Komvuxarbete i fysik

# Interaktiv simulering av fysiska moment

Christoffer Lundell
2022/06/01

Handledare: Christer Svensson

## Sammanfattning

Den här rapporten redogör för information kring skapandet, användandet och innehåll av ett interaktivt simuleringsprogram som har utformats med hjälp av Godot. Det huvudsakliga syftet är att visa sambandet mellan fysiska formler och objekt i rörelse. Programmet kommer underlätta i förklaringar på fysiska fenomen genom att ge grafiska exempel på vanliga problem  i fysik på gymnasienivå. Källkod till projektet finns tillgängligt på sidan https://github.com/HoppenR/KA och färdigkompilerat program till Windows finns på https://github.com/HoppenR/KA2/releases/tag/v1.0.0 kallad KAv1.zip.

# Innehållsförteckning


1. Sammanfattning
2. Innehållsförteckning
3. Inledning
4. Teori
    a. Snett kast
    b. Fjäderrörelse
5. Metod och material
6. Resultat
7. Figurer och tabeller
8. Diskussion och slutsats
9. Källförteckning

# Inledning

## *Bakgrund*

Samband mellan fysiska formler och visuella instrument har alltid varit viktigt i förståelsen av fysikområdet och undervisningen därav.

## *Syfte*

Målet med det här arbetet är att använda olika grundläggande verktyg och fysisk teori för att sammanställa ett fungerande program som utför olika fyska simulationer. Samt att göra det möjligt att lättare utvidga programmets funktionalitet.

## *Teori*

I simulationerna Godot anropas funktioner kontinuerligt varje tidsintervall, vilket gör det lämpligt att använda stegmetoder som tar hänsyn till äldre värden under simulationens gång för att räkna ut de nya värdena.

### Snett kast

Utan luftmotstånd ändras hastigheten enligt följande, med ett varierande tidsintervall $\Delta u$ .
(Lägg märke till att variablernas index är sekventiella, vilket kan vara missvisande eftersom tiden mellan dem *inte* är konstant)

$v_{x0}=v_0\cdot\cos(\alpha)$ och $v_{y0}=v_0\cdot\sin(\alpha)$

$v_{y1}=v_0+\Delta t\cdot(-g)$
$v_{y2}=v_0+\Delta t\cdot(-g)$
…
$v_{y(n)}=v_{y(n-1)}+\Delta t\cdot(-g)$

Med luftmotstånd ändras hastigheten/accelerationen enligt följande, med tidsintervallet $\Delta u$ som också varierar.

$(t_0=0)$

$v_{x0}=v_0\cdot\cos(\alpha)$ $\quad$ $v_{y0}=v_0\cdot\sin(\alpha)$

$a_{x0}=\frac{-(k\cdot v_0)}{m}\cdot v_{x0}$ $\quad$ $a_{y0}=(-g)-\frac{k\cdot v_0}{m}\cdot v_{y0}$

$(t_1=t_0+\Delta u_0)$

$v_{x1}=v_{x0}\cdot a_{x0}\cdot\Delta t$ $\quad$ $v_{y1}=v_{y0}+a_{y0}\cdot\Delta t$

$v_1=\sqrt{v_{x1}^2\cdot v_{y1}^2}$

$a_{x1}=\frac{-(k\cdot v_1)}{m}\cdot v_{x1}$ $\quad$ $a_{y1}=(-g)-\frac{k\cdot v_1}{m}\cdot v_{y1}$

…

$(t_n=t_{n-1}+\Delta u_{n-1})$

$v_{x(n)} = v_{x(n-1)} + a_{x(n-1)} \cdot \Delta t$ $\quad$ $v_{y(n)} = v_{y(n-1)} + a_{y(n-1)} \cdot \Delta t$

$v_n = \sqrt{v_{x(n)} + v_{y(n)}}$

$a_{x(n)} = \frac{-(k \cdot v_n)}{m} \cdot v_{x(n)}$ $\quad$ $a_{y(n)} = (-g) - \frac{k \cdot v_n}{m} \cdot v_{y(n)}$

där $k$ är den följande konstanten.

$k = ½ \cdot C \cdot \rho \cdot A$ varav

$C$ utgör luftmotståndskoefficienten

$\rho$ utgör luftens densitet

$A$ är föremålets tvärsnittsarea

Vid varje steg ändras positionen med

$x_0 = 0 \quad y_0 = 0$

$x_n = x_{n-1} + v_{x(n-1)} \cdot \Delta t$

$y_n = y_{n-1} + v_{y(n-1)} \cdot \Delta t$

Motsvarigheten i programmeringsspråket GDScript i Godot blir följande

Utan luftmotstånd:

```
func _physics_process(delta):
	var a_y = -g
	velocity.y += a_y * delta
	translation += velocity * delta
```

Med luftmotstånd där $C$ för en sfär är en konstant $0{,}47$ ; $\rho$ har valts till att vara konstanten $\frac{1{,}2\,kg}{m^3}$ och tvärsnittsarean på bollen är $A = r^2 \cdot \pi = 0{,}5^2 \cdot \pi \approx 0{,}785\,m$ eftersom radien på bollen är 0,5m blir det följande motsvarigheten.

```
func _physics_process(delta):
	var k = 0.5 * 0.47 * 1.2 * 1 * pi
	var v = sqrt(pow(velocity.x, 2) + pow(velocity.y, 2))
	var m = $"Knappar/ViktGlidare".value
	var a_x = -(k * v) / m * velocity.x
	var a_y = -g - (k * v) / m * velocity.y
	velocity.x += a_x * delta
	velocity.y += a_y * delta
	translation += velocity * delta
```

## Fjäderrörelse

Totalkraften på fjädern ändras enligt följande med ett varierande tidsintervall $\Delta u$ .
(Lägg märke till att variablernas index är sekventiella, vilket kan vara missvisande eftersom tiden mellan dem *inte* är konstant)

$(t_0 = 0)$
$x_0 = 0$

$(t_1 = t_0 + \Delta u_0)$
$F_1 = -k \cdot x_0 + g \cdot m$
$a_1 = \frac{F_1}{m}$
$v_1 = a_1 \cdot \Delta u_0$
$x_1 = x_0 + v_1 \cdot \Delta u_0$

$(t_2 = t_1 + \Delta u_1)$
$F_2 = -k \cdot x_1 + g \cdot m$
$a_2 = \frac{F_2}{m}$
$v_2 = a_2 \cdot \Delta u_1$
$x_2 = x_1 + v_2 \cdot \Delta u_1$

…

$(t_n = t_{n-1} + \Delta u_{n-1})$
$F_n = -k \cdot x_{n-1} + g \cdot m$
$a_n = \frac{F_n}{m}$
$v_n = a_n \cdot \Delta u_{n-1}$
$x_n = x_{n-1} + v_n \cdot \Delta u_{n-1}$

Där k är fjäderkonstanten med enheten N/m, x är utsträckningslängden på fjädern ifrån dess jämviktsläge och m är dess konstanta massa. Minustecknet beror på att kraften från fjädern alltid är i motsatt riktning till dess utsträckning/ihopdragning från dess jämviktsläge.

I GDScript blir det motsvariga det följande.

```
onready var equilibrium = $"MeshInstance".get_aabb().size.y
func _physics_process(delta):
	var F = -k * (x-equilibrium) + g * m
	a = F / m
	v += a * delta
	x += v * delta
	$"MeshInstance".scale.y = x / equilibrium
```

### *Metod och material*

Godot

För arbetet behövdes en grafikmotor där själva fysiken gick att kontrollera själv. Godot var en bra kandidat eftersom den är en färdig spelmotor men det går att använda olika typer av objekt där den inbyggda bearbetningen av fysik inte används, för att sedan kunna implementera den själv.

Blender

För vissa simulationer behövdes det tillgång till modeller mer komplexa än vanliga geometriska figurer, där användes Blender för att utforma dessa modeller.

## *Resultat*

Den slutgiltiga produkten är ett minimalistiskt exempel på hur fysiska fenomen utspelas och kontrolleras med hjälp av kod och fysikformler som gör det lätt att koppla ihop dessa fenomen med en mer abstrakt fysikteori.

## *Figurer och tabeller*

[Flytta all kod ner till bilder hitåt?]

[Bilder på programmet?]

## *Diskussion och slutsats*

I längduträkningen i simulationen av ”Snett kast” utan luftmotstånd finns det ofta en felmarginal på mellan 0,02 och 0,06m, detta kan bero på flera olika saker.

För det första kan det bero på att i Godot beropas den användardefinierade`_physics_process()` funktionen i varje objekt enligt intervallen $\Delta t = \frac{1}{bilder/sekund}$ . Så när bildhastigheten ändras blir uträkningen odeterministisk eftersom positionen bara kan kontrolleras var $\Delta t$ sekund.

Samt att dessa reella tal avrundas i uträkningsrutan till att bara visa två decimaltal, vilket kan få felmarginalen att verka större än den är ifall de två talen avrundas åt olika håll.

Förutom vid extremt onormala bildhastigheter kommer bollen i simulationen ”Snett kast” utan luftmotstånd alltid landa ganska nära den föruträknade längden.

Eftersom alla variabler i fjäderrörelsens uträkning egentligen bara beror på fjäderns utsträckning, betyder det att denna resonans kommer att pågå i evighet. Detta är ganska olikt fjäderrörelse i verkliga livet, men är bra nog för att ge en ungefärlig bild över hur det fungerar på ett förenklat sätt.

### *Källförteckning*

https://docs.godotengine.org/en/stable/

Formelblad