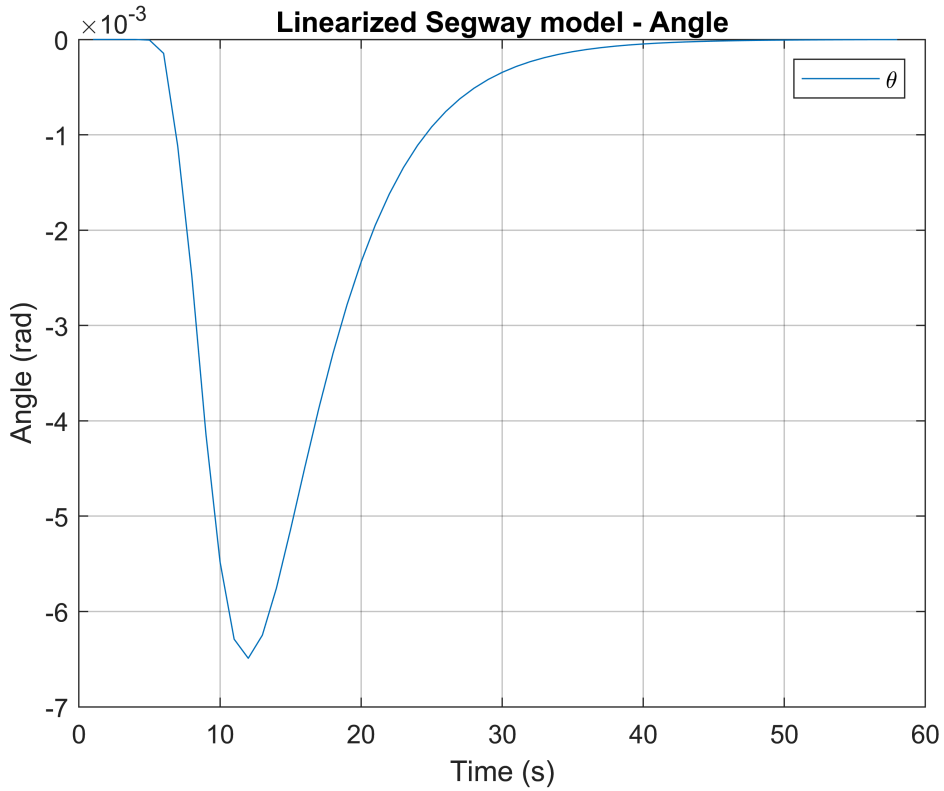
```
Kr = pinv(dcgain(syscl))
```

```
Kr = 2×4
  -14.4135         0   39.1890         0
   87.0866         0 -195.1534         0
```

```
simuLin0ctr = sim("SegwayLinearized0Controlled");
figure()
plot(simuLin0ctr.lin0ctr(:, 1)); % Plot x
legend('x');
xlabel("Time (s)")
ylabel("Position (m)")
grid on;
title("Linearized Segway model - Position")
```

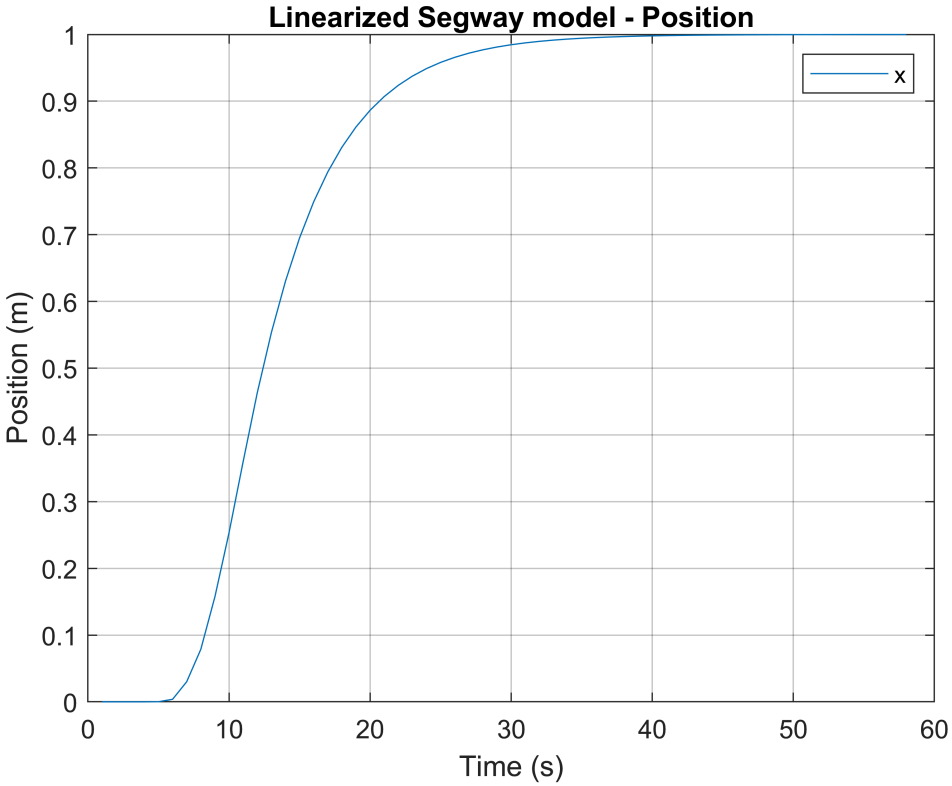



```
figure()
plot(simuLin0ctr.lin0ctr(:, 3)); % Plot theta
xlabel("Time (s)")
ylabel("Angle (rad)")
legend('{\theta}');
grid on;
title("Linearized Segway model - Angle")
```

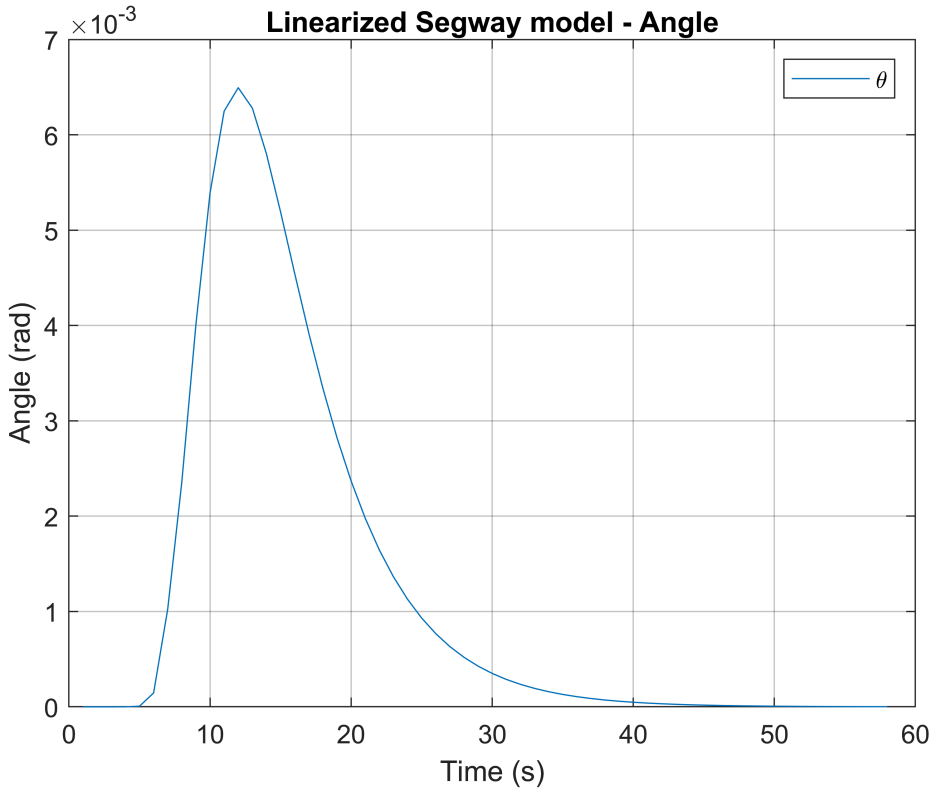


Pole placement - To another position

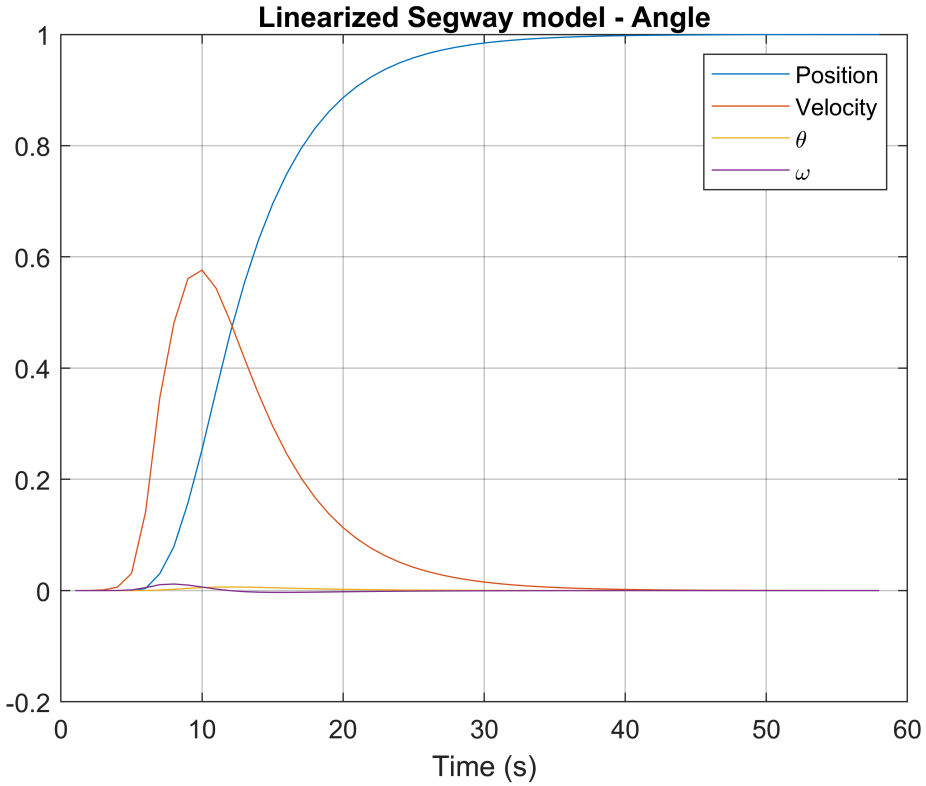
```
simuPos = sim("SegwayControlledNot0");
figure()
plot(simuPos.values(:, 1)); % Plot x
legend('x');
xlabel("Time (s)")
ylabel("Position (m)")
grid on;
title("Linearized Segway model - Position")
```




```
figure()
plot(simuPos.values(:, 3)); % Plot theta
xlabel("Time (s)")
ylabel("Angle (rad)")
legend('{\theta}');
grid on;
title("Linearized Segway model - Angle")
```




```
figure()
plot(simuPos.values(:, 1)); % Plot x
hold on;
plot(simuPos.values(:, 2)); % Plot v
plot(simuPos.values(:, 3)); % Plot theta
plot(simuPos.values(:, 4)); % Plot thetadot
hold off;
legend('{Position}', '{Velocity}','{\theta}',  '\omega')
xlabel("Time (s)")
grid on;
title("Linearized Segway model - Angle")
```



Pole placement - Constant speed

```
C = [0 0 0 0
    0 1 0 0
    0 0 0 0
     0 0 0 0];
A = linmod('SegwayModel')
```

```
A = struct with fields:
             a: [4×4 double]
             b: [4×0 double]
             c: [0×4 double]
             d: []
     StateName: {4×1 cell}
    OutputName: {0×1 cell}
     InputName: {0×1 cell}
     OperPoint: [1×1 struct]
            Ts: 0
```

```
A = A.a
```

```
A = 4×4
         0    1.0000         0         0
         0         0   12.7338         0
         0         0         0    1.0000
         0         0   21.7567         0
```

```
Acl = A-B*K;
syscl = ss(Acl, B, C, D);
Kr = pinv(dcgain(syscl))
```

```
Kr = 2×4
     0     0     0     0
     0     0     0     0
```

```
simuV = sim("SegwayControlledV");
figure()
x = plot(simuV.values(:, 1)); % Plot x
hold on;
xd = plot(simuV.values(:, 2)); % Plot x dot
th = plot(simuV.values(:, 3)); % Plot theta
thd = plot(simuV.values(:, 4)); % Plot omega
xlabel("Time (s)")
hold off;
legend('{Position}', '{Velocity}','{\theta}',  '\omega')
grid on;
title("Linearized Segway model")
```