Conceptueel schaalmodel van een autonome Marsrover

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

13 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 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

De lijst met productvereisten hielp bij het vinden van het beste idee. Het ontwerp dat het best voldeed aan de vereisten, een vierwieler met een stuurmechanisme gelijkaardig aan dat van een auto, kreeg verdere uitwerking. Hierna volgt een besluit over de combinatie van sensoren waarmee de wagen uitgerust is om de omgeving zo goed mogelijk te kunnen aftasten. Rekening houdend met de prijs bleek een afstandssensor, gemonteerd op een servomotor het meest voordelig. Verder is er een elektromotor om de wagen aan te drijven en een servomotor om het stuurmechanisme aan te drijven.

Een preciezere beschrijving van het mechanisch en elektronisch gedeelte volgt

in de volgende paragrafen.

2.1 Materialen

In het bouwen van de wagen zijn verschillende materialen en onderdelen gebruikt. De belangrijkste zijn hieronder terug te vinden.

De bodemplaat van de wagen is van MDF hout en uitgesneden met behulp van een lasercutter. Het sturingsmechanisme is gebouwd uit Lego Technics en de wielen hebben een doorsnede van 60 mm. Voor de overbrenging zijn er zes tandwielen van 26 mm en twee kleinere tandwielen van 5,2 mm doorsneden. Een MM28 motor zorgt ervoor dat de wagen kan rijden.

De wagen verzamelt zijn informatie met behulp van een IR-afstandssensor en een Arduino Uno-controller verwerkt deze informatie dan. Twee servomotors bewegen het sturingsmechanisme en de IR-sensor. Het geheel is verbonden met behulp van kabels die op een Printed Circuit Board gesoldeerd zijn. Een battery pack levert stroom aan de elektrische componenten.

De volledige materiaallijst is opgenomen in Bijlage 8.10.

2.2 Mechanisch

2.3 Elektronisch

De aansturing van de Mars Rover gebeurt via een Arduino Uno-controller. Deze microcontroller kan een analoog of digitaal signaal versturen naar het elektronisch gedeelte van het wagentje. Voor het realiseren van een autonome rover staan verschillende sensoren en actuatoren op de wagen gemonteerd die onderling met elkaar communiceren via de controller.

Het meten van de omgeving gebeurt via een optische afstandssensor, gemonteerd op een servomotor. Die heeft een bereik van 0° tot 180° en richt de afstandssensor beurtelings vooruit en zijdelings. Op die manier kan de omgeving in drie richtingen in kaart gebracht worden. De configuratie van een servomotor met een afstandssensor is goedkoper dan drie afstandssensoren die in de te verkennen richtingen gefixeerd worden, maar heeft als nadeel dat de afstand tot obstakels niet continu in elke richting bepaald kan worden.

De afstandssensor bestaat uit een infraroodzender en een lichtsensor. De sensor meet het tijdsverschil tussen het verzenden van het licht en het moment wanneer de lichtsensor een gereflecteerd signaal terug ontvangt en bepaalt zo de afstand tot een obstakel.[3] De keuze viel op deze sensor omdat die de meest bruikbare informatie geeft van alle sensors. Met een lichtsensor is het lastig werken zonder lichtbronnen en als er gewerkt wordt met geleiding (de aluminiumstrip) bestaat de output enkel uit 'ja of neen', zonder verdere informatie over de plaats van de rover in zijn omgeving.

Aan het einde van het parcours zal de afstandssensor zowel vooraan, links als rechts het signaal krijgen dat er een obstakel in de weg staat. Dan zal er een LED-licht knipperen.

Om de rover aan te drijven wordt een MM28 DC-motor gebruikt. Het inbrengen van een elektronische schakelaar, namelijk een relais, realiseert de ompoling van de motor. Zo is de rover in staat om in beide richtingen te kunnen rijden. Ook de snelheid kan gevarieerd worden door het gebruik van een MOSFET-transistor zodat de rover in de bochten zijn snelheid kan vertragen.

Voor de besturing van de rover wordt één servomotor gebruikt die aangesloten is op een stuurmechanisme. Aan de hand van een stuursignaal, gebaseerd op pulsbreedtes, kan de servomotor in de gewenste hoek geplaatst worden.

Aangezien er geen grote stromen door de Arduino-controller mogen lopen (maximaal grootteorde van enkele tientallen mA), moeten de signalen en de eigenlijke stroomtoevoer voor de motoren gesplitst worden. Daarom wordt gebruik gemaakt van een MOSFET-transistor. Dit soort transistor sluit de stroomkring met de batterij en de motoren van zodra de drempelspanning overschreden wordt. Deze drempelspanning wordt verkregen tussen gate- en sourcepin van de MOSFET.[4]

2.4 AI

De aansturende software op de Arduinocontroller 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 er voor het wagentje geen muur staat en 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[1]. Op die manier kan dit onderdeel gemakkelijker onderzocht worden op problemen als het aangesloten is aan de Arduino IDE[2].

De volledige code bevindt zich in Bijlage 8.9.

[1] Interface om data naar de aangesloten computer te versturen.

[2] Integrated Development Environment (geïntegreerde ontwikkelomgeving).

3 Experimenten

3.1 Rolweerstand

Het kennen van de rolweerstand is belangrijk bij het berekenen van de snelheid van de wagen en voor het verlies van elektrische energie in warmtevorming.

De rolweerstand wordt berekend door de omzetting van potentiële energie naar kinetische energie te bestuderen. Met behulp van de wet van behoud van energie kan de verloren energie ten gevolge van de wrijving berekend worden aan de hand van het tweede postulaat van Newton. Verdere uitleg en uitgebreide berekeningen staan in Bijlage 8.2.

Uit experimenten blijkt dat de gebruikte wielen een wrijvingscoëfficiënt van 0.06 hebben.

3.2 Luchtweerstand

3.3 Koppeloverdracht

De ideale koppeloverdracht moet berekend worden om te leren hoe de elektromotoren zo efficiënt mogelijk gebruikt kunnen worden. Een slecht gekozen overdrachtsverhouding zou leiden tot energieverspilling. Verdere uitleg en uitgebreide berekeningen staan in Bijlage 8.4.

De optimale overbrenging blijkt 1/18 te bedragen. Deze is echter in praktijk niet realiseerbaar, omdat de achterwielen van de wagen dan weinig grip zouden hebben op de ondergrond. Daarom wordt gewerkt met een overbrenging van 1/125.

3.4 Snelheidsmeting

3.5 Animatiefilmpje

Een lineaire vergelijking bleek een erg goede benadering voor de bewegingsvergelijking van het wagentje (zie Bijlage 8.6). 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 Bijlage 8.6. 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 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. De respectievelijke middens tussen de achterwielen en de voorwielen bevinden 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 Lijst van gebruikte symbolen

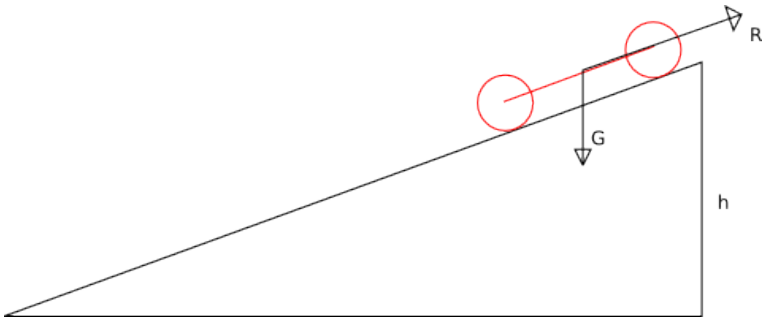
m = massa
g = valversnelling
G = universele gravitatieconstante ($6.6754 * 10^{-11} \frac{m^3}{s^2 * kg}$)
v = snelheid
x = verplaatsing
a = versnelling
μ = rolweerstandscoëfficiënt
Cd = luchtweerstandscoëfficiënt
F_{motor} = voortdrijvende kracht door motor
F_{rol} = rolweerstand
F_{lucht} = luchtweerstand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

8.2 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.
Het vrij-lichaamsdiagram van de situatie bevindt zich in Figuur 1.



Figuur 1: Vrij-lichaamsdiagram rolweerstand

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. [3]

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

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.

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

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:

$$\mu = 0.067$$

[3]Deze vergelijking werd afgeleid door team 419 in P&O 1 opdracht 3

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.

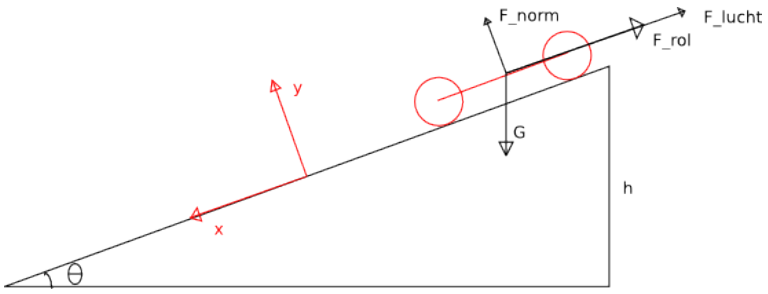
$$g = \frac{G * m_{Mars}}{(r_{Mars})^2} \tag{6}$$

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

$$F_{rol} = 0.12N$$

8.3 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 (zie Figuur 2) 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.



Figuur 2: Vrij-lichaamsdiagram luchtweerstand

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 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 Afstandsmeter als snelheidsmeter

In combinatie met een apparaat dat het verschil in golflengte meet, zou de afstandsmeter gebruikt kunnen worden als snelheidsmeter. Dit kan door het Doppler effect. De voorwaartse verplaatsing van de wagen zal er immers voor zorgen dat de golflengte korter wordt. Door deze verandering van golflengte kan de snelheid berekend worden.

De snelheid van elektromagnetische golven is gegeven, namelijk $299792458\frac{m}{s}$. De golflengte van de afstandssensor is ook gegeven.

$$\lambda = \frac{780}{10^{19}}$$

De frequentie van de afstandsmeter is de lichtsnelheid gedeeld door de golflengte.

$$f' = \frac{f}{1 - \frac{v_{source}}{c}} \tag{15}$$

De uitgezonden golf ondervindt al de effecten van het Doppler effect en dus wordt de nieuwe frequentie gegeven door de volgende formule.

$$f' = \frac{f}{1 - \frac{v_{source}}{c}} \tag{16}$$

De golf die de wagen bereikt is terug beïnvloed door het Doppler effect.

$$f_{percieved} = \left(1 + \frac{v_{source}}{c}\right) * f' \tag{17}$$

Ten slotte kan met de volgende formule de snelheid van de wagen berekend worden.

$$v_{source} = \frac{\left(\frac{f_{percieved}}{f}\right) * c - c}{1 + \frac{f_{percieved}}{f}} \tag{18}$$

8.6 Animatiefilmpje

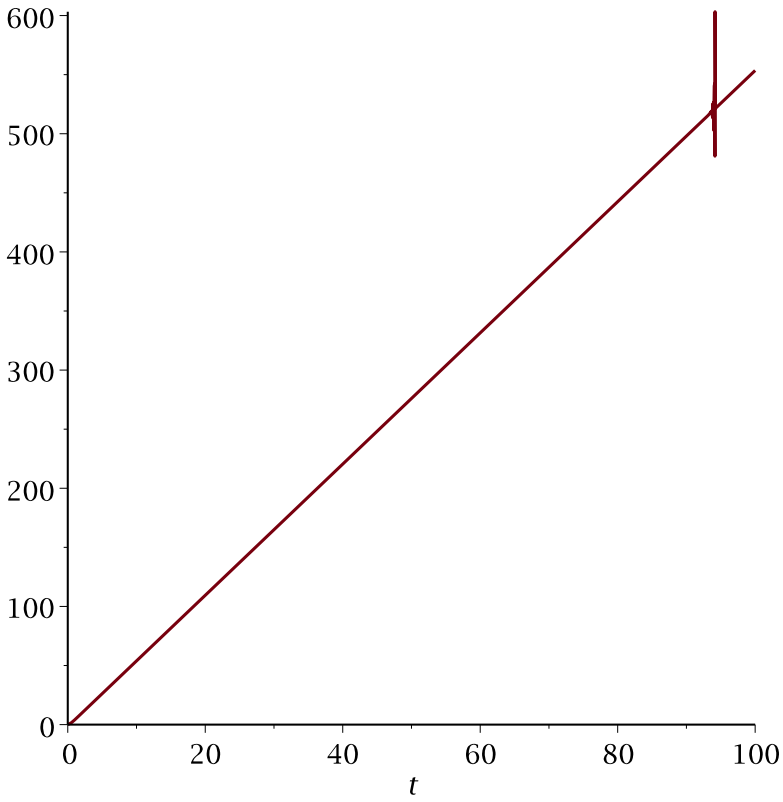
Bewegingsvergelijking

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:
```

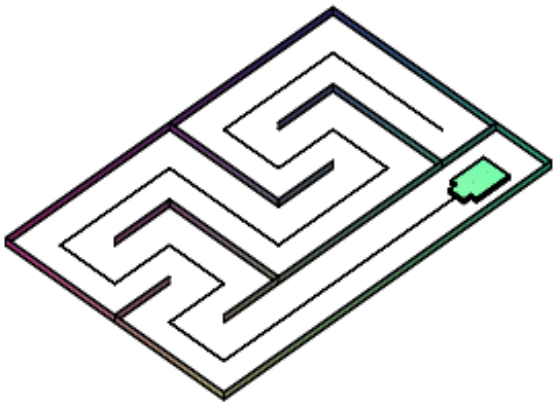
```
    end do;
    tmp_matrix := [op(tmp_matrix), current_polygon];
  end do;
  polygonplot3d(tmp_matrix, scaling = constrained, axes = none);
  end proc:
> plot_wagentje_angle := proc(translation, angle)

    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);
```



8.7 Technische tekening

8.8 Elektrische schakeling

8.9 Programmacode

Listing 1: basis_sketch.ino

```
#include <Servo.h>
const unsigned int DCP[] = {13, 2, 3, 4, 5, 6, 7, 8, 9};
const unsigned int ACP[] = {13, A0, A1, A2, A3, A4, A5};
#include "LedOutput.h"
#include "Motor.h"
#include "MyServo.h"
#include "DrivingManager.h"
#include "PushButton.h"
#include "MusicPlayer.h"

#define FRONT 90
#define LEFT 0
#define RIGHT 180

DrivingManager manager;
PushButton start_button;
LedOutput end_led;
MusicPlayer music_player;

#define RADAR_SERVO DCP[6]
#define RADAR_SENSOR ACP[1]
#define STEERING_SERVO DCP[5]
#define ELECTRO_MOTOR DCP[3]
#define RELAY DCP[4]
#define PUSHBUTTON ACP[2]
#define LED DCP[7]

/*
DIGITAL PINS (BLACK TO BROWN)
2 1
4 3
6 5
8 7
*/

void setup(){
  Serial.begin(9600);
  for(int i = 1; i <= 6; ++i){
    pinMode(ACP[i], INPUT);
  }
  manager.setup(RADAR_SERVO, RADAR_SENSOR, STEERING_SERVO, ELECTRO_MOTOR, RELAY
  start_button.setup(PUSHBUTTON);
  end_led.setup(LED);
```

```
  //music_player.setup(DCP[8]);
  manager.start();
}


void loop(){
  /*if(start_button.getValue()){
    manager.start();
  }*/
  manager.loop();
  if(manager.isReady()){
    end_led.set_signal_led(true);
  }
}
```

Listing 2: DistanceSensor.h

```
#ifndef DISTANCESENSOR_H
#define DISTANCESENSOR_H
#define FAR 999999
#define CLOSE 0
class DistanceSensor{
  public:
    DistanceSensor();
    void setup(int pin);
    double getDistance();
    void debug();
  private:
    int pin;
    double getDistanceByVoltage(double V);
    double getVoltage();
};
#endif
```

Listing 3: DistanceSensor.ino

```
DistanceSensor::DistanceSensor(){
}
void DistanceSensor::setup(int pin){
   this->pin = pin;
   pinMode(this->pin, INPUT);
}
double DistanceSensor::getDistance(){
  double v = this->getVoltage();
  if(v < 0.4)
   return FAR;
  if(v >= 2.6)
   return CLOSE;
  double d = this->getDistanceByVoltage(v);
  return d;
}
```

```
double DistanceSensor::getDistanceByVoltage(double V){
  //linearizing based on https://www.sparkfun.com/datasheets/Sensors/Infrared/G
  return 1.0/(V*(0.25-0.025)/2.7)-0.42;
}
double DistanceSensor::getVoltage(){
  int val = analogRead(this->pin);
  return ((double)val)*0.0049;
}
void DistanceSensor::debug(){
  Serial.println(this->getDistance());
  delay(1000);
}
```

Listing 4: DrivingManager.h

```
#ifndef DRIVINGMANAGER_H
#define DRIVINGMANAGER_H
#include "LedOutput.h"
#include "Motor.h"
#include "MyServo.h"
#include "DistanceSensor.h"

class DrivingManager{
  public:
    DrivingManager();
    void start();
    void stop();
    bool isReady();
    void setup(int radar_servo_pin, int radar_sensor_pin, int steering_servo_pi
    void loop();
  protected:
    MyServo radar_servo;
    MyServo steering_servo;
    Motor motor;
    DistanceSensor radar_sensor;
    bool active;
    bool isActive();
    double measureDistance(int angle);
    void handleFrontDistance(double distance);
    void handleSideDistance(double left_distance, double right_distance);
    void turn(int turn_direction, int angle, int duration);
    int steer_angle;
    int turn_angle;
    int turn_duration;
    int steer_duration;
    double final_distance;
    bool ready;
    void steeringServoDemo();
    void motorDemo();
    void radarServoDemo();
```

```
    void sensorDemo();
    void radarDemo();
};
#endif
```

Listing 5: DrivingManager.ino

```
#include "DrivingManager.h"
DrivingManager::DrivingManager(){
  this->active = false;
  this->ready = false;
  this->steer_angle=30;
  this->turn_angle=70;
  this->turn_duration=300;
  this->steer_duration=100;
  this->final_distance=5.0;
}
void DrivingManager::start(){
  this->active = true;
  this->ready = false;
}
void DrivingManager::stop(){
  this->active = false;
  this->ready = true;
}
bool DrivingManager::isReady(){
  return this->ready;
}
void DrivingManager::setup(int radar_servo_pin, int radar_sensor_pin, int steer
  this->radar_servo.setup(radar_servo_pin);
  this->radar_sensor.setup(radar_sensor_pin);
  this->steering_servo.setup(steering_servo_pin);
  this->motor.setup(motor_pin, relay_pin);

  this->radar_servo.setAngle(FRONT);
  this->radar_servo.setAngle(FRONT);
  this->motor.setMotorSpeed(0);
}
bool DrivingManager::isActive(){
  return this->active;
}
double DrivingManager::measureDistance(int angle){
  this->radar_servo.setAngle(180-angle);
  this->motor.setMotorSpeed(0.0);
  this->radar_servo.wait();
  return this->radar_sensor.getDistance();
}
void DrivingManager::handleFrontDistance(double distance){
   if(distance > 20){
      this->motor.setMotorSpeed(1.0);
```

```
    }else if(distance < 10){
      this->stop();
       this->motor.setMotorSpeed(0.0);
    }else{
      this->motor.setMotorSpeed(1.0);
    }
}
void DrivingManager::turn(int turn_direction, int angle, int duration){
  //Serial.println("turning");
  int wheel_angle = FRONT;
  if(turn_direction == LEFT){
    wheel_angle -= angle;
  }else if(turn_direction == RIGHT){
    wheel_angle += angle;
  }
  this->steering_servo.setAngle(wheel_angle);
  delay(200);
  this->motor.setMotorSpeed(0.8);
  delay(duration);
  this->steering_servo.setAngle(FRONT);
  //TODO: how long would this take?
  //Important! U-turns!
}
void DrivingManager::handleSideDistance(double left_distance, double right_dist
  if(left_distance > 30){
    this->turn(LEFT, this->turn_angle, this->turn_duration);
  }else if(right_distance > 30){
    this->turn(RIGHT, this->turn_angle, this->turn_duration);
  }else{
    //correct driving angle
    if(abs(left_distance-right_distance) >= 4.0){
      if(left_distance > right_distance){
        this->turn(RIGHT, this->steer_angle, this->steer_duration);
      }else{
        this->turn(LEFT, this->steer_angle, this->steer_duration);
      }
    }
  }
}
void DrivingManager::loop(){
  //this->radar_servo.setAngle(FRONT);
  //this->steering_servo.setAngle(FRONT);
  // return;
  this->radarDemo();
  return;
  //this->steering_servo.setAngle(FRONT);
  //this->radar_servo.setAngle(FRONT);
  //return;
  //this->sensorDemo();
  //return;
```

```
//this->radarServoDemo();
//return;
//this->steeringServoDemo();
//return;



//this->motorDemo();
//return;

//echte code

if(!this->isActive())
  return;

//backup
double front_distance_2 = this->radar_sensor.getDistance();
Serial.println(front_distance_2);
if(front_distance_2 < 20){
  this->motor.setMotorSpeed(0.0);
  this->stop();
  return;
}
this->steering_servo.setAngle(FRONT);
this->motor.setMotorSpeed(0.8);
delay(400);
this->motor.setMotorSpeed(0.0);
delay(500);
return;
//end of backup



//measure front distance, if too close: stop
this->motor.setMotorSpeed(0);
 double front_distance = this->measureDistance(FRONT);
this->handleFrontDistance(front_distance);
delay(500);
//measure left distance, if too close: steer right. if nothing left: turn
double left_distance = this->measureDistance(LEFT);
//TODO: should we check FRONT? how much additional time is needed for this?
double front_distance2 = this->measureDistance(FRONT);
//measure right distance, if too close: steer left. If nothing right: turn
double right_distance = this->measureDistance(RIGHT);
 delay(100);
 this->radar_servo.setAngle(FRONT);
this->handleSideDistance(left_distance, right_distance);
if(front_distance <= this->final_distance && left_distance <= 30 && right_dis
  this->motor.setMotorSpeed(0);
  this->stop();
```

```
    return;
  }
  delay(500);
}

void DrivingManager::steeringServoDemo(){
  this->steering_servo.setAngle(30);
  delay(1000);
  this->steering_servo.setAngle(90);
  delay(1000);
  this->steering_servo.setAngle(150);
  delay(1000);
  this->steering_servo.setAngle(FRONT);
  delay(1000);
}
void DrivingManager::radarServoDemo(){
  int interval = 200;
  this->radar_servo.setAngle(LEFT);
  delay(interval);
  this->radar_servo.setAngle(FRONT);
  delay(interval);
  this->radar_servo.setAngle(RIGHT);
  delay(interval);
  this->radar_servo.setAngle(FRONT);
  delay(interval);
}

void DrivingManager::motorDemo(){

  //motor wordt per 3 seconden omgekeerd van richting

   this->motor.setMotorSpeed(1.0);
  delay(3000);
  this->motor.setMotorSpeed(0.0);
  delay(3000);
  this->motor.setMotorSpeed(-1.0);
  delay(3000);
  this->motor.setMotorSpeed(0.0);
  delay(3000);
}
void DrivingManager::sensorDemo(){
  this->radar_sensor.debug();
}
void DrivingManager::radarDemo(){
  double front_distance, left_distance, right_distance;
  front_distance = this->measureDistance(FRONT);
  left_distance = this->measureDistance(LEFT);
  this->measureDistance(FRONT);
  right_distance = this->measureDistance(RIGHT);
  Serial.print(front_distance);
```

```
  Serial.print(" ");
  Serial.print(left_distance);
  Serial.print(" ");
  Serial.println(right_distance);
}
```

Listing 6: LedOutput.h

```
#ifndef LEDOUTPUT_H
#define LEDOUTPUT_H
class LedOutput{
  public:
    void setup(int pin);
    void loop();
    void set_signal_led(bool value);
  private:
    int led_pin;
    bool led_on;
};
#endif
```

Listing 7: LedOutput.ino

```
#include "LedOutput.h"
void LedOutput::setup(int pin){
  this->led_pin = pin;
  pinMode(led_pin, OUTPUT);
  this->set_signal_led(false);
}
void LedOutput::set_signal_led(bool value){
  this->led_on = value;
  if(this->led_on){
    digitalWrite(this->led_pin, HIGH);
  }else{
    digitalWrite(this->led_pin, LOW);
  }
}
```

Listing 8: Motor.h

```
#ifndef MOTOR_H
#define MOTOR_H
class Motor{
  public:
    void setup(int pin, int relay_pin);
    void setMotorSpeed(double motor_speed);
  private:
    int motor_pin;
    int relay_pin;
    double motor_speed;
};
#endif
```

Listing 9: Motor.ino

```
#include "Motor.h"

void Motor::setup(int pin, int relay_pin){
  this->motor_pin = pin;
  this->relay_pin = relay_pin;
  this->motor_speed = 0;
  pinMode(this->motor_pin, OUTPUT);
  pinMode(this->relay_pin, OUTPUT);
}

void Motor::setMotorSpeed(double motor_speed){
  motor_speed = min(max(motor_speed, -1.0), 1.0);
  this->motor_speed = motor_speed;
  if(this->motor_speed < 0){
    digitalWrite(this->relay_pin, HIGH);
  }else{
    digitalWrite(this->relay_pin, LOW);
  }
  Serial.println((int)(this->motor_speed*255));
  analogWrite(this->motor_pin, (int)(abs(this->motor_speed)*255.0));
}
```

Listing 10: MusicPlayer.h

```
#ifndef MUSICPLAYER_H
#define MUSICPLAYER_H
class Song;
class MusicPlayer{
  public:
    MusicPlayer();
    void loop();
    void setup(int pin);
    void setSong(Song* song);
    Song* getSong();
  protected:
    Song* song;
    int pin;
};

class Song{
  public:
    Song(int notes_amount, int* tones, int* note_durations);
  protected:
    int* tones;
    int* note_durations;
    int notes_amount;
};
#endif
```

Listing 11: MusicPlayer.ino

```
MusicPlayer::MusicPlayer(){
}
void MusicPlayer::setup(int pin){
  this->pin = pin;
  pinMode(this->pin, OUTPUT);
}
void MusicPlayer::loop(){

}
void MusicPlayer::setSong(Song *s){
  this->song = s;
}
Song* MusicPlayer::getSong(){
  return this->song;
}
Song::Song(int notes_amount, int* tones, int* note_durations){
  this->notes_amount = notes_amount;
  this->tones = tones;
  this->note_durations = note_durations;
}
```

Listing 12: MyServo.h

```
#ifndef MYSERVO_H
#define MYSERVO_H

class MyServo{
  public:
    void setup(int pin);
    void setAngle(int angle);
    void wait();
  private:
    Servo servo;
    int angle;
    int convertAngleToMicroseconds(int angle);
};
#endif
```

Listing 13: MyServo.ino

```
#include "MyServo.h"

void MyServo::setup(int pin){
  pinMode(pin, OUTPUT);
  servo.attach(pin);
}
void MyServo::wait(){
  //TODO: delay a certain amount of milliseconds until the servo is at the righ
  delay(1000);
}
```

```
void MyServo::setAngle(int angle){
  this->angle = angle;
  this->servo.write(this->angle);
}
```

Listing 14: PushButton.h

```
#ifndef PUSHBUTTON_H
#define PUSHBUTTON_H
class PushButton{
  public:
    void setup(int pin);
    bool getValue();
  private:
    int pin;
};
#endif
```

Listing 15: PushButton.ino

```
void PushButton::setup(int pin){
  this->pin = pin;
  pinMode(this->pin, INPUT);
}

bool PushButton::getValue(){
  return analogRead(this->pin) < 500;
}
```

8.10 Materiaallijst

	Aantal	**Component-nummer / product-informatie**	**Prijs Per Stuk (€)**	**Totaal (€)**
Mechanisch gedeelte				
Hout MDF	1	60cm*30cm*3mm	1	1
Onderdelen Lego*		Geschatte kostprijs: €15	0	0
Duct Tape			0	0
Schroefogen	9		0.12	1.08
Bouwpakket	2		2.6	5.2
Tandwiel 50/10	10	841027	0.1	1
Tandwiel 30/10	10	841016	0.085	0.85
Zeskantmoeren	20	M3		0
Bouten	30	3Mx10mm	0.08	2.4
Rondellen	80		0.03	2.4
Draadstang	1	M3 x 1000	2.09	2.09
Elektronica				
Arduino	1		0	0
Relais	1	RTE24005AP	3	3
Mosfet	2	IRF3707ZPBF		0
Weerstand	2	10k	0.05	0.1
Servomotor	2	T010050	11	22
Elektromotor	2	MM28	4	8
Afstandsensor	1	GP2D120XJ00F	16	16
LED	1	L-934 GD	0.1	0.1
Administratie en verzendkosten				1
			Totaal	**66.22**

*De LEGO-onderdelen werden niet aangekocht maar in bruikleen van een teamlid gebruikt.

8.11 Financiën

8.12 Ganttchart

8.13 Poster

9 Referenties

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

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

[3] Sharp. GP2Y0A41SK0F datasheet. Raadpleging: 2014-05-06. [Online]. Available: http://www.sharp.co.jp/products/device/doc/opto/gp2y0a41sk_e.pdf

[4] bitwizard.nl. (2012) Mosfets. Raadpleging: 2014-05-06. [Online]. Available: http://prive.bitwizard.nl/mosfets.pdf