Conceptueel schaalmodel van een autonome Marsrover

Floris Kint Selwin Konijn Evert Leeuws
Urban Lemmens Jan-Uwe Lorent
Michiel Vanschoonbeek Vincent Vliegen Ruben Verhulst

12 mei 2014

Inhoudsopgave

1 Inleiding

Bij het verkennen van de ruimte en andere hemellichamen zijn er verschillende factoren die tot problemen kunnen leiden. De grote afstand is daar één van. Op een andere planeet zou een verkenningsrover hierdoor een vertraging van inkomende en uitgaande signalen ondervinden. Deze vertraging zou tot problemen kunnen leiden bij het besturen van de rover. Dit probleem kan opgelost worden door de rover voor een deel autonoom te maken. Deze kan dan zelf een veilige weg vinden en obstakels ontwijken.

De marsrover Curiosity is een voorbeeld van zo'n autonome rover. Deze bezit 17 camera's waarmee hij zijn omgeving kan aftasten. Aan de hand van de verzamelde informatie kan de Curiosity een veilig pad over het marsoppervlak vinden. Dit spaart tijd uit voor de bestuurders van de rover op aarde.[1, 2]

De specifieke opdracht bestaat uit het ontwerpen van een wagentje dat autonoom een vooraf onbekend parcours moet afleggen. Het parcours is maximaal 12 m lang, minimaal 40 cm breed en de afbakenende muren zijn 18 cm hoog. De af te leggen weg bestaat uit rechte hoeken en er zijn geen plaatsen waarin het wagentje vast kan komen te zitten (er zijn geen doodlopende zijwegen). Het wagentje moet het parcours zo snel mogelijk afleggen en bij het bereiken van de finish een visueel of auditief signaal geven. De totaalprijs van het ontwerp mag maximaal €80 bedragen.

Met behulp van een afstandssensor verzamelt de wagen informatie over de omgeving. De Arduino Uno-controller ontvangt de metingen. Daarna verwerkt deze de informatie en zendt dan weer aan de hand van computercode signalen uit naar de motoren en het stuurmechanisme zodat de wagen op de juiste baan blijft. Op deze manier kan de wagen autonoom een vooraf onbekend parcours afleggen, zoals de opdracht vereist.

In dit verslag zal in het eerste deel de definitieve conceptkeuze, met name het mechanische en elektronische ontwerp aan bod komen. Ook de materiaalkeuze voor de rover komt hier aan bod. De volgende sectie "Experimenten" bevat de ontwerpberekeningen en de uitgevoerde experimenten. Deze zijn ondersteund met informatie uit bijlages A en B. Daarna volgt een bespreking van de resultaten van de demonstratie. Als laatste wordt het behalen van de deadlines besproken. Doorheen het verslag zullen er enkele belangrijke bevindingen aan bod komen. De theoretische optimale overbrengingsverhouding kan bijvoorbeeld niet behaald worden. De luchtweerstand is pas merkbaar bij hoge snelheden en kan dus verwaarloosd worden bij het rijden op het parcours.

2 Conceptkeuze en ontwerp

2.1 Materialen

2.2 Mechanisch

2.3 Elektronisch

2.4 AI

De aansturende software op de Arduinocontroler laat de radarsensor afwisselend links, vooruit en rechts de afstand meten. Afhankelijk van de gemeten waarden

stuurt de software het wagentje bij. Van zodra voor de afstandssensor vooruit een muur opmerkt, stopt het wagentje. Afhankelijk van de gemeten waarden links en rechts, weet de software aan welke kant het volgende deel van het traject zich bevindt. Indien zich aan beide kanten een muur bevindt, is de rover op de bestemming aangekomen en knippert een LED. Wanneer de waarden links en rechts indiceren dat het wagentje zich tussen twee muren bevindt, maar dat de ene muur dichterbij is dan de andere, stuurt de software de baan van de wagen bij.

De software bestaat uit verschillende klassen die abstractie maken van de hardwarespecifieke commando's om de sensoren in te lezen en de actoren aan te sturen. Zo is er een MyServo-, LedOutput-, Motor-, DistanceSensor-, MusicPlayer en PushButtonklasse. Deze laatste twee werden niet gebruikt in het uiteindelijke ontwerp, maar het programmeren van de skeletten gebeurde reeds in een eerste fase van het project.

De DrivingManagerklasse is de centrale unit van waaruit de wagen aangestuurd wordt. Deze klasse neemt de beslissingen zoals hierboven beschreven. DrivingManager kan de objecten manipuleren door middel van duidelijke commando's, zonder details over de onderliggende commando's. De methode setMotorSpeed in de klasse Motor regelt zo bijvoorbeeld de snelheid van de motoren en de staat van de relais, zodat ook achteruit gereden kan worden.

Om het testen van de aparte sensoren en actoren te vergemakkelijken, staan er ook enkele demo's in de DrivingManager-klasse. Deze zijn bedoeld om de specifieke elementen te testen. De methode radarDemo roteert bijvoorbeeld de servo waarop de afstandssensor gemonteerd is en de ingelezen waarden van de afstandssensor worden dan (geconverteerd naar centimeter) doorgestuurd via de Serial Monitor. Op die manier kan dit onderdeel gemakkelijker onderzocht worden op problemen als het aangesloten is aan de Arduino IDE[1].

3 Experimenten

3.1 Rolweerstand

3.2 Luchtweerstand

3.3 Koppeloverdracht

3.4 Animatiefilmpje

De experimenteel bepaalde bewegingsvergelijking van het karretje bleek erg goed benaderd te kunnen worden door een lineaire vergelijking (zie '8.9 Bijlage 9: Animatiefilmpje'). Het animatiefilmpje simuleert de beweging van het karretje dan ook alsof het een constante snelheid heeft gedurende het volledige parcours.

De Maplefile bevindt zich in '8.9 Bijlage 9: Animatiefilmpje'. Met behulp van de voorgedefinieerde plot3d-functies (polygonplot3d), plot Maple het parcours en het wagentje in een opeenvolging van frames. Verschillende zelfgedefinieerde procedures structureren het plotten van een frame door het onder te verdelen in enkele deeltaken. Zo zijn er de procedures `plot_wagentje` en `plot_wall_part` die voor een gegeven positie (en rotatie voor het wagentje), de

[1]IDE: Integrated Development Environment (geïntegreerde ontwikkelomgeving).

nodige matrixbewerkingen uitvoeren en vervolgens het gevraagde object plotten. Het parcours is gedefinieerd als een opeenvolging van punten. De muren en het wagentje worden opengetrokken op basis van hun coordinaten in het vlak. De bovenkant van de muren en het karretje zijn loodrechte translaties van het grondvlak. Telkens worden de overeenkomstige zijden als rechthoek geplot, zodat het model gesloten is. Het midden tussen de achterwielen en de voorwielen bevindt zich altijd op de lijn van het parcours. Met behulp van eenvoudige goniometrie wordt het karretje gedurende het rijden geroteerd zodat altijd aan deze voorwaarde voldaan is en de constante snelheid over de lijn behouden blijft. Deze vorm van rotatie is slechts een model en komt niet overeen met de bochten die het werkelijke ontwerp maakt. De werkelijke rover stuurt enkel met de voorwielen en kan niet in rechte hoeken draaien, wat in het animatiefilmpje wel gebeurt.

4 Resultaten demonstratie

5 Teamefficiëntie en deadlines

6 Discussie

7 Besluit

test 2

8 Bijlagen

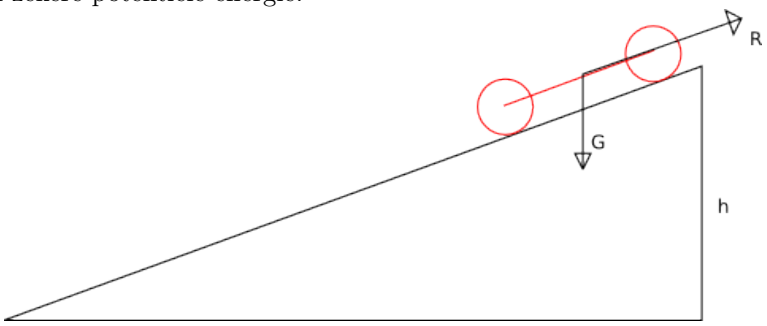
8.1 Bijlage 1: Lijst van gebruikte symbolen

m = massa
g = valversnelling
G = universele gravitatieconstante ($6.6754 * 10^{-11} \frac{m^3}{s^2 * kg}$)
v = snelheid
x = verplaatsing
a = versnelling
μ = rolweerstandscoëfficiënt
Cd = luchtweerstandscoëfficiënt
F_{motor} = voortdrijvende kracht door motor
F_{rol} = rolweerstand
F_{lucht} = luchtweerstand
θ = hoek
A = oppervlakte
ρ = dichtheid
η = overbrenging
ω = hoeksnelheid
R = straal
T = koppel van motor

8.2 Bijlage 2: Berekening rolweerstand

8.2.1 De rolweerstand op Aarde

Een belangrijke eigenschap van de Rover is de rolweerstand. Dit heeft een grote invloed in de efficiëntie van het rijden. Met een grote rolweerstand komt de wagen veel sneller tot stilstand dan bij een kleine rolweerstand. Het bepalen van de rolweerstand is niet moeilijk. Met behulp van een schuine plank is het mogelijk het verband te zoeken tussen de hoogte van waarop de wagen is losgelaten en de afstand die de wagen heeft afgelegd nadat het van de plank reed.
Wanneer de wagen boven op de plank staat, heeft het door de zwaartekracht een zekere potentiële energie.



$$E_{pot} = m * g * h \tag{1}$$

Wanneer de wagen naar beneden rolt, zet het die potentiële energie om in kinetische energie.

$$E_{kin} = \frac{m * v^2}{2} \tag{2}$$

Aan de onderkant van de helling is alles omgezet in kinetische energie. De snelheid v is nu te berekenen door de vorige 2 formules aan elkaar gelijk te stellen. Voor het berekenen van deze snelheid wordt de wrijving met de grond tijdens het rijden op de helling verwaarloosd. Dit is echter een redelijke veronderstelling doordat de wrijving zeer klein is ten opzichte van de zwaartekracht.
Terwijl de wagen verder rolt, verliest het deze energie door de wrijving met de grond. Uit de bewegingsvergelijking hieronder kan de versnelling a berekend worden. [2]

$$x = \frac{1}{2} * \left(\frac{-v}{a}\right)^2 + v * \left(\frac{-v}{a}\right) \tag{3}$$

Terwijl de wagen verder rolt, spelen er drie krachten op in. Dit zijn de zwaartekracht, de normaalkracht en de wrijvingskracht. Aangezien de wagen horizontaal rijdt, heeft alleen de wrijvingskracht invloed op de versnelling. Met behulp van het 2de Postulaat van Newton kan de rolweerstand bepaald worden.

$$F_{rol} = F_{res} = m * a \tag{4}$$

Ten slotte wordt de wrijvingscoëfficiënt μ bepaald met behulp van de definitie van de rolweerstand.

$$\mu = \frac{F_{rol}}{F_{normaal}} = \frac{F_{rol}}{m * g} \tag{5}$$

[2] Deze vergelijking werd afgeleid door team 208 in P&O opdracht 3

Om de onnauwkeurigheid van meetresultaten zo goed mogelijk te beperken, werd dit experiment zes keren herhaald. De definitieve wrijvingscoëfficiënt is het gemiddelde van de zes bekomen coëfficiënten. deze waarde bedraagt:

$$\mu = 0.067$$

8.2.2 De rolweerstand op Mars

Het vinden van de rolweerstand op Mars gebeurt op een zeer gelijkaardige manier als het vinden van de rolweerstand op Aarde. Het enige verschil is de gravitatieconstante g. Om de gravitatieconstante van Mars te berekenen, bestaat de volgende formule.

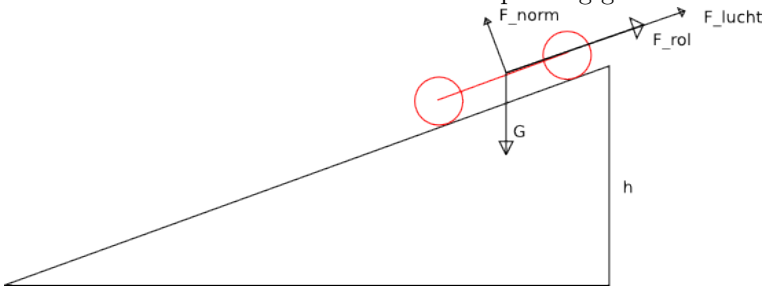
$$g = \frac{G * m_{Mars}}{(r_{Mars})^2} \tag{6}$$

Wanneer deze nieuwe gravitatieconstante ingevuld wordt in de vorige vergelijkingen, wordt de nieuwe rolweerstand

$$F_{rol} = 0.12N$$

8.3 Bijlage 3: Berekening luchtweerstand

Een ander belangrijke eigenschap van de rover is de luchtweerstand. Deze heeft maar weinig invloed bij lage snelheden, maar begint zeker mee te tellen zodra de snelheid stijgt. Het bepalen van de luchtweerstand gebeurt in twee stappen. Eerst wordt met behulp van een krachtenevenwicht een differentiaalvergelijking opgesteld en opgelost. Hiermee wordt de theoretische baan van de wagen berekend. Dit staat wel nog altijd in functie van een onbekende Cd, de luchtweerstandscoëfficiënt.
Vervolgens wordt met behulp van video-analyse de feitelijke baan opgemeten. De werkelijke baan en de theoretische baan worden met alkaar vergeleken. Ten slotte wordt er een kleinste-kwadratische oplossing gezocht in voor Cd.



8.3.1 Theoretische baan

Wanneer de rover naar beneden rolt, werken er vier krachten op in, namelijk de zwaartekracht, de normaalkracht, de rolweerstand en de luchtweerstand. Voor het opstellen van de bewegingsvergelijking wordt de x-as evenwijdig met de helling gelegd. Zo is de versnelling alleen maar in de x-richting en moeten alleen maar de x-componenten van de krachten beschouwd worden.

$$F_{rol} = -\mu * m * g * cos(\theta) \tag{7}$$

$$F_{lucht} = -\frac{1}{2} * \rho_{lucht} * A * Cd * \left(\frac{dx}{dt}\right)^2 \tag{8}$$

$$F_{zwaartekracht} = m * g * sin(\theta) \tag{9}$$

De bewegingsvergelijking wordt dus

$$F_{zwaartekracht} - Frol - F_{lucht} = m * \left(\frac{d^2x}{dt^2}\right)^2 \tag{10}$$

Deze vergelijking kan opgelost worden in functie van Cd.

8.3.2 Feitelijke baan

8.4 Bijlage 4: Berekening ideale overbrenging

Wanneer de rover in beweging is, werken er in totaal drie krachten op in. Deze krachten zijn de voortdrijvende kracht door de motor, de wrijvingskracht door de rolweerstand en de wrijvingskracht door de luchtweerstand. De bewegingsvergelijking is dus:

$$F_{motor} - F_{rol} - F_{lucht} = m * a \tag{11}$$

De wrijvingskracht door de rolweerstand wordt bepaald met behulp van de volgende vergelijking.

$$F_{rol} = \mu * m * g \tag{12}$$

De wrijvingskracht door de luchtweerstand wordt bepaald met behulp van deze vergelijking.

$$F_{lucht} = \frac{1}{2} * \rho_{lucht} * Cd * A * v^2 \tag{13}$$

De aandrijvende kracht van de motor wordt verkregen met behulp van de volgende formule.

$$F_{motor} = \frac{\eta}{R_{wiel}} * \frac{T_{max} - \left(T_{max} * \frac{dx}{dt}\right)}{\omega_{max} * R_{wiel}} \tag{14}$$

Door vergelijkingen 12, 13 en 14 in te vullen in vergelijking 11 kan de vergelijking volledig opgelost worden. Als beginvoorwaarden wordt de snelheid en de plaats van de wagen op tijdstip 0 gelijk gesteld aan 0. De oplossing van de vergelijking is een vergelijking met de tijd in functie van de onbekende η. Deze wordt geoptimaliseerd voor een minimale tijd om 2.5 meter af te leggen.
De optimale overbrengingsverhouding blijkt na numeriek in te vullen gelijk te zijn aan $\frac{1}{18}$

8.5 Bijlage 5: Technische tekening

8.6 Bijlage 6: Programmacode

8.7 Bijlage 7: Financiën

8.8 Bijlage 8: Ganttchart

8.9 Bijlage 9: Animatiefilmpje

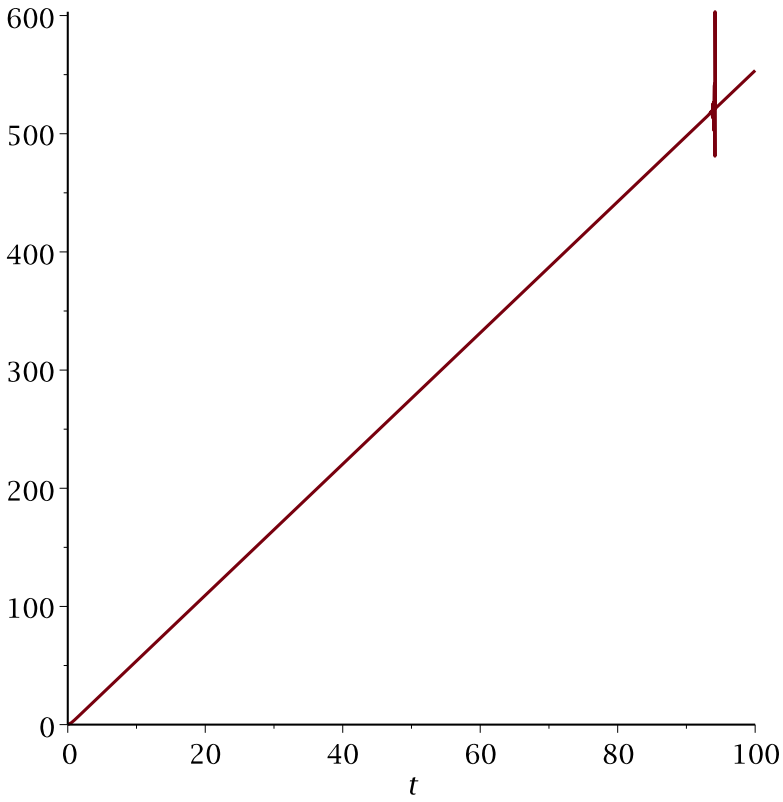
Bewegingsvergelijking

Maak van de bewegingsvergelijking een functie in t. (Experimenteel bepaald op basis van een eerste ontwerp).

```
> qr:=subs(overbr=18,x(t)):
> q:=t->-(151665298680333059 73/21550000000000000)*t-
  (1/258600000000000000)*t*sqrt
  (33648005365069128406930902587880596648976)-(266000/1293)*ln
  (13459202146027651362772361035152238659590 4/
  (181998358416399671676*exp((1/532000000000000000 0)*t*sqrt
  (33648005365069128406930902587880596648976))+sqrt
  (33648005365069128406930902587880596648976)*exp(
  (1/532000000000000000 0)*t*sqrt
  (33648005365069128406930902587880596648976))
  -181998358416399671676+sqrt
  (33648005365069128406930902587880596648976))^2):
> speed := evalf(diff(q(t), t));
```

$$speed := -1413.117884 + \frac{5.184275238\ 10^{23}\ e^{3.448006135\ t}}{3.654322848\ 10^{20}\ e^{3.448006135\ t} + 1.4355680\ 10^{18}} \qquad \textbf{(1.1)}$$

```
> plot(q(t), t = 0..100); #Quasi-lineaire bewegingsvergelijking
```



Animatiefilm

```
> restart:
> with(LinearAlgebra): with(VectorCalculus): with(plots): with(ArrayTools):
```

Animatie in 3D van het karretje dat een parcours volgt bestaande uit enkel rechte hoeken.

```
> translate_polygon := proc(polygon, translation)
    description "translate polygon";
    local result, i;
    result := LinearAlgebra[Copy](polygon):
    for i from 1 to Size(polygon, 1) do result[i] := polygon[i] + translation: end
       do;
    result;
    end proc:
> #De dimensies van het grondvlak van het karretje.
    front_width := 4:
    board_width := 7:
    board_length := 11:
```

```
board_height := 1:
front_length := 2:
board_matrix := Matrix([[-front_width/2, board_length/1, 0],
[front_width/2, board_length/1, 0],
[front_width/2, board_length/1 - front_length, 0],
[board_width/2, board_length/1 - front_length, 0],
[board_width/2, 0, 0],
[-board_width/2, 0, 0],
[-board_width/2, board_length/1 - front_length, 0],
[-front_width/2, board_length/1 - front_length, 0]]):
car_polygons := []:
#top_board: bovenkant
#bottom_board: onderkant
top_board := translate_polygon(board_matrix, Vector([0, 0, board_height/2],
    orientation = row)):
bottom_board := translate_polygon(board_matrix, Vector([0, 0,
    -board_height/2], orientation = row)):
car_polygons := [op(car_polygons), top_board]:
car_polygons := [op(car_polygons), bottom_board]:
#zijvlakken van het karretje (verbinding tussen boven- en ondervlak)
for i from 0 to Size(board_matrix, 1) - 1 do car_polygons
    := [op(car_polygons), Matrix([[top_board[i+1]], [top_board[(i+1)
    mod Size(board_matrix, 1) + 1]], [bottom_board[(i+1)
    mod Size(board_matrix, 1) + 1]], [bottom_board[i+1]]])]: end do:
> plot_wagentje :=proc(translation, rotation)
description "plot het wagentje in 3d, getransleerd + ter plaatse geroteerd";
local tmp_matrix, i, rotation_transposed, j, current_polygon;
rotation_transposed := rotation^+ :
tmp_matrix := []:
for j from 1 to Size(car_polygons, 2) do
  current_polygon := LinearAlgebra[Copy](car_polygons[j]);
  for i from 1 to Size(car_polygons[j], 1) do
  current_polygon[i] := VectorMatrixMultiply(car_polygons[j][i],
    rotation_transposed) + translation:
```

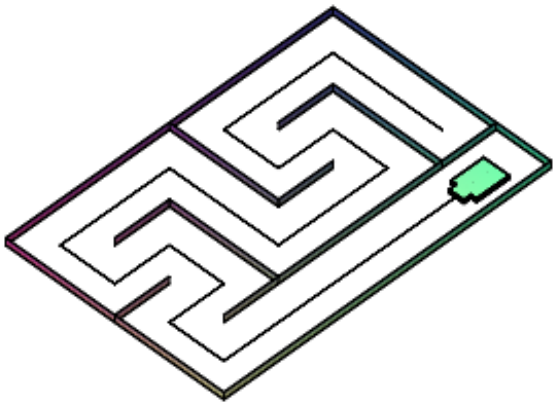
```
    end do;
    tmp_matrix := [op(tmp_matrix), current_polygon];
  end do;
  polygonplot3d(tmp_matrix, scaling = constrained, axes = none);
  end proc:
> plot_wagentje_angle := proc(translation, angle)

    description "plot het wagentje in 3d, gegeven een translatievector en de
    hoek rond de z-as waarrond het wagentje geroteerd wordt";
  local rotation_matrix, theta;
  theta := angle-Pi/2 :
  rotation_matrix := Matrix([[-cos(theta), -sin(theta), 0], [-sin(theta),
    cos(theta), 0], [0, 0, 1]]) :
  plot_wagentje(translation, rotation_matrix);
  end proc:
>
> unit := 15 :#relatieve dimensie van het het parcours
  plot_wall_part := proc(p1, p2)
  description "plot een stuk van een muur van het parcours"
  local translation, result :
  translation := Vector([0, 0, 2], orientation = row);
  result := Matrix(4, 3) :
  result[1] := p1 :
  result[2] := p2:
  result[3] := p2 + translation:
  result[4] := p1 + translation:
  polygonplot3d(result);
  end proc:
  parcours := [ ]:
  #rand van het parcours
  border := Matrix([[0, 0, 0], [6·unit, 0, 0], [6·unit, 4·unit, 0], [0, 4·unit, 0], [0,
    0, 0]]) :
  #binnenmuren
  inner1 := Matrix([[0, 3·unit, 0], [5·unit, 3·unit, 0]]) :
  inner2 := Matrix([[unit, 3·unit, 0], [unit, unit, 0], [2·unit, unit, 0]]) :
  inner3 := Matrix([[2· unit, 2· unit, 0], [3·unit, 2· unit, 0], [3·unit, 0, 0]]) :
  inner4 := Matrix([[4·unit, 3·unit, 0], [4·unit, unit, 0], [5·unit, unit, 0]]) :
  inner5 := Matrix([[5·unit, 2·unit, 0], [6·unit, 2·unit, 0]]) :
  parcours := [border, inner1, inner2, inner3, inner4, inner5]:

  plot_wall := proc( )
  description "plot alle muren van het parcours"
  local i, all_parts, j;
  all_parts := [ ]:
  for j from 1 to Size(parcours, 2) do
  for i from 1 to Size(parcours[j], 1) - 1 do
  all_parts := [op(all_parts), plot_wall_part(parcours[j][i ], parcours[j][i
```

```
        +1])]:
    end do:
   end do:
  return all_parts:
   end proc:
   wall_plot := plot_wall( ):
> #de checkpoints van het pad dat het wagentje moet afleggen
   path := [[unit/2, 7/2 unit, 0], [11/2 unit, 7/2 unit, 0], [11/2 unit, 5/2 unit, 0], [9/2 unit,
       5/2 unit, 0], [9/2 unit, 3/2 unit, 0], [11/2 unit, 3/2 unit, 0], [11/2 unit, 1/2 unit, 0],
       [7/2 unit, 1/2 unit, 0], [7/2 unit, 5/2 unit, 0], [3/2 unit, 5/2 unit, 0], [3/2 unit, 3/2 unit, 0],
       [5/2 unit, 3/2 unit, 0], [5/2 unit, 1/2 unit, 0], [1/2 unit, 1/2 unit, 0], [1/2 unit, 5/2 unit
       - board_length, 0], [1/2 unit, 5/2 unit, 0]]:
> path_plot := pointplot3d(path, connect = true, color = black):
> speed := t→1:#Quasi-lineaire bewegingsvergelijking (zie bovenaan de file)
> current_step := 1:
   current_position := Vector(path[1], orientation = row):
   displays := [ ]:
   distance_left := 0:
   current_speed := 0:
   current_rotation := 0:
   #het wagentje beweegt zich aan constante snelheid

      #het midden tussen de achterwielen en het midden tussen de voorwielen
      bevinden zich altijd op de lijnen van het pad

      #in de bochten roteert het wagentje zich zo dat aan bovenstaande
      voorwaarde voldaan blijft
   while current_step < nops(path) do
    if distance_left = 0 then
     displays := [op(displays), display([path_plot, op(wall_plot),
      plot_wagentje_angle(current_position, current_rotation)], scaling
      = constrained)]:
     distance_left := speed(current_speed):
     current_speed := distance_left:
    else
     distance_to_go := Student[Precalculus][Distance](Vector(path[current_step
      +1], orientation = row), current_position):
     if distance_to_go < distance_left then
      #in the next part
      current_step := current_step + 1:
      current_rotation := arctan(path[current_step + 1][2]
      - path[current_step][2], path[current_step + 1][1]
```

```
      - path[current_step][1]):
      distance_left := distance_left - distance_to_go:
      current_position := Vector(path[current_step], orientation = row):
    else
      current_position := current_position + (distance_left
     *(Vector(path[current_step + 1] - path[current_step], orientation = row)))
     /(Norm(Vector(path[current_step + 1] - path[current_step], orientation
      = row))):

      if distance_to_go > distance_left + board_length then
        #still straight in this part

      else
        #at the corner
        back_corner_distance := distance_to_go - distance_left:
        next_angle := CrossProduct(Vector(path[current_step + 1]
      - path[current_step]), Vector(path[current_step + 2]
      - path[current_step]))[3]:
        d_angle := arccos(back_corner_distance / board_length) * signum(next_angle):
        current_rotation := arctan(path[current_step + 1][2]
      - path[current_step][2], path[current_step + 1][1]
      - path[current_step][1]) + d_angle;
      end if;
      distance_left := 0:

    end if:
  end if:
end do:
```

Error, invalid subscript selector

```
> display(displays, insequence = true);
```



8.10 Bijlage 10: Poster

9 Referenties

[1] NASA. (3333) Nasa curiosity. Raadpleging: 2014-04-29. [Online]. Available: http://mars.jpl.nasa.gov/msl/multimedia/interactives/learncuriosity/index-2.html

[2] ——. (2222) Nasa curiosity news. Raadpleging: 2014-04-29. [Online]. Available: http://www.jpl.nasa.gov/news/news.php?release=2013-259