Roller Coaster Simulering
TNM085 - Modelleringsprojekt

Amanda Tydén - amaty364
Jacob Nyman - jacny980
Johanna Granström - johgr565
Viktor Sandberg - viksa368
Emilie Ho - emiho502

Handledare: Anna Lombardi och Ulf Sannemo
Examinator: Anna Lombardi

17 februari 2020

Sammanfattning

I detta projektet har en enkel simulering av en berg- och dalbana skapats. En berg- och dalbana är ett slutet system med vagn som åker efter en bestämd bana. Tanken med projektet är att titta på olika typer av banor och simulera dessa för att se hur de fungerar i praktiken. För att kunna simulera en berg- och dalbana behövdes en differentialekvation över de fysikaliska parametrarna som verkar på vagnen. För att skapa differentialekvation började det med att titta på vilka krafter som verkade i ett lutande plan. Med hjälp av denna enkla modell av banan kunde en differentialekvation fås fram. Denna ekvation användes sedan med Eulers stegmetod för att kunna approximera rörelsen av vagnen längs med banan. Resultatet av projektet blev en simulering av berg- och dalbanan i Matlab samt en grafisk applikation i ramverket Unity. Applikationen i Unity har möjligheten att ändra krafterna som verkar på systemet samt vagnens massa. De slutsatser som kunnat dras efter projektet är att det var svårt att arbeta med kollisionshantering mot banan. Det resultat som skapats har därför undgått den biten. Det var mycket lärorikt att lära sig om processen för att skapa en simulering av ett verkligt system. Det projektgruppen framförallt lärt sig är att ett detaljerat system kan bli väldigt komplicerat i ett tidigt skede av projektet.

Innehåll

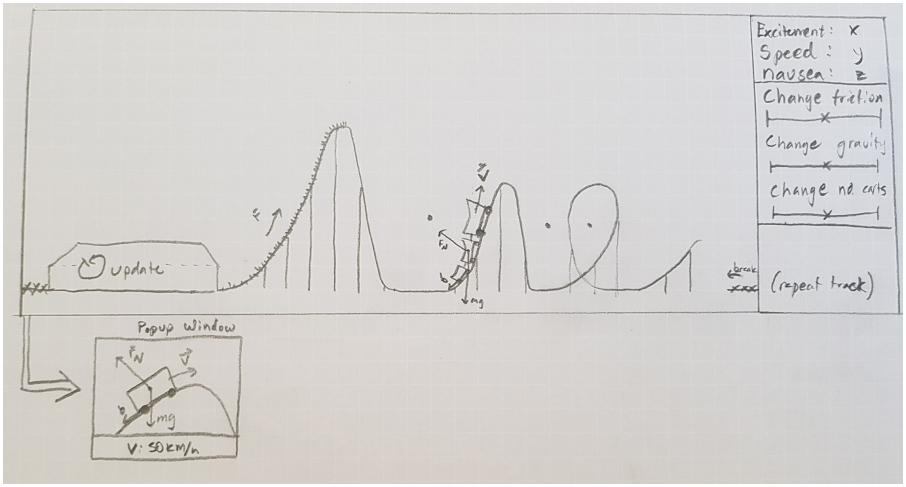
Figurer

1. Inledning

Detta projekt handlar om att göra en enkel simulering av en berg- och dalbana och de fysikaliska krafter som påverkar systemet. Att simulera olika system innan de konstrueras i verkligheten är ett essentiellt verktyg för att säkerställa stabilitet. Om beräkningar och simuleringar inte genomförs innan konstruktionen av ett system, är det lätt att förbise olika situationer och risker som kan inträffa. I detta projekt är det väsentligt att ta hänsyn till aspekter som bland annat vind, gravitationskraft och vagnens massa.

1.1 Syfte

Syftet med projektet är att skapa en grafisk simulering av en berg- och dalbana med fysiska egenskaper likt grundskissen i Figur 1.1. Detta ska genomföras med hjälp av beräkningar från *Matlab*.



Figur 1.1: *Skiss av grundidé*

2. Teori och bakgrund

En berg- och dalbana är ett slutet system där en vagn följer en bestämd kurva. Den initiala tanken var att simulera olika situationer som finns i verkliga berg- och dalbanor. Exempelvis olika nivåer på toppar och dalar, samt olika varianter på loopar. Med hjälp av dessa olika typer av banor kan sedan de nödvändiga parametrar om hur banan skulle fungera i praktiken fås ut. En berg- och dalbana får till exempel inte utsätta personer för för höga g-krafter [1]. Det som är intressant att undersöka är hur systemet påverkas av ändrad gravitation, vagnens massa och ändrad motstånd. Simuleringen kommer till en början genomföras i Matlab för att sedan implementeras i *Unity*.

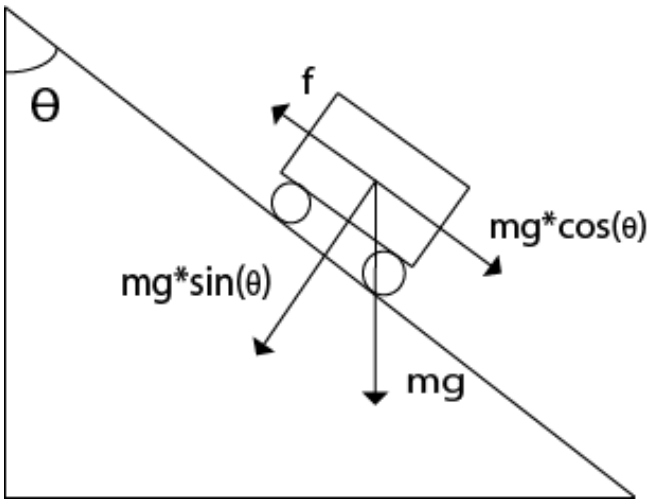
2.1 Objekt i ett lutande plan

För att underlätta framtagandet av en differentialekvation utfördes simuleringen först längs med ett lutande plan för den önskade kurvan som kan ses i Figur 1.1. Detta på grund av att lättare kunna se vilka krafter som påverkar vagnen och det ger enklare förhållanden. Newtons andra lag säger att kraften är lika med massan multiplicerat med accelerationen enligt 2.1. Detta kan skrivas om och ger ett uttryck för acceleration. [3]

$$F(t) = m \cdot a(t) \tag{2.1}$$

F = kraft, m = massa, a = acceleration

Kraftuppdelningen av gravitationskraften för ett lutande plan visas i Figur 2.1. Gravitationen delas upp i olika komponenter med hjälp av trigonometri av en rätvinklig triangel. [3]



Figur 2.1: *Kraftkomponentuppdelning för ett lutande plan*

Det är önskvärt att kunna beräkna lutningen för planet då projektet senare kommer utvecklas till en

kurva med olika vinklar i varje punkt. Lutningen på planet kan således beräknas med ekvationen i 2.2. Med hjälp av detta blir det möjligt att beräkna accelerationen för en bana som har varierande lutning [4].

$$\theta = arctan((y_1 - y_2)/(x_1 - x_2)) \tag{2.2}$$

Luftmotståndet är en kraft som påverkar objekt i rörelse. Ett objekt med stor area faller långsammare än ett med liten area då luftmotståndet verkar som en motriktad kraft till hastigheten. Exempel på detta är att se på en fjäder som faller. En fjäder har stor yta och liten massa vilket gör att den faller långsammare än exempelvis en metallkula som har liten area och stor massa. Det totala luftmotståndet beror därför på en luftmotståndskoefficient, luftens densitet, tvärsnittsarean samt hastigheten på objektet. Ekvationen för luftmotstånd kan ses i 2.3. [3]

$$F_d(t) = 1/2 \cdot C \cdot \rho \cdot A \cdot v(t)^2 \tag{2.3}$$

$F_d(t)$ = luftmotstånd, C = luftmotståndskoefficient, ρ = luftens densitet, A = tvärsnittsarea, $v(t)$ = hastighet

Sammanställs alla krafter för systemet kan en differentialekvation skapas enligt 2.4 och 2.5.

$$a_x = (g \cdot cos(\theta) \cdot sin(\theta)) - 1/2 \cdot C \cdot \rho \cdot A \cdot v_x(t) \cdot |v_x(t)| \tag{2.4}$$

$$a_y = (g \cdot cos(\theta) \cdot sin(\theta)) - 1/2 \cdot C \cdot \rho \cdot A \cdot v_y(t) \cdot |v_y(t)| \tag{2.5}$$

2.2 Eulers stegmetod

Eulers stegmetod är en metod för att lösa differentialekvationer med ett givet initialvärde. En approximation görs genom att beräkna nästa steg i kurvan utifrån tangentens riktning i den aktuella punkten. Eulers stegmetod används ofta för att göra enklare simuleringar men kan fortfarande göra approximeringar med god precision. Uttrycken för Eulers stegmetod ses i 2.6 och 2.7. [2]

$$y'(t) = f(t, y) \tag{2.6}$$

$$y(t + h) = y(t) + h \cdot f(t, y(t)) \tag{2.7}$$

I uttrycken motsvarar *h* steglängden, ju mindre steglängd desto noggrannare approximering [2].

För detta projekt kommer Euler att anpassas till variablerna som kan ses i 2.8 och 2.9 där $a(t)$ är accelerationen för vagnen och $v(t)$ hastigheten.

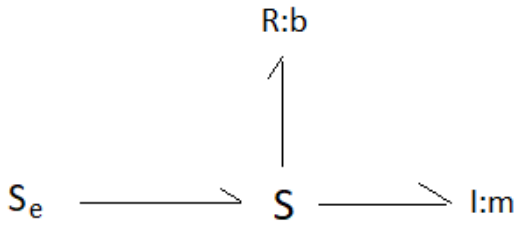
$$v'(t) = a(t) \tag{2.8}$$

$$v(t + h) = v(t) + h \cdot a(t) \tag{2.9}$$

För att kunna genomföra en simulering längs en bestämd bana så behövs ett matematiskt uttryck för banan. Bezierkurvor är matematiska representationer av kurvor och linjer vilket gör dessa passande för att bygga upp banan som ska simuleras. Varje linje definieras genom olika ankarpunkter samt linjerna som dras mellan punkterna [5]. På detta vis är det möjligt att beräkna vinkeln i varje punkt längs banan och utifrån det approximera acceleration och hastighet med hjälp av Eulers stegmetod.

3. Modellering och Implementation

Systemet simulerades i Matlab med hjälp av differentialekvationerna 2.4 och 2.5 i kombination med Eulers stegmetod 2.8 och 2.9. Bindningsgrafen för systemet kan ses i Figur 3.1.



Figur 3.1: *Bindningsgraf för en berg- och dalbana.* S_e *är källan är gravitationen,* m *är massan på vagnen och* b *är friktion i form av luftmotstånd [2].*

3.1 Implementation i Matlab

Initialt approximerades positionen för vagnen i Matlab. Detta var dock problematiskt då felet ökade exponentiellt vilket gjorde att vagnen kom längre och längre bort från banan. För att lösa detta approximerades endast hastigheten vilket användes för att plotta den fördefinierade banan. På detta vis hade vagnen alltid kontakt med banan och approximationen gav hastigheten för vagnen.

Det beslutades att berg- och dalbanan endast skulle simuleras i två dimensioner eftersom ytterligare dimensioner innebar fler beräkningar. För Euler-approximation så delades accelerationen upp i x- och y-led med hjälp av trigonometri för att särskilja de olika komponenterna. Hade en tredje dimension funnits hade det varit önskvärt med en z-komponent av accelerationen.

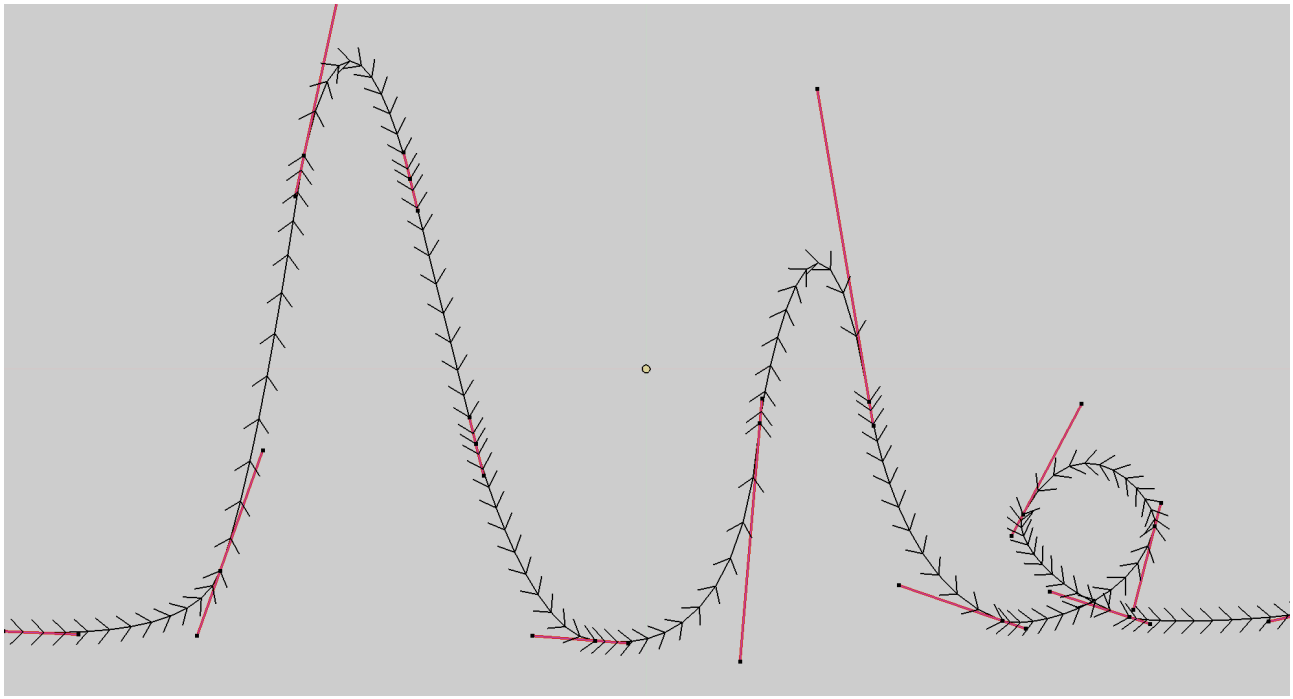
För att bestämma vagnens acceleration längs en kurva, studerades först fallet med ett lutande plan. Eftersom vagnen har hjul kommer rullfriktionen vara relativt liten, därför kan denna friktionskraft bortses ifrån. Den friktion som främst påverkar systemet är luftmotståndet. På grund av luftmotståndet är det viktigt att ta hänsyn till om hastigheten är negativ eller positiv i x- och y-led då luftmotståndet alltid är motriktad hastigheten. Accelerationen som studerades i Matlab hade endast yttre påverkan av gravitationen och luftmotståndet. När en differentialekvation för ett lutande plan utformats så vidareutvecklades den till att fungera för kurvor med varierande lutning. Det som återstod var att beräkna lutningen för varje punkt i kurvan, vilket kan göras genom ekvationen 2.2. Vinklarna för varje punkt i banan beräknades i förväg och det är dem som fås fram genom ekvationen 2.2 och är det som $alpha(i)$ i Figur 3.2.

```
% Acceleration framåt
ax(i) = (g*cos(alpha(i))*sin(alpha(i)))-((C*ro*A*(vx(i)^2)*sign(vx(i))/(2*m)));
ay(i) = (g*cos(alpha(i))*cos(alpha(i)))-((C*ro*A*(vy(i)^2)*sign(vy(i))/(2*m)));
%Hastighet framåt
vx(i+1) = vx(i)+h*ax(i);
vy(i+1) = vy(i)+h*ay(i);
```

Figur 3.2: *Euler-approximering av hastigheten i x- och y-led*

Den framtagna differentialekvationen (ekvation 2.4 & 2.5) som används i Euler-approximationen kan ses i Figur 3.2. Differentialekvationen består av accelerationen och med den som utgångspunkt approximeras hastigheten. Accelerationen beror således på vinkeln och hastigheten i varje punkt. Detta ledde till en variation av acceleration och hastighet längs med banan.

Vid implementation av en kurva till skillnad från ett lutande plan, användes Bezierkurvor för att rita upp banan. Bezierkurvor skapar ett matematiskt uttryck i form av koordinater för hela banan, alltså finns lutningen i alla punkter och alla positioner som vagnen kommer befinna sig på i ett uttryck. Med dessa punkter går det således att beräkna acceleration och hastighet med hjälp av den framtagna differentialekvationen. Genom att uttrycka banan med hjälp av Bezierkurvor kan de önskade formerna för banan enkelt beskrivas till skillnad från att uttrycka banan som en funktion. [5]

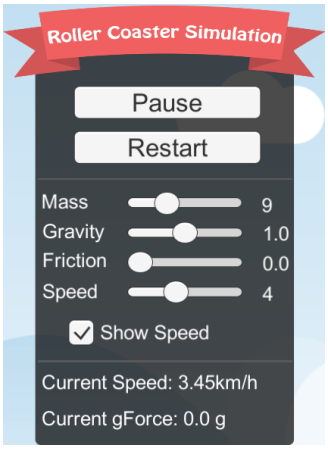


Figur 3.3: *Den slutgiltiga Bezierkurvan*

3.2 Implementation i Unity

Efter implementationen av differentialekvationen i Matlab användes ramverket Unity för att göra en grafiskt tilltalande version av systemet. I Unity utformades den önskade miljön och skapandet av en berg- och dalbana som vagnen skulle följa. Unity gav möjligheten att enkelt implementera fysik och realistiska beteenden på objekt utan att kräva implementation av komplicerade metoder. Det skapades även möjligheter för att ändra begynnnelsevärden så att användaren kan se hur systemet beter sig vid

andra inmatningar. Detta är samlat i ett användargränssnitt där massa, gravitation, luftmotstånd och kedjehastighet kan ändras. Syftet med att implementera ändringar av värden är för att göra det möjligt för användaren att göra egna simuleringar av systemet och se hur det beter sig vid olika inmatningar. Det implementerades även en funktion för att visa hastighetsvektorn för vagnen i färdriktningen. I Figur 3.4 visas användargränssnittet där reglagen ändrar på vagnens fysikaliska egenskaper. *Mass*-reglaget motsvarar vagnens massa, *Gravity* är en multiplikator mot den ursprungliga gravitationen. *Friction* motsvarar luftmotståndet samt *Speed* ändrar hastigheten på kedjan i början av banan.

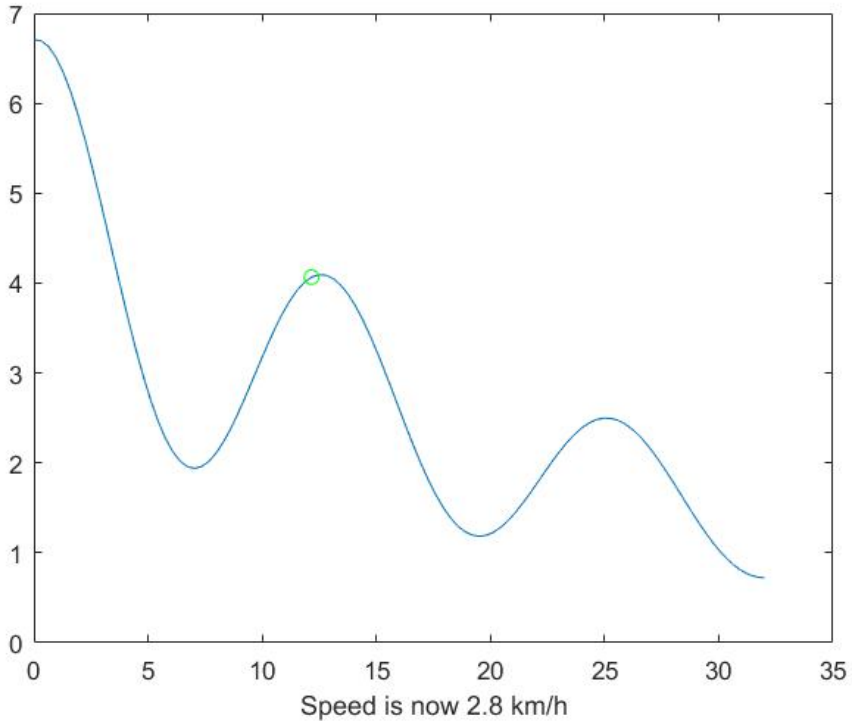


Figur 3.4: *Användargränssnitt på slutprodukten*

Modellering av vagn och bana gjordes i *Blender*, som är ett gratis program för 3D-modellering. Modellering av banan gjordes med hjälp av Bezierkurvor vilket ledde till att vagnen kunde åka jämnt över banan. 3D-modellerna exporterades från Blender i *.fbx*-format för att sedan lätt kunna importeras i Unity.

4. Resultat

Resultatet delas upp i två delar. Ett resultat från beräkningarna i Matlab och ett från den grafiska implementeringen i Unity. En stillbild från simulering i Matlab kan ses i Figur 4.1.



Figur 4.1: *Simulering i Matlab där vagnens position uppdateras med hastighetsapproximationen*

Slutresultatet av projektet som skapades i Unity kan ses i Figur 4.2 på nästa sida. Figuren visar hur användargränssnittet är utformat samt de parametrar som går att ändra i simuleringen för att få alternativa resultat.

Figur 4.2: *Slutprodukten i Unity*

5. Diskussion

Det är inte möjligt att skapa en helt perfekt simulering men bör eftersträvas, det uppkommer alltid oförutsägbara situationer, förutsättningar kan förändras, detta är något som är oundvikligt. Eulers stegmetod användes för att approximera nästkommande punkt på kurvan men eftersom Eulers metod inte är den noggrannaste kan den orsaka att viss osäkerhet. Det finns metoder som ger en större säkerhet än Euler som till exempel Runge-Kuttas metod [2]. För att få en ännu bättre säkerhet i resultat skulle en annan metod kunnat användas.

Ett stort problem som funnits under projektets gång har varit hur vagnen ska följa banan. Idén var att approximera positionen för vagnen med Eulers stegmetod i Matlab. Problemet som uppstod var att felet ökade exponentiellt vilket gjorde att positionen för vagnen hamnade längre och längre bort ifrån den tänkta banan. Lösningen för att kunna följa banan var att endast approximera hastigheten för vagnen och använda den för att styra uppspelningsfrekvensen.

Planen var att genomföra projektet i Matlab och sedan i JavaScript. Under processens gång uppkom det problem med den framtagna differentialekvationen i Matlab när den skulle implementeras i JavaScript. Därför användes en annan metod i Javascript som utgick från hur långt vagnen hade färdats på banan. Avstånden mellan första punkten och resterande punkter sparades i en lista. Listan jämfördes med den approximerade positionen för att hitta en punkt som låg på kurvan. För att jämföra dessa användes två olika funktioner. Den ena beräknade det euklidiska avståndet och den andra det närmsta värdet till den approximerade positionen utifrån listan med avstånd som skapats. Dessvärre kunde implementationen i JavaScript inte färdigställas på grund av problematik med kollisionshantering.

När JavaScript inte fungerade valdes det att implementera simuleringen i Unity. Genom att använda Unity kunde de problem som uppstod när systemet skulle implementeras i Javascript undvikas. Med användning av inbyggd kollisionsmotor och grafikimplementation gick det att skapa den önskade miljön. Det visade sig att Unity gjorde det enkelt att använda realistiska fysikaliska samband, men med en nackdel i form av att att sambanden inte enkelt gick att redigera eller skriva om. Det resulterade i att differentialekvationen inte gick att implementera som önskat och att bara små delar av den ursprungliga ekvationen kom med i slutresultatet i Unity.

Avgränsningar som gjorts för projektet är att endast arbeta i två dimensioner, alltså svänger banan varken till höger eller vänster. Dessutom har det valts att bortse från friktionen mellan banan och vagnen i differentialekvationen och endast räkna med gravitation och luftmotstånd. De faktorer som ansågs ha liten påverkan på systemet har medvetet valts bort då det inte är känts relevant att ta med i simuleringen.

När det gäller vidareutveckling av systemet finns det stor utvecklingspotential. En tredje dimension hade kunnat medföra nya funktionaliteter och nya kraftmoment att ta hänsyn till. Vid implementering av en tredje dimension hade banan exempelvis kunnat vara vinklad i sidled, vilket skulle påverka både viktfördelningen av vagnen samt dess hastighet. Banan hade även kunnat utvecklats vidare med till exempel olika material och utformning av räls. En mer avancerad bana kräver mer beräkningar och noggrannare simulering.

6. Slutsats

Simuleringen som skapats är en rimlig representation av en enklare variant av en berg- och dalbana där enbart gravitation och luftmotstånd har beräknats i två dimensioner. Den slutsats som kunnat dras efter skapandet av simuleringen är att kollisionshantering är svårt att genomföra utan en fysikmotor, speciellt på ytor som inte är plana. Detta var något som i ett tidigt skede av projektet inte verkade vara ett problem, men ju längre projektet pågick desto mer beroende blev utvecklingen av fungerande kollisioner. Den framtagna differentialekvationen i Matlab är rimlig då resultatet levde upp till förväntningarna av hur ett liknande system skulle bete sig i verkligheten. Den slutsats som kan dras är att ett system kan verka enkelt i ett tidigt skede när det diskuteras på en generell nivå, men kan bli en hel vetenskap om en detaljerad representation skapas.

Litteraturförteckning

[1] ABC News, *Can Fast Roller Coasters Cause Brain Injury?*, ABC News, 2017-07-01, hämtad: 2018-03-09
http://abcnews.go.com/GMA/story?id=125989&page=1

[2] Ljung L och Glad T. *Modellbygge och simulering*. Upplaga 2:5. Lund: Studentlitteratur; 2004.

[3] David Halliday, Robert Resnick, Jearl Walker, *Fundamentals of Physics*, Ninth Edition, International Student Version, John Wiley & Sons 2011

[4] Neumann E, *Roller Coaster*, myPhysicsLab, 2018-02-05, hämtad: 2018-03-08
https://www.myphysicslab.com/roller/roller-single-en.html

[5] Kantor I, *Bezier curve*, JavaScript.info, 2018, hämtad: 2018-03-09
https://javascript.info/bezier-curve