Systematische analyse van het “ABAG” -regel algoritme

F.E.J. Hekker

Bachelor Eindproject

Identiteitsnummer: 0897373

Begeleider: prof.dr. H.P.J. Bruyninckx

Technische Universiteit Eindhoven
Faculteit Werktuigbouwkunde
Control Systems Technology

Technische Universiteit Eindhoven
Eindhoven 31 januari 2018

Inhoudsopgave

Symbolenlijst

k	de huidige tijdstap
y_k	gemeten snelheid
y_k^d	gewenste snelheid
u_k	koppel voor de motor
α	fout filter factor
$\bar{e}_b$	grenswaarde voor verandering in de bias
δ_b	stapgrootte verandering in de bias
$\bar{e}_g$	grenswaarde voor verandering in de gain
δ_g	stapgrootte verandering in de gain
$\bar{e}_k$	gefilterde fout
b_k	bias
g_k	gain
r	radius van het wiel
F	kracht
m	massa van het systeem
a	versnelling
F_w	weerstandskracht
t	tijd
v	de snelheid van de robot
v_{links}	de snelheid van het linker wiel
v_{rechts}	de snelheid van het rechter wiel
θ	de hoek die de snelheid vector van de robot maakt met de x-as
l	de afstand tussen de wielen van de robot
x	de plaats van de robot op de x-as
y	de plaats van de robot op de y-as
$a_{voorspel}$	de afstand waarop de robot virtueel verder rijdt

Introductie

Het doel van dit project is om beter te kijken naar het "ABAG"(Adaptive-Bias / Adaptive-Gain) regel algoritme als in een artikel van Franchi and Mallet [1] beschreven. Het voordeel van dit algoritme zou zijn dat er weinig kalibratie nodig is, het zeer robuust is tegen veranderingen in het systeem en er geen zware berekening moeten worden gedaan. Dit laatste zorgt ervoor dat de rekentijd zeer laag is en er dus meer samples kunnen worden genomen.

In het artikel is het algoritme enkel getest op drones maar in dit project wordt er gekeken naar de toepasbaarheid op rijdende robots. Om dit te testen zullen er een aantal modellen worden gemaakt. Eerst een model met één wiel om inzicht te geven in de werking van het algoritme en vervolgens een nieuw model om de samenwerking van meerdere wielen te beoordelen. Dit verslag zal dan ook als volgt in elkaar zitten:

Eerst zal de code worden uitgelegd. Alle variabelen zullen worden besproken en de werking van het algoritme wordt duidelijk gemaakt. Hierna wordt er naar het model gekeken dat gebruikt is om op de snelheid van één wiel te regelen. De toepassing wordt geÃ«valueerd. Vervolgens wordt dit model verbeterd door een passieve rem toe te voegen. Ook zal het model van een robot met twee wielen worden getoond. Hierbij wordt er ook naar plaatsbepaling gekeken. Ten slotte wordt er gekeken naar de invloed van een aantal parameters op de resultaten van dit model.

1 Werking ABAG

Om het "ABAG"-algoritme beter uit te leggen zal het volgende probleem worden gebruikt:
Er is een wiel dat alleen naar voren en achter kan bewegen op verschillende snelheden. Het doel is om het wiel een vooraf vast gestelde snelheid te laten rijden door de kracht die de motor levert in te stellen. Hierbij is het probleem dat er een aantal onbekenden zijn, zo kan bijvoorbeeld de wrijving van de vloer waarop het wiel rijdt variÃ«ren. Dit zou een andere snelheid geven bij de zelfde kracht.
Om dit te voorkomen word het "ABAG"-algoritme ingezet. Dit algoritme kijkt naar het verschil tussen de snelheid die het wiel heeft en de snelheid die gewenst is voor het wiel; dit heet de fout. Door naar deze fout te kijken (en alle fouten die eerder op dit traject zijn gemaakt) schat het ABAG algoritme de kracht die de motor moet leveren om bij de gewenste snelheid te komen. Hieronder wordt uitgelegd hoe het algoritme dit signaal schat. Dit wordt gedaan met hulp van de code die te zien is in Algoritme 1. Deze code komt uit een onderzoek van Franchi and Mallet 1. In dit onderzoek wordt ook uitgelegd waarom de code op deze manier is gemaakt.

1.1 Variabelen

De ingangssignalen in dit voorbeeld zijn de snelheid die gewenst is (wordt y_k^d genoemd in de code) en de gemeten snelheid die het wiel op dat moment heeft (wordt y_k genoemd in de code). Het uitgangssignaal is wat er naar de aandrijving van het wiel word gestuurd (wordt u_k genoemd in de code). Dit signaal ligt altijd tussen 0 en 1 en wordt later met de maximale snelheid vermenigvuldigd om tot de gewenste snelheid te komen. Dit is zo gedaan om de regelaar relatief te maken en dus voor meer verschillende situaties inzetbaar. Door de hele code heen wordt met het subscript k de huidige tijdstap aangegeven.
Voordat er berekeningen kunnen worden gemaakt zullen de waarden van een aantal parameters gekozen moeten worden. Alle parameters moeten tussen de 0 en 1 liggen door het eerder genoemde relatieve karakter van de code. De volgende parameters zijn van belang:

- α, hoe hoger deze waarde is hoe meer de vorige waardes voor de fout ($\bar{e}_k$) meetellen ten opzichte van de nieuwste waarde.
- $\bar{e}_b$ en $\bar{e}_g$, deze waarden bepalen wanneer de bias danwel de gain veranderd moeten worden. Dit treedt in werking wanneer de fout groter is dan deze parameter.
- δ_b en δ_g, dit staat voor de stap-grootte. Wanneer er een verandering moet komen in de bias of gain, zal dit aangeven hoe groot die verandering is.

Voor een meer diepgaande uitleg van de parameters zie paragraaf 2.2

1.2 Code

Algorithm 1 ABAG

1: $k = 0$ $u_0 = \bar{e}_0 = g_0 = b_0 = 0$
2: **while** ++k **do**
3: $\quad \bar{e}_k = \alpha \bar{e}_{k-1} + (1 - \alpha) sgn(y_k - y_k^d)$
4: $\quad b_k = sat[0, 1](b_{k-1} + \delta_b hside(|\bar{e}_k| - \bar{e}_b) sgn(\bar{e}_k - \bar{e}_b))$
5: $\quad g_k = sat[0, 1](g_{k-1} + \delta_g sgn(|\bar{e}_k| - \bar{e}_g))$
6: $\quad u_k = sat[0, 1](b_k + g_k sgn(y_k - y_k^d))$
7: **end**

Nu de variabelen duidelijk zijn is het tijd om beter naar de code te kijken. Dit zal per regel worden gedaan:

1. Op de eerste tijdstap (k=0) zijn alle te berekenen variabelen 0, omdat er nog weinig bekend is over het systeem.

3. $\bar{e}_k$ is de fout en wordt berekend door te kijken of y_k^d groter of kleiner is dan y_k dit gebeurt door de sgn (sign functie geeft -1 voor alles kleiner dan 0 anders geeft hij 1). Hierdoor hoeft de fout niet exact bepaald te worden zolang het teken maar goed is. Dit geeft het algoritme een grotere robuustheid. Het signaal dat uit de sign functie komt wordt samen genomen met alle voorgaande fouten door middel van een laagdoorlaatfilter(low-pass). Hoe sterk de nieuwste fout meetelt ligt aan de grote van α

4. De bias b_k wordt alleen her-berekend als $|\bar{e}_k|$ groter is dan $\bar{e}_b$ anders zal de heaviside functie 0 worden (heaviside wordt in de code afgekort met hside en geeft 0 als zijn invoer kleiner is dan 0 anders geeft hij 1). Dit zorgt er voor dat de bias niet continu verandert maar alleen als hij buiten de marge komt.
   Daarna word er gekeken of de bias hoger of lager moet worden door de sign functie van $\bar{e}_k - \bar{e}_b$. De verandering zal ter grote van δ_b zijn en opgeteld worden bij de vorige bias. Door de saturatie (afgekort met sat) kan de bias alleen tussen de 0 en 1 zitten, als hij daar buiten dreigt te gaan zal deze functie zorgen dat b_k de dichtstbijzijnde grenswaarde wordt.

5. Net als bij de bias wordt er bij de gain gekeken hoe het moet veranderen door de sign van $|\bar{e}_k| - \bar{e}_g$ te nemen. Deze verandering heeft de grootte van δ_g.
   Ook hier wordt de saturatie gebruikt om de gain binnen de gewenste grenzen te houden.

6. Als laatste wordt het uitgangssignaal berekend door de bias op te tellen bij de gain. Het teken van de gain wordt bepaald door de sign van $y_k - y_k^d$. Dit zorgt er voor dat het signaal kleiner wordt als het te groot was en andersom. Ook hier werkt de saturatie.

Door dit algoritme lang genoeg te laten draaien zal y_k stabiliseren rond y_k^d binnen de grenzen die worden aangegeven door $\bar{e}_b$ en $\bar{e}_b$.

2 Implementatie en resultaten

Om het ABAG algoritme te kunnen testen is er een simulatie gemaakt in het programma Simulink [2]. De simulatie werkt als volgt: het gewenste signaal gaat eerst door de regelaar. Daarna wordt het door een systeem gevoerd dat de robot nadoet (hier de plant genoemd). Het laatste blok dat te zien is onthoudt het uitgangssignaal van die tijdstap om hem op de volgend tijdstap terug te kunnen geven aan de regelaar. Dit is te zien in Figuur 1. Voor een meer gedetailleerde kijk op zowel de regelaar als de plant zie Appendix A.1

Figuur 1: Een overzicht van het model

2.1 Plant

De plant is in dit geval een vrij simpel systeem. Het gaat er vanuit dat zijn ingangssignaal dat hij van de regelaar krijgt het gewenste koppel is. Dit word door de volgende berekeningen omgezet naar de snelheid.

$$u_k = r * F$$

Waar bij u_k het koppel is, r de straal van het wiel en F de kracht die als volgt is gedefinieerd:

$$F = m * a + F_w$$

Hier is m de massa van de robot, a de versnelling en F_w de weerstandskracht. Deze twee formules substitueren geeft:

$$a = \frac{u_k * m}{r} - \frac{F_w}{m}$$

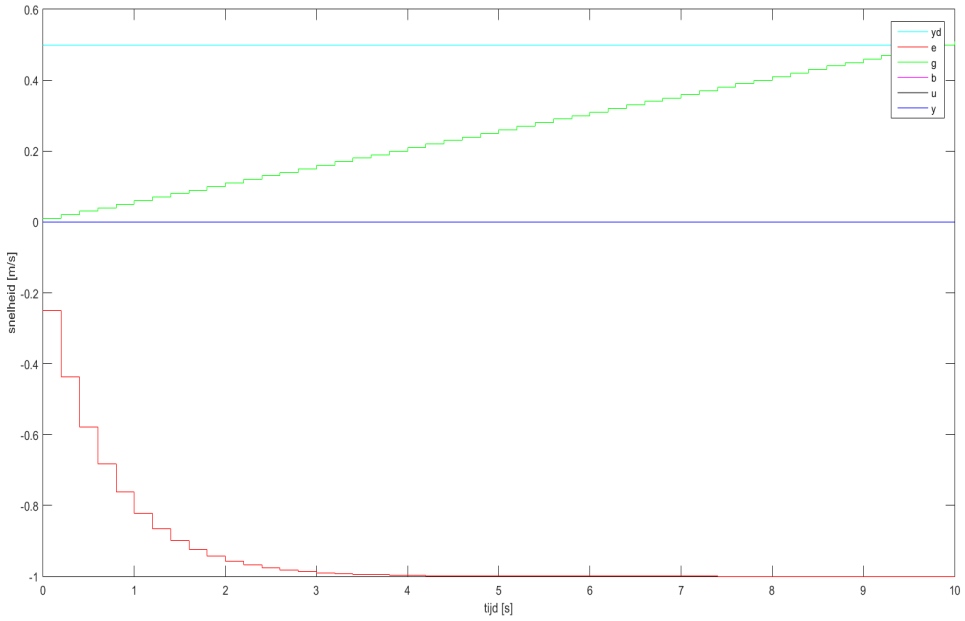
Na integratie kan de snelheid(y_k) gevonden worden

$$y_k = \frac{u_k * m}{r} - \frac{F_w}{m} dt$$

t staat hier voor de tijd.
Deze formules zijn ook terug te lezen in het boek van Meriam and Kraige [3]
Alle onbekende variabelen hebben hier een realistische schatting als waarde gekregen. In een praktisch experiment hoeft geen plant gebruikt te worden dus is schatting niet aan de orde.

2.2 Resultaten

Met het voorgenoemde model wordt het resultaat (te zien in Figuur 2) behaald.



Figuur 2: resultaat van het eerste model

Hier zijn y_k, u_k en b_k over de tijd gelijk aan nul.
Dit is niet het verwachte resultaat. Het uitgangssignaal zou langzaam naar y_k^d toe moeten groeien maar dat is niet het geval. Met een kritische blik op de code is dit te verklaren.
Deze grafiek zal als voorbeeld dienen. Op de eerste tijdstap is de gewenste waarde (y_k^d) hier groter dan de huidige waarde(y_k). Dit geeft de volgende consequenties (per regel in de code, zie paragraaf 1.2):

- In dit geval is sign($y_k - y_k^d$)=-1. Hierdoor is $e_k = -1 + \alpha$ en aangezien α altijd kleiner dan 1 is, wordt $\bar{e_k}$ dus negatief.
- Dat zorgt er voor dat $\bar{e}_k$ nooit groter kan zijn dan $\bar{e}_b$. Eerder is namelijk vastgesteld dat $\bar{e}_b$ altijd tussen de 0 en 1 is. Dit heeft tot gevolg dat de sign functie $\bar{e}_k - \bar{e}_b$ altijd negatief is. Dus b_k kan alleen maar negatief of nul zijn (nul door de heaviside functie). Door de saturatie wordt b_k 0.
- Aangezien g_k alleen naar de absolute waarde van $\bar{e_k}$ kijkt kan hij wel groeien.
- Door de eerste consequentie geldt hier dat $u_k = b_k - g_k$. Omdat $b_k = 0$ en g_k is altijd positief (ook hier weer door de saturatie) kan u_k alleen maar negatief worden wat, zoals in het resultaat te zien is naar 0 wordt teruggebracht.

Op deze manier kan het systeem dus niet de gewenste snelheid bereiken.

2.2.1 Veranderingen in het model

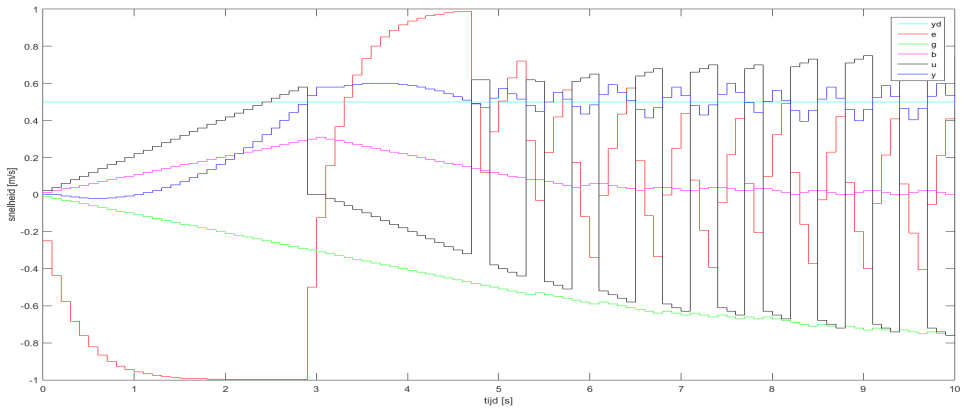
Om hierboven genoemde redenen werkte het model nog niet naar behoren. Hierop zijn de volgende aanpassingen gedaan.
Ten eerste zijn de vierde en vijfde regel van de code aangepast. Waar eerst de nieuwe waarde bij de vorige werd opgeteld, worden ze in deze versie van elkaar afgetrokken. Dit zorgt er voor dat de bias en gain omhoog gaan als de snelheid(y_k) te laag is.
Ook zou het mooi zijn als het wiel zowel voor- als achteruit zou kunnen rijden. Daartoe zijn alle saturaties omgezet van [0,1] naar [-1,1]. Het onderstaande algoritme laat deze veranderingen zien.

Algorithm 2 veranderd ABAG

1: $k = 0\ u_0 = \bar{e}_0 = g_0 = b_0 = 0$
2: **while** ++k **do**
3: $\bar{e}_k = \alpha\bar{e}_{k-1} + (1-\alpha)sgn(y_k - y_k^d)$
4: $b_k = sat[-1,1](b_{k-1} - \delta_b hside(|\bar{e}_k| - \bar{e}_b)sgn(\bar{e}_k - \bar{e}_b))$
5: $g_k = sat[-1,1](g_{k-1} - \delta_g sgn(|\bar{e}_k| - \bar{e}_g))$
6: $u_k = sat[-1,1](b_k + g_k sgn(y_k - y_k^d))$
7: **end**

2.2.2 Resultaat verbeteringen

De aanpassingen leiden tot het resultaat zichtbaar in Figuur 3. De eerst paar seconden loopt y_k naary_k^d. Wanneer hij daar overheen schiet reageert u_k meteen door te dalen. y_k reageert vertraagd door de integratie die in de plant wordt gedaan (die is nodig om van de versnelling naar de snelheid te gaan). Na ongeveer 6.5 s bereikt het systeem een steady-state. y_k blijf binnen de marges die aangegeven zijn in $\bar{e}_g$ en $\bar{e}_b$. Dit is het verwachte en gewenste resultaat.

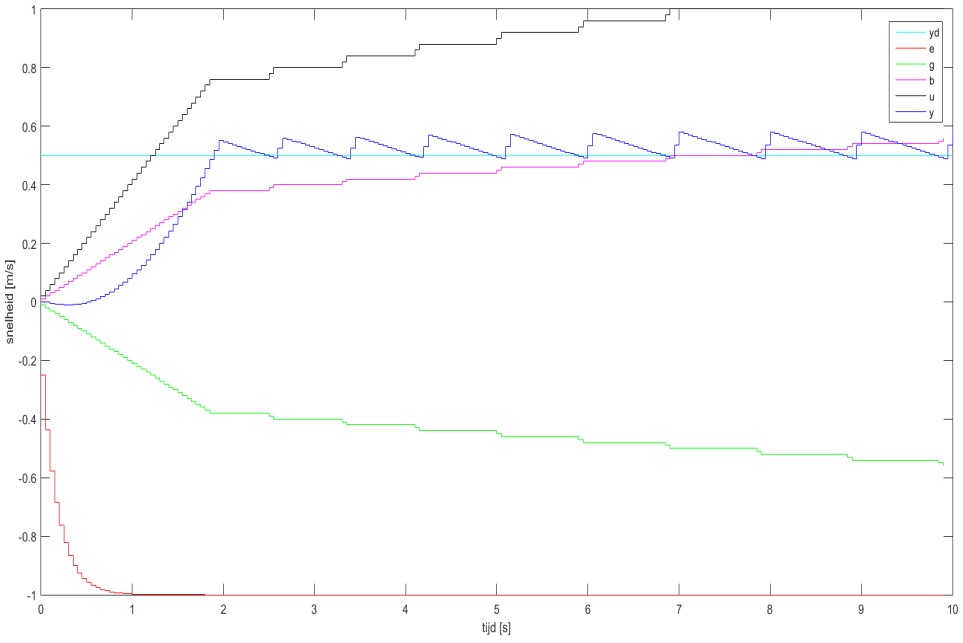


Figuur 3: resultaat van het verbeterde model

3 Passief remmen

In het vorige model werd er actief geremd. Dat wil zeggen; wanneer de robot te snel gaat zal er een negatief koppel worden gegeven waardoor hij remt. Dit kan ook op een andere manier. In de plant is ook wrijving meegenomen die altijd een ander teken heeft dan de snelheid. Om te remmen kun je dus ook de wrijving zijn werk laten doen. Dit is een kwestie van geen koppel meer geven. In het volgende model is precies dat gedaan.
In eerste instantie lijkt het misschien een goed idee om het uitgangssignaal u_k te overschrijven als het groter is dan y_k^d zodat het op die momenten 0 is. Dit resulteert echter in een sterke afname in g_k en b_k zolang het signaal niet weer onder de gewenste snelheid komt. Dit zou er voor zorgen dat u_k veel lager is en dus minder invloed heeft wanneer het algoritme weer zijn werk moet doen.

Om dit te voorkomen wordt er eerst gekeken naar y_k^d en $\text{sgn}(y_k - y_k^d)$. Wanneer beiden hetzelfde teken hebben wordt $u_k = 0$, zo niet dan wordt regelaar aangeroepen en doet het ABAG algoritme zijn werk.
Het nieuwe model dat te zien is in Appendix A.2 geeft het volgende resultaat:



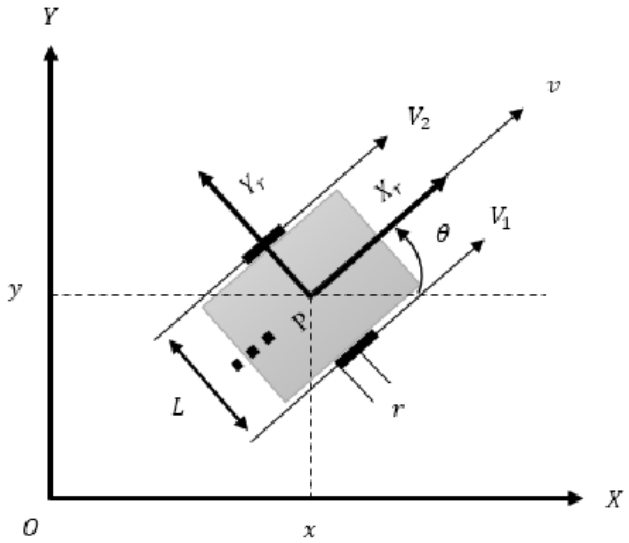
Figuur 4: resultaat van het model met passieve rem

In dit Figuur is goed zichtbaar dat de regelaar stil ligt wanneer y_k te ver boven y_k^d uitkomt. De wrijving zorgt er voor dat het uitgangssignaal vanzelf weer laag genoeg word. Wanneer dit gebeurt grijpt de regelaar weer in.
De plateaus zijn tijden waarop de regelaar stil ligt. Aan het einde van de grafiek is het zelfs zichtbaar (voor al bij de lijn van g_k) dat er geen waarden worden gegenereerd maar dat in

de voorgaande gevallen de grafiek naar het volgende datapunt is doorgetrokken voor betere leesbaarheid.

4 Samenwerking tussen meerdere wielen

Nu het gelukt is om van één wiel de snelheid te regelen, kan er naar de positie gekeken worden. Om dit wat interessanter te maken wordt het nieuwe systeem een zogenaamde "differential drive". Dit houdt in dat er twee wielen zijn die los van elkaar kunnen worden aangestuurd. Het systeem kan dus van richting veranderen door de relatieve snelheid van de wielen te veranderen. De "differential drive" en variabelen die later worden gebruikt worden zijn uitgebeeld in Figuur 5. Hoewel bij dit soort systemen vaak alleen de aangedreven wielen worden afgebeeld is er in de praktijk ook een zwenkwieltje nodig voor de stabiliteit. Dit wieltje heeft echter maar een minieme invloed op de besturing van het systeem en zal daarom hierna niet meer worden genoemd.



Figuur 5: het systeem met twee wielen

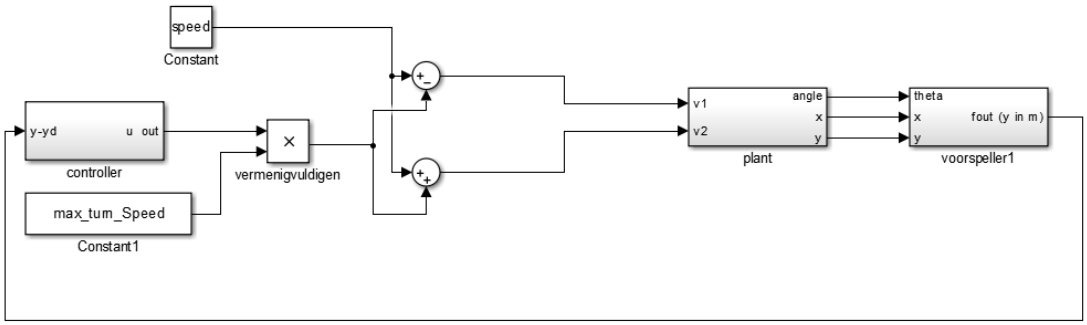
4.1 Model

Voor dit systeem is een nieuw model gemaakt. Dit is te zien in Figuur 6. Het model gaat uit van bepaalde punten:

1. Er wordt een vooraf gedefinieerd traject gevolgd
2. De robot rijdt een constante snelheid
3. De maximale snelheid van de wielen is bekend

Uit punt twee volgt dat, om te rechtdoor te rijden, beide wielen de vooraf bepaalde, constante snelheid moeten maken. Om te draaien moet het ene wiel sneller gaan draaien dan het andere.

De constante wordt behouden door het ene wiel even veel langzamer te laten gaan dan het andere wiel sneller.
Hoe groot dit verschil in snelheid moet zijn wordt berekend door het ABAG-algoritme. Het algoritme geeft als uitkomst het deel van de maximale snelheid dat bij de constante snelheid van de wielen moet worden opgeteld om de juiste draai te maken. Dit geeft de snelheden van beide wielen. Met de snelheden van beide wielen kan de plant de plaats van de robot berekenen. De plaats van de robot is nodig om afwijking van het traject te bepalen. Dit wordt gedaan in de zogenaamde "voorspeller". De afwijking, of fout, die de "voorspeller" berekent wordt het ingangssignaal van de regelaar. Dit model is uitgebeeld in Figuur 6 en verder uitgewerkt in Appendix A.3



Figuur 6: model van het systeem met twee wielen

4.1.1 Plant

Het doel van de plant is om met de snelheden van het linker (v_2) en het rechter (v_1) wiel de plaats van de robot te berekenen (hier aangegeven in (X,Y)). Aangezien dit een standaard probleem is zal hier niet te diep op in worden gegaan. Meer informatie kan worden gevonden in Ribeiro and Lima [4] waar dit op gebaseerd is.
De plaats wordt gevonden door eerst de snelheid van het hele systeem(v) te berekenen. Ook de verandering in de hoek tussen de assenstelsels van de robot en de echte wereld wordt berekend (θ). Hiervoor is ook de afstand tussen de wielen (l nodig). Met deze twee variabelen kan daarna de verplaatsing van de robot worden berekend. Alle genoemde variabelen zijn terug te vinden in Figuur 5.
Dit wordt op de volgende manier gedaan:

$$v = 0.5v_1 + 0.5v_2$$
$$\dot{\theta} = \frac{1}{l}v_1 + \frac{1}{l}v_2$$

Hier mee kan berekend worden dat:

$$\dot{x} = vcos(\theta)$$
$$\dot{y} = vsin(\theta)$$

Door middel van integratie van deze berekeningen kan de plaats worden gevonden.

4.1.2 Voorspeller

De naam van de voorspeller is zo gekozen omdat hij niet naar de huidige/feitelijke plaats kijkt van de robot maar naar de plaats die de robot krijgt als hij doorgaat op de ingeslagen weg. Op deze plaats wordt er gekeken naar de afwijking van het gewenste traject. Dit is de fout die door wordt gegeven aan de regelaar. Deze voorspelling wordt gedaan om een preciezere en stabielere weg te rijden.

De voorspeller krijgt van de plant de snelheid waarmee de robot zich verplaatst en de hoek die de snelheid maakt met de X-as. Hiermee kan hij als volgt de voorspelde plaats berekenen waar de robot zou zijn als hij een afstand $a_{voorspel}$ door rijdt:

$$x_{voorspeld} = a_{voorspel} cos(\theta) + X$$

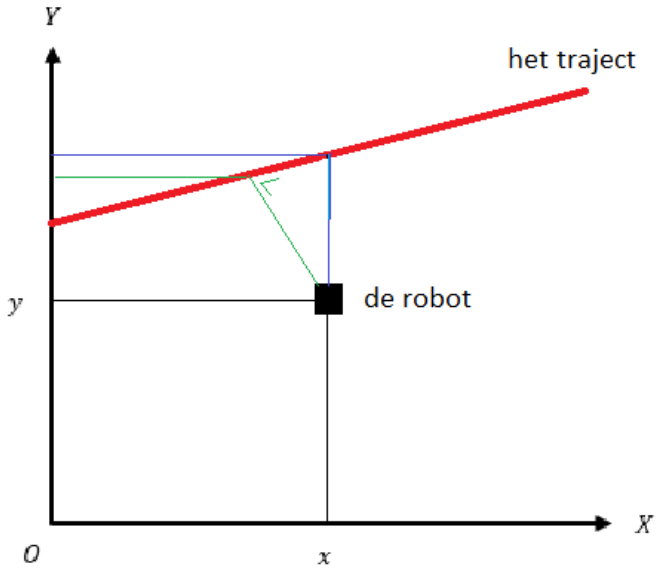
$$y_{voorspeld} = a_{voorspel} sin(\theta) + Y$$

Daarna wordt de afstand vanaf de voorspelde plaats van de robot tot het gewenste traject berekend. Deze "fout" is nodig als input voor het ABAG-algoritme om te weten wanneer en hoe die in werking moet treden en kan op verschillende manieren worden berekend.

De beste manier is om een met behulp van raak-lijnen de kortste afstand van het traject naar de plaatst te vinden. Hoewel dit het meest secuur resultaat geeft is het erg bewerkelijk. Het differentiëren wat daar voor nodig is, is ook iets wat middels de ABAG-regelaar vermeden wordt in verband met de snelheid van de berekeningen en de kosten van de elektronica. Omdat er minder berekeningen per seconde worden gemaakt, wordt het resultaat minder accuraat. Samengevat ondermijnt dit het simpele karakter van het ABAG-algoritme.

Het alternatief dat hier is toegepast, is kijken naar het verschil tussen de y-coördinaten bij het x-coördinaat van de robot. Deze manier is minder precies. Dit is echter geen probleem omdat het "ABAG"-algoritme alleen maar naar het teken, en niet de waarde, van de fout kijkt. Dat is wat het zo robuust maakt.

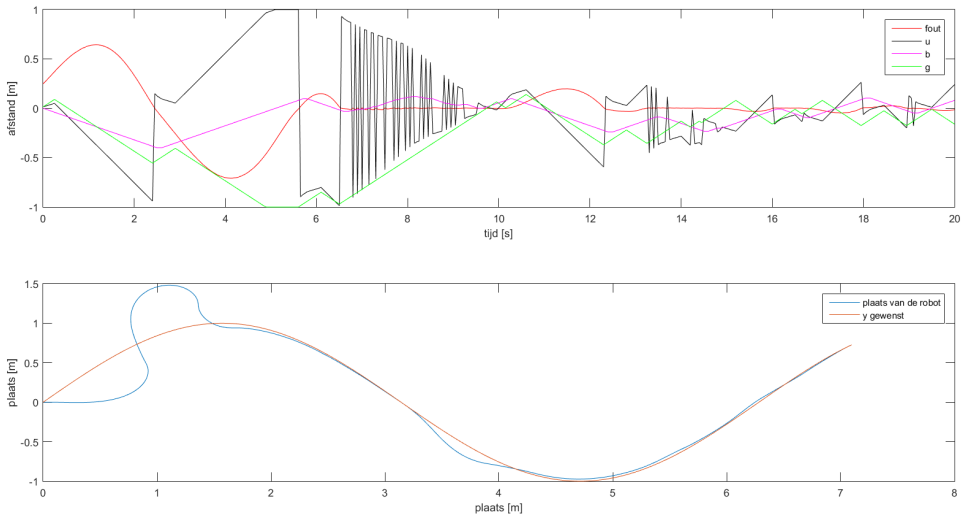
Beide manieren zijn te zien in Figuur 7. De eerste manier is getekend in groen en geeft de kleinst mogelijke fout. De tweede manier is getekend in blauw en geeft de makkelijkst te berekenen fout.
Het is belangrijk dat beide manieren om tot foutberekening te komen een negatieve waarde geven wanneer de robot te ver naar links is en een positieve waarde als hij te ver naar rechts zit. Het ABAG-algoritme kijkt enkel naar het teken van de fout.



Figuur 7: Twee manieren om de fout te berekenen. De groenen lijn is een raaklijn met het traject. De blauwe lijn kijkt naar het traject op hetzelfde x-coördinaat als de robot

4.2 Resultaten

Om aan te tonen dat dit systeem gecontroleerd een hoek om kan gaan/ een bocht kan maken wordt hem nu de opdracht gegeven een sinustraject te volgen. Het resultaat daarvan is in Figuur 8 te zien.



Figuur 8: resultaat van het “differential drive” model

In de grafiek die de plaats aangeeft is te zien dat de robot enige tijd nodig heeft om het gewenste traject te volgen. Dit is ook terug te zien in de grafiek daar boven. Het uitgangssignaal (u_k) heeft eerst twee grootte uitschieters om daarna langzaam steeds kleiner te worden. Ook de fout gaat richting 0.
De rest van de plaats grafiek blijft er naar verwachting uitzien. De robot volgt het traject binnen de gestelde afwijking.

Echter de tijds grafiek ziet er niet zo mooi uit als voorheen. Er is veel chatter ontstaan, het uitgangssignaal fluctueert snel tussen zijn positieve en negatieve waarde. Dit is niet wenselijk want deze abrupte veranderingen in de snelheid geven veel trillingen.
Ook maakt het uitgangssignaal vaak zaagtanden (het signaal loop sterk op om vervolgens in een keer de andere kant op te gaan). Dit heeft veel frequentie-inhoud.
Beide kunnen door hun trillingen zorgen voor de excitatie van de eigenfrequentie van de robot. Dit betekend dat de robot versterkt mee gaat trillen. Hierdoor is er meer kans op falen.

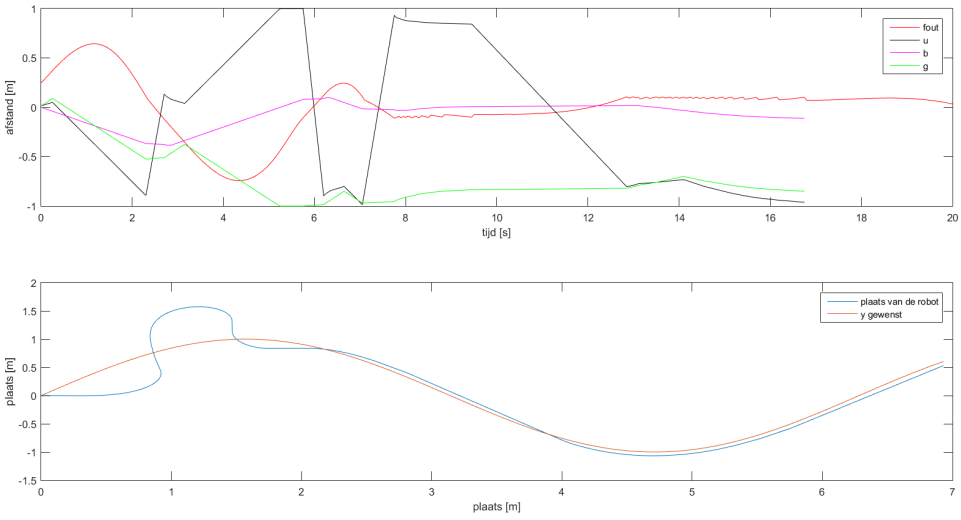
4.2.1 Onderdrukken van chatter

Zoals hier voor is duidelijk gemaakt, is de chatter die zichtbaar is in Figuur 8 slecht voor het systeem . Om dit tegen te gaan is de regelaar aangepast.
Als de robot dicht genoeg bij zijn traject is, houdt de regelaar op met werken. Dit is op dezelfde manier geïmplementeerd als de passieve rem. Het verschil is dat bij het model van de passieve rem de regelaar ophield als het wiel sneller ging dan de gewenste snelheid. Maar in het nieuwe model houdt de regelaar op als de robot binnen een vooraf bepaalde marge van het traject is.

De chatter ontstond doordat de robot het traject vaak doorkruist wanneer de fout klein is. Dit gebeurt omdat hij zo dicht mogelijk bij het traject wil blijven maar altijd kleine afwijkingen heeft. De aangepaste regelaar verhelpt dit door aan te geven dat de regelaar niet hoeft te werken als de robot zo dichtbij het traject is dat de fout toch te verwaarlozen is. Natuurlijk werkt de regelaar buiten die marge wel zodat de robot in de buurt van het traject blijft.

4.2.2 Resultaat verbeteringen

Het resultaat van dit model is hier onder weergeven.
In de plaats grafiek is te zien dat alle chatter geëlimineerd is. Er zijn nog wel enkele uitschieters waarin de waarde van u_k snel verandert maar zolang die niet te dicht achter elkaar zitten kan dat geen kwaad.
Ook hier is weer te zien dat de variabelen van de regelaar niet tot het einde van de tijd een waarde hebben. Dit gebeurt wanneer de robot binnen de marge blijft. Op dat moment is de regelaar niet aan en geeft dus geen nieuwe waarden.
In de plaats grafiek is te zien dat de robot op nauwkeurigheid inlevert. Om te beginnen doet hij er langer over om het traject te vinden. Daarnaast blijft de robot vaak een kleine afstand van het traject af rijden. Dit is de marge.
Dit is een trade-off; wanneer de marge kleiner wordt is er meer chatter en vice versa.



Figuur 9: resultaat van het chatter onderdrukkende model

De manier waarom dit model is gemaakt kan ook gebruikt worden om een robot aan te sturen met meer dan twee aandrijfbaren wielen. Om dit te doen hoeft alleen de plant veranderd te worden / vervangen te worden door de robot zelf. Daarom zal dit verder niet meer worden uitgewerkt.

5 Invloed van parameters

Nu het duidelijk is dat de "ABAG"-regelaarr werkt in meerdere situaties, wordt er gekeken naar de invloed van de constante die van te voren moeten worden ingesteld, en hoe deze gekozen kunnen worden.

5.1 Invloed van de constanten

Er zijn maar vijf variabelen die veranderd kunnen worden: de twee grenswaarden, de twee stapgrootten en de laagdoorlaatfilter factor. Om een goede vergelijking te kunnen maken zal er gekeken worden naar de gemiddelde absolute fout die de robot maakt over een tijdsbestek van 10 seconden. De fout is hier gedefinieerd als $|y_k - y_k^d|$. De waarden die worden onderzocht zijn gebaseerd op de waarden die gebruikt zijn in Franchi and Mallet [1]. De niet onderzochte waarden zullen constant blijven over de verschillenden experimenten.
De resultaten zullen worden vergeleken met de code zodat er conclusies kunnen worden getrokken over de manier waarop deze variabelen gekozen kunnen worden.
Dit experiment zal gedaan worden in het model dat één wiel aanstuurt. De reden hiervoor is dat dit het simpelste en meest robuuste systeem is. Er is dus minder kans dat het experiment wordt beïnvloed door dingen die buiten de regelaar gebeuren.

5.1.1 Grenswaarden

De grenswaarden bepalen hoe groot de afwijking moet zijn voordat er ingrepen word. Wanneer deze waarden kleiner worden zal de werkelijke snelheid dichter bij de gewenste snelheid komen maar hij zal ook meer fluctueren. Als dit fluctueren doorzet wordt het chatter, wat vermeden moet worden. Wanneer de bijbehorende stapgrootte te groot is zal de fout ook groter worden.

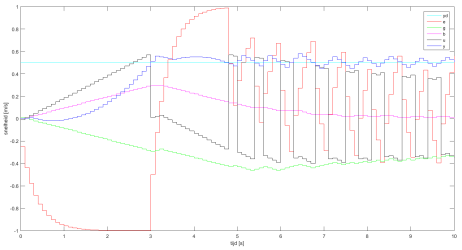
In Tabel 1 is te zien dat de gemiddelde fout kleiner wordt wanneer de e_b kleiner word. Dit bevestigt dat kleinere grenswaarden kleinere fouten geven.
De e_g daarentegen bereikt de kleinste fout rond de 0.4. Dit komt doordat er een vaak een kleine vertraging in het systeem zit. Wanneer y_k dichtbij y_k^d komt zal een regelaar met een kleine e_g nog naar y_k^d toe sturen. Maar de fout was in de praktijk al kleiner dan de regelaar aangaf en y_k zal aan y_k^d voorbij schieten. Bij een hoger e_g was het signaal op dat moment gelijk gebleven en zou y_k nog langzaam naar y_k^d toe gaan.

Tabel 1: de gemiddelde afwijking bij verschillende grenswaarden

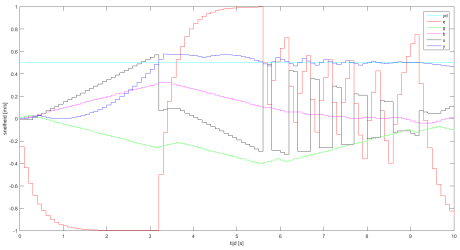
e_b	gemiddelde fout (m/s)	e_g	gemiddelde fout (m/s)
1	0.1986	1	0.5
0.9	0.1667	0.9	0.2133
0.8	0.1661	0.8	0.1674
0.7	0.1575	0.7	0.1548
0.6	0.1597	0.6	0.1456
0.5	0.1536	0.5	0.1400
0.4	0.1487	0.4	0.1357
0.3	0.1480	0.3	0.1385
0.2	0.1469	0.2	0.1389
0.1	0.1449	0.1	0.1449
0	0.1466	0	0.1487

De bias moet de trage dynamica aangeven. Het kijkt niet naar iedere verandering maar zoekt naar de grote lijnen. Het zou het goed zijn om de e_b zo klein mogelijk te maken. Op deze manier kan de bias een grote precisie bereiken.
Dit is in tegenstelling tot de gain die de snelle dynamica nabootst. Om hier niet te veel last van chatter te krijgen moet de e_g hoger zijn.

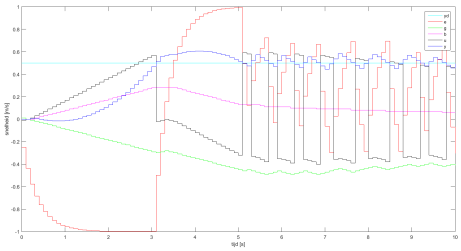
In de grafieken van Figuur 10 is ook zichtbaar dat het minder lang duurt voordat het systeem in een steady-state komt als e_g klein is. Dit is te verklaren met hulp van de fout. Op een gegeven moment wordt de fout even nul om vervolgens door te schieten en positief te worden. Dit zorgt er voor dat g_k even niet mag veranderen. Als e_g groter is, is dit moment langer.
Wanneer e_g groter wordt zijn er minder fluctuaties in het signaal als het in de steady state is. Dit heeft weer te maken met de eerder genoemde vertraging
e_b heeft een minder grote invloed. Maar het is wel zichtbaar dat een lage e_b bij een hoge e_g de meest preciese steady-state heeft.



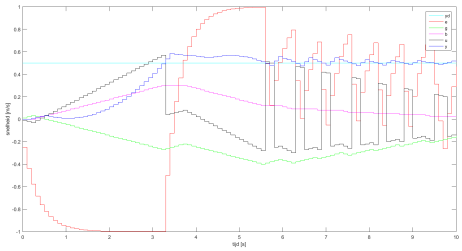
(a) lage e_g en e_b



(b) hoge e_g en lage e_b



(c) lage e_g en hoge e_b



(d) hoge e_g en e_b

Figuur 10: resultaten één wiel model voor verschillende e_g en e_b

5.1.2 Stapgrootte

De stapgrootten hebben een vergelijkbare trade-off. Aan de ene kant geeft een kleinere stapgrootte meer precisie. Met kleinere stappen kan y_k dichterbij y_k^d komen zonder er aan voorbij te schieten.
Aan de andere kant zorgt een kleinere stapgrootte er ook voor dat het langer duurt om bij de gewenste waarde uit te komen. Dit is vooral problematisch wanneer y_k^d niet constant is. In dat geval zou y_k^d al kunnen veranderen voordat y_k in de buurt kwam. Het gevaar is dat y_k en y_k^d dan niet meer op elkaar lijken.

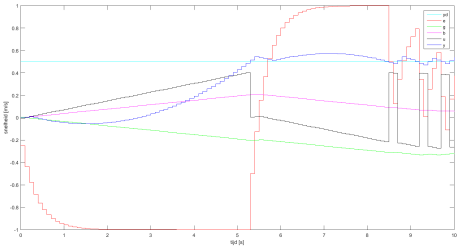
In Tabel 2 worden deze stellingen bevestigd. De kleinste afwijking wordt gevonden bij een stapgrootte van 1/16 en 1/64 respectievelijk voor δ_g en δ_b.

Tabel 2: de gemiddelde afwijking bij verschillende stapgrootten

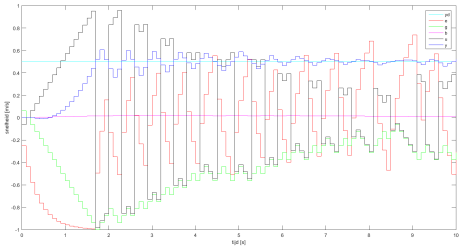
δ_g	gemiddelde fout (m/s)	δ_b	gemiddelde fout (m/s)
1	0.0975	1	0.1500
1/4	0.0956	1/4	0.2024
1/16	0.1140	1/16	0.2258
1/64	0.1388	1/64	0.1340
1/256	0.1915	1/256	0.1665
1/1024	0.3826	1/1024	0.1900
1/4096	0.4645	1/4096	0.1954

Naast de verbeterde precisie, helpt een kleinere δ_b tegen chatter. Dit komt voornamelijk doordat de bias minder snel kan veranderen en een stabielere basis vormt voor het signaal.
Tot bepaalde hoogte zorgt een kleiner δ_g ook voor een stabieler signaal. Samen zorgen deze stapgrootten er voor dat een steady-state eerder wordt bereikt.

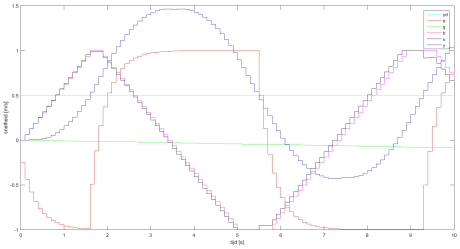
In Figuur 11 is te zien dat een lage δ_g er voor zorgt dat het heel lang duurt voordat het systeem in een steady-state komt.
Het is interessant om, in Figuur 11b, te zien dat de bias bijna niet verandert. Hoewel dit figuur een van de meest preciezere steady-state signalen geeft is dit niet wenselijk.
Als er een verandering is in de wrijving of andere onbekenden in het systeem zou dit een grotere invloed hebben. De bias zorgt er namelijk voor dat er een lijn is die ongeveer hetzelfde blijft voor het uitgangssignaal dat nodig is, het geeft de trend aan.
De gain zou deze fouten op moeten vangen, zodat de bias ongeveer gelijk kan blijven en het systeem snel kan herstellen. In deze situatie daarentegen "vergeet" het systeem wat hij daarvoor geleerd heeft over hoe dit signaal te volgen omdat de gain moet veranderen en er geen basis is om naar terug te keren.



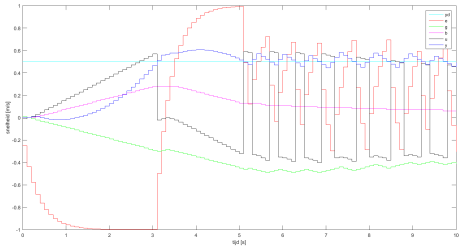
(a) lage δ_g en δ_b



(b) hoge δ_g en lage δ_b



(c) lage δ_g en hoge δ_b



(d) hoge δ_g en δ_b

Figuur 11: resultaten één wiel model voor verschillende δ_g en δ_b

5.1.3 laagdoorlaatfilter

De laatste waarde die ingesteld moet worden is de factor van het laagdoorlaatfilter. Een hogere waarde voor α betekent dat de oudere e_k meer meetellen dan de huidige e_k. Dit zorgt er voor dat er een verschil is tussen de fout als y_k om y_k^d heen oscilleert of er net bij in de buurt is gekomen.

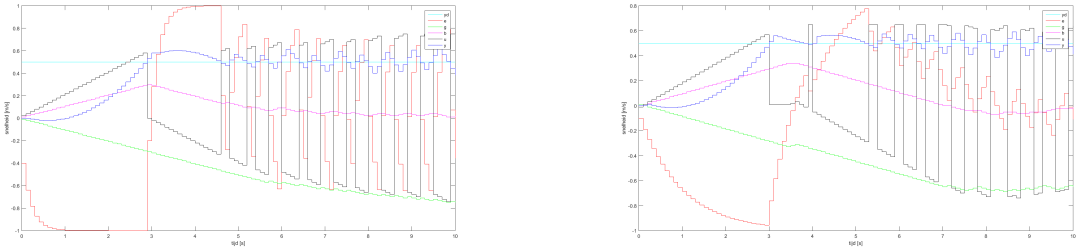
In het eerste geval zal het laagdoorlaat filter, in combinatie met de sign functies die helpen de bias en gain te bepalen, ervoor zorgen dat er geen veranderingen in het signaal komen. In het tweede geval zullen de bias en gain nog wel verbeterd worden.

Hiermee is het laagdoorlaatfilter de "ABAG"-regelaar ingebouwde manier om chatter tegen te gaan. In tabel 3 is dan ook te zien dat de gemiddelde afwijking kleiner wordt wanneer α groeit.

Tabel 3: De gemiddelde afwijking bij verschillende sterktes van het laagdoorlaatfilter

α	gemiddelde afwijking
1	1.6599
0.9	0.1431
0.8	0.1442
0.7	0.1445
0.6	0.1449
0.5	0.1456
0.4 tot 0	0.1506

In Figuur 12 is het nadeel te zien van een hoge α: het duurt veel langer voordat het systeem in een steady-state komt. Doordat er meer naar de "oude" fout word gekeken, zal het totaal minder snel veranderen als de afwijking van y_k naar y_k^d groter wordt. Als de fout minder groot is zal er ook een minder hevige reactie komen van de bias en gain.



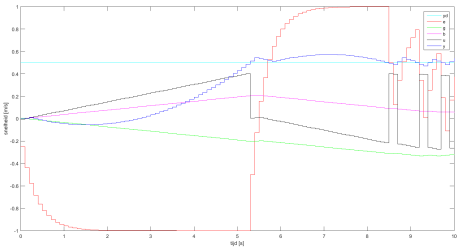
(a) lage α (b) hoge α

Figuur 12: resultaten één wiel model voor verschillende α

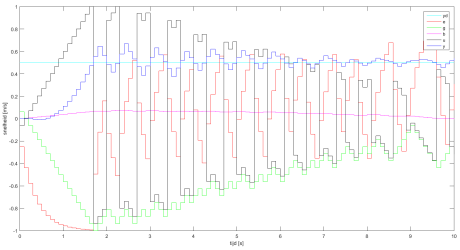
5.2 Samenwerking tussen de parameters

De stapgrootte en grenswaarden beïnvloeden elkaar ook. Het is niet zinnig om grotere stappen te maken dan de grenswaarden. Dit haalt het effect van de grenswaarden onderuit aangezien u_k zo snel verandert dat het niet binnen de grenzen hoeft te komen.
Andersom zorgen te kleine stapgrootten ervoor dat het heel lang duurt voordat de fout de grenswaarde kan bereiken, wat ook niet gewenst is.

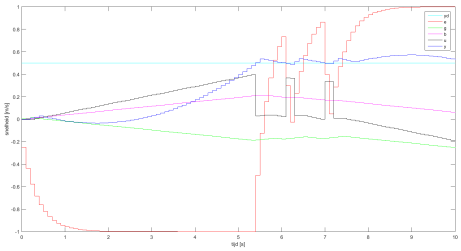
In Figuur 13 wordt er gekeken naar de invloed van δ_g en e_g op het systeem. Ook dit figuur bevestigt dat een lage δ_g erg late of geen steady-state geeft. Het laat ook zien dat een hoge e_g een preciezer signaal geeft, maar het heeft hier minder invloed.



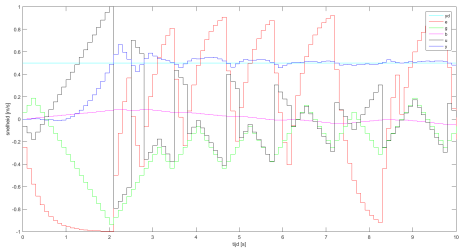
(a) lage δ_g en e_g



(b) hoge δ_g en lage e_g



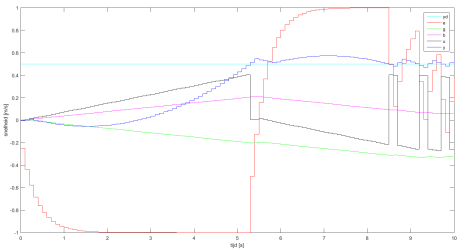
(c) lage δ_g en hoge e_g



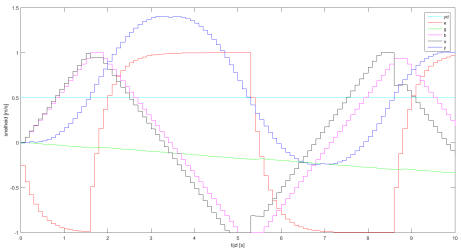
(d) hoge δ_g en e_g

Figuur 13: resultaten één wiel model voor verschillende e_g en δ_g

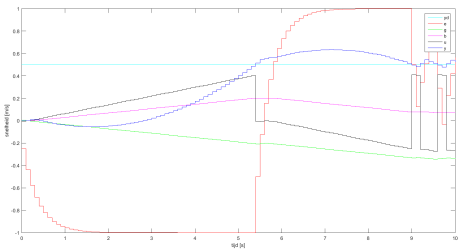
Figuur 14 worden de δ_b en e_b met elkaar vergeleken. Het word hier uit duidelijk dat e_b geen groot effect heeft op een lichte vertraging na. Een lagere δ_b geeft ook een vertraging maar dit effect is meer uitgesproken.



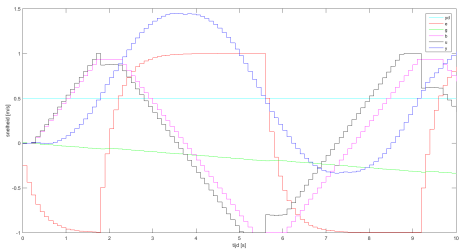
(a) lage δ_b en e_b



(b) hoge δ_b en lage e_b



(c) lage δ_b en hoge e_b



(d) hoge e_g en δ_b

Figuur 14: resultaten één wiel model voor verschillende e_b en δ_g

6 Conclusie

Het doel van het "ABAG"- algoritme is dat er alleen door middel van simpele berekeningen een regelaar wordt gemaakt die zorgt dat de fout van een systeem binnen een aangegeven marge blijft.
Dit wordt gedaan door middel van een bias die de grote lijnen, ofwel de trage dynamica, nabootst en een gain die de veranderingen opvangt, oftewel de snelle dynamica nabootst.
Het systeem kan worden beïnlvoed door 5 instelbare parameters die een trade-off vormen tussen preciezie, chatter en snel in de buurt van het gewenste signaal komen.
Uit de resultaten van de, in dit verslag gepresenteerde modellen, kan geconcludeerd worden dat het "ABAG"-algoritme niet alleen op de snelheidsbepaling van drones maar ook op de snelheids- en plaats bepaling van rijdende robots werkt. Het laatste model suggereert ook dat het algoritme gebruikt kan worden onafhankelijk van het aantal wielen mits goed geïmplementeerd.

7 Discussie

Zoals eerder genoemd in paragraaf 2.2 kon het originele algoritme niet gebruikt worden in de modellen. Het lijkt echter wel succesvol gebruikt te zijn in Franchi and Mallet [1]. Hoewel het probleem simpel genoeg is opgelost blijft nu nog onduidelijk waardoor het in het origineel wel werkt maar in deze experimenten niet.
Daarnaast blijft het de vraag of dit algoritme beter is dan bekendere systemen zoals een PID-regelaar. ABAG is zeker makkelijker om in te stellen aangezien de PID-regelaar uitgebreide experimenten nodig heeft. Maar waarschijnlijk heeft de PID-regelaar een grotere precisie aangezien ABAG gemaakt is om niet te verbeteren als de fout klein genoeg is. Hier zou meer onderzoek naar moeten worden gedaan.
De praktische bruikbaarheid van dit algoritme boven andere is dus nog niet bewezen.

Referenties

[1] Antonio Franchi and Anthony Mallet. Adaptive closed-loop speed control of BLDC motors with applications to multi-rotor aerial vehicles. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, (978):5203–5208, 2017. ISSN 10504729. doi: 10.1109/ICRA.2017.7989610. URL `http://ieeexplore.ieee.org/document/7989610/`.

[2] MathWorks. Simulink. URL `https://nl.mathworks.com/products/simulink.html`.

[3] J. L. (James L.) Meriam and L. G. (L. Glenn) Kraige. *Dynamics*. Wiley, 2012. ISBN 9781118083451.

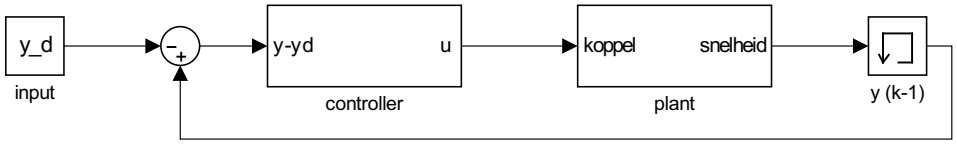
[4] Maria Isabel Ribeiro and Pedro Lima. KINEMATICS MODELS OF KINEMATICS MODELS OF MOBILE ROBOTS MOBILE ROBOTS MOBILE ROBOTICS course. 2002. URL `http://users.isr.ist.utl.pt/~mir/cadeiras/robmovel/Kinematics.pdf`.

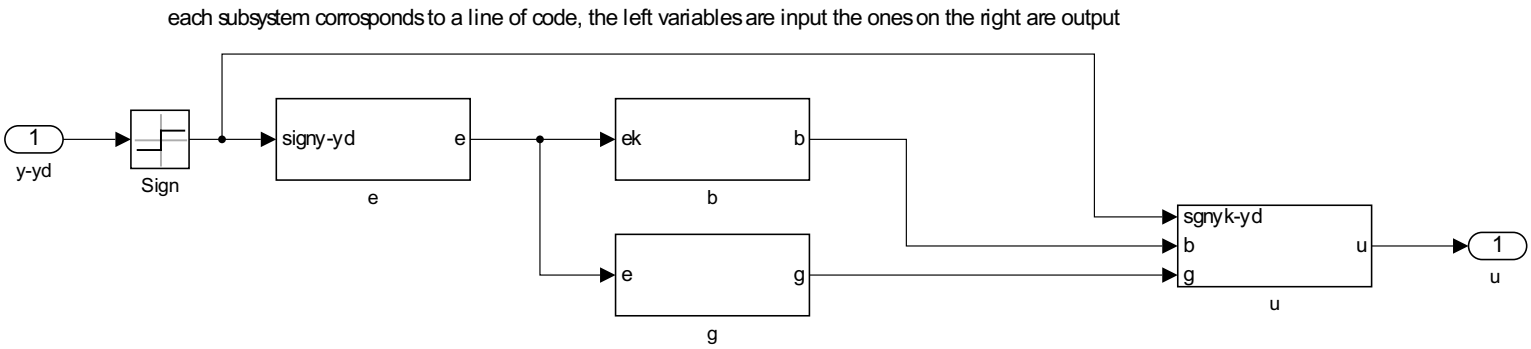
A modellen

Hieronder zullen alle gebruikte modellen volgen. De modellen staan in volgorde waarin ze genoemd zijn in het verslag en zijn waar mogelijk hetzelfde gehouden. Daarom zullen alleen de veranderde delen getoond worden na het eerste model.

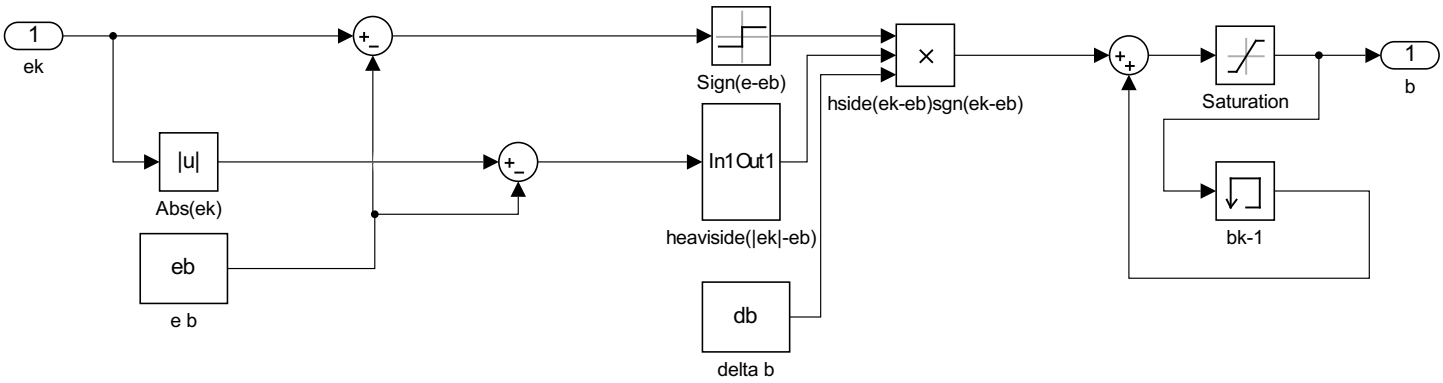
Bij elk model is als eerste het overzicht gegeven. Daarna wordt de inhoud van de "subsystems", dit zijn de rechthoekige blokken, getoond. Sommige "subsystems" hebben ook weer een "subsystem" in zich. In dit geval wordt eerst elke niveau naar beneden laten zien, daarna de volgorde van links naar rechts.

A.1 model voor één wiel

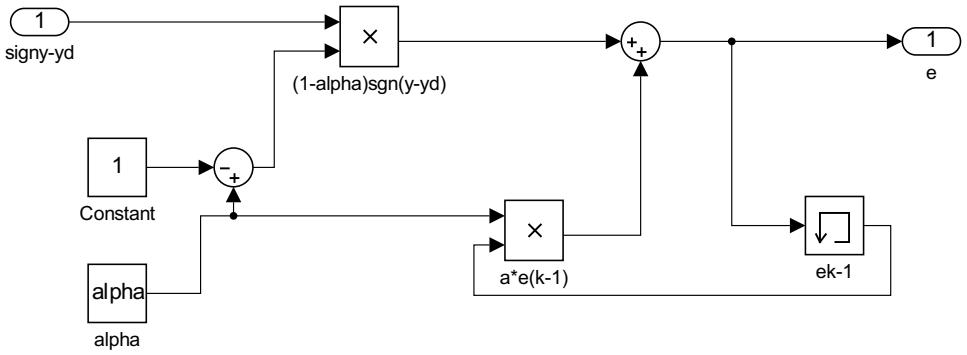



each subsystem corrosponds to a line of code, the left variables are input the ones on the right are output
1
y-yd
Sign
signy-yd
e
e
ek
b
b
e
g
g
sgnyk-yd
b
g
u
u
1
u

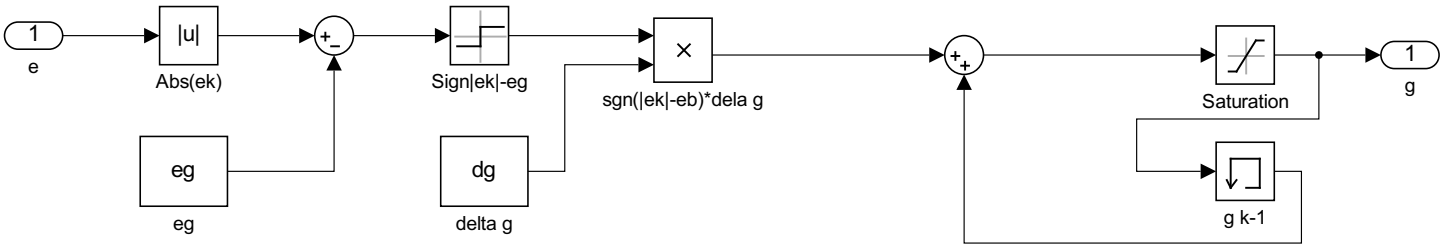
bk=sat[0,1](b(k-1)+db hside(|ek|-eb)sgn(ek-eb))

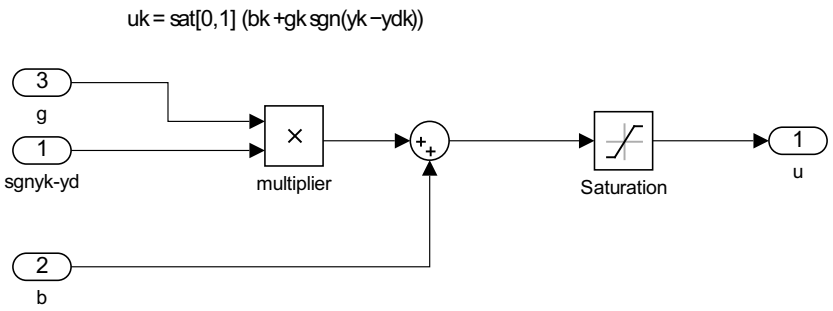


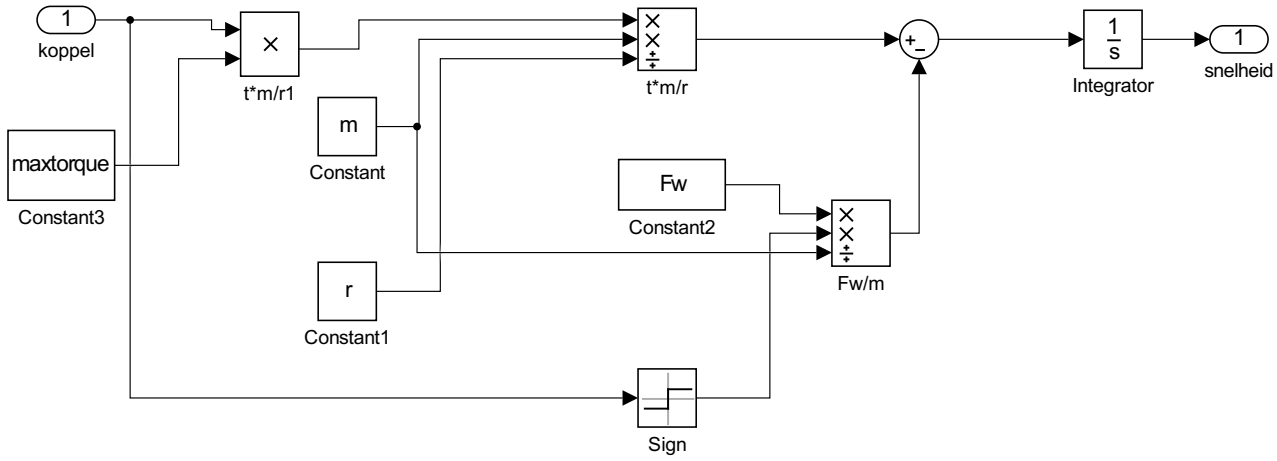
ek=alpha e(k-1) +(1-alpha)sgn(y-yd)



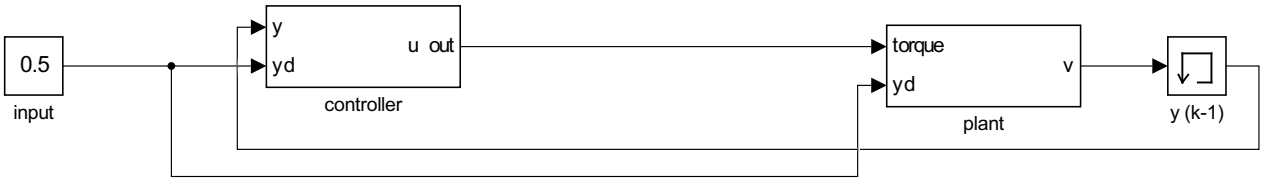
gk = sat[0,1] (g(k−1) +dg sgn(|ek|− eg))

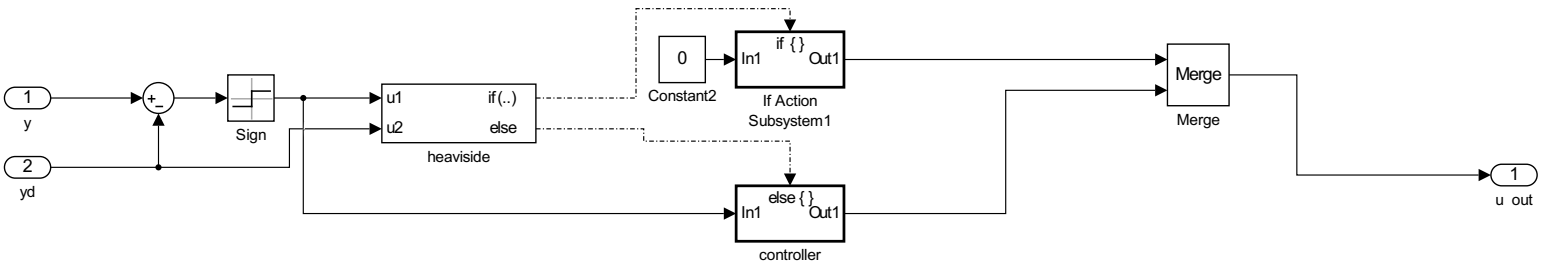



uk = sat[0,1] (bk +gk sgn(yk −ydk))
3
g
1
sgnyk-yd
multiplier
2
b
Saturation
1
u

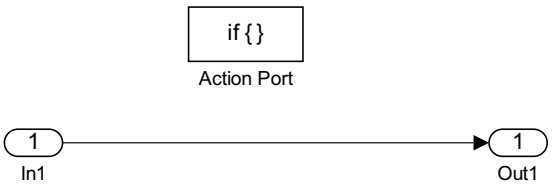

1
koppel
t*m/r1
m
Constant
maxtorque
Constant3
r
Constant1
t*m/r
Fw
Constant2
Fw/m
Sign
1
s
Integrator
1
snelheid

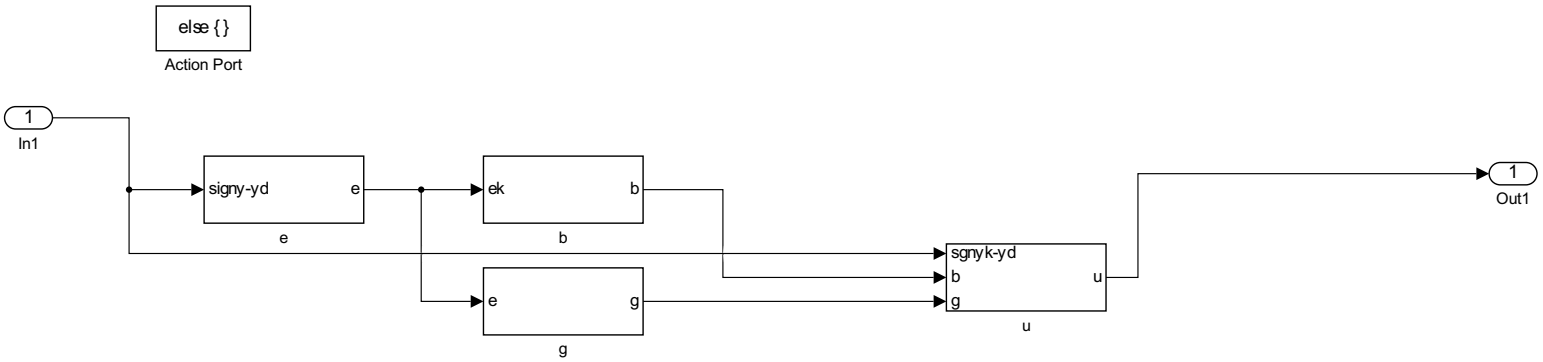
A.2 model met een passieve rem



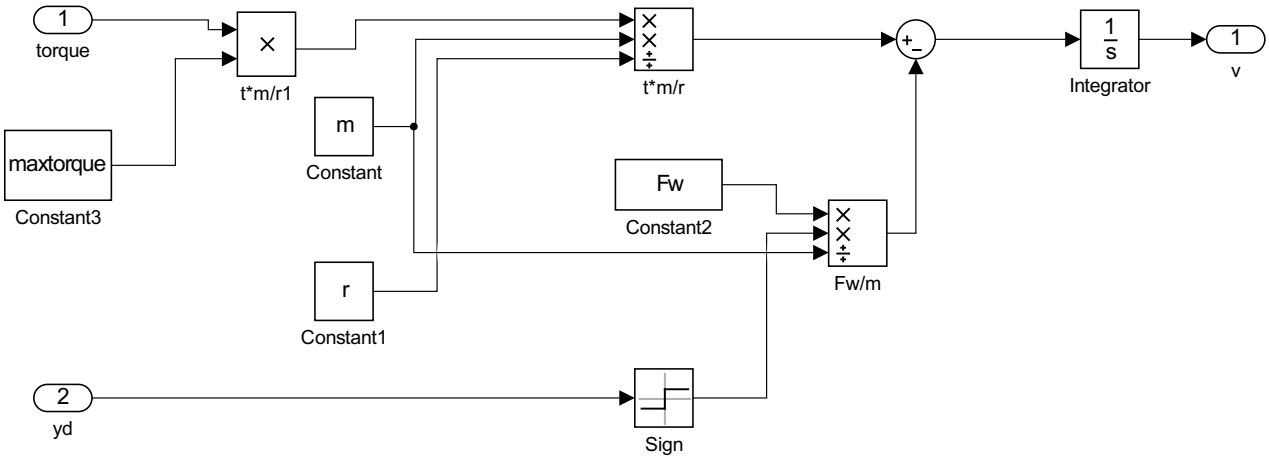


if ((u1 > 0)&(u2>0))|((u1<0)&(u2<0)) so if the |y|>|yd| than the system outputs zero
else it will normaly run the controller

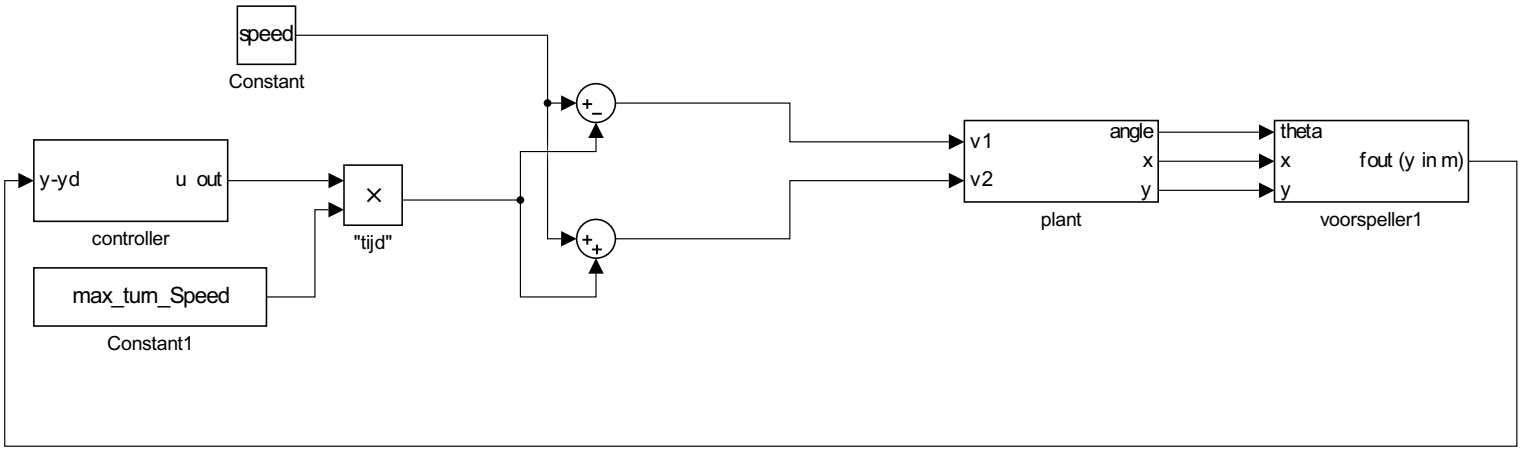

if { }
Action Port
1
In1
1
Out1

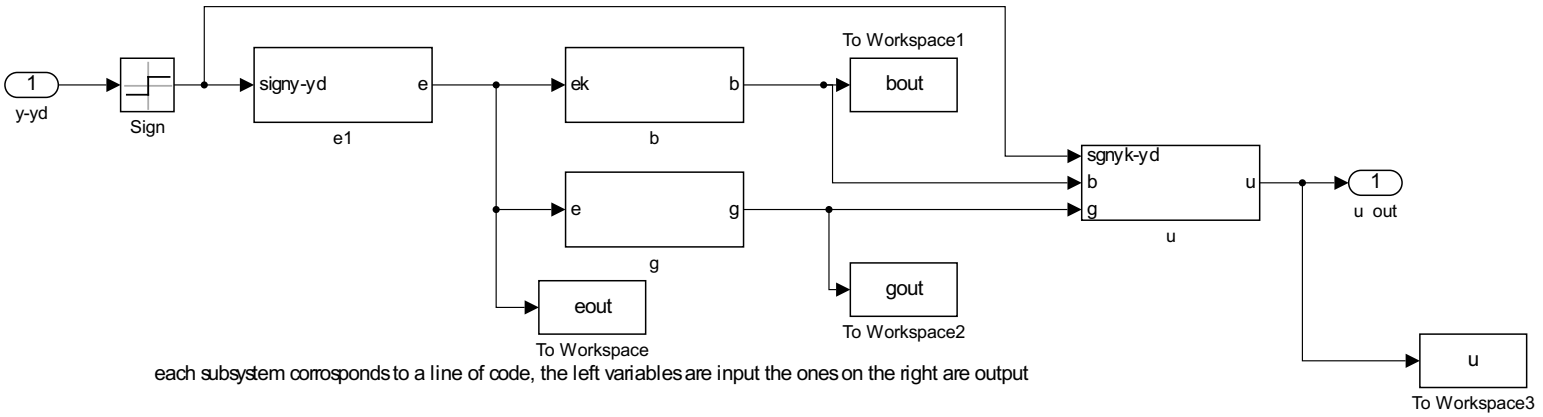


each subsystem corrosponds to a line of code, the left variables are input the ones on the right are output


1
torque
t*m/r1
maxtorque
Constant3
m
Constant
r
Constant1
t*m/r
Fw
Constant2
Fw/m
+
−
1
s
Integrator
1
v
2
yd
Sign

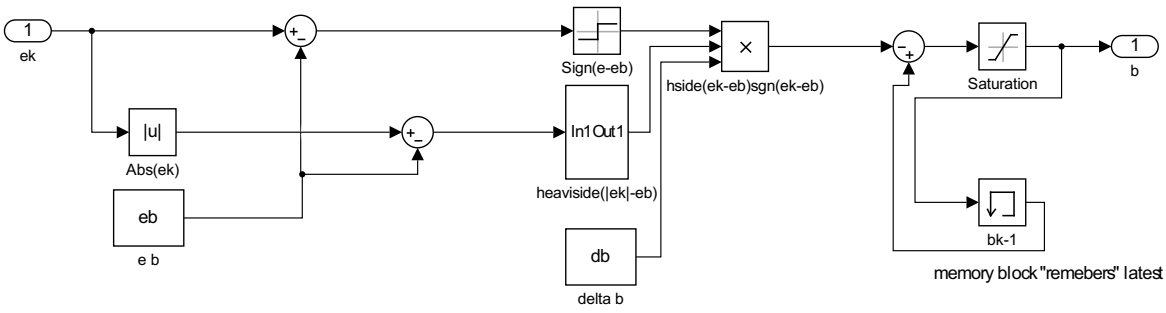
A.3 model voor twee wielen



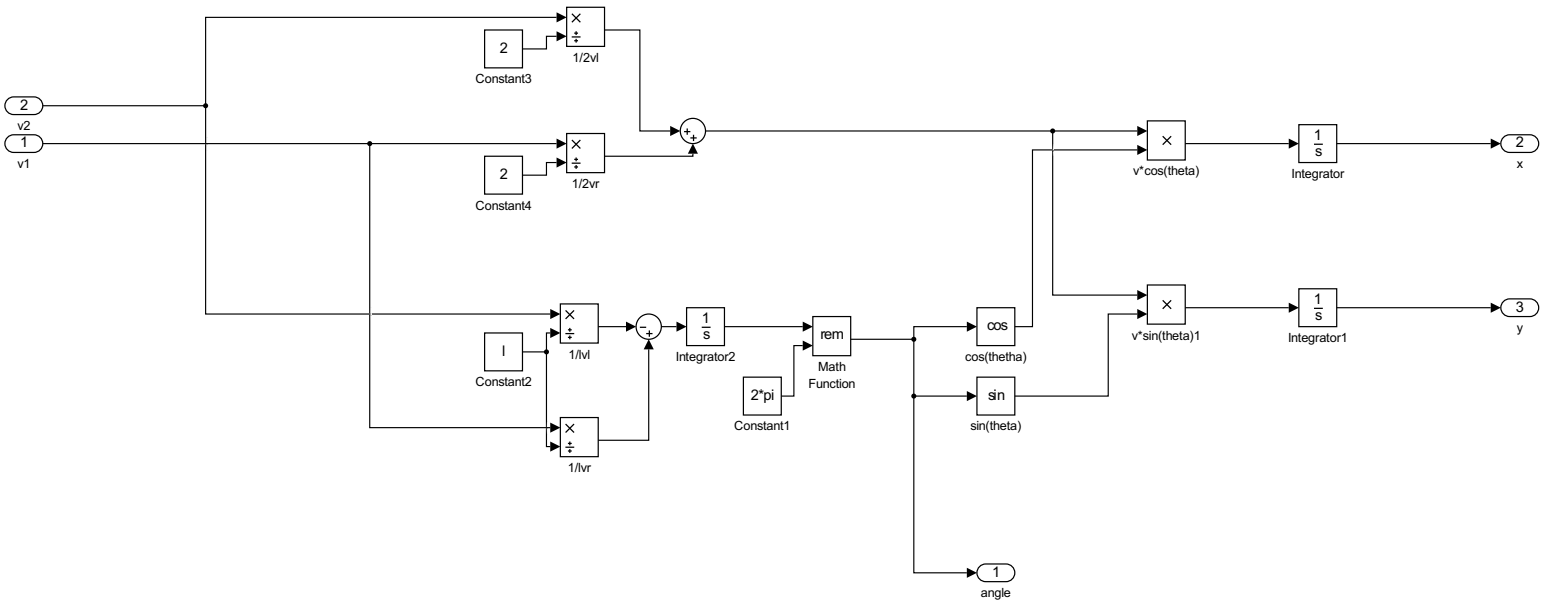


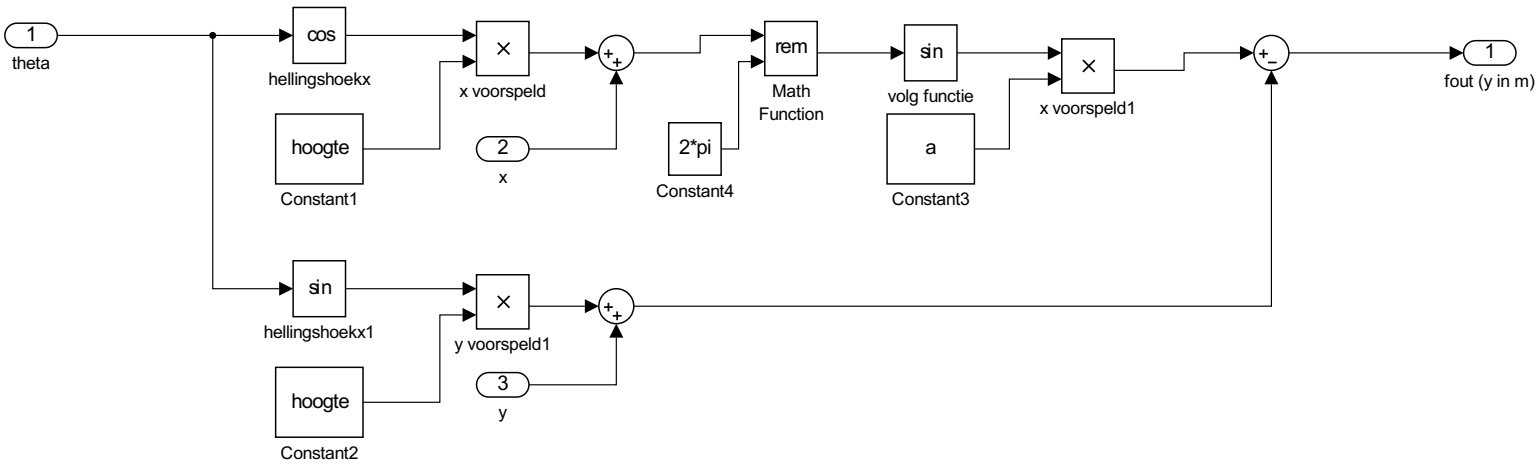
each subsystem corrosponds to a line of code, the left variables are input the ones on the right are output

bk=sat[0,1](b(k-1)+db hside(|ek|-eb)sgn(ek-eb))



memory block "remebers" latest output so it can be passed on to the next time step


2
v2
1
v1
2
Constant3
1/2vl
2
Constant4
1/2vr
1
Constant2
1/lvl
1/lvr
Integrator2
2*pi
Constant1
rem
Math
Function
cos
cos(thetha)
sin
sin(theta)
v*cos(theta)
v*sin(theta)1
Integrator
Integrator1
2
x
3
y
1
angle


1
theta
cos
hellingshoekx
hoogte
Constant1
x voorspeld
2
x
rem
Math
Function
2*pi
Constant4
sin
volg functie
a
Constant3
x voorspeld1
1
fout (y in m)
sin
hellingshoekx1
hoogte
Constant2
y voorspeld1
3
y