12/9/2020

PHYSICS PROJECT

NS101

SAAD BIN KHALID - 20K-0161

M. MUDABIR – 20K-0273

1F – 1

"Swinging Atwood's Machine"

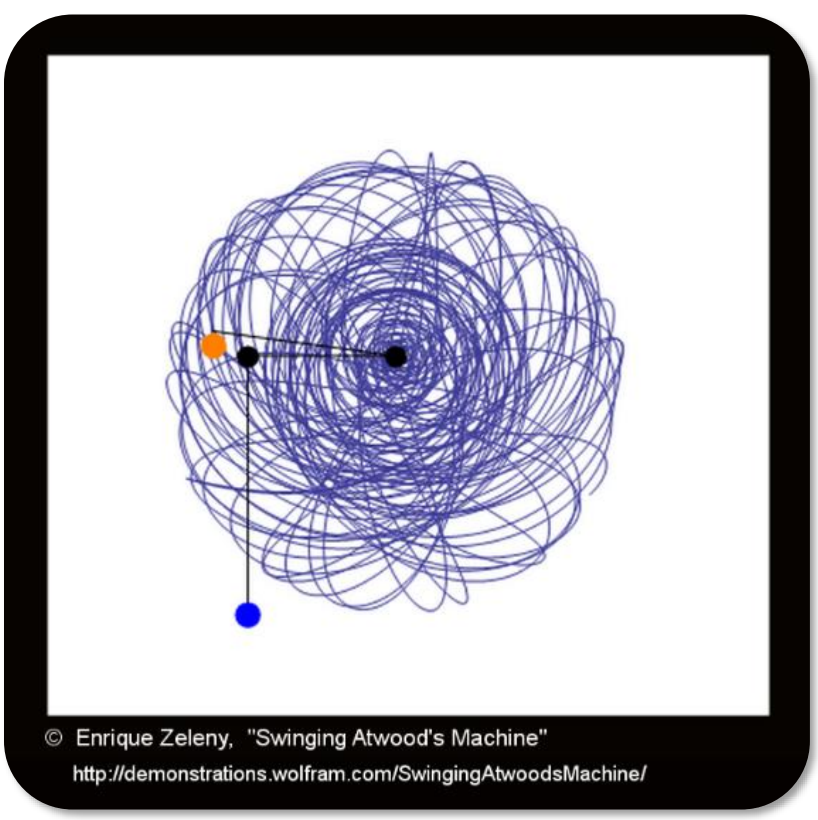
Problem Statement

Simulate a pulley-less Swinging Atwood's Machine (SAM) with varying values of mass ratio (μ) and release angle (θ). Use suitable initial values and visualization elements.

Solution

*SAM is a complex form of Atwood's Machine, having two degrees of freedom, in which two masses, **m** and **M**, are tied together with a string that passes over two massless & frictionless pulleys. When the lighter body, say **m**, is disturbed from its equilibrium position through an angle, it will perform perpetual and quasi-periodic oscillations given by the equations derived from the system's Langrangian while **M** will perform vertical non-uniform oscillations such that the string retains its original length.*

In simulation, motion will start from desired values of μ and θ, and the simulation will be terminated after a given interval of steps is completed.



Procedure

*We will use an online platform of **GlowSript 3.0 VPython** on* trinket.io *for simulation. Further elements and properties for this simulation are described below.*

ANIMATION:

*For animation we will employ **while loop** till the desired interval of steps is completed.*

OBJECTS IN SIMULATION:

⇒ *Two spheres of suitable different radii as our pendulum and counter-weight.*
- *Named as* 'pendulum' *and* 'block'.

⇒ *Three cylinders of suitable lengths as the three pats of our string.*
- *Named as* 'thread1', thread2', *and* 'thread3'.

⇒ *Two circles of relatively small radii as our **imaginary** pulleys.*
- *Named as* 'pulley' *and extruded twice.*

VARIABLES:

⇒ m : *mass of pendulum*
⇒ M : *mass of block.*
⇒ μ *: mass ratio*
⇒ L *: total length of string*
⇒ r *: initial length of pendulum*
⇒ d *: distance b/w pulleys*
⇒ R *: initial length of block*
⇒ θ *: release angle measured counter-clockwise from – y-axis*
⇒ v *: rate of change of 'r' w.r.t time*
⇒ a *: rate of change of 'v' w.r.t time*
⇒ ω *: angular velocity*
⇒ α *: angular acceleration*
⇒ t *: time at any instant*
⇒ T *: terminating time*

CONSTANTS:

- ⇒ g = *9.8 m/sec^2 : acceleration due to gravity on Earth*
- ⇒ dt = *0.0001 sec : interval of time for a single step*

INITIAL VALUES:

- ⇒ *Following values will be user-defined, but for this example, the chosen values are:*
  - μ = 4.5
  - L = 45 units
  - r = 10 units
  - d = 20 units
  - R = 15 units
  - θ = 90°
  - T = 100 sec
- ⇒ *Following values, starting from 0, will be updated inside the loop as per the equations:*
  - v = a = ω = α = t = *0*

Equations

Other than some fundamental equations, the used equations are:

$$a = \frac{r\omega^2 + g(\cos\theta - \mu)}{1 + \mu}$$

$$\alpha(t) = -\frac{2v\omega + g\sin\theta}{r}$$

Code Block

```
GlowScript 3.0 VPython
from vpython import *

# variables:
μ = 4.5
L = 45
r = 10
d = 20
R = L - d - r
θ = radians(90)
g = 9.8
v = 0
ω = 0
a = 0
α = 0
t = 0
dt = 0.0001
T = 100

# system objects:

## colors
pulley_color = vec(0.5, 0.2, 0)
thread_color = vec(0.7, 0.7, 0.7)

## labels
s = 'μ : ' + μ + '\nθ : '+ degrees(θ)
text(text = s, color = color.white, pos = vec(-20,10,0), height = 2, width = 2)

## physical:
thread1 = cylinder(pos = vec(0,0,0), axis = vec( r*sin(θ), -r*cos(θ),0), radius = 0.2, color = thread_color )
thread2 = cylinder(pos = vec(-d,0,0), axis = vec(0,R,0), radius = 0.2, color = thread_color )
thread3 = cylinder(pos = thread1.pos, axis = thread2.pos, radius = 0.2, color = thread_color )

pendulum = sphere(pos = thread1.axis, radius = 1, color = color.orange, make_trail = True,
trail_color = color.red)
block = sphere(pos = thread2.axis, radius = 2, color = color.green )

### pulleys:
pulley = shapes.circle(radius = 1)
extrusion( path = [ vec(0,0,0.3) , vec(0,0,-0.3)], shape = pulley , color = pulley_color )
axle1 = cylinder(pos = vec(0,0,0.3), axis = vec(cos(θ), sin(θ), 0), radius = 0.1)
extrusion( path = [ vec(-d,0,0.3) , vec(-d,0,-0.3)], shape = pulley , color= pulley_color )
axle2 = cylinder(pos = vec(-d,0,0.3), axis = vec(cos(θ),sin(θ),0), radius = 0.1)

while t <  T:
 rate(1 / dt)

# animating:
 X = pendulum.pos = thread1.axis = vec( r*sin(θ), -r*cos(θ),0)
```

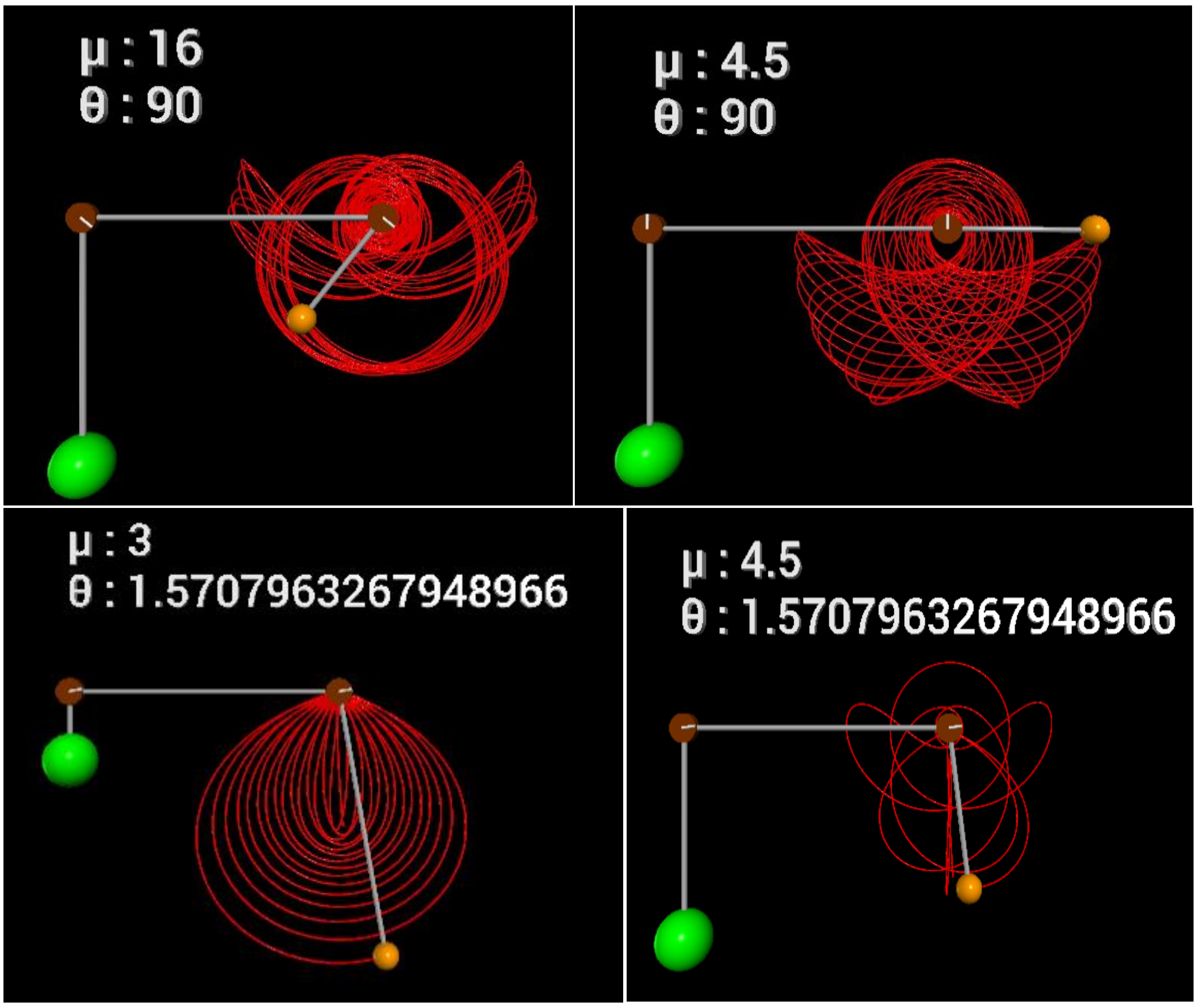
```
thread2.axis = vec( 0,-R,0)
block.pos = vec(-d,-R,0)
axle1.axis = vec(cos(θ), sin(θ), 0)
axle2.axis = vec(cos(θ), sin(θ), 0)

# updating:
l = mag(X)                                        # length of pendulum
a = ( l*(ω*ω) + g*(cos(θ) - μ ) ) / ( 1+μ )
α = -( 2*(v*ω) + g*sin(θ) ) / l
v += a*dt
ω += α*dt
θ += ω*dt
r += v*dt
R = L - d - r
t += dt

print("terminated")
```

Our simulation shows a smooth SAM with the help of trail function in vpython. [Click here for the link to simulation]

Conclusion

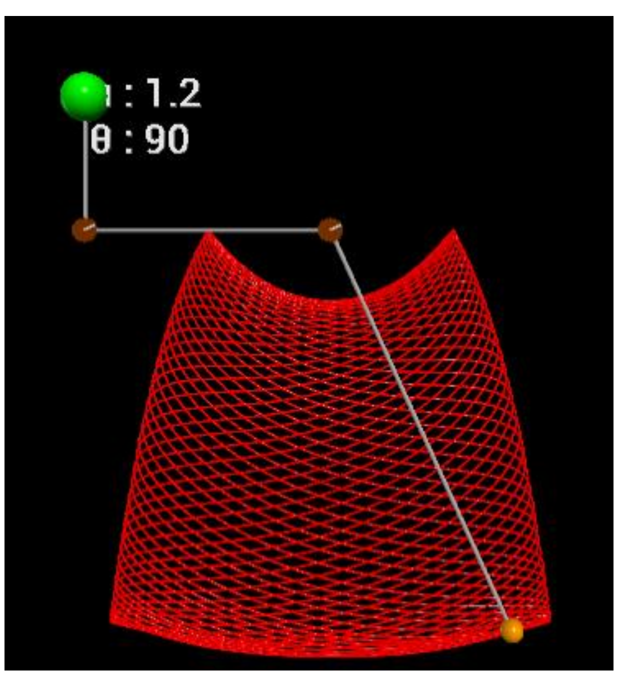
- *The Snapshots show a successful simulation of the said problem.*
- *To make the motion smooth, accurate and free of possible errors, we have used a smaller time step value for dt i.e. 0.0001. Even though a faster rate can sometimes result in a false depiction of the phenomenon, a slower rate can be infuriating and can take a very long time to generate an understandable design. Hence in some cases, faster rate is better than slower rates.*
- *We have also tried to simulate a realistic rotational motion of our imaginary pulleys, and the values of μ and θ are also displayed with the simulation.*

Snapshots:

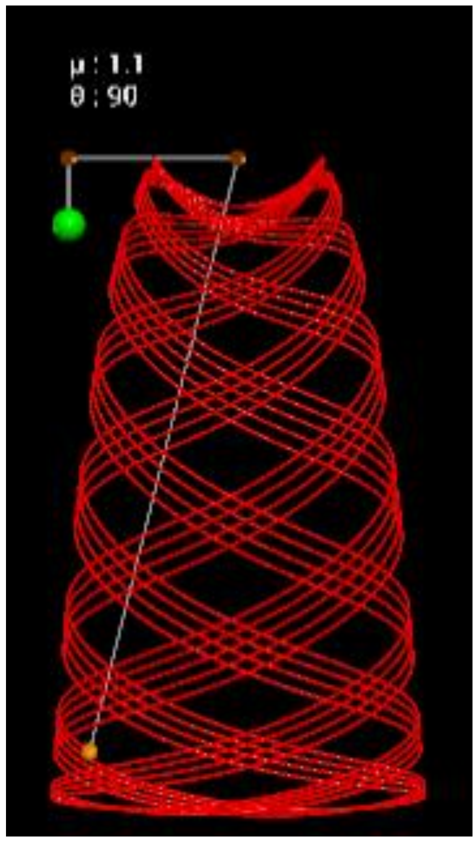


A Loose End:

It is worth to note an unusual loose end of our program: when values of μ *become close to 1,* R *starts to surpass its boundary i.e the value of* r *increases unexpectedly such that* r > L*. A snapshot of this problem is shown below:*



A simple solution to this issue is to increase the total length of string L before running the code at such values of μ, so that the value of R stays below the x axis. An example is shown below:



Citations

References

[1] Wikipedia : "Swinging Atwood's Machine".

[2] Research Gate : "Swinging Atwood machine: experimental and numerical results, and a theoretical study".