

Point-to-Point Navigatie

voor een kostenefficiënte robot in de landbouw

Axel WILLEKENS

Promotor: dr. ing. Geoffrey Ottoy

Masterproef ingediend tot het behalen van
de graad van master of Science in de
industriële wetenschappen: Elektronica - ICT
Elektronica

Co-promotor(en): dr. ir. Simon Cool (ILVO)

Academiejaar 2018 - 2019

©Copyright KU Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promotor(en) als de auteur(s) is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, kan u zich richten tot KU Leuven Technologiecampus Gent, Gebroeders De Smetstraat 1, B-9000 Gent, +32 92 65 86 10 of via e-mail iiw.gent@kuleuven.be.

Voorafgaande schriftelijke toestemming van de promotor(en) is eveneens vereist voor het aanwenden van de in deze masterproef beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Voorwoord

Gedurende deze masterproef kreeg ik de kans me te verdiepen in de robotica, een actueel onderwerp dat volledig bij mijn interesseveld aansluit.

Een wetenschappelijk en technisch project als dit had ik natuurlijk niet kunnen aanvatten zonder de kennis die ik gedurende de voorbije vier jaar aan de KU Leuven opdeed. Ook dank voor de praktische organisatie van dit project. Aan het Instituut voor Landbouw, Visserij en Voedingsonderzoek (ILVO), afdeling Agrotechniek, kwam ik terecht in een ambitieuze, innovatieve en professionele onderzoeksgroep met een enorme expertise in techniek en wetenschap.

Dr. ir. Simon Cool, onderzoeksingenieur aan het ILVO, leerde ik kennen als uitmuntend promotor. Steeds kon ik bij hem met mijn vragen terecht. Zijn kritische blik gaf ongetwijfeld aanleiding tot een verrijking van deze masterproef. Evenveel dank aan Dr. Ing. Geoffrey Ottoy, onderzoeker en docent aan de Technologiecluster Elektrotechniek (ESAT) KU Leuven. Voor het advies mijn thesis op een grondige wetenschappelijke manier aan te vatten ben ik hem nog steeds dankbaar.

Speciale dank aan Prof. Dr. Erdal Kayakan, actief aan de universiteit van Turing die me meer heeft bijgebracht over autonome navigatie en Prof. Dr. Ing Dejan Seatović, actief aan de Hochschule für Techniek Rapperswil, die me op weg zette met de implementatie in ROS. Ook dank aan Guido Kips, docent landmeten aan de KU Leuven die me veel bijleerde over verschillende GPS systemen en coördinaatreferentiesystemen.

...

Abstract

To reduce the labour cost and operating time on the field, there is an ongoing trend towards heavier and wider machinery in the agriculture. This, however, has a negative impact on soil quality, biodiversity and compaction, which result in yield reduction¹. Multiple small scale autonomous robots could solve these problems, while reducing the labour cost even more. To speed-up the ongoing introduction process of robotics in agriculture, ILVO² is building a small-scale sensing robot built with price-efficient hardware and (open-source) software. This article focuses on the development of a point-to-point navigation system for this robot, i.e., make it a self-driving robot.

The robot is equipped with a Real-Time Kinematic Global Positioning System (RTK-GPS) and an Inertial Measurement Unit (IMU). The Robot Operating System (ROS)³ is responsible for reading the GPS and IMU data and steering the robot in the desired direction using a point-to-point navigation algorithm, i.e., the robot navigates from one point to another from a set of predefined points (coordinates).

To that end, both a Fuzzy Logic⁴ and Sliding Mode Control (SMC)⁵ algorithm have been evaluated in simulations. The SMC-based control algorithm benefits an easy implementation and depends on very little parameters. The Fuzzy Logic algorithm is more complex to implement, but is more intuitive for the programmer. Because of the good stability and the limited set of implementation parameters the SMC-based algorithm is chosen to implement on the robot platform. This is tested under varying conditions (in the field) and the results, mainly the navigation error, are being compared to the simulation, i.e., less than 10 cm error once the robot is in a stable state.

Keywords: Autonomous point-to-point navigation, sliding Mode Control, Fuzzy logic, Lyapunov functions, PID-tuning

¹E. Fugelsnes, E. Lie, "Heavy agricultural machinery damages the soil," The Research Council of Norway, June 2006. [online:] https://www.forskningsradet.no/en/Newsarticle/Heavy_agricultural_machinery_damages_the_soil/1253966195787

²Instituut voor Landbouw-, Visserij- en Voedingsonderzoek

³ROS website: <http://www.ros.org/>

⁴C. C. Lee, "Fuzzy logic in control systems: fuzzy logic controller. II," in IEEE Transactions on Systems, Man, and Cybernetics, vol. 20, no. 2, pp. 419-435, March-April 1990.

⁵V. Utkin, J. Guldner, J. Shi, S. Ge, F. Lewis. Sliding Mode Control in Electro-Mechanical Systems. Boca Raton: CRC Press, 2009.

Inhoudsopgave

Voorwoord	v
Abstract	vii
Inhoud	xi
Figurenlijst	xv
Tabellenlijst	xvii
Symbolenlijst	xix
Afkortingenlijst	xxiii
1 Inleiding	1
1.1 Robotplatform	3
1.2 Structuur van de masterproef	4
2 Literatuurstudie en simulatie	5
2.1 Inleiding	5
2.2 Notaties en uitgangspunt navigatie algoritme	7
2.3 Opstellen van een kinematisch model	7
2.3.1 Berekening v_x en ω_z i.f.v. \dot{X} , \dot{Y} , $\dot{\theta}$	8
2.3.2 Kinematisch model in Simulink	9
2.4 Controle algoritmes	10
2.4.1 Sliding Mode Control (SMC)	10
2.4.2 Fuzzy PID	23
2.4.3 Type-2 Fuzzy Neural Network (T2FNN) in combinatie met Proportional Derivative (PD) controller	33
2.4.4 Vergelijking SMC versus Fuzzy PID algoritme	35

2.5 Sensoren	36
2.5.1 Kalman filtering model	36
2.5.2 Simulatie	40
2.5.3 Resultaten	42
2.5.4 Besluit	47
2.6 Coördinatensystemen	49
3 Implementatie	51
3.1 Inleiding	51
3.2 Hardware	52
3.3 Software	53
3.3.1 reach_rs_driver	54
3.3.2 measure_gps_cos	54
3.3.3 imu_node	55
3.3.4 imu_driver	55
3.3.5 point_creator	56
3.3.6 odometry_source	58
3.3.7 drive_controller	59
3.3.8 motor_driver	62
3.3.9 keyboard_steering	62
4 Evaluatie	65
4.1 Inleiding	65
4.2 Methodiek bepaling fout	65
4.2.1 Rechtlijnig traject	65
4.2.2 Niet-rechtlijnig traject	68
4.3 Evaluatie SMC algoritme	69
5 Algemene Conclusie en Future Work	73
5.1 Algemene Conclusie	73
5.2 Future Work	74
A Lyapunov functies	77
A.1 Definitie	77
A.2 Voorbeeld	77
A.3 Gebruikte stabiliteitscriterium en theorema	78

B Sliding Mode Control (SMC)	79
B.1 Eenvoudig voorbeeld	79
B.2 Algemeen	80

Lijst van figuren

1.1 Illustratie precisielandbouw cyclus	2
1.2 Voorbeelden van bestaande robotplatformen	2
2.1 Robot in groot inertiaal assenstelsel en naamgeving	7
2.2 Kinematisch model non-holonomic robot	8
2.3 Kinematisch model gebruikt in Simulink	9
2.4 Toekenning coördinaatstelsels robot bij gebruik van SMC	11
2.5 Illustraties ter verduidelijking van vergelijking 2.2	13
2.6 Principe Sliding Mode Control	15
2.7 Saturatie functie gekozen als switching mode functie	15
2.8 Simulaties voor startpositie $P = (0, 0, -\frac{\pi}{4})^T$	17
2.9 Simulaties voor startpositie $P = (-1.5, -1, \frac{3\pi}{4})^T$	18
2.10 Simulaties voor startpositie $P = (0, 0, -\frac{\pi}{4})^T$ met storing	18
2.11 Simulaties voor startpositie $P = (0, 0, -\pi/4)^T$ na invoering parameter $k_3 = 1$	20
2.12 Simulatie SMC controlealgoritme in Simulink	21
2.13 Simulatie SMC controlealgoritme met storingen in Simulink	22
2.14 Illustratie ter verduidelijking van de terminologie in de sectie Fuzzy PID	23
2.15 Gekozen input MF's voor de hoekfout θ_e en de afstandsfout d_e	26
2.16 Gekozen output MF voor de snelheid v	26
2.17 Gekozen output MFs voor de verandering van de PID parameters dKp , dKi , dKd . .	27
2.18 Implementatie PID blok	28
2.19 Simulaties voor startpositie $P = (0, 0, -\frac{\pi}{4})^T$ met Fuzzy logica	28
2.20 Simulaties voor startpositie $P = (0, 0, -\frac{\pi}{4})^T$ met $v = 1$ m/s zonder Fuzzy netwerk . .	29
2.21 Simulaties voor startpositie $P = (-1.5, -1, \frac{3\pi}{4})^T$ met Fuzzy logica	29
2.22 Simulaties voor startpositie $P = (-1.5, -1, \frac{3\pi}{4})^T$ met $v = 1$ m/s zonder Fuzzy netwerk	30
2.23 Simulatie Fuzzy PID controlealgoritme in Simulink	31
2.24 Simulatie PID controlealgoritme met $v = 1$ m/s in Simulink	32

2.25 Implementatie T2FNN controlealgoritme door prof. Kayacan	33
2.26 Simulatie T2FNN controlealgoritme in Simulink	34
2.27 Simulaties voor startpositie $P = (0, 0, -\frac{\pi}{4})^T$ met SMC algoritme	35
2.28 Simulaties voor startpositie $P = (0, 0, -\frac{\pi}{4})^T$ met Fuzzy PID algoritme	35
2.29 Kalman filter responsie voor x	43
2.30 Kalman filter responsie voor v_x	43
2.31 Kalman filter responsie voor y	44
2.32 Kalman filter responsie voor v_y	44
2.33 Kalman filter responsie voor θ	45
2.34 Kalman filter responsie voor ω	45
2.35 Kalman filter responsie voor x met $\sigma_x = \sigma_y = 0.02$	46
2.36 Kalman filter responsie voor y met $\sigma_x = \sigma_y = 0.02$	47
2.37 Simulatie Kalman filter model Simulink	48
2.38 Links: geografisch coördinatenstelsel (longitude-latitude) – Rechts: cartografisch coördinatenstelsel (UTM)	50
3.1 Robot hardware implementatie	52
3.2 Blokdiagram hardware implementatie	53
3.3 Rosnodes (ovalen), Rostopics (rechthoeken, volle pijlen) en Rosservices (stippellijn pijlen)	54
3.4 Correctie magnetometer data. blauw: originele magnetometer data; rood: gecompenseerde data	55
3.5 Links: assenstelsel gebruikt bij UTM – rechts assenstelsel gebruikt door IMU	57
3.6 Illustratie parameters robottraject – (X_{curr}, Y_{curr}) huidige positie van de robot – (X_r, Y_r) trajectpunt ter bepaling van x_e , y_e en θ_e	58
3.7 Illustratie ter bepaling van v_l en v_r	60
3.8 Lineaire fit van de <i>DACwaarde</i> [/] in functie van de <i>Wielsnelheid</i> [$\frac{m}{s}$] ter bepaling van $m = 421.3$ en $b = -71.43$ ($DACwaarde = m \cdot Wielsnelheid + b$)	62
3.9 RobotTool webapplicatie	63
4.1 Verduidelijking berekening fout	66
4.2 Nauwkeurigheidstest: $d_{traject} = 18 \text{ mm}$, $d_{switch} = 50 \text{ cm}$, $arrayLength = 30$	70
4.3 Responsiviteitstest: Traject1	71
4.4 Responsiviteitstest: Traject2	71
4.5 Responsiviteitstest: printscren robotTool webapplicatie	72
A.1 Verloop $\frac{dV}{dt}$ voor $V(x_1, x_2) = a \cdot x_1^2 + b \cdot x_2^2$ waarbij $\frac{dV}{dt} < 0$	78

B.1	vb: Switching surface	79
B.2	vb: Staten diagram	79
B.3	Switching surface	80
B.4	Statendiagram	80

Lijst van tabellen

2.1 GPS systemen met hun nauwkeurigheid	49
3.1 Meting ter bepaling van het verband tussen de DAC waarde in functie van de snelheid $\left[\frac{m}{s} \right]$	61

Lijst van symbolen

Algemeen

Symbol	Omschrijving	Eenheid
\vec{V}_G	Lineaire snelheidsvector van het zwaartepunt in het groot inertiaal assenstelsel	[/]
\vec{v}	Lineaire snelheidsvector van het zwaartepunt in het klein lokaal assenstelsel	[/]
$\vec{\omega}$	Hoeksnelheidsvector van het zwaartepunt in het klein lokaal assenstelsel	[/]
P_{index}	Coördinaat of punt	[/]
v_l	Snelheid van de linkse wielen van de robot	[m/s]
v_r	Snelheid van de rechtse wielen van de robot	[m/s]
ϕ	Roll – oriëntatie van de robot in het groot inertiaal assenstelsel	[rad]
ψ	Pitch – oriëntatie van de robot in het groot inertiaal assenstelsel	[rad]
R	Rotatiematrix	[/]
σ	Standaard afwijking	[/]

Controle algoritmes

Symbol	Omschrijving	Eenheid
X	X-positie robot in het groot inertiaal assenstelsel	[m]
X_r	Gewenste X-positie van de robot in het groot inertiaal assenstelsel	[m]
x_e	Fout van de huidige x-positie van de robot (x) t.o.v. de gewenste x-positie (x_r) in het klein lokaal assenstelsel	[m]
Y	Y-positie robot in het groot inertiaal assenstelsel	[m]
Y_r	Gewenste X-positie van de robot in het groot inertiaal assenstelsel	[m]
y_e	Fout van de huidige y-positie van de robot (y) t.o.v. de gewenste y-positie (y_r) in het klein lokaal assenstelsel	[m]
v_X	Snelheid van de robot in de X-richting in het groot inertiaal assenstelsel	[m/s]
v_x	Snelheid van de robot in de x-richting in het klein lokaal assenstelsel	[m/s]
v_Y	Snelheid van de robot in de Y-richting in het groot inertiaal assenstelsel	[m/s]
v_y	Snelheid van de robot in de y-richting in het klein lokaal assenstelsel	[m/s]

v	Lineaire snelheid van de robot – impliciet wordt hier de snelheid in de x-richting (v_x) bedoeld	[m/s]
θ	Yaw (heading) – oriëntatie van de robot in het groot inertiaal assenstelsel	[rad]
θ_r	Gewenste heading van de robot in het groot inertiaal assenstelsel	[rad]
θ_e	Fout van de huidige heading van de robot (θ) t.o.v. de gewenste heading (θ_r) in het groot inertiaal assenstelsel	[rad]
ω	Hoeksnelheid van de robot – impliciet wordt hier de hoeksnelheid rond de z-as (ω_z) bedoeld	[rad/s]
ω_r	Gewenste hoeksnelheid van de robot (requested corner velocity) – impliciet wordt hier de hoeksnelheid rond de z-as ($\omega_{z,r}$) bedoeld	[rad/s]
P	Huidige staat van de robot in het groot inertiaal assenstelsel (posture)	[/]
u	Actie die de robot verwacht wordt aan te nemen	[/]
P_r	Gewenste staat in het groot inertiaal assenstelsel (requested posture)	[/]
p_e	Fout die de huidige staat maakt t.o.v. de gewenste staat in het klein lokaal assenstelsel (error posture)	[/]
$d_{traject}$	Afstand tussen trajectpunten	[m]
d_{switch}	Afstand tot waarop de robot een trajectpunt nadert alvorens het trajectpunt wordt geüpdate	[m]
k_1, k_2, k_3	Parameters SMC algoritme	[/]
d_e	Afstandsfout in het groot inertiaal assenstelsel, berekend als de kortste afstand tussen de positie van de robot en het traject	[m]
μ	Membership functie voor Fuzzy logica	[/]

Kalman filtering

Symbol	Omschrijving	Eenheid
X_k	X-positie robot in het groot inertiaal assenstelsel op tijdstip $t = k \cdot T_s$	[m]
$X_{GPS,k}$	X-positie, gemeten door de GPS, in het groot inertiaal assenstelsel op tijdstip $t = k \cdot T_s$	[m]
Y_k	Y-positie robot in het groot inertiaal assenstelsel op tijdstip $t = k \cdot T_s$	[m]
$Y_{GPS,k}$	Y-positie, gemeten door de GPS, in het groot inertiaal assenstelsel op tijdstip $t = k \cdot T_s$	[m]
$v_{X,k}$	Snelheid van de robot in de X-richting in het groot inertiaal assenstelsel op tijdstip $t = k \cdot T_s$	[m/s]
$v_{X,GPSt,k}$	Snelheid van de robot in de X-richting, gemeten door de GPS, in het groot inertiaal assenstelsel op tijdstip $t = k \cdot T_s$	[m/s]
v_r	Gewenste lineaire snelheid van de robot (requested velocity) – impliciet wordt hier de snelheid in de x-richting ($v_{x,r}$) bedoeld	[m/s]
$v_{k,actie}$	Lineaire snelheid die de robot verwacht wordt aan te nemen op tijdstip $t = k \cdot T_s$ – impliciet wordt hier de snelheid in de x-richting ($v_{x,k,actie}$) bedoeld	[m/s]

$v_{Y,k}$	Snelheid van de robot in de Y-richting in het groot inertiaal assenstelsel op tijdstip $t = k \cdot T_s$	[m/s]
$v_{Y,GPSt,k}$	Snelheid van de robot in de Y-richting, gemeten door de GPS, in het groot inertiaal assenstelsel op tijdstip $t = k \cdot T_s$	[m/s]
θ_k	Yaw (heading) – oriëntatie van de robot in het groot inertiaal assenstelsel op tijdstip $t = k \cdot T_s$	[rad]
$\theta_{IMU,k}$	Yaw (heading) – oriëntatie van de robot, gemeten door de IMU, in het groot inertiaal assenstelsel op tijdstip $t = k \cdot T_s$	[rad]
ω_k	Hoeksnelheid van de robot op tijdstip $t = k \cdot T_s$ – impliciet wordt hier de hoeksnelheid rond de z-as ($\omega_{z,k}$) bedoeld	[rad/s]
$\omega_{IMU,k}$	Hoeksnelheid van de robot, gemeten door de IMU, op tijdstip $t = k \cdot T_s$ – impliciet wordt hier de hoeksnelheid rond de z-as ($\omega_{z,IMU,k}$) bedoeld	[rad/s]
$\omega_{k,actie}$	Hoeksnelheid die de robot verwacht wordt aan te nemen op tijdstip $t = k \cdot T_s$ – impliciet wordt hier de hoeksnelheid rond de z-as ($\omega_{z,IMU,k}$) bedoeld	[rad/s]
T_s	Sampling interval	[s]
x_k	Huidige staat van de robot in het groot inertiaal assenstelsel op tijdstip $t = k \cdot T_s$	[/]
$\hat{x}_{n m}$	Schatting van de huidige staat van de robot in het groot inertiaal assenstelsel op tijdstip $t = n \cdot T_s$ waarbij alle $m \leq n$ observaties in rekening zijn gebracht	[/]
z_k	Kalman observatie van de huidige staat van de robot in het groot inertiaal assenstelsel op tijdstip $t = k \cdot T_s$	[/]
u_k	Actie genomen door de robot op tijdstip $t = k \cdot T_s$	[/]
F_k	Staten transitie model op tijdstip $t = k \cdot T_s$	[/]
B_k	Controle input model op tijdstip $t = k \cdot T_s$	[/]
Q_k	Covariantiematrix tussen de geobserveerde waarden t.g.v. procesruis op tijdstip $t = k \cdot T_s$	[/]
H_k	Observatiemodel op tijdstip $t = k \cdot T_s$	[/]
R_k	Covariantiematrix tussen de geobserveerde waarden t.g.v. observatieruis op tijdstip $t = k \cdot T_s$	[/]
K_k	Gain die in een volgende schatting bepaald in welke mate de geobserveerde waarden z_k en de vorige schatting $x_{k k-1}$ op tijdstip $t = k \cdot T_s$ doorwegen	[/]
$P_{n m}$	Covariantiematrix van de fout van het geschatte signaal voor op het tijdstip $t = n \cdot T_s$ waarbij alle $m \leq n$ observaties in rekening zijn gebracht	[/]
I	Eenheidsmatrix	[/]
μ	Gemiddelde	[/]

Lijst van afkortingen

9-DoF 9-Degrees Of Freedom. 3

ALS Autocovariance Least-Squares. 47

DAC Digitaal Analoog Convertor. 60

DGPS Differential Global Positioning System. 3, 49

ETRS European Terrestrial Reference System. 49

FEL Feedback Error Learning. 33

FLEPOS Flemish Positioning Service. 49

FNN Fuzzy Neural Network. 5, 6

GPGGA Global Positioning System Fix Data. 3

GPRMC Global Positioning Recommended minimum specific GPS/Transit data. 3

GPS Global Positioning System. 2, 49

ICR Instantaneous Centers of Rotation. 59

ILVO Instituut voor Landbouw, Visserij en Voedingsonderzoek. 2

IMU Internal Measurement Unit. 3

ITRS International Terrestrial Reference System. 49

LIDAR Laser Imaging Detection And Ranging. 2

MF Membership Function. 24

NMEA National Marine Electronics Association. 3

PD Proportioneel Differentieel. 6

PID Proportioneel Integrerend Differentieel. 5

ROS Robot Operating System. 3

RPI Raspberry Pi. 3

RPM Rotaties Per Minuut. 60

RTK Real Time Kinematic. 3, 49

SMC Sliding Mode Control. 5

SSM Site-Specific crop Management. 1

T2FNN Type-2 Fuzzy Neural Network. 33

TSK Takagi-Sugeno-Kang. 33

UTM Universal Transverse Mercator coordinate system. 49, 56

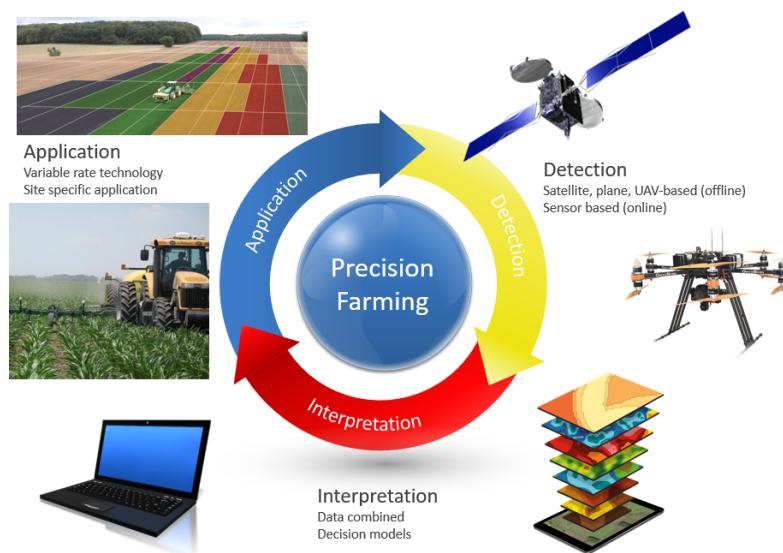
WGS84 World Geodetic System 1984. 49

Hoofdstuk 1

Inleiding

Precisielandbouw heeft tot doel de efficiëntie van de landbouw te verhogen door toepassing van technologie op het veld of in beschutte teelten. Hierbinnen past ook het thema robotisatie van landbouwvoertuigen. Robots zijn in staat zeer precies en op een intelligente wijze taken uit te voeren. Daarenboven kunnen ze tijdens het behandelen van bodem of gewas informatie vergaren over de bodem, het gewas en de omgeving die achteraf verder kan worden geanalyseerd. Met behulp van deze geanalyseerde gegevens kunnen betere bedrijfsbeslissingen worden genomen en kan een betere operationele handeling op plantniveau worden gekozen. Waar in de klassieke landbouw beslissingen worden genomen op basis van de behoefté van het veld, kan men hier tegemoet komen aan de individuële behoefté van de plant. Dit leidt tot een efficiëntere productie (meer opbrengst) met minder milieu-impact (minder inputs: meststoffen, gewasbeschermingsmiddelen, ...). In precisielandbouw onderscheidt men nog de volgende deeldomeinen: Site-Specific crop Management (SSM), Smart farming en Digital farming. SSM duidt op de optimalisatie van de inputs en de agronomische praktijken om de bodem en het gewas op elkaar af te stemmen zoals ze variëren op het veld. Hiermee overlapt het deeldomein Smart farming, waarbij de focus wordt gelegd op beslissingsondersteunende tools door middel van dataverwerking. Digital Farming wordt gezien als het consequent toepassen van de precisielandbouw- en van de smart farming approach, het connecteren van processen binnen en buiten de grenzen van de boerderij en het gebruik van web-gebaseerde data platformen in combinatie met big data analysemethoden. In figuur 1.1 is de cyclus van precisielandbouw weergegeven. Waar detectie, interpretatie en applicatie hier nog elk afzonderlijk gebeuren, zullen deze stappen door verdere robotisatie gebundeld kunnen worden in één systeem. Autonome navigatie is hierbij een hoeksteen.

Naast het aspect van precisielandbouw bieden robots een oplossing voor het probleem van bodemcompactie. Vandaag de dag is er in de landbouw een trend naar steeds bredere en zwaardere machines om de arbeidsuren en -kosten te drukken. Echter heeft dit een negatieve impact op de bodemkwaliteit, de biodiversiteit en de bodemcompactie wat aanleiding geeft tot een verminderde opbrengst. Meerdere kleine robots zouden dit probleem een halt toeroepen terwijl de arbeidsuren en -kosten toch laag blijven.



Figuur 1.1: Illustratie precisielandbouw cyclus

Het onderzoeksgebied agrotechniek aan het Instituut voor Landbouw, Visserij en Voedingsonderzoek (ILVO) doet onderzoek naar precisielandbouw. Sinds enkele jaren is het ILVO in samenwerking met hogescholen en universiteiten, voornamelijk in het kader van bachelor- en masterproeven, bezig met het ontwikkelen en testen van fieldrobotplatformen. Autonome navigatie bleek hierbij steeds een enorme en vaak onderschatte uitdaging. Navigatie op het land, in de serre of in de boomgaard kan gebeuren op verschillende manieren. Een eerste voorbeeld is visuele rijdetectie waarbij de robot tussen de plantenrijen rijdt op basis van camerabeelden. Een tweede voorbeeld maakt gebruik van Laser Imaging Detection And Ranging (LIDAR)'s, waarbij de robot nageeft tussen bomenrijen. Dit laatste is toegepast bij de Autonomous Orchard robot ontwikkeld aan de universiteit van Wageningen (figuur 1.2b). Echter kan er met dergelijke systemen niet geplant worden vanwege het ontbreken van de boom- of plantenrijen. Om de volledige autonomie van een robot op het veld te verzekeren is er dikwijls een Global Positioning System (GPS) nodig. Een systeem dat de autonome navigatie bewerkstelligt met onder andere een GPS is de Smart Weeding Robot van Ecorobotix (figuur 1.2c).



Figuur 1.2: Voorbeelden van bestaande robotplatformen

Gelijkwaardige projecten tonen aan dat de minimum vereiste hardware voor autonome navigatie een GPS en een Internal Measurement Unit (IMU) is [1][2]. Ook dit onderzoek beperkt zich hier toe. De meeste van de huidige tractoren maken reeds gebruik van GPS om bodembewerkingen of teeltmaatregelen uit te voeren. Wel moet er eerst rond het veld gereden worden alvorens het traject kan worden berekend, wat dus niet volledig automatisch is. Het bedrijf Octinion dat een aardbijplukrobot ontwikkelde voor in serres maakt gebruik van beacons voor de positionering van de robot. Dit is een goede oplossing voor in een serre, gezien het GPS-signal hier van mindere kwaliteit is, maar heeft als nadeel dat er beacons moeten worden uitgezet. De piste van de beacons is in dit onderzoek niet verder doorgaand. Er is ingezet op het gebruik van GPS.

Als uitgangspunt van deze masterproef worden verschillende algoritmes onderzocht voor autonome point-to-point navigatie van een robotplatform vertrekende van sensordata uit een GPS en een IMU. Verder zal gebruik gemaakt worden van open-source software om zo tot een goedkope implementatie voor autonome navigatie op het veld te komen. Door middel van simulatie wordt voor elk algoritme nagegaan of het stabiel is. Vervolgens wordt het meest belovende algoritme geïmplementeerd in software. De implementatie gebeurt in Robot Operating System (ROS) en wordt in de praktijk getest. ROS is een krachtig ontwikkelingsplatform voor robot software waarbij men in staat is hardware- en software functies te abstracteren tot nodes. ROS maakt vervolgens communicatie tussen deze nodes mogelijk wat de ontwikkelaar toelaat grote projecten modulair op te bouwen. Tijdens de praktijktjesten wordt gebruik gemaakt van een robotplatform dat reeds op het ILVO aanwezig is.

1.1 Robotplatform

Het robotplatform dat in deze masterproef zal worden benut, beschikt over vier wielen, elk aangedreven door een tweepolige borstelloze DC motor (type EC-max 40 serie van Maxon motor) die op zijn beurt wordt aangedreven door een motordriver (type 4-Q-EC EC DECV 50/5 305259 ook van Maxon motor). Verder is er op de robot een Real Time Kinematic (RTK) - GPS gemonteerd van de fabrikant *Emlid Reach* en een 9-Degrees Of Freedom (9-DoF)¹ IMU. Het autonome point-to-point navigatie algoritme zal draaien op een Raspberry Pi (RPI) waarop het besturingssysteem Lubuntu boot, met daarop *ROS kinetic*.

De *Emlid Reach* GPS module beschikt over meerdere uitvoerformaten: LLH (Longitude Latitude Height), XYZ (XYZ coördinaten met als oorsprong het centrum van de aarde) en de National Marine Electronics Association (NMEA) formaten waaronder Global Positioning System Fix Data (GPGGA) en Global Positioning Recommended minimum specific GPS/Transit data (GPRMC). Interessante data die hieruit kan worden gehaald is een timestamp, de latitude-longitude coördinaten en de quality flag die de variantie op de longitude-latitude coördinaten bepaalt. De quality flag bestaat uit een nummer van 1 tot 5 die de status en daarmee de nauwkeurigheid van de GPS weergeeft. Hierbij is 1: *Fix*; 2: *Float*; 3: *Reserved*; 4: *Differential Global Positioning System (DGPS)*; 5: *Single*. Bij *Fixed* bedraagt de nauwkeurigheid van de GPS $\approx 0.03\text{ m}$ en bij *Float* $\approx 0.3\text{ m}$. Bij

¹9 vrijheidsgraden in 3 dimensies (typisch versnelling, hoeksnelheid en magnetisch veld)

andere statussen is het toestel onnauwkeuriger dan 1 m. Enkel de status *Fix* is dus nauwkeurig genoeg om autonoom te navigeren. Om te verhinderen dat de robot over de gewassen rijdt, is het nodig te stoppen indien de GPS de vereiste nauwkeurigheid niet haalt.

Als IMU wordt gebruik gemaakt van een Sparkfun Razor IMU 9DOF. Uit de magnetometer haalt men de grootte van het magneetveld in de x-, y- en z-richting uitgedrukt in Gauss (1 Gauss = 10^{-4} Tesla). Hieruit wordt een quaternion berekend die overeen komt met de oriëntatie van de robot. Een quaternion is een uitbreiding van complexe getallen die dikwijls wordt aangewend voor relatieve plaatsbepaling. Ook data uit de gyro- en accelerometer worden benut.

Een heikel punt bij autonome navigatie is dat er rekening moet gehouden worden met de onnauwkeurigheden van de sensoren en actuatoren van het robotplatform. Plantenrijen op een veld staan doorgaans 50 tot 75 cm uit elkaar. Een autonome robot wordt verwacht dermate nauwkeurig te zijn opdat hij de teelt geen schade toebrengt. Een absolute nauwkeurigheid van bijvoorbeeld 10 cm zou ideaal zijn.

1.2 Structuur van de masterproef

Na Hoofdstuk 1 'Inleiding' volgt Hoofdstuk 2 'Literatuurstudie en simulatie' waarin zal worden ingegaan op verschillende algoritmes voor autonome point-to-point navigatie, maar ook kalmanfiltering van de sensordata wordt hierin besproken. Verder wordt elk algoritme uitgewerkt en gesimuleerd om de stabiliteit ervan na te gaan. Per besproken algoritme volgt een kort besluit waarin de stabiliteit en de voor- en nadelen worden toegelicht. Ook het principe van kalmanfiltering wordt hier geformuleerd en aan simulatie onderworpen. Het Hoofdstuk 3 'Implementatie' focust zich vooral op hoe het algoritme in ROS wordt geïmplementeerd. De bespreking van de nauwkeurigheid en de testen van het uitgekozen algoritme komen aan bod in het Hoofdstuk 4 'Evaluatie'. Hoofdstuk 5 betreft de 'Algemene Conclusie en Future Work'.

Hoofdstuk 2

Literatuurstudie en simulatie

2.1 Inleiding

Een belangrijk onderdeel van deze masterproef is het vinden van een passend point-to-point navigatie algoritme voor een robotplatform. Hiervoor zullen verschillende controlealgoritmes uit de literatuur op hun stabiliteit worden getest door middel van simulaties in Simulink (*Matlab 9.5.0.944444 (R2018b)*). Navigatie-algoritmes doen een beroep op wiskundige methoden om na te gaan of een systeem naar zijn evenwicht gaat. Hieronder volgen de twee voornaamste. Het bekendste voorbeeld is de leer van Proportioneel Integrerend Differentieel (PID)-regeling waarbij de stabiliteit van het systeem kan worden nagegaan door middel van een aantal stabiliteitscriteria zoals bijvoorbeeld het stabiliteitscriterium van Nyquist. De tweede wiskundige methode om te testen of een systeem stabiel is maakt gebruik van Lyapunov functies. Lyapunov functies behoren tot het studiedomein van de (evenwichtspunten bij) differentiaalvergelijkingen, waarbij tevens enkele stabiliteitscriteria zijn beschreven om na te gaan of een systeem al dan niet stabiel is [3]. In de literatuur zijn er een aantal methoden die bovenstaande stabiliteitstheorieën implementeren om een robuuste controlewet of controlealgoritme te bekomen. De voorbeelden die in dit onderzoek worden besproken zijn respectievelijk gebaseerd op Sliding Mode Control (SMC)[4], Fuzzy logica[5] en Fuzzy Neural Network (FNN)[6].

De meeste robots zijn *non-holonomic* robots. Om tot een goed algoritme voor point-to-point navigatie van een *non-holonomic* robot te komen moet er rekening worden gehouden met het niet-lineaire gedrag en de onzekerheden van het systeem. In de wiskunde wordt een *non-holonomic* system gedefinieerd als een systeem waarbij de overgang van staat *A* naar staat *B* afhankelijk is van de afgelegde weg [7]. Noemt men

- C de kromme die de afgelegde weg in de parameterruimte voorstelt
- \vec{F} de functie die de transitie van staat *A* naar staat *B* bewerkstelt,

dan is de lijnintegraal bij een *non-holonomic* system $\int_C \vec{F} \cdot d\vec{P}$ enkel te berekenen als men in elk punt van de kromme C de parameterwaarden kent. \vec{F} noemt men dan een niet conservatieve potentiaalfunctie.

Bij een holonomic system daarentegen is de overgang tussen twee staten onafhankelijk van de aangelegde weg. Kortom hier is de lijnintegraal $\int_C \vec{F} \cdot d\vec{P} = \rho(B) - \rho(A)$ en \vec{F} een conservatieve potentiaalfunctie.

Bij een robot zijn er maar enkele van het totaal aantal parameters in de parameterruimte te beïnvloeden en daardoor gekend. Andere parameters hangen af van omstandigheden (ruw terrein, robot slipt, ...) en hun invloed op de andere parameters in de parameterruimte is onbekend. Hieruit volgt dat vertrekende van staat A de exacte staat B niet kan berekend worden omdat het gevolgde traject onbekend is vanwege de parameters waarvan de invloed niet te bepalen is. Dit geeft wellicht aanleiding tot het niet-lineaire gedrag van het robot systeem. Kalmanfiltering tracht de volgende staat (staat B) te gaan voorspellen (geen exacte berekeningen). Toch moet dit nog steeds op regelmatige basis worden afgetoetst met behulp van sensordata [8].

Zoals eerder vermeld zijn de twee wiskundige methoden die vaak worden aangewend om de stabiliteit van een systeem na te gaan de PID-leer en Lyapunov functies. De meest bekende is de PID-leer. Het is niet mogelijk de robot te stabiliseren met een vast stel PID parameters (een vaste feedbacklus) [4]. PID regeling gaat immers uit van een lineair of gelineariseerd systeem terwijl een *non-holonomic* robot een niet-lineair systeem is. PID regeling zou volgens [5] wel mogelijk zijn indien de PID-parameters dynamisch veranderen naargelang de situatie (bijvoorbeeld afhankelijk van de afstand tot het punt naar waar gereden moet worden, afhankelijk van de snelheid, ...). Dit kan men bekomen door middel van Fuzzy logica. Een andere mogelijkheid is om een traditionele Proportioneel Differentieel (PD) controller te gebruiken in combinatie met een intelligente controller. Een Fuzzy Neural Network (FNN) kan men trainen om op die manier de juiste correcties te doen aan de output van de traditionele PD of PID controller [9].

De andere wiskundige methode die vaak gebruikt wordt om de stabiliteit van een systeem te controleren zijn Lyapunov functies. Hierbij wordt een SMC controller gebruikt om het systeem in een toestand te duwen waar het asymptotische stabiliteitscriterium van Lyapunov voldaan is [4]. SMC als trajectory tracking controlealgoritme heeft als belangrijkste voordeelen dat het een snelle respons biedt en dat het opgewassen is tegen de onzekerheden van het systeem en storingen van buitenaf. SMC is een niet-lineaire controlestrategie waarbij men een switching functie opstelt. De switching functie dient te convergeren naar nul, hierdoor bevindt het systeem zich in sliding mode. De nodige achtergrond voor deze studie van Lyapunov functies en van SMC staan gebundeld in respectievelijk Bijlage A en Bijlage B.

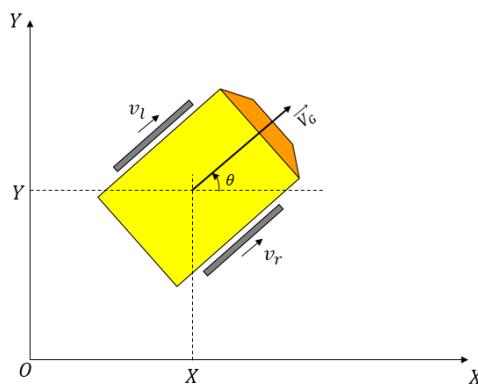
Verder maakt men in de literatuur nog onderscheid tussen time-based trajectory algoritmes en space-based trajectory algoritmes [6]. Bij beide algoritmes moeten bepaalde punten in de ruimte aangelegd worden, met als enige verschil dat bij een time-based trajectory algoritme deze punten binnen een bepaalde tijd moeten zijn gepasseerd. Naast de hoeksnelheid wordt daar dus ook de longitudinale snelheid geregeld door de controller. Dit houdt in dat de robot een ander gedrag vertoont op vlak van snelheid en versnelling wanneer het te bereiken punt verder of dichter ligt. Dit onderscheid wordt in deze masterproef gebruikt als typologie voor de verschillende controle-algoritmen. Voor de toepassing besproken in deze masterproef volstaat in principe een space-based trajectory algoritme.

Opmerking: De Simulink- en Matlab files, gebruikt in dit hoofdstuk, zijn terug te vinden op de githubpagina <https://github.com/axelwillekens/Point-to-Point-Navigation>.

2.2 Notaties en uitgangspunt navigatie algoritme

In deze masterproef wordt consequent omgegaan met naamgeving van vectoren in een assenstelsel. Wanneer de naam van de vector een hoofdletter bevat, duidt dit erop dat de vector zich in het groot inertiaal assenstelsel bevindt. Bestaat de naam van de vector enkel uit kleine letters dan bevindt deze zich in het klein lokaal assenstelsel waarvan de x-as samenvalt met de richting van de robot.

Algoritmes voor autonome navigatie vereisen minstens dat de sensoren de meest elementaire staat $P = (X, Y, \theta)^T$ van de robot kunnen bepalen. Waarbij X en Y respectievelijk de X-positie en de Y-positie van het zwaartepunt van de robot zijn in het groot inertiaal assenstelsel en θ de hoek tussen de heading (richting waarin de robot staat georiënteerd) en de X-as (figuur 2.1). Als resultaat geven dergelijke algoritmes de matrix $u = (v, \omega)^T$ met v de lineaire snelheid van de robot en ω de hoeksnelheid van de robot, dewelke de linker- en rechterwielsnelheid bepalen (v_L en v_R in figuur 2.1). Met de gebruikte hardware is men eveneens in staat nog extra informatie te vergaren zoals snelheidscomponenten v_X en v_Y en de hoeksnelheid ω van de robot. De staat van de robot kan dan als volgt worden uitgebreid $P = (X, v_X, Y, v_Y, \theta, \omega)^T$.



Figuur 2.1: Robot in groot inertiaal assenstelsel en naamgeving

2.3 Opstellen van een kinematisch model

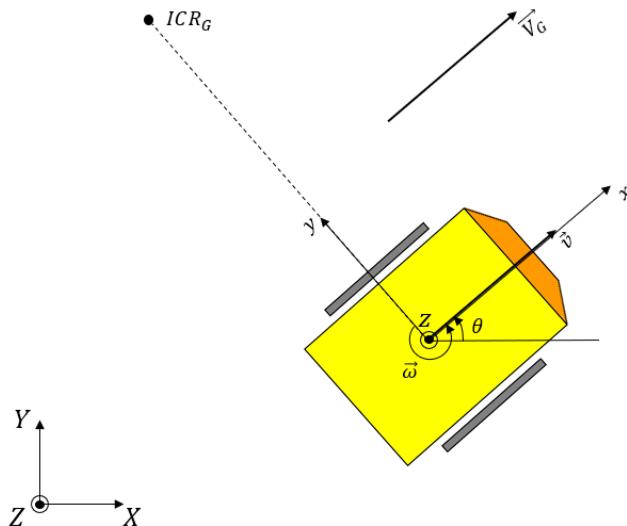
Uit elk algoritme voor autonome navigatie krijgt men een vector $u = (v, \omega)^T$, waarbij v de lineaire snelheid is en ω de hoeksnelheid is die de robot verwacht wordt aan te nemen. Deze snelheden moeten afgebeeld worden op v_L en v_R , respectievelijk de snelheid van de linkse en de snelheid van de rechtse wielen. Deze berekeningen werden gebaseerd op [10]. In figuur 2.2 is een vereenvoudigd kinematisch model gegeven van een *non-holonomic* robot.

De volgende veronderstellingen werden hierbij gemaakt:

1. Het massa zwaartepunt van de robot ligt op het geometrische zwaartepunt van de robot.
2. Indien er meerdere wielen zijn aan een zijde van de robot, dan draaien deze steeds aan dezelfde snelheid.

Op figuur 2.2 is er een groot inertiaal assenstelsel (X,Y) gedefinieerd en een klein lokaal assenstelsel (x,y) waarvan de x-as steeds in de rijrichting van de robot en de oorsprong steeds in het zwaartepunt van de robot ligt. Verder worden de volgende vectoren gedefinieerd:

- $\vec{V}_G = (\dot{X}, \dot{Y}, \dot{Z}) = (\dot{X}, \dot{Y}, 0)$ de snelheidsvector van het zwaartepunt in het groot inertiaal assenstelsel
- v_x afkomstig van de snelheidsvector van het zwaartepunt $\vec{v} = (v_x, v_y, v_z) = (v_x, 0, 0)$ in het klein lokaal assenstelsel.
- ω_z afkomstig van de hoeksnelheidsvector rond het zwaartepunt $\vec{\omega} = (0, 0, \omega_z)$ in het klein lokaal assenstelsel



Figuur 2.2: Kinematisch model non-holonomic robot

2.3.1 Berekening v_x en ω_z i.f.v. $\dot{X}, \dot{Y}, \dot{\theta}$

Stelt men vervolgens $P = (X, Y, \theta)^T$, dan kan men de afgeleide naar de tijd schrijven als \dot{P} in het groot inertiaal assenstelsel (X,Y): $\dot{P} = \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix}$

Beschouw verder de matrix V in het klein lokaal assenstelsel (x,y) : $V = \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix}$

Dan geldt $\dot{P} = R \cdot V$ met R de rotatiematrix die het assenstelsel in de positieve zin (tegen wijzerzin)

$$\text{verdraait: } R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix}$$

Gezien steeds geldt dat $v_y = 0$ vindt men:

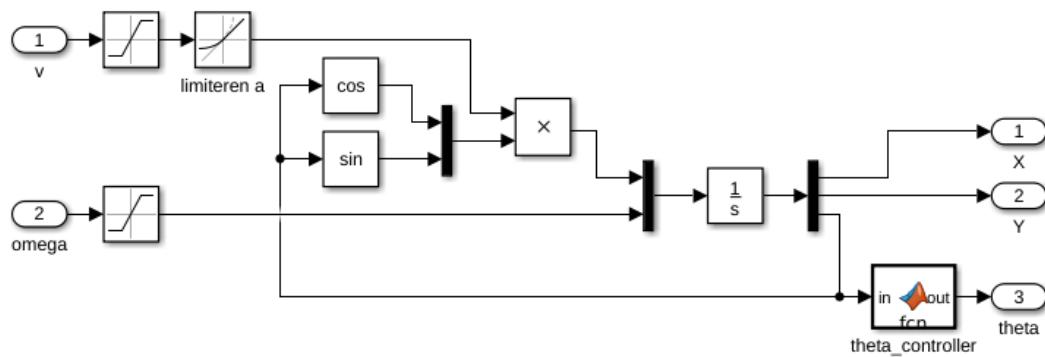
$$\begin{cases} \dot{X} = v_x \cdot \cos(\theta) \\ \dot{Y} = v_x \cdot \sin(\theta) \\ \dot{\theta} = \omega_z \end{cases} \Rightarrow \begin{cases} v_x = \frac{\dot{X}}{\cos(\theta)} \\ v_x = \frac{\dot{Y}}{\sin(\theta)} \\ \omega_z = \dot{\theta} \end{cases} \Rightarrow \begin{cases} v_x = \frac{\dot{X}}{\cos(\theta)} \\ \omega_z = \dot{\theta} \end{cases}$$

Opm: Wenst men een rotatie toe te passen in wijzerzin, dan bekomt men

$$R(-\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{ dit is nog van toepassing in subsectie 2.4.1.}$$

2.3.2 Kinematisch model in Simulink

Het kinematisch model van de robot in Simulink is gebaseerd op [5] en is te zien in figuur 2.3.



Figuur 2.3: Kinematisch model gebruikt in Simulink

Elke controlealgoritme zal uiteindelijk de lineaire snelheid v_x en de hoeksnelheid ω_z leveren. Het kinematisch model wordt hier beperkt tot de essentie. In de praktijk zullen v_x en ω_z gemapt worden naar v_l en v_r wat beschreven zal worden in sectie 3.3.7. Deze stap wordt bij de simulaties voor het testen van de controlealgoritmes buiten beschouwing gelaten gezien dit geen bijdrage levert tot het nagaan of het algoritme al dan niet stabiel is.

Het blok "limiteren a" is een rate limiter in Simulink, het stelt een limiet op de afgeleide van v , de versnelling dus. Ook de hoeksnelheid ω wordt gelimiteerd door een saturatieblok toe te voegen. Uit het "multiplier" blok komt $[v \cdot \cos(\theta), v \cdot \sin(\theta)] = [\dot{X}, \dot{Y}]$, die multiplexen met ω levert $[\dot{X}, \dot{Y}, \dot{\theta}]$. Na integratie bekomt men $[X, Y, \theta]$, de staat van de robot in het groot inertiaal assenstelsel. Het blok "theta_controller" zorgt ervoor dat θ wordt gemapt in het interval $[-\pi, \pi]$. De Matlab code van dit blok is terug te vinden in listing 2.1. Merk nog op dat de beginvoorwaarden van het integratieblok de beginpositie van de robot weergeeft.

```

1 function out = fcn(in)
2
3 out = mod(in, 2*pi);
4
5 % nodig voor interval -180, +180
6 if out > pi
7     out = out - 2*pi;
8 end

```

Listing 2.1: Matlab code "theta_controller" blok (figuur 2.3)

2.4 Controle algoritmes

2.4.1 Sliding Mode Control (SMC)

2.4.1.1 Algorimte

SMC als trajectory tracking controlealgoritme heeft als belangrijkste voordelen dat het een snelle respons biedt en het opgewassen is tegen de onzekerheden van het systeem en storingen van buitenaf. SMC is een niet-lineaire controlestrategie waarbij men een switching functie opstelt. De switching functie dient te convergeren naar nul, op deze manier bevindt het systeem zich in sliding mode. In sliding mode is geweten dat het systeem stabiel is. De switching mode functie wordt namelijk opgesteld vertrekende van een stabiele Lyapunov functie. Hieronder past men de methodiek van SMC toe zoals beschreven in [4].

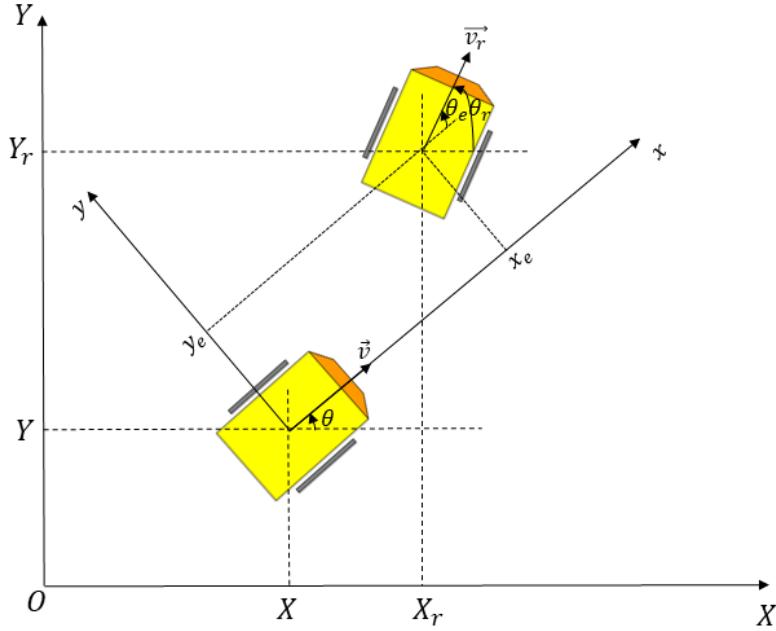
Beschouw figuur 2.4 waaruit men de volgende matrixen kan aflezen:

- De huidige positiematrix van de robot in het groot inertiaal assenstelsel (current posture):

$$P = \begin{bmatrix} X \\ Y \\ \theta \end{bmatrix}$$

- De gewenste positiematrix van de robot in het groot inertiaal assenstelsel (required posture):

$$P_r = \begin{bmatrix} X_r \\ Y_r \\ \theta_r \end{bmatrix}$$



Figuur 2.4: Toekenning coördinaatstelsels robot bij gebruik van SMC

- De errormatrix van de robot in het klein lokaal assenstelsel (figuur 2.4):

$$p_e = \begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_r - X \\ Y_r - Y \\ \theta_r - \theta \end{bmatrix} = \begin{bmatrix} \cos(\theta) \cdot (X_r - X) + \sin(\theta) \cdot (Y_r - Y) \\ -\sin(\theta) \cdot (X_r - X) + \cos(\theta) \cdot (Y_r - Y) \\ \theta_r - \theta \end{bmatrix}$$

- De snelheiderrormatrix van de robot in het klein lokaal assenstelsel:

$$\dot{p}_e = \begin{bmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{bmatrix} = \begin{bmatrix} \cos(\theta) \cdot (\dot{X}_r - \dot{X}) - \sin(\theta) \cdot \omega \cdot (X_r - X) + \sin(\theta) \cdot (\dot{Y}_r - \dot{Y}) + \cos(\theta) \cdot \omega \cdot (Y_r - Y) \\ -\sin(\theta) \cdot (\dot{X}_r - \dot{X}) - \cos(\theta) \cdot \omega \cdot (X_r - X) + \cos(\theta) \cdot (\dot{Y}_r - \dot{Y}) - \sin(\theta) \cdot \omega \cdot (Y_r - Y) \\ \omega_r - \omega \end{bmatrix} \quad (2.1)$$

$$= \begin{bmatrix} \omega [\cos(\theta)(Y_r - Y) - \sin(\theta)(X_r - X)] - [\sin(\theta)\dot{Y} + \cos(\theta)\dot{X}] + [\cos(\theta)\dot{X}_r + \sin(\theta)\dot{Y}_r] \\ -\omega [\cos(\theta)(X_r - X) + \sin(\theta)(Y_r - Y)] + [\sin(\theta)\dot{X} - \cos(\theta)\dot{Y}] + [\cos(\theta)\dot{Y}_r - \sin(\theta)\dot{X}_r] \\ \omega_r - \omega \end{bmatrix}$$

$$= \begin{bmatrix} y_e \omega - v + v_r \cos(\theta_e) \\ -x_e \omega + v_r \sin(\theta_e) \\ \omega_r - \omega \end{bmatrix} \quad (2.2)$$

Hierbij wordt de laatste stap nader toegelicht. Hieronder worden de gelijkheden tussen vol-

gende termen aangetoond:

$$\cos(\theta)(Y_r - Y) - \sin(\theta)(X_r - X) = y_e \quad (2.3)$$

$$\cos(\theta)(X_r - X) + \sin(\theta)(Y_r - Y) = x_e \quad (2.4)$$

$$\sin(\theta)\dot{Y} + \cos(\theta)\dot{X} = v \quad (2.5)$$

$$\sin(\theta)\dot{X} - \cos(\theta)\dot{Y} = 0 \quad (2.6)$$

$$\cos(\theta)\dot{X}_r + \sin(\theta)\dot{Y}_r = v_r \cos(\theta_e) \quad (2.7)$$

$$\cos(\theta)\dot{Y}_r - \sin(\theta)\dot{X}_r = v_r \sin(\theta_e) \quad (2.8)$$

- De gelijkheden 2.3 en 2.4 zijn af te leiden uit figuur 2.4.
- De termen 2.5 en 2.6, termen met betrekking tot v , zijn op te lossen met behulp van figuur 2.5a. Hieruit worden de volgende twee gelijkheden gehaald:

$$\begin{aligned} \sin(\theta) &= \frac{\dot{Y}}{v} \Leftrightarrow \dot{Y} = \sin(\theta) \cdot v \\ \cos(\theta) &= \frac{\dot{X}}{v} \Leftrightarrow \dot{X} = \cos(\theta) \cdot v \end{aligned}$$

Substitutie van twee bovenstaande vergelijkingen in de term van vergelijking 2.5 levert:

$$\begin{aligned} \sin(\theta)\dot{Y} + \cos(\theta)\dot{X} &= \sin(\theta)^2 \cdot v + \cos(\theta)^2 \cdot v \\ &= [\sin(\theta)^2 + \cos(\theta)^2] \cdot v \\ &= v \end{aligned}$$

Substitutie van twee bovenstaande vergelijkingen in de term van vergelijking 2.6 levert:

$$\begin{aligned} \sin(\theta)\dot{X} - \cos(\theta)\dot{Y} &= \sin(\theta)\cos(\theta) \cdot v - \cos(\theta)\sin(\theta) \cdot v \\ &= 0 \end{aligned}$$

- De termen 2.7 en 2.8, termen met betrekking tot v_r , zijn op te lossen met behulp van figuur 2.5b. Hieruit haalt men de volgende twee gelijkheden:

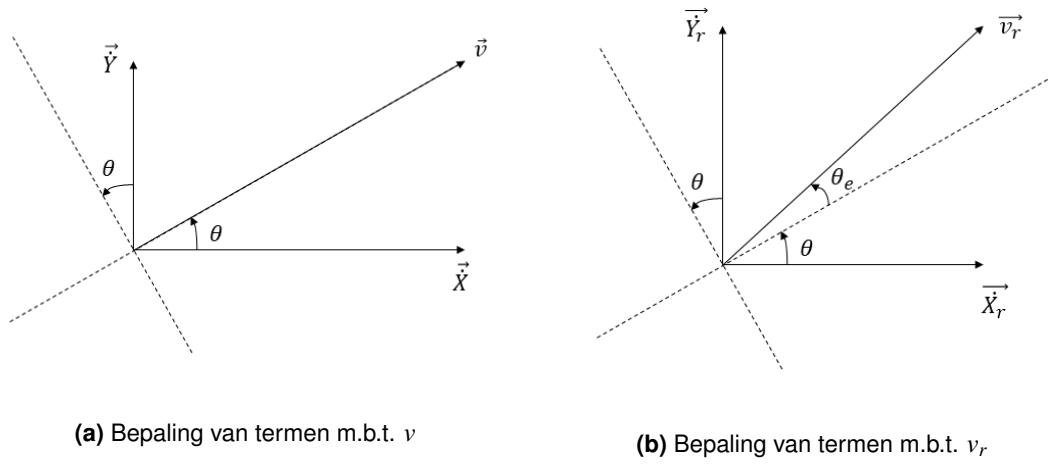
$$\begin{aligned} \sin(\theta + \theta_e) &= \frac{\dot{Y}_r}{v_r} \\ \Leftrightarrow \dot{Y}_r &= v_r \cdot \sin(\theta)\cos(\theta_e) + v_r \cdot \cos(\theta)\sin(\theta_e) \\ \cos(\theta + \theta_e) &= \frac{\dot{X}_r}{v_r} \\ \Leftrightarrow \dot{X}_r &= v_r \cdot \cos(\theta)\cos(\theta_e) - v_r \cdot \sin(\theta)\sin(\theta_e) \end{aligned}$$

Substitutie van twee bovenstaande vergelijkingen in de term van vergelijking 2.7 levert:

$$\begin{aligned} \cos(\theta)\dot{X}_r + \sin(\theta)\dot{Y}_r &= v_r \cdot \cos(\theta)^2\cos(\theta_e) - v_r \cdot \cos(\theta)\sin(\theta)\sin(\theta_e) + v_r \cdot \sin(\theta)^2\cos(\theta_e) + v_r \cdot \sin(\theta)\cos(\theta)\sin(\theta_e) \\ &= [\sin(\theta)^2 + \cos(\theta)^2] \cdot v \cdot \cos(\theta_e) \\ &= v \cdot \cos(\theta_e) \end{aligned}$$

Substitutie van twee bovenstaande vergelijkingen in de term van vergelijking 2.8 levert:

$$\begin{aligned}
 & \cos(\theta)\dot{Y}_r - \sin(\theta)\dot{X}_r \\
 &= -v_r \cdot \cos(\theta)\sin(\theta)\sin(\theta_e) + v_r \cdot \cos(\theta)^2\sin(\theta_e) + v_r \cdot \sin(\theta)\cos(\theta)\sin(\theta_e) + v_r \cdot \sin(\theta)^2\cos(\theta_e) \\
 &= [\sin(\theta)^2 + \cos(\theta)^2] \cdot v \cdot \sin(\theta_e) \\
 &= v \cdot \sin(\theta_e)
 \end{aligned}$$



Figuur 2.5: Illustraties ter verduidelijking van vergelijking 2.2

Het maken van een sliding mode controller bestaat uit twee stappen:

1. Het kiezen van de juiste switching functies $s(t)$, opdat het systeem het gewenste gedrag vertoont indien $s(t) = 0$. In deze situatie worden er twee switching functies gezocht ($s_1(t)$ en $s_2(t)$), op deze manier zal men twee controlewetten kunnen afleiden: één voor de snelheid v en één voor de hoeksnelheid ω . De twee switching functies samen vormen de switching surface.
2. Het uitzoeken van de juiste controlewetten opdat het systeem in slidingmode raakt en daarin blijft ($s_1(t) = 0$ en $s_2(t) = 0$).

Het vinden van de switching surface kan vereenvoudigd worden door de eerste switching functie $x_e = 0$ te kiezen [11]. Voor het zoeken van de tweede switching functie wordt gebruik gemaakt van Lyapunov functies. Lyapunov functies worden gebruikt om de stabiliteit van differentiaalvergelijkingen van een systeem aan te tonen. De Lyapunov functie methode omvat stabilitetscriteria en theorema om de stabiliteit van een systeem na te gaan. De wiskundige achtergrond voor Lyapunov functies staat beschreven in [3]. De noodzakelijke achtergrondkennis voor deze studie is terug te vinden in Bijlage A.

De eerste switching functie is dus reeds bepaald: $x_e = 0$. Nu dient men op zoek te gaan naar de tweede switching functie. Hierbij wordt gebruik gemaakt van Lyapunov functies. Kiest men een

Lyapunov kandidaatfunctie van de vorm $V(X) = V(x_e, y_e) = a \cdot x_e^2 + b \cdot y_e^2$. Hierboven werd x_e reeds nul gesteld en werd $b = \frac{1}{2}$ gekozen. Dan wordt deze uitdrukking vereenvoudigd tot

$$V(y_e) = V_y = \frac{1}{2} \cdot y_e^2$$

Afleiden naar de tijd en substitutie van y_e (zie stelsel in vergelijking 2.2):

$$\dot{V}_y = y_e \dot{y}_e = y_e(-x_e \omega + v_r \sin(\theta_e)) = -x_e y_e \omega - v_r y_e \sin(\theta_e) \quad (2.9)$$

Nu moet de vergelijking 2.9 voldoen aan de derde voorwaarde van een Lyapunov functie (namelijk: $\frac{dV}{dt} \leq 0 \quad \forall X \in U$: terug te vinden in Bijlage A). Hieronder is aangetoond dat de derde voorwaarde voor een Lyapunov functie voldaan is indien men $\theta_e = -\arctan(v_r y_e)$ invult in vergelijking 2.9.

$$\dot{V}_y = y_e \dot{y}_e = y_e(-x_e \omega + v_r \sin \theta_e) = -x_e y_e \omega - v_r y_e \sin(\arctan(v_r y_e))$$

Zolang $v_r y_e \sin(\arctan(v_r y_e)) \geq 0$ geldt dat $\dot{V}_y \leq 0$.

Het systeem is volgens het asymptotische stabiliteitstheorema van de Lyapunov functie theorie (terug te vinden in Bijlage A) asymptotisch stabiel. In het evenwichtspunt $V_y \equiv 0$ geldt dat $y_e = 0$. Dit evenwicht wordt bereikt als $x_e \rightarrow 0$ en $\theta_e \rightarrow -\arctan(v_r y_e)$. Ook θ_e zal dus, naarmate men de requested posture P_r nadert, naar nul convergeren. Wanneer de robot aankomt bij de requested posture P_r zal dus $x_e = 0$, $y_e = 0$ en dus ook $\theta_e = 0$.

Men weet dat indien $x_e = 0$ en $\theta_e = -\arctan(v_r y_e)$ het systeem asymptotisch stabiel is volgens het asymptotische stabiliteitstheorema van Lyapunov. Met behulp van SMC dwingt men het systeem nu in deze toestand door $x_e = 0$ en $\theta_e + \arctan(v_r y_e) = 0$ als kandidaat switching functies te kiezen. De switching surface ziet er dan als volgt uit:

$$s = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} x_e \\ \theta_e + \arctan(v_r y_e) \end{bmatrix} \quad (2.10)$$

In figuur 2.6 wordt het principe voor de bepaling van de controlewet van SMC toegelicht. Voor een korte uitleg over hoe SMC in zijn werk gaat wordt verwezen naar bijlage B. Naargelang de switching functie $s(x)$ zal het systeem $f_1(x)$ of $f_2(x)$ uitvoeren. Deze functies zijn bepaald door \dot{s} , de afgeleiden van switching functie naar de tijd. Veronderstelt men nu dat de afgeleide van de switching functie $\dot{s} = -k \cdot \text{sat}(s)$ (figuur 2.7). $\text{sat}(s)$ stelt hier de saturatiefunctie voor. Dan moet dit ook gelden voor de afgeleiden van de switching mode functies zoals beschreven in vergelijking 2.10.

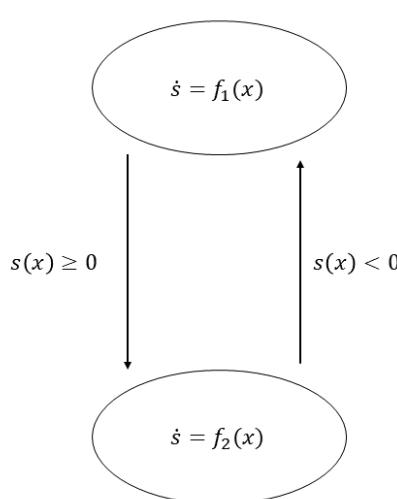
$$\dot{s} = \begin{bmatrix} \dot{s}_1 \\ \dot{s}_2 \end{bmatrix} = \begin{bmatrix} -k_1 \cdot \text{sat}(s_1) \\ -k_2 \cdot \text{sat}(s_2) \end{bmatrix} = \begin{bmatrix} \dot{x}_e \\ \dot{\theta}_e + \frac{y_e}{1+(v_r y_e)^2} v_r + \frac{v_r}{1+(v_r y_e)^2} \dot{y}_e \end{bmatrix}$$

Na substitutie van de \dot{x}_e , \dot{y}_e en $\dot{\theta}_e$ uit de formules van vergelijking 2.2 komt men:

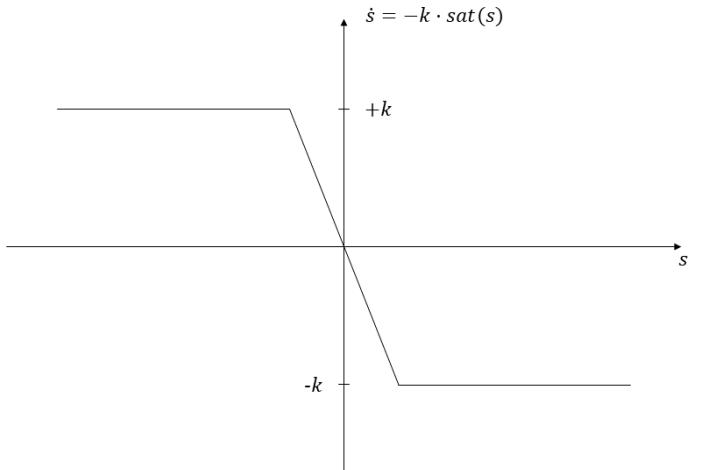
$$\begin{bmatrix} -k_1 \cdot \text{sat}(s_1) \\ -k_2 \cdot \text{sat}(s_2) \end{bmatrix} = \begin{bmatrix} y_e \omega - v_r \cos(\theta_e) \\ \omega_r - \omega + \frac{y_e}{1+(v_r y_e)^2} v_r + \frac{v_r}{1+(v_r y_e)^2} (-x_e \omega + v_r \sin(\theta_e)) \end{bmatrix}$$

Waaruit men haalt dat:

$$\begin{aligned} \begin{bmatrix} v \\ \omega \end{bmatrix} &= \begin{bmatrix} y_e \omega + v_r \cos(\theta_e) + k_1 \cdot \text{sat}(s_1) \\ \omega_r + \frac{y_e}{1+(v_r y_e)^2} v_r + \frac{v_r}{1+(v_r y_e)^2} (v_r \sin(\theta_e)) + k_2 \cdot \text{sat}(s_2) \end{bmatrix} \\ \Leftrightarrow \begin{bmatrix} v \\ \omega \end{bmatrix} &= \begin{bmatrix} y_e \omega + v_r \cos(\theta_e) + k_1 \cdot \text{sat}(s_1) \\ \omega_r + \frac{y_e \cdot v_r}{1+(v_r y_e)^2} + \frac{v_r^2 \cdot \sin(\theta_e)}{1+(v_r y_e)^2} + k_2 \cdot \text{sat}(s_2) \end{bmatrix} \text{ waarbij } x_e = 0 \end{aligned} \quad (2.11)$$



Figuur 2.6: Principe Sliding Mode Control



Figuur 2.7: Saturatie functie gekozen als switching mode functie

2.4.1.2 Simulatie

Dit controlealgoritme wordt gesimuleerd in Simulink. Hierbij wordt gebruik gemaakt van het kinematisch model van de robot beschreven in subsectie 2.3.2. Het totale Simulink schema van de simulatie is te zien in figuur 2.12.

De meest essentiële blokken worden hieronder kort toegelicht:

- "X_r, Y_r generator": genereert de trajectpunten, dit zijn de coördinaten waar het systeem naartoe moet. Merk op dat ook de huidige coördinaten zijn teruggekoppeld, gezien het trajectpunt (X_r, Y_r) enkel wordt vernieuwd indien de robot d_{switch} [m] verwijderd is van het gewenste punt. De aangesloten pulsgenerator zorgt ervoor dat het simulink blok van het type "Enabled and Triggered Subsystem" een updatefrequentie heeft van 10 Hz wat overeenkomt met de updatefrequentie van de GPS, dewelke de bottleneck vormt van de gebruikte sensoren.
- "theta_r": berekent de gewenste hoek θ_r . De matlab code van de functie is terug te vinden in listing 2.2. Merk op dat de Matlab functie *atan2* een waarde teruggeeft in het interval $[-\pi, \pi]$. Het verschil van θ_r met de huidige hoek θ levert vervolgens θ_e .

```

1 function theta_r = fcn(u)
2 theta_r = atan2(u(2),u(1));

```

Listing 2.2: Matlab code "theta_r" blok

- "smc regelaar": implementeert het hierboven beschreven SMC controlealgoritme. De Matlab code is terug te vinden in listing 2.3

```

1 function [v,omega] = fcn(pos_e, theta_e, theta)
2 v_r = 1;
3 omega_r = 0;
4 a_r = 0;
5 x_e = cos(theta)*pos_e(1) + sin(theta)*pos_e(2);
6 y_e = -sin(theta)*pos_e(1) + cos(theta)*pos_e(2);
7 k2 = 1;
8 k1 = 1;
9 omega = omega_r + (y_e*a_r)/(1+(v_r*y_e)^2) + (v_r^2*sin(theta_e))
    /(1+(v_r*y_e)^2) + k2*min(1, max(-1, theta_e + atan(v_r*y_e)));
10 v = y_e*omega + v_r*cos(theta_e) + k1*min(1, max(-1,x_e));

```

Listing 2.3: Matlab code "smc regelaar" blok

- "calctheta_e": tijdens de berekening van de hoekfout θ_e kunnen zich complicaties voordoen. Verplaats de robot zich in de negatieve x-richting, dan is het mogelijk dat bijvoorbeeld $\theta_r = 178^\circ$ en $\theta = -178^\circ$ waardoor $\theta_e = \theta_r - \theta = 178^\circ - (-178^\circ) = 356^\circ$. Dit probleem wordt voorkomen in dit blok. De Matlab code is terug te vinden in listing 2.4.

```

1 function theta_e = fcn(theta_r,theta)
2
3 theta_e = theta_r - theta;
4
5 if theta_e > pi
6     theta_e = theta_e - 2*pi;
7 elseif theta_e < -pi
8     theta_e = theta_e + 2*pi;
9 end

```

Listing 2.4: Matlab code "calctheta_e" blok

- "model robot": het gebruikte kinematisch model zoals beschreven in subsectie 2.3.2.

Voor een bepaalde posture $P = (X, Y, \theta)^T$ wordt het gevolgde traject van de robot getekend en de fout ten opzichte van het ideale traject in meter bepaald. Het ideale traject voldoet aan de rechte r met vergelijking $X = Y$. Bevindt de robot zich in een punt $P_1(X_1, Y_1)$, dan wordt de vergelijking van

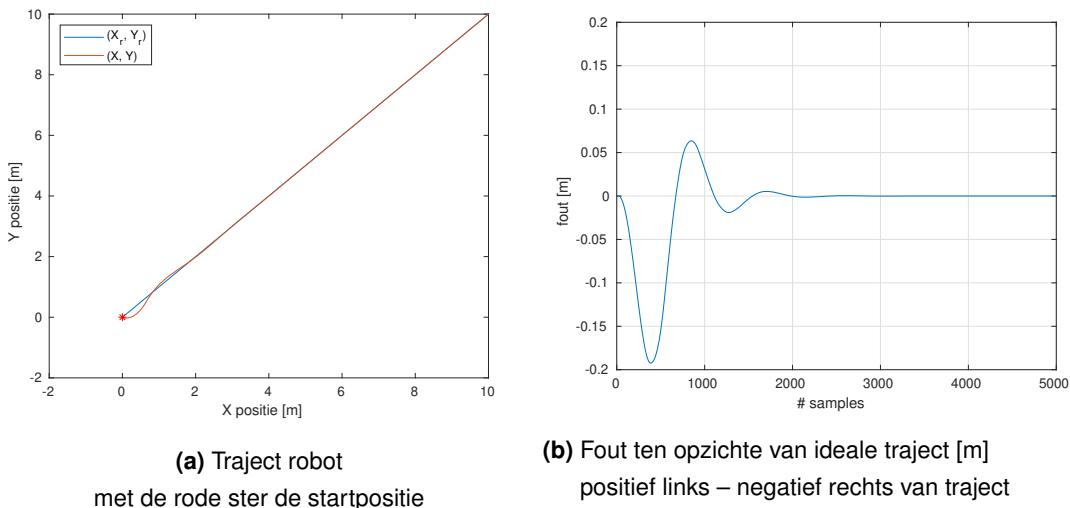
de rechte r_{\perp} loodrecht op r en door het punt $P_1(X_1, Y_1)$ als volgt bepaald:

$$\begin{aligned} r_{\perp} : Y - Y_1 &= a \cdot (X - X_1) \\ \Leftrightarrow Y &= a \cdot X - a \cdot X_1 + Y_1 \text{ wanneer } a = -1 \\ \Leftrightarrow Y &= -X + X_1 + Y_1 \end{aligned}$$

Uit het stelsel $\begin{cases} Y = -X + X_1 + Y_1 \\ X = Y \end{cases}$ haalt men dat $X = Y = \frac{X_1 + Y_1}{2}$. Het punt $P_2\left(\frac{X_1 + Y_1}{2}, \frac{X_1 + Y_1}{2}\right)$ is dus het punt het dichtste bij P_1 . De afstand tussen P_1 en P_2 bepaalt de fout ten opzichte van het ideale traject.

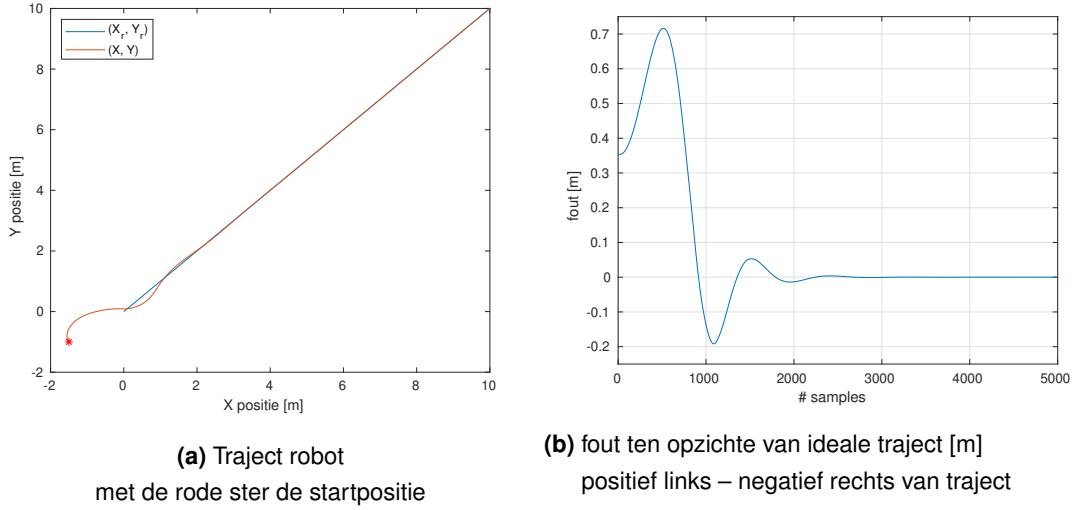
2.4.1.3 Resultaten

De resultaten van de simulaties met startpositie $P = (0, 0, -\frac{\pi}{4})^T$ zijn te zien in figuur 2.8. In figuur 2.8a is te zien dat het systeem naar de gewenste toestand terugkeert. Hierbij is geen offset fout meer waar te nemen, zoals te zien in figuur 2.8b. Het systeem stabiliseert na ≈ 2000 metingen.



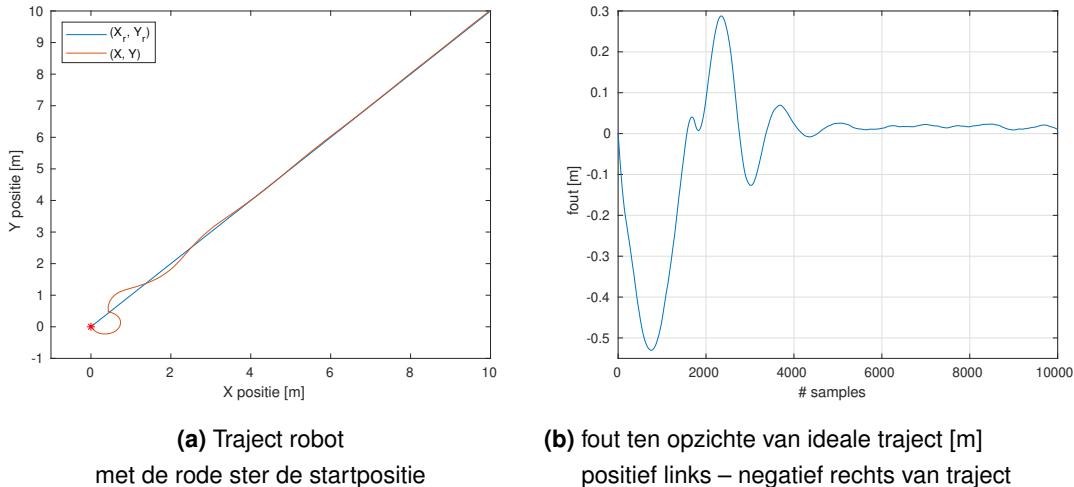
Figuur 2.8: Simulaties voor startpositie $P = (0, 0, -\frac{\pi}{4})^T$

De resultaten van de simulaties met startpositie $P = (-1.5, -1, \frac{3\pi}{4})^T$ zijn te zien in figuur 2.9. In figuur 2.9a is te zien dat het systeem naar de gewenste toestand terugkeert. Hierbij is geen offset fout meer waar te nemen, zoals te zien in figuur 2.9b. Het systeem stabiliseert na ≈ 2000 metingen. Uit de simulatie blijkt dat niet elke uitgangspositie aanleiding geeft tot een stabiel systeem. De robot staat best dicht bij de start van het traject.



Figuur 2.9: Simulaties voor startpositie $P = (-1.5, -1, \frac{3\pi}{4})^T$

Als laatste test wordt extra ruis toegevoegd, zowel aan de hoek als aan de positie. Deze ruis stelt de onzekerheid van de sensoren voor. De maximale fout op de IMU bedraagt $\approx 4^\circ$ en de maximale fout op de GPS in fixed mode 3 cm. Tijdens het simuleren wordt bij elke meting (puls van pulsgenerator) een random waarde gesuperponeerd binnen het interval $[-4^\circ, 4^\circ]$ op θ en een random waarde binnen het interval $[-\frac{0.03}{\sqrt{2}}, \frac{0.03}{\sqrt{2}}]$ op zowel de X- als Y-positie. Het nieuwe simulink blokschema vindt u terug op figuur 2.13. De resultaten van de simulaties met startpositie $P = (0, 0, -\frac{\pi}{4})^T$ met storingen zijn te zien in figuur 2.10. In figuur 2.10a is te zien dat het systeem naar de gewenste toestand terugkeert en in de buurt van dit evenwicht blijft. In figuur 2.10b is te zien dat de offsetfout schommelt rond de 2 cm na ongeveer 5000 metingen.



Figuur 2.10: Simulaties voor startpositie $P = (0, 0, -\frac{\pi}{4})^T$ met storing

De stabiliteit en de responsiviteit van het algoritme is naast de parameters k_1 en k_2 afhankelijk van

de volgende drie parameters:

1. De afstand tussen de trajectpunten waar de robot naartoe rijdt $d_{traject}$. De trajectpunten zijn de punten (X_r, Y_r) gegenereerd door het "X_r, Y_r generator" blok te zien in figuur 2.12. Betere resultaten werden bekomen wanneer deze trajectpunten dichter op elkaar lagen. Dit is vanzelfsprekend, hoe dichter de punten op elkaar liggen, hoe meer ze een lijn benaderen wat noodzakelijk is voor de snelle werking van het besproken algoritme. In bovenstaande simulatie was de afstand tussen twee opeenvolgende trajectpunten $d_{traject} = 30$ cm gekozen. Een verdere discussie van deze parameter volgt in subsectie 3.3.5.
2. De afstand d_{switch} . Wanneer de robot zich binnen een straal d_{switch} bevindt ten opzichte van het trajectpunt (X_r, Y_r) waarnaar de robot rijdt, wordt dit punt geüpdate naar het volgende trajectpunt waar de robot naar moet rijden. d_{switch} vindt u eveneens terug in de simulatiefile te zien in figuur 2.12. Het is van belang dat d_{switch} wordt afgestemd op $d_{traject}$. In bovenstaande simulatie werd $d_{switch} = 25$ cm gesteld. Een verdere discussie van deze parameter volgt in subsectie 3.3.5.
3. De updatefrequentie. Zoals uit sectie 2.5.1 nog zal duidelijk worden, kan men door toepassen van Kalman filtering de staat van de robot schatten ook zonder een meting. In bovenstaande simulatie werd de updatefrequentie van 10 Hz naar 20 Hz verhoogd.

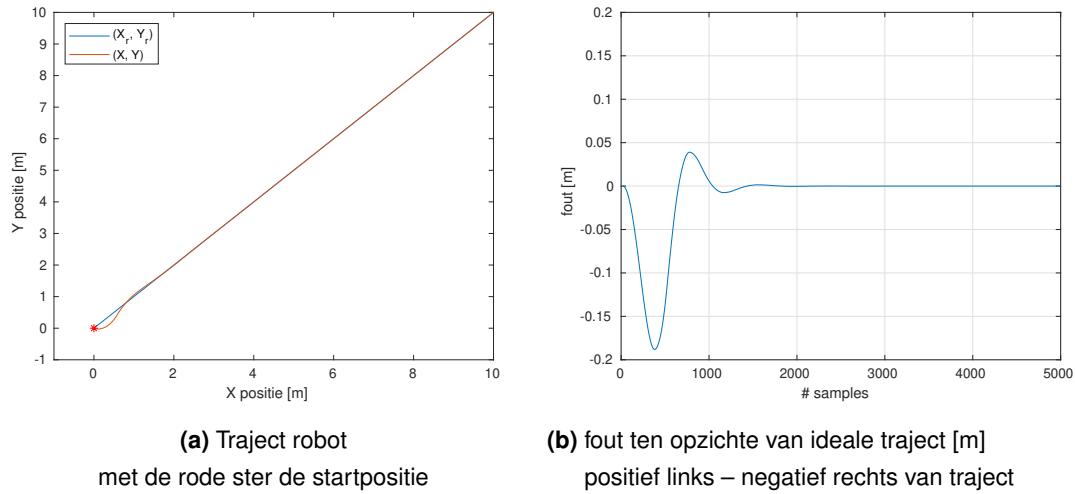
In de simulaties hierboven werd steeds $v_r = 1$ m/s en $\omega_r = 0$ rad/s gekozen. Uit ondervinding werd vastgesteld dat het evenwicht nog sneller wordt bereikt indien ω_r evenredig is met de grootte van de hoekfout θ_e met evenredigheidsconstante k_3 . De resultaten voor $k_3 = 1$ zijn te zien in figuur 2.11. De wijzigingen in de Matlab code voor het het blok "smc regelaar" zijn te zien in listing 2.5.

```

1 function [v,omega] = fcn(pos_e, theta_e, theta)
2 k1 = 1;
3 k2 = 1;
4 k3 = 1;
5
6 v_r = 1;
7 omega_r = k3*theta_e;
8 a_r = 0;
9
10 x_e = cos(theta)*pos_e(1) + sin(theta)*pos_e(2);
11 y_e = -sin(theta)*pos_e(1) + cos(theta)*pos_e(2);
12
13 omega = omega_r + (y_e*a_r)/(1+(v_r*y_e)^2) + (v_r^2*sin(theta_e))/(1+(
14     + k2*min(1, max(-1, theta_e + atan(v_r*y_e)))) ;
15 v = y_e*omega + v_r*cos(theta_e) + k1*min(1, max(-1,x_e));

```

Listing 2.5: Matlab code "smc regelaar" blok na invoering k_3



Figuur 2.11: Simulaties voor startpositie $P = (0,0,-\pi/4)^T$ na invoering parameter $k_3 = 1$

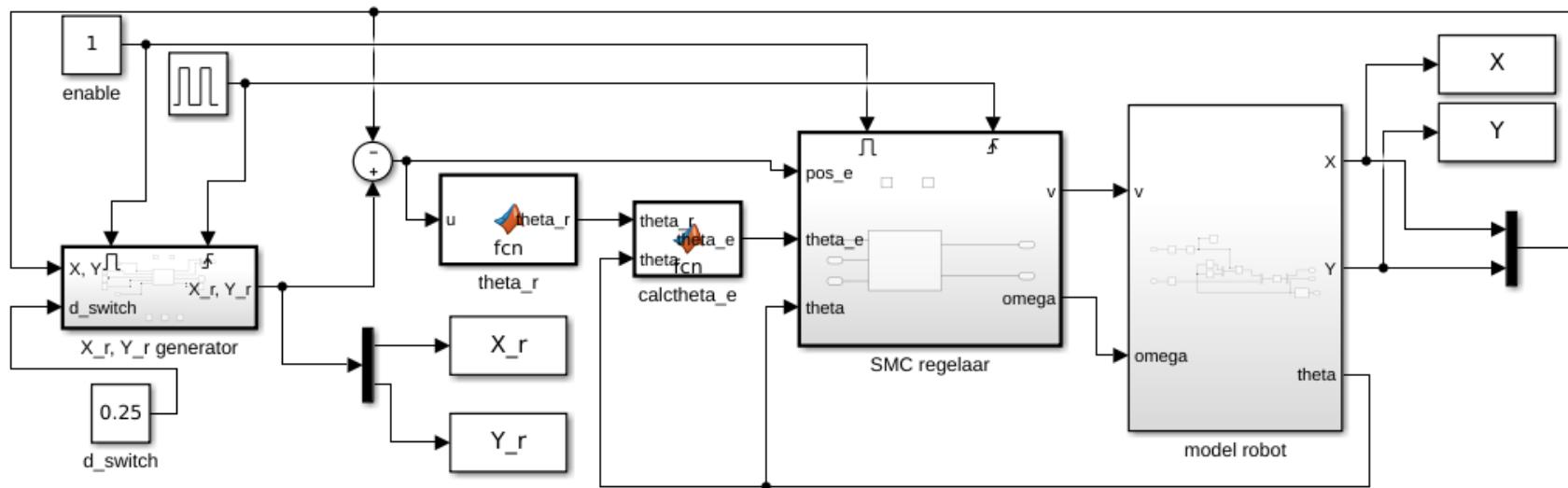
2.4.1.4 Besluit

Uit de simulaties kan men besluiten dat deze methode, met een controlealgoritme gebaseerd op SMC, een stabiel algoritme is gezien het naar de gewenste toestand convergeert. Bovendien is het verschil tussen de simulatie met storing en de simulatie zonder storingen klein. Men kan besluiten dat het een systeem is dat goed functioneert onafhankelijk van de storingen op de gebruikte sensoren. Het grote voordeel van het systeem is dat er slechts twee parameters k_1 en k_2 te tunen zijn. Ook de volgende parameters hebben een belangrijke invloed op het gedrag van de robot.

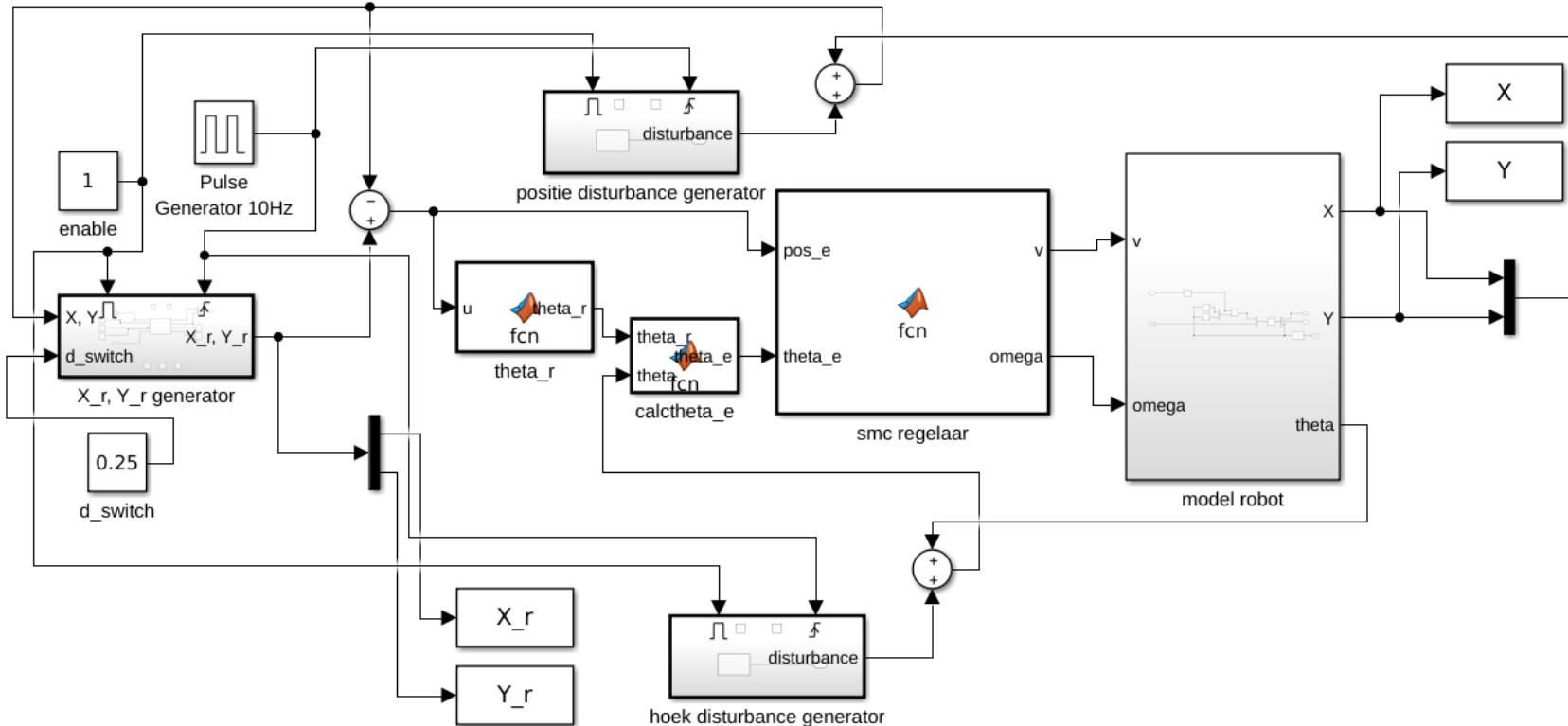
1. de onderlinge afstand van de punten waar de robot naartoe rijdt: $d_{traject}$
2. de afstand waarbij het punt waar naartoe gereden wordt, wordt vernieuwd: d_{switch}
3. de updatefrequentie van v_r en ω_r

De snelheid waarmee naar de evenwichtstoestand wordt gegaan en de grootte van de gemaakte fout kan nog verder worden geminimaliseerd door ω_r evenredig te laten variëren met de hoekfout θ_e met evenredigheidsconstante k_3 .

Een ander voorbeeld dat gebruik maakt van Sliding mode control en Lyapunov functies is terug te vinden in [12]. Hierin wordt gewezen op de "pure rolling constraints" om een goede werking van het algoritme te verzekeren. Dit biedt een nadeel voor dit algoritme gezien het niet onoverkomelijk is dat een robot in het veld slijpt of weglijdt. Wel zou de stabiliteit verzekerd blijven bij onzekerheden op de sensoren, bovendien biedt het algoritme een snelle respons bij de juiste parameterkeuze.



Figuur 2.12: Simulatie SMC controlealgoritme in Simulink



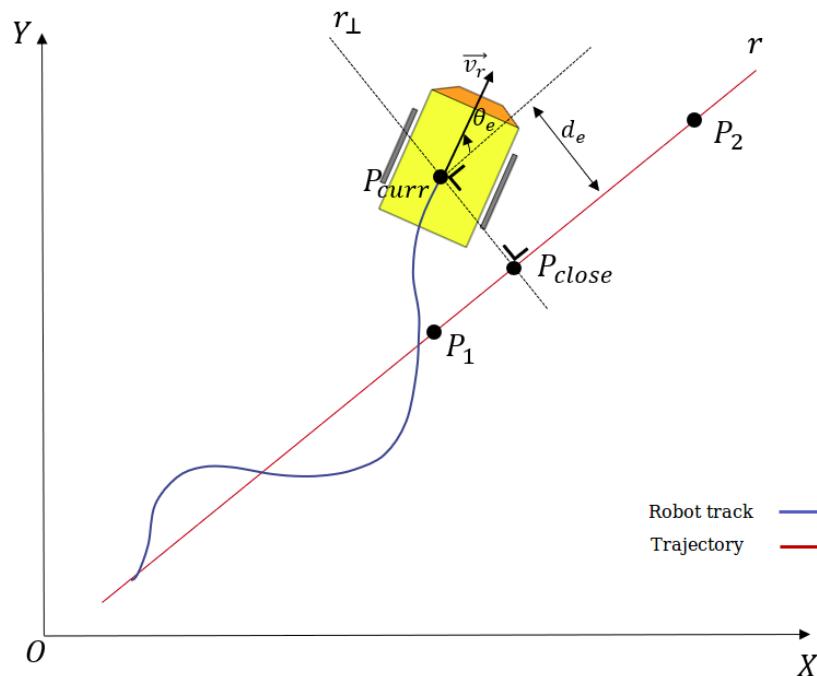
Figuur 2.13: Simulatie SMC controlealgoritme met storingen in Simulink

2.4.2 Fuzzy PID

2.4.2.1 Algoritme

In [5] staat een controle time-based algoritme beschreven, waarbij zowel de snelheid als de hoek-snelheid worden bepaald door een PID-controller waarvan de PID-parameters dynamisch worden aangepast door een Fuzzy controller.

Voor dit controlealgoritme wordt naast de hoekfout θ_e ook de afstandsfout d_e gedefinieerd. De afstandsfout drukt de kortste afstand uit van het zwaartepunt van de robot tot het gewenste traject. Dit is weergegeven in figuur 2.14. Ook als het traject niet recht is kan d_e nog steeds worden bepaald zoals weergegeven op figuur 2.14. Door op een bochtig parcours P_1 en P_2 voldoende dicht bij P_i te kiezen wordt het traject als het ware gelineariseerd.



Figuur 2.14: Illustratie ter verduidelijking van de terminologie in de sectie Fuzzy PID

Beschouw de volgende punten op figuur 2.14:

- $P_1(X_1, Y_1)$ is het vorige punt waarnaar de robot reed.
- $P_2(X_2, Y_2)$ is het huidige punt waarnaar de robot rijdt.
- $P_{curr}(X_{curr}, Y_{curr})$ is het punt waar de robot zich op dit ogenblik bevindt (current point).
- $P_{close}(X_{close}, Y_{close})$ is het snijpunt van de rechten r en r_{\perp} (closest point).

Dan heeft de rechte r de vergelijking:

$$\begin{aligned} Y - Y_1 &= a \cdot (X - X_1) \\ \Leftrightarrow Y &= a \cdot X + (Y_1 - a \cdot X_1) \text{ waarbij } a = \frac{Y_2 - Y_1}{X_2 - X_1} \end{aligned} \quad (2.12)$$

r_{\perp} heeft bijgevolg een richtingscoëfficiënt $-a$ en gaat door het punt P_{curr} , hieruit volgt zijn vergelijking:

$$\begin{aligned} Y - Y_{curr} &= -a \cdot (X - X_{curr}) \\ \Leftrightarrow y &= -a \cdot X + (Y_{curr} + a \cdot X_{curr}) \text{ waarbij } a = \frac{Y_2 - Y_1}{X_2 - X_1} \end{aligned} \quad (2.13)$$

Uit de vergelijkingen 2.12 en 2.13 kunnen nu het snijpunt P_{close} van de rechten r en r_{\perp} bepalen:

$$\begin{cases} Y &= a \cdot X + (Y_1 - a \cdot X_1) \\ Y &= -a \cdot X + (Y_{curr} + a \cdot X_{curr}) \end{cases} \text{ waarbij } a = \frac{Y_2 - Y_1}{X_2 - X_1}$$

Na uitrekenen van het bovenstaande stelsel vinden we P_{close} :

$$P_{close} \left(\frac{a \cdot (X_{curr} - X_1) + Y_{curr} + Y_1}{2 \cdot a}, \frac{a \cdot (X_{curr} - X_1) + Y_{curr} + Y_1}{2} + (Y_1 - a \cdot X_1) \right) \text{ waarbij } a = \frac{Y_2 - Y_1}{X_2 - X_1} \quad (2.14)$$

Alhoewel een space-based trajectory algoritme hier zou volstaan, is het toch van belang dat de snelheid van de robot wordt geregeld. Zo is het wenselijk dat de robot vertraagt als θ_e en/of d_e zeer groot worden en een aanvaardbare snelheid heeft indien $\theta_e \approx 0$ en $d_e \approx 0$. Dit gedrag kan bekomen worden door de lineaire snelheid v fuzzy te regelen met fuzzy input sets voor θ_e en d_e . De verandering in de lineaire snelheid heeft natuurlijk ook een invloed op het gedrag van de robot, bijgevolg is het nodig hiermee rekening te houden bij het fuzzy regelen van de PID-parameters ter bepaling van hoeksnelheid ω . Ook hiervoor worden input sets voor θ_e en d_e gebruikt. De output van de snelheid v en de output van de verandering van de PID-parameters zullen worden bepaald door middel van centroid defuzzificatie.

Zoals beschreven in [6] ligt Fuzzy logic veel dichter bij het menselijke denken. Waar inputs dikwijls binair worden geïnterpreteerd (waar of vals, 0 of 1), neemt een Fuzzy Membership Function (MF) waarden aan in een interval van $[0, 1]$. Voor meer informatie over de Fuzzy Logic Theory wordt verwezen naar hoofdstuk 2 en 3 van [6]. Voor het ontwikkelen van een snelheidscontrolealgoritme van de robot zal er gebruik worden gemaakt van Type-1 Fuzzy sets. In tegenstelling van Type-2 Fuzzy sets heeft een Type-1 Fuzzy set MF μ slechts één waarde voor x . $\mu(x)$ is dus uniek. Bij een Type-2 Fuzzy set MF kan $\mu(x)$ meerdere waarden aannemen binnen de onder- en bovengrens voor de waarde x . Een Type-2 Fuzzy set MF bestaat immers uit een lower- en upper MF: $\underline{\mu}(x)$ en $\overline{\mu}(x)$. Op die manier vangt een Type-2 Fuzzy set de onnauwkeurigheden van het systeem op. Voor de snelheidscontroller zal een Type-1 Fuzzy set volstaan gezien de nauwkeurigheid hiervan van ondergeschikt belang is. Bij het regelen van de PID-parameters zou een Type-2 Fuzzy set wel interessant kunnen zijn. Hierdoor zou de fout op de GPS-coordinates en de fout op de IMU

in rekening kunnen worden gebracht in respectievelijk d_e en θ_e . Eerst wordt de werking van het algoritme met Type-1 Fuzzy set MFs getest, achteraf is dan nog steeds de overgang naar Type-2 Fuzzy set MFs mogelijk.

Zoals eerder vermeld is hier gekozen voor een Fuzzy logic netwerk waarvan de input Fuzzy sets afhankelijk zijn van de hoekfout θ_e en de afstandsfout d_e . Er wordt gebruik gemaakt van triangular MF's, volgens [5] voldoen deze aan de real-time vereisten en ze zijn eenvoudig in gebruik. Daar de robot zal vertragen bij een grotere θ_e wordt verwacht dat de afstandsfout d_e vermindert ten opzichte van een robot waarbij de snelheid niet Fuzzy geregeld is. Om dit algoritme te testen vergelijkt men de afstandsfout van een systeem met en zonder PID Fuzzy logic.

2.4.2.2 Simulatie

Dit controlealgoritme wordt in Simulink gesimuleerd. Hierbij wordt het kinematisch model van de robot beschreven in subsectie 2.3.2 gebruikt. Het totale Simulink schema van de simulatie met Fuzzy regeling is te zien in figuur 2.23. Het schema zonder fuzzy regeling is te zien in figuur 2.24.

De meest essentiële blokken worden hieronder kort toegelicht:

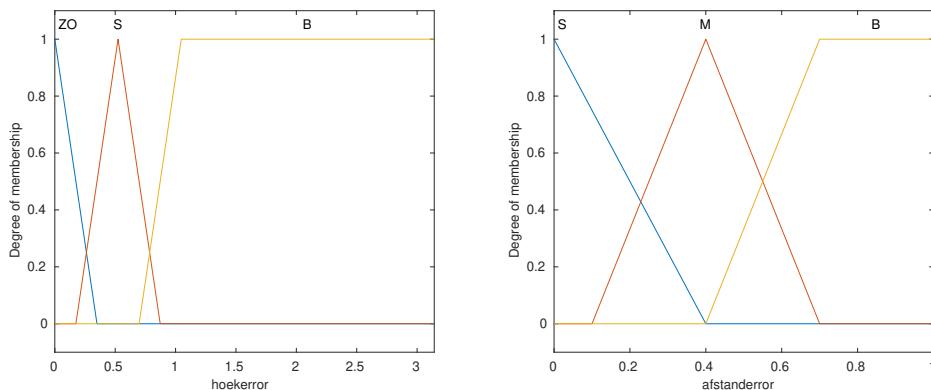
- "X_r, Y_r generator": genereert de trajectpunten waar het systeem naartoe moet. Merk op dat ook de huidige coördinaten zijn teruggekoppeld, gezien een trajectpunt (X_r, Y_r) enkel wordt vernieuwd indien de robot d_{switch} [m] verwijderd is van het dit punt. De aangesloten pulsgenerator zorgt ervoor dat het simulink blok van het type "Enabled and Triggered Subsystem" een updatefrequentie heeft van 10 Hz wat overeenkomt met de updatefrequentie van de GPS, dewelke de bottleneck vormt van de gebruikte sensoren.
- "theta_r": berekent de gewenste hoek θ_r . De matlab code van de functie is terug te vinden in listing 2.2. Merk op dat de Matlab functie *atan2* een waarde teruggeeft in het interval $[-\pi, \pi]$. Het verschil van θ_r met de huidige hoek θ levert vervolgens θ_e .
- "model robot": het gebruikte kinematisch model zoals beschreven in subsectie 2.3.2.
- "Fuzzy Logic speed": Fuzzy logic control block dewelke de Fuzzy logica voor de snelheid implementeert, zoals hierboven beschreven. De gekozen input MFs zijn te zien in figuur 2.15 en de gekozen output MFs in figuur 2.16. De gebruikte Fuzzy rule set is terug te vinden in Listing 2.6. In de Fuzzy rules wordt de volgende naamgeving gebruikt:

- voor de hoekfout: ZO → zero / S → small / B → big
- voor de afstandfout: S → small / M → medium / B → big
- voor de speed: S → slow / F → fast / VF → very fast

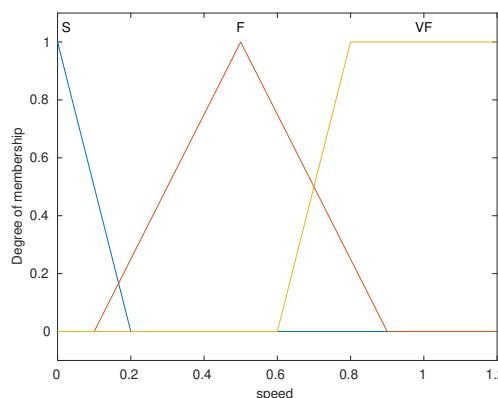
1. If (hoekfout is ZO) and (afstandfout is S) then (speed is VF)
2. If (hoekfout is ZO) and (afstandfout is M) then (speed is F)
3. If (hoekfout is ZO) and (afstandfout is B) then (speed is S)
4. If (hoekfout is S) and (afstandfout is S) then (speed is F)

5. If (hoekerror is S) and (afstanderror is M) then (speed is S)
6. If (hoekerror is S) and (afstanderror is B) then (speed is S)
7. If (hoekerror is B) and (afstanderror is S) then (speed is S)
8. If (hoekerror is B) and (afstanderror is M) then (speed is S)
9. If (hoekerror is B) and (afstanderror is B) then (speed is S)

Listing 2.6: Fuzzy Rule set voor de snelheid Fuzzy logica



Figuur 2.15: Gekozen input MF's voor de hoekfout θ_e en de afstandsfout d_e



Figuur 2.16: Gekozen output MF voor de snelheid v

- "Fuzzy Logic omega": Fuzzy logic control block dewelke Fuzzy logica voor de PID-parameters ter bepaling van ω implementeert, zoals hierboven beschreven. De gekozen input MFs zijn dezelfde als deze van de Fuzzy logica voor de snelheid en zijn te zien in figuur 2.15. De output MFs zijn te zien in figuur 2.17. De gebruikte Fuzzy rule set is terug te vinden in Listing 2.7. In de Fuzzy rules wordt de volgende naamgeving gebruikt:

- voor de hoekfout: ZO → zero / S → small / B → big
- voor de afstandfout: S → small / M → medium / B → big

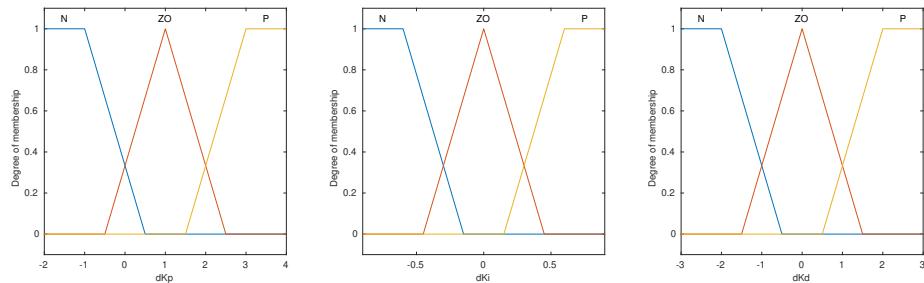
– voor de PID parameters: N → negative / ZO → zero / P → positive

```

1  1. If (afstanderror is S) and (hoekerror is ZO)
2      then (dKp is ZO)(dKi is P)(dKd is N)
3  2. If (afstanderror is S) and (hoekerror is SMALL)
4      then (dKp is ZO)(dKi is P)(dKd is N)
5  3. If (afstanderror is S) and (hoekerror is BIG)
6      then (dKp is P)(dKi is P)(dKd is N)
7  4. If (afstanderror is M) and (hoekerror is ZO)
8      then (dKp is ZO)(dKi is ZO)(dKd is ZO)
9  5. If (afstanderror is M) and (hoekerror is SMALL)
10     then (dKp is P)(dKi is ZO)(dKd is ZO)
11 6. If (afstanderror is M) and (hoekerror is BIG)
12     then (dKp is P)(dKi is ZO)(dKd is P)
13 7. If (afstanderror is B) and (hoekerror is ZO)
14     then (dKp is P)(dKi is ZO)(dKd is P)
15 8. If (afstanderror is B) and (hoekerror is SMALL)
16     then (dKp is P)(dKi is ZO)(dKd is P)
17 9. If (afstanderror is B) and (hoekerror is BIG)
18     then (dKp is P)(dKi is ZO)(dKd is P)

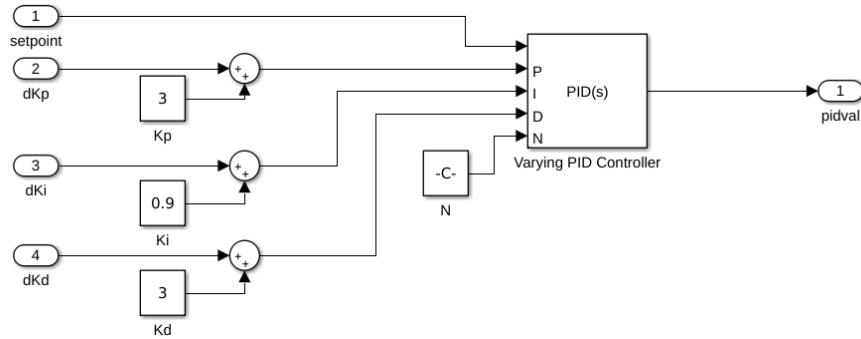
```

Listing 2.7: Fuzzy Rule set voor de hoeksnelheid Fuzzy logica



Figuur 2.17: Gekozen output MFs voor de verandering van de PID parameters dK_p , dK_i , dK_d

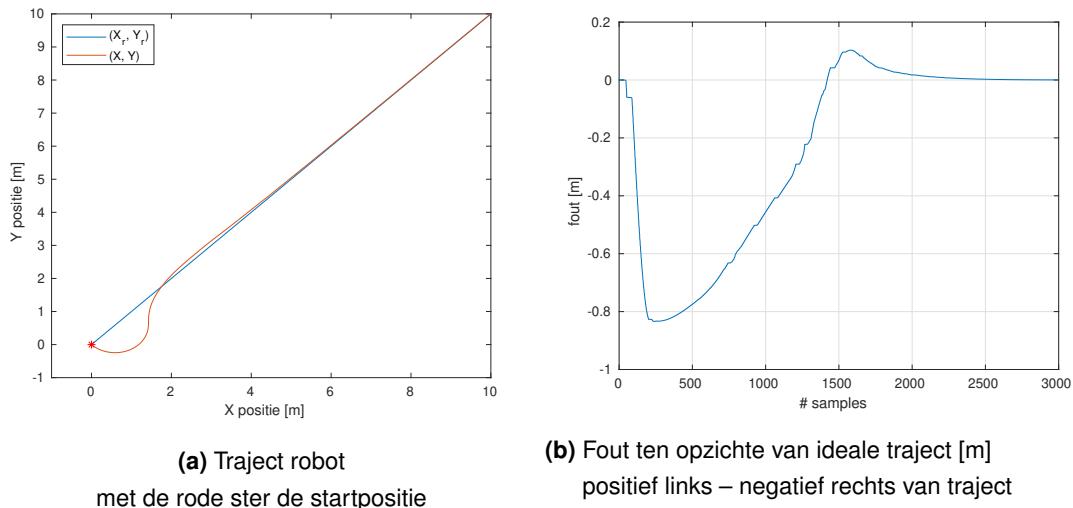
- "pid": bevat een "varying pid controller". De implementatie van het blok is te zien in figuur 2.18. De gebruikte formule in een "Varying PID Controller" blok is $K_p + K_i \frac{1}{s} + K_d \frac{N}{1+N \frac{1}{s}}$. Kies de constante voldoende groot $N \gg 1$ waardoor deze formule benaderend kan geschreven worden als $K_p + K_i \frac{1}{s} + K_d s$. De constante blokken zijn bepaald door PID tuning volgens Ziegler en Nichols en zijn de waarden waarrond de PID parameters schommelen.



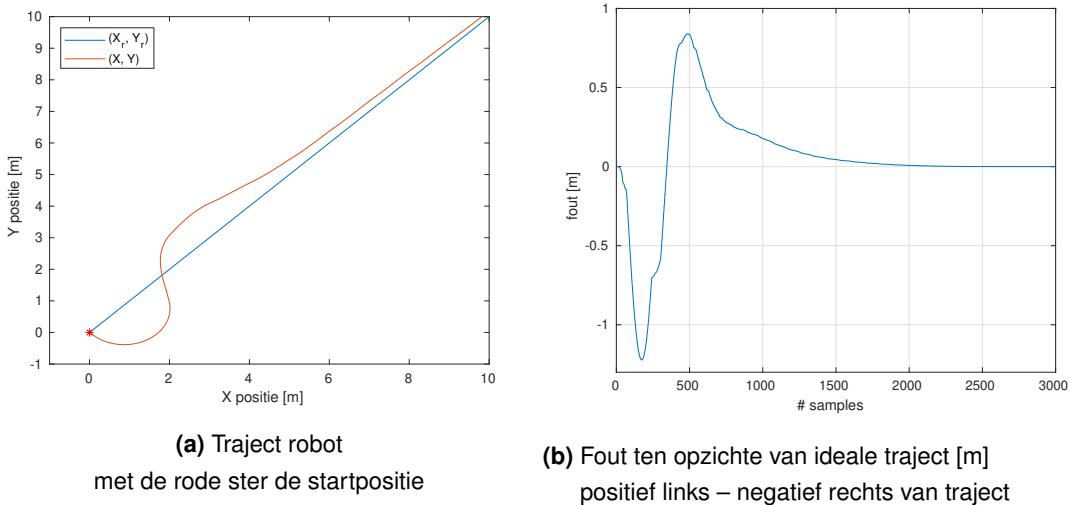
Figuur 2.18: Implementatie PID blok

2.4.2.3 Resultaten

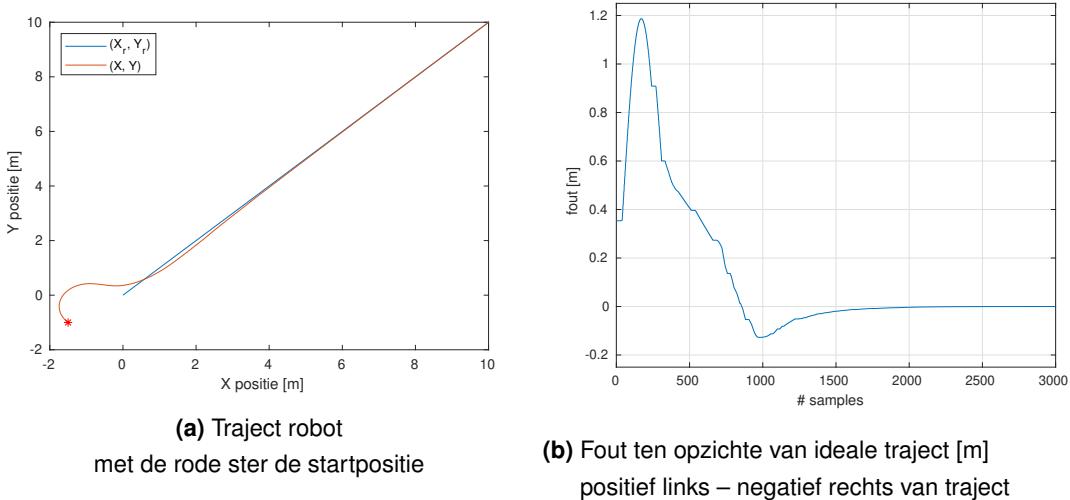
Analoog aan de sectie resultaten in paragraaf 2.4.1 is het algoritme in eerste instantie getest door de robot een lijn te laten volgen en te kijken wat de fout is ten opzichte van het ideale traject. Hierbij werd uitgegaan van de startposities $P = (0, 0, -\frac{\pi}{4})^T$ en $P = (-1.5, -1, \frac{3\pi}{4})^T$. Door het vergelijken van figuur 2.19 met figuur 2.20 en figuur 2.21 met figuur 2.22 stelt men vast dat de fout ten opzichte van het ideale traject weldegeleijk steeds kleiner blijft bij de regeling met Fuzzy logica, dan bij PID-regeling zonder Fuzzy logica. Op basis van deze resultaten is af te leiden dat de fout ongeveer met een factor 1/3 afneemt.



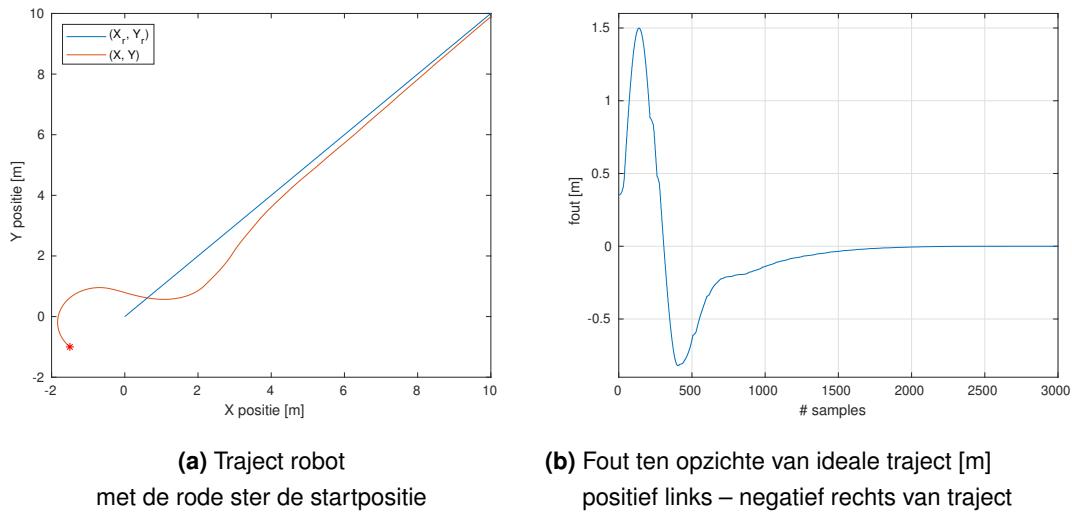
Figuur 2.19: Simulaties voor startpositie $P = (0, 0, -\frac{\pi}{4})^T$ met Fuzzy logica



Figuur 2.20: Simulaties voor startpositie $P = (0, 0, -\frac{\pi}{4})^T$ met $v = 1 \text{ m/s}$ zonder Fuzzy netwerk



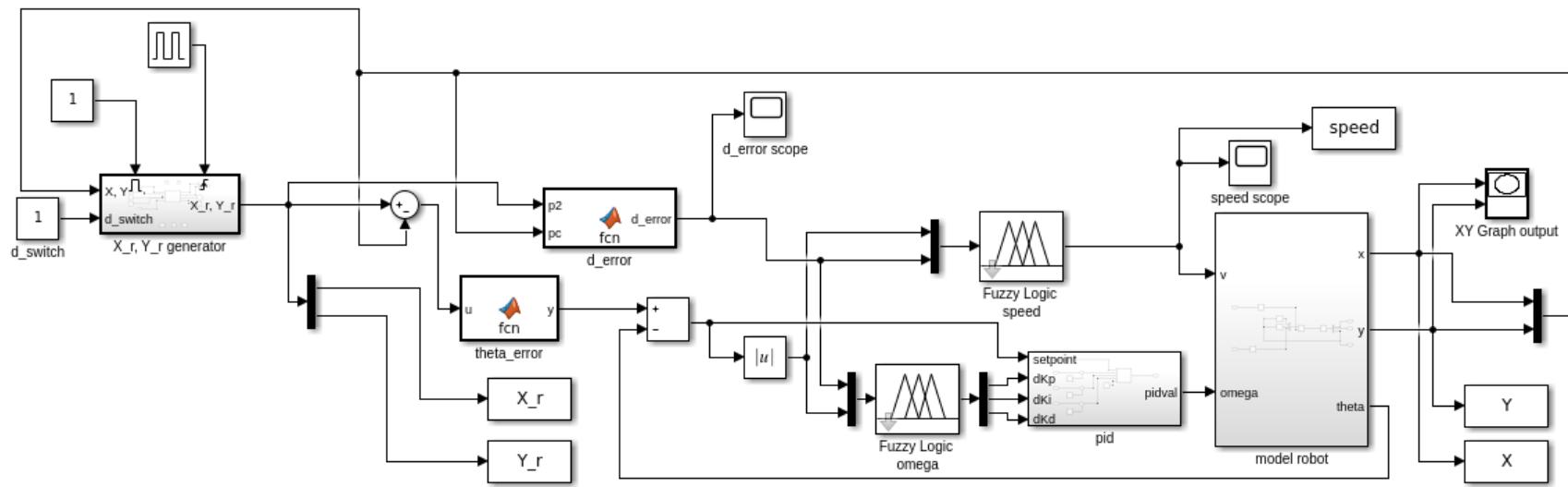
Figuur 2.21: Simulaties voor startpositie $P = (-1.5, -1, \frac{3\pi}{4})^T$ met Fuzzy logica



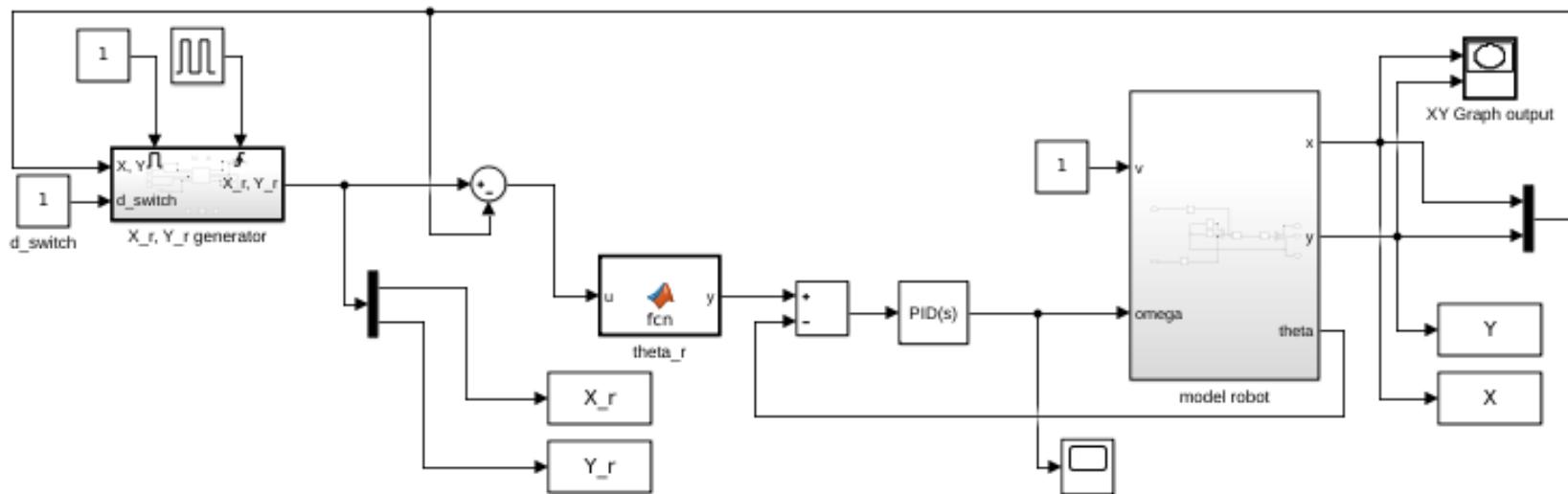
Figuur 2.22: Simulaties voor startpositie $P = (-1.5, -1, \frac{3\pi}{4})^T$ met $v = 1$ m/s zonder Fuzzy netwerk

2.4.2.4 Besluit

Fuzzy logica biedt een mogelijkheid om toch een PID-regelaar te gebruiken in een onzeker niet-lineair systeem. Het vermindert de gemaakte fout van de robot ten opzichte van het gewenste traject doordat men op het juiste moment de gewenste PID-parameter kan laten doorwegen. Zo wordt bijvoorbeeld de I-parameter hoog gemaakt wanneer het systeem naar het gewenste pad convergeerde om zo offset fouten weg te werken. Door gebruik te maken van Type-2 Fuzzy MFs zou men ook in staat zijn de onzekerheden van GPS en IMU in het algoritme te verwerken. Echter is er ook een groot nadeel verbonden aan het gebruik van Fuzzy logica. Alle MFs, PID-parameters en Fuzzy rules moeten namelijk op voorhand gezocht en bepaald worden, dit wordt nog complexer wanneer Type-2 MFs zouden worden gebruikt, gezien deze uit een upper en lower functie bestaan.



Figuur 2.23: Simulatie Fuzzy PID controlealgoritme in Simulink

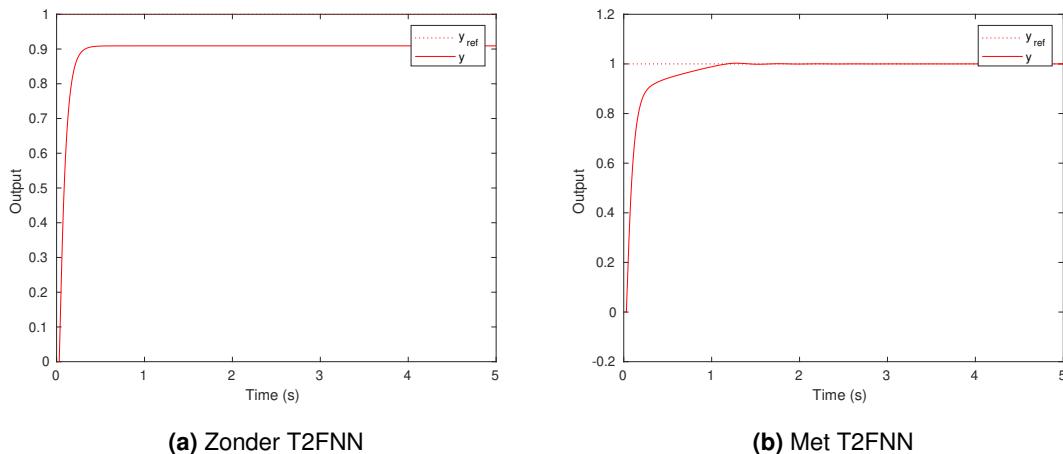


Figuur 2.24: Simulatie PID controlealgoritme met $v = 1$ m/s in Simulink

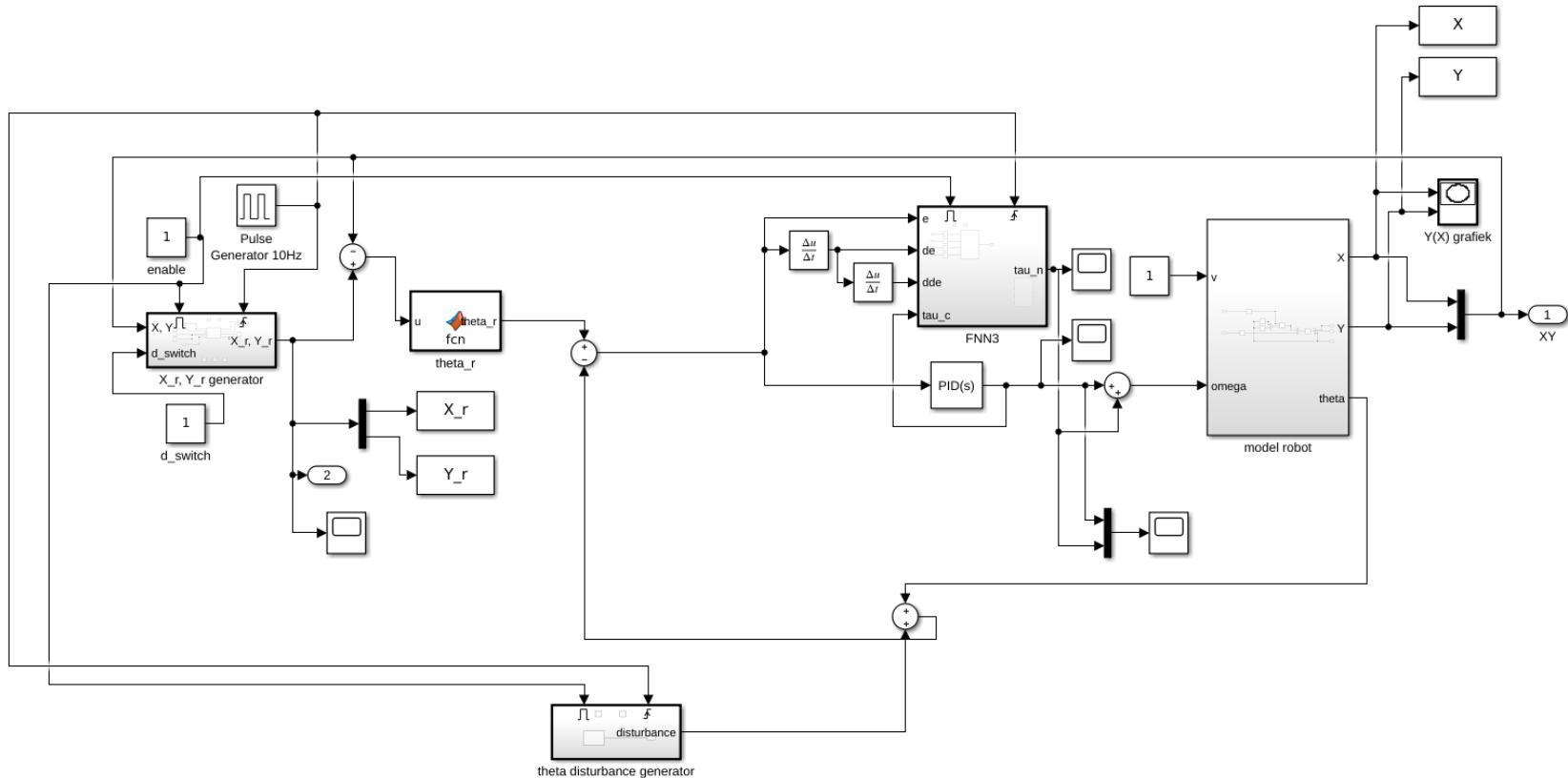
2.4.3 Type-2 Fuzzy Neural Network (T2FNN) in combinatie met Proportional Derivative (PD) controller

In [6] staat beschreven hoe men een traditionele PD controller combineert met een intelligente Takagi-Sugeno-Kang (TSK) Type-2 Fuzzy Neural Network (T2FNN) controller. Met behulp van een leeralgoritme gebaseerd op SMC zal de output van de traditionele PD controller gebruikt worden om de intelligente T2FNN controller te trainen. Dit principe noemt men Feedback Error Learning (FEL). De reden dat een intelligente controller voor point-to-point navigatie zich opdringt is de moeilijkheid om de juiste PD parameters te vinden omwille van de verschillende omstandigheden waarin de agrorobot zich bevindt en de onzekerheden op de sensorwaarden [6].

Dit soort controller werd tijdens de literatuurstudie eveneens grondig onderzocht op basis van de publicatie [6]. Bij een Matlab simulatie geïmplementeerd door prof. Kayacan is eveneens de correcte werking van het systeem vast te stellen, zoals te zien in figuur 2.25. Mijn eigen simulatie van het algoritme, waarvan het schema te zien is in figuur 2.26, heeft echter geen goede resultaten opgeleverd. Ter volledigheid wordt deze hier toch vermeld.



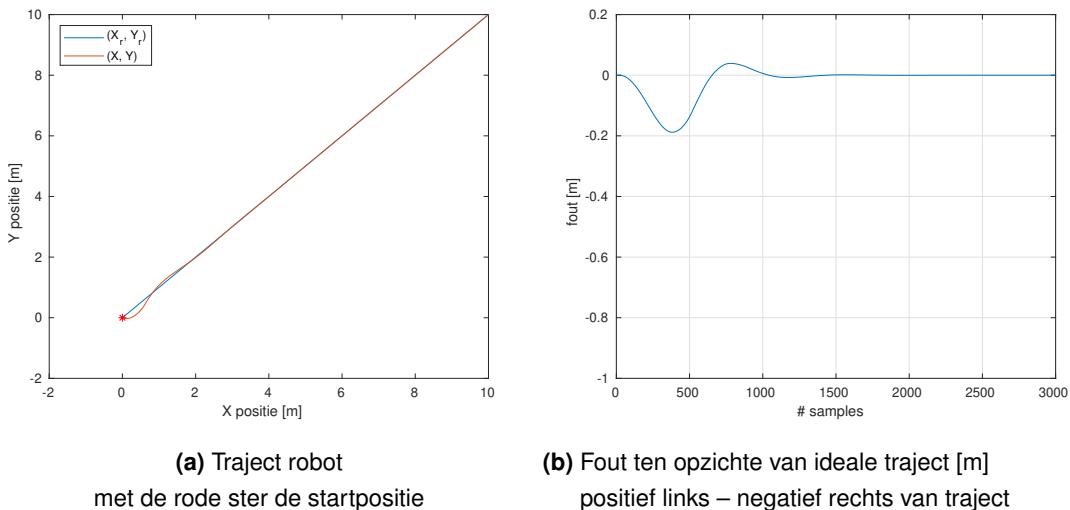
Figuur 2.25: Implementatie T2FNN controlealgoritme door prof. Kayacan



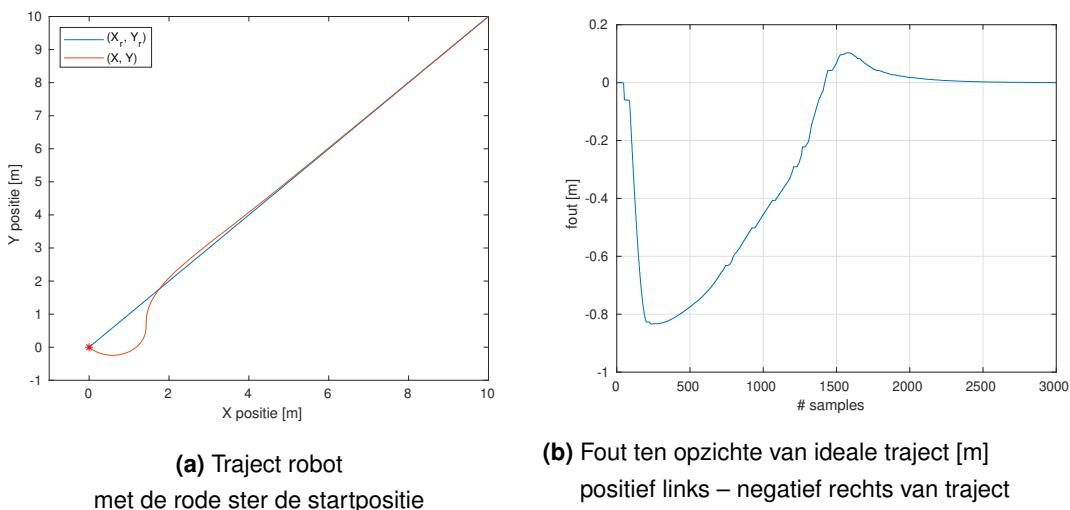
Figuur 2.26: Simulatie T2FNN controlealgoritme in Simulink

2.4.4 Vergelijking SMC versus Fuzzy PID algoritme

De drie algoritmen SMC, Fuzzy PID en T2FNN kunnen dienen om autonome point-to-point navigatie toe te passen op het veld. Hieronder worden het SMC en Fuzzy PID algoritme met elkaar vergeleken. In figuur 2.27 en figuur 2.28 zijn het traject en de fout van respectievelijk het SMC algoritme en het Fuzzy PID algoritme in dezelfde schaal weergegeven voor startpositie $P = (0, 0, -\frac{\pi}{4})^T$. Uit deze figuren is op te maken dat beide algoritmen de fout naar nul doen convergeren. Wel doet het SMC algoritme dit sneller. Bovendien is de gemaakte fout kleiner dan bij het Fuzzy PID algoritme.



Figuur 2.27: Simulaties voor startpositie $P = (0, 0, -\frac{\pi}{4})^T$ met SMC algoritme



Figuur 2.28: Simulaties voor startpositie $P = (0, 0, -\frac{\pi}{4})^T$ met Fuzzy PID algoritme

2.5 Sensoren

De perfecte sensor bestaat niet, vandaar is het van belang sensoren te combineren om tot juiste informatie te komen. Een methode die in de literatuur dikwijls wordt gebruikt om dit te bekomen is Kalman filtering. Kalman filtering maakt in elk tijdsinterval gebruik van geobserveerde variabelen om een schatting te maken van onbekende variabelen. Deze schatting van de onbekende variabelen zou nauwkeuriger moeten zijn dan het gebruik van enkel de geobserveerde waarden. De onnauwkeurigheid (ruis) van de schatting en van de observatie worden immers in rekening gebracht. Hierbij wordt verondersteld dat de ruis op de schatting en op de observatie normaal verdeeld is volgens $\mathcal{N}(\mu, \sigma^2)$ waarbij μ het gemiddelde en σ^2 de variantie is. De schatting wordt bepaald door een model.

Zoals hierboven vermeld wordt Kalman filtering toegepast per tijdsinterval. In wat hieronder volgt zal voor de eenvoud het tijdstip $t = k \cdot T_s$ kortweg genoteerd worden als tijdstip k , waarbij T_s de periode is voor het maken van de schattingen. De staat van de robot op een tijdstip $t = k$ wordt bepaald door de vector $(X_k, v_{X,k}, Y_k, v_{Y,k}, \theta_k, \omega_k)^T$. In plaats van rechtstreeks gebruik te maken van de metingen gedaan door de GPS en IMU, kan men de staat van de robot nauwkeuriger bepalen door Kalman Filtering. In wat volgt wordt nog steeds rekening gehouden met de notaties van de assenstelsels zoals geformuleerd in sectie 2.2. Wordt bijvoorbeeld het volgende verondersteld:

- $x_k = (X_k, v_{X,k}, Y_k, v_{Y,k}, \theta_k, \omega_k)^T$ de staat van de robot op het tijdstip $t = k$.
- $z_k = (X_{GPS,k}, v_{X,GPS,k}, Y_{GPS,k}, v_{Y,GPS,k}, \theta_{IMU,k}, \omega_{IMU,k})^T$ de metingen op het tijdstip $t = k$ bepaald door de GPS en de IMU.
- $u_k = (v_{k,actie}, \omega_{k,actie})^T$ de actie van de robot bepaald door een controlealgoritme. u_k is dus de snelheid en de hoeksnelheid die op $t = k$ op de motoren wordt gezet.

In de plaats van $x_k = z_k$ te stellen op het tijdstip $t = k$ wordt x_k met behulp van Kalman filtering voorspelt. Met een model wordt de volgende staat berekend, vervolgens wordt een Gain berekend die de gewichtsverdeling tussen deze berekende staat aan de hand van het model en de metingen bepaalt.

2.5.1 Kalman filtering model

Voor deze toepassing kan het volgende stelsel opgesteld worden:

$$\left\{ \begin{array}{l} X_k = X_{k-1} + v_{X,k-1} \cdot T_s + v_{k,actie} \cdot \cos(\theta) \cdot T_s + \text{procesruis} \\ v_{X,k} = v_{X,k-1} + v_{k,actie} \cdot \cos(\theta) + \text{procesruis} \\ Y_k = Y_{k-1} + v_{Y,k-1} \cdot T_s + v_{k,actie} \cdot \sin(\theta) \cdot T_s + \text{procesruis} \\ v_{Y,k} = v_{Y,k-1} + v_{k,actie} \cdot \sin(\theta) + \text{procesruis} \\ \theta_k = \theta_{k-1} + \omega_{k-1} \cdot T_s + \omega_k \cdot T_s + \text{procesruis} \\ \omega_k = \omega_{k-1} + \omega_{k,actie} + \text{procesruis} \end{array} \right. \quad (2.15)$$

Ter illustratie wordt de eerste vergelijking toegelicht. De positie X_k op het tijdstip $t = k$ is gelijk aan de som van de volgende termen:

- X_{k-1} : de positie op het vorige tijdstip $t = k - 1$ in het groot inertiaal assenstelsel. (informatie uit een vorige schatting)
- $v_{X,k-1} \cdot T_s$: de afstand afgelegd door de robot ten gevolge van de snelheidscomponent in de X-richting van het groot inertiaal assenstelsel op het tijdstip $t = k - 1$. (bepaald door metingen)
- $v_{k,actie} \cdot \cos(\theta) \cdot T_s$: de afstand afgelegd ten gevolge van de snelheidscomponent in de X-richting van het groot inertiaal assenstelsel die de robot zal aannemen in het komende tijdsinterval. (gevolg van de actie)
- de procesruis: ruis ten gevolge van het schatten.

Op een analoge wijze zijn de andere vergelijkingen in het stelsel 2.15 te verklaren.

In matrixvorm ziet het bovenstaande stelsel 2.15 er als volgt uit:

$$\begin{bmatrix} X_k \\ v_{X,k} \\ Y_k \\ v_{Y,k} \\ \theta_k \\ \omega_k \end{bmatrix} = \begin{bmatrix} 1 & T_s & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & T_s & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & T_s \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_{k-1} \\ v_{X,k-1} \\ Y_{k-1} \\ v_{Y,k-1} \\ \theta_{k-1} \\ \omega_{k-1} \end{bmatrix} + \begin{bmatrix} \cos(\theta) T_s & 0 \\ \cos(\theta) & 0 \\ \sin(\theta) T_s & 0 \\ \sin(\theta) & 0 \\ 0 & T_s \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_{k,actie} \\ \omega_{k,actie} \end{bmatrix} + w_k \quad (2.16)$$

Uit vergelijking 2.16 kan men een eerste vergelijking uit het Kalman filtering model, zoals beschreven in [8], herkennen. Bij dit model gaat men ervan uit dat de huidige staat van het systeem op tijdstip k kan geschreven worden als

$$x_k = F_k x_{k-1} + B_k u_k + w_k \quad (2.17)$$

Hieronder worden nog eens kort de overeenkomstige termen van vergelijking 2.16 en vergelijking 2.17 overlopen.

- x_k en x_{k-1} stellen de staat van het systeem voor op de respectievelijke tijdstippen $t = k$ en $t = (k - 1)$.
- F_k is het staten transitie model dat hier als volgt wordt gedefiniëerd:

$$F_k = \begin{bmatrix} 1 & T_s & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & T_s & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & T_s \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

- B_k is het controle input model dat hier als volgt wordt gedefinieerd indien de snelheid tussen de tijdsintervallen constant wordt beschouwd [9]:

$$B_k = \begin{bmatrix} \cos(\theta) T_s & 0 \\ \cos(\theta) & 0 \\ \sin(\theta) T_s & 0 \\ \sin(\theta) & 0 \\ 0 & T_s \\ 0 & 1 \end{bmatrix}$$

- w_k wordt verondersteld normaal verdeeld te zijn volgens $w_k \sim \mathcal{N}(0, Q_k)$ waarbij Q_k een covariantiematrix is tussen de geobserveerde waarden ten gevolge van de procesruis. Kies Q_k hier de eenheids-diagonaalmatrix : $I(6)$.

Een tweede stelsel dat in deze situatie kan worden opgesteld is het volgende:

$$\left\{ \begin{array}{l} X_{GPS,k} = X_k + \text{observatieruis} \\ v_{X,GPS,k} = v_{X,k} + \text{observatieruis} \\ Y_{GPS,k} = Y_k + \text{observatieruis} \\ v_{Y,GPS,k} = v_{Y,k} + \text{observatieruis} \\ \theta_{IMU,k} = \theta_k + \text{observatieruis} \\ \omega_{GPS,k} = \omega_k + \text{observatieruis} \end{array} \right. \quad (2.18)$$

Ter illustratie wordt de eerste vergelijking toegelicht. De meting $X_{GPS,k}$ op het tijdstip $t = k$ is gelijk aan de som van de volgende termen:

- X_k : de positie van de robot in het groot inertiaal assenstelsel op het huidige tijdstip $t = k$. (informatie uit een vorige schatting)
- de ruis ten gevolge van de observatie (houdt rekening met de onnauwkeurigheid van de sensor)

Op een analoge wijze zijn de andere vergelijkingen van het stelsel 2.18 te verklaren.

In matrixvorm ziet het bovenstaande stelsel 2.18 er als volgt uit:

$$\begin{bmatrix} X_{GPS,k} \\ v_{X,GPS,k} \\ Y_{GPS,k} \\ v_{Y,GPS,k} \\ \theta_{IMU,k} \\ \omega_{GPS,k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_k \\ v_{X,k} \\ Y_k \\ v_{Y,k} \\ \theta_k \\ \omega_k \end{bmatrix} + v_k \quad (2.19)$$

Uit vergelijking 2.19 kan men de tweede vergelijking uit het Kalman filtering model, zoals beschreven in [8], herkennen. Bij dit model gaat men ervan uit dat de huidige observatie van het systeem

op tijdstip k kan geschreven worden als

$$z_k = H_k x_k + v_k \quad (2.20)$$

Hieronder worden kort de overeenkomstige termen van vergelijking 2.19 en vergelijking 2.20 overlopen.

- H_k is het observatiemodel dat hier als een eenheidsmatrix wordt gedefinieerd: $I(6)$
- v_k wordt verondersteld normaal verdeeld te zijn volgens $v_k \sim \mathcal{N}(0, R_k)$ waarbij R_k een covariantiematrix is tussen de geobserveerde waarden ten gevolge van de observatieruis, die hier als volgt wordt definiëerd:

$$R_k = \begin{bmatrix} \sigma(X_k, X_k) & \sigma(X_k, v_{X,k}) & \sigma(X_k, Y_k) & \sigma(X_k, v_{Y,k}) & \sigma(X_k, \theta_k) & \sigma(X_k, \omega_k) \\ \sigma(v_{X,k}, X_k) & \sigma(v_{X,k}, v_{X,k}) & \sigma(v_{X,k}, Y_k) & \sigma(v_{X,k}, v_{Y,k}) & \sigma(v_{X,k}, \theta_k) & \sigma(v_{X,k}, \omega_k) \\ \sigma(Y_k, X_k) & \sigma(Y_k, v_{X,k}) & \sigma(Y_k, Y_k) & \sigma(Y_k, v_{Y,k}) & \sigma(Y_k, \theta_k) & \sigma(Y_k, \omega_k) \\ \sigma(v_{Y,k}, X_k) & \sigma(v_{Y,k}, v_{X,k}) & \sigma(v_{Y,k}, Y_k) & \sigma(v_{Y,k}, v_{Y,k}) & \sigma(v_{Y,k}, \theta_k) & \sigma(v_{Y,k}, \omega_k) \\ \sigma(\theta_k, X_k) & \sigma(\theta_k, v_{X,k}) & \sigma(\theta_k, Y_k) & \sigma(\theta_k, v_{Y,k}) & \sigma(\theta_k, \theta_k) & \sigma(\theta_k, \omega_k) \\ \sigma(\omega_k, X_k) & \sigma(\omega_k, v_{X,k}) & \sigma(\omega_k, Y_k) & \sigma(\omega_k, v_{Y,k}) & \sigma(\omega_k, \theta_k) & \sigma(\omega_k, \omega_k) \end{bmatrix}$$

Hierbij is $\sigma(a, b) = \mathbb{E}[(a - \mathbb{E}(a))(b - \mathbb{E}(b))]$ de covariatie tussen a en b . Merk op dat hierin de definitie van de variantie kan in worden herkend: $\sigma(a, a) = \mathbb{E}[(a - \mathbb{E}(a))(a - \mathbb{E}(a))] = \mathbb{E}[(a - \mathbb{E}(a))^2] = \sigma_a^2 = \text{var}(a)$. Hieruit volgt dat alle elementen op de hoofddiagonaal gelijk zijn aan hun respectieve varianties. Verder geldt er nog dat $\sigma(a, b) = \sigma(b, a)$. Een covariantiematrix is hier steeds een symmetrische matrix ten opzichte van de hoofddiagonaal. Ook geldt

- $\sigma(a, b) = \sigma(b, a) > 0$ indien b toeneemt bij een toename van a of omgekeerd
- $\sigma(a, b) = \sigma(b, a) < 0$ indien b afneemt bij een toename van a of omgekeerd.

Kalman filtering kan beschreven worden in een enkele formule, maar om het concept verstaanbaarder te maken wordt ze doorgaans opgesplitst in twee acties de voorspellende ("predict") en de uitvoerende ("update") actie. Eerst moet nog geduid worden dat $\hat{x}_{n|m}$ staat voor de geschatte waarde van x op het tijdstip n , waarbij alle $m \leq n$ observaties in rekening zijn gebracht. $P_{n|m}$ stelt de covariantiematrix van de fout van het geschatte signaal voor op het tijdstip n , waarbij alle $m \leq n$ observaties in rekening zijn gebracht.

predict

$$\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} + B_k u_k \quad (2.21)$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \quad (2.22)$$

update

$$K_k = \frac{P_{k|k-1} H_k^T}{R_k + H_k P_{k|k-1} H_k^T} \quad (2.23)$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k (z_k - H_k \hat{x}_{k|k-1}) \quad (2.24)$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} (I - K_k H_k)^T + K_k R_k K_k^T \quad (2.25)$$

Vergelijking 2.25 kan bij een optimale gain K_k , waarbij de residuële fout wordt geminimaliseerd, herschreven worden als $P_{k|k} = (I - K_k H_k)P_{k|k-1}$. De afleiding staat beschreven in [8].

In elke update fase wordt er een gain K_k berekend. Deze gain is functie van de covariantiematrix van de fout op het geschatte signaal $P_{k|k-1}$ en de covariantiematrix van de geobserveerde waarden ten gevolge van de observatieruis R_k , zoals te zien in vergelijking 2.24. Bijgevolg zal deze gain in vergelijking 2.25 bepalen in welke verhouding de geobserveerde waarden z_k en de waarden van de vorige schatting $\hat{x}_{k|k-1}$ zullen doorwegen ter bepaling van de nieuwe schatting $\hat{x}_{k|k}$.

2.5.2 Simulatie

Vooraleer er kan gesimuleerd worden, dient men nog de startvoorwaarden in te vullen. De staat $x_{0|0}$ en de covariantie hierop $P_{0|0}$ moeten op het tijdstip $t = 0$ gekend zijn. Om de Kalman filter te testen wordt ervoor gekozen de waarden van de actiematrix $u = (v_k, \omega_k)^T$ te gebruiken uit de simulatie zoals te zien in figuur 2.24. Hierin is het controlealgoritme voor de hoeksnelheid ω een PID-regeling en wordt de snelheid constant gehouden $v = 1$ m/s. Hierbij is uitgegaan van een startpositie $P = (2, -1, \pi)^T$. Bijgevolg is de staat op $t = 0$ gekend: $x = 2$, $y = -1$, $v_0 = 1$ m/s, $\theta_0 = \pi$, $\omega_0 = -0.8727$ (uit de resultaten van de simulatie gehaald).

$$x_{0|0} = \begin{bmatrix} X_0 \\ v_0 \cdot \cos(\theta_0) \\ Y_0 \\ v_0 \cdot \sin(\theta_0) \\ \theta_0 \\ \omega_0 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \\ -1 \\ 0 \\ \pi \\ -0.8727 \end{bmatrix}$$

Ondanks dat men 100% zeker is van deze startwaarden mag de initiële covariantiematrix $P_{0|0}$ toch niet gelijk worden gesteld aan de nulmatrix, anders zou $P_{k|k}$ voor altijd nul zijn. Daarom stelt men de covariantiematrix arbitrair gelijk aan de eenheidsmatrix $P_{0|0} = I$.

Het totale Simulink schema van de simulatie van de Kalman filter is te zien in figuur 2.37.

De meest essentiële blokken worden hieronder kort toegelicht:

- "Plant": Dit is een subsysteem dat het model van de robot implementeert zoals beschreven in figuur 2.3. De ingang u van dit blok stelt de actie $u_k = (v_{k,actie}, \omega_{k,actie})^T$ voor in de Kalman filter op het tijdstip $t = k$. De uitgang y is de exacte staat van de robot. Op het tijdstip $t = k$ wordt dit dus $x_k = (X_k, v_{X,k}, Y_k, v_{Y,k}, \theta_k, \omega_k)^T$.
- "Gaussian noise . . .": Dit is een Matlab-functie waarmee gaussiaanse observatieruis wordt toegevoegd. De matlab code om de observatieruis ten gevolge van de onzekerheid op de x-coördinaat (door de onnauwkeurigheid van de GPS) te implementeren is te zien in listing 2.8. Hierbij genereert randn(1) een waarde die normaal verdeeld is volgens $\mathcal{N}(0, 1)$. Verder is hier gesteund op de volgende eigenschap: indien $X \sim \mathcal{N}(\mu_X, \sigma_X^2)$ en a en b constanten zijn, dan geldt voor $Y = a \cdot X + b$ dat $Y \sim \mathcal{N}(a \cdot \mu_X + b, a^2 \cdot \sigma_X^2)$. In dit geval is $X \sim \mathcal{N}(0, 1)$. Na

de bewerking $Y = \sigma_{des} \cdot X + \mu_{des}$ (index $des \rightarrow$ desired), krijgt men de gewenste normale verdeling voor Y : $Y \sim \mathcal{N}(\mu_{des}, \sigma_{des}^2)$.

```

1 function y = fcn()
2 standard_dev = 0.08;
3 mu = 0;
4 y = standard_dev.*randn(1) + mu;
```

Listing 2.8: Matlab code Gaussian noise x_sens blok

- "Kalman filter": Enabled and Triggered Subsystem dewelke het Kalman algoritme implementeert. De gebruikte functie is te zien in listing 2.9. De functie "kalmanf" implementeert de formules 2.21 tot 2.25.

```

1 function y = fcn(u,z)
2
3 persistent x P
4 %%%%%% initial conditions %%%%%%
5 if isempty(x)
6     % [x, vx, y, vy, theta, omega]
7     x = [2; -1; -1; 0; pi; -0.8727];
8 end
9 if isempty(P)
10    P = eye(6); % can't be zero, otherwise it will stay zero for
11    ever!
12 end
13 theta_o = z(5);
14 Ts = 0.1;
15
16 %standard variations
17 varx = 0.08^2;
18 varvx = 0.1^2;
19 vary = 0.08^2;
20 varvy = 0.1^2;
21 vartheta = (10.*(pi/180))^2;
22 varomega = (4.*((pi/180)))^2;
23
24 %%%%% the matrices %%%%%%
25 s.X = x;
26 s.P = P;
27 s.z = z;
28 s.u = u;
29 % A is F in text!!
```

```

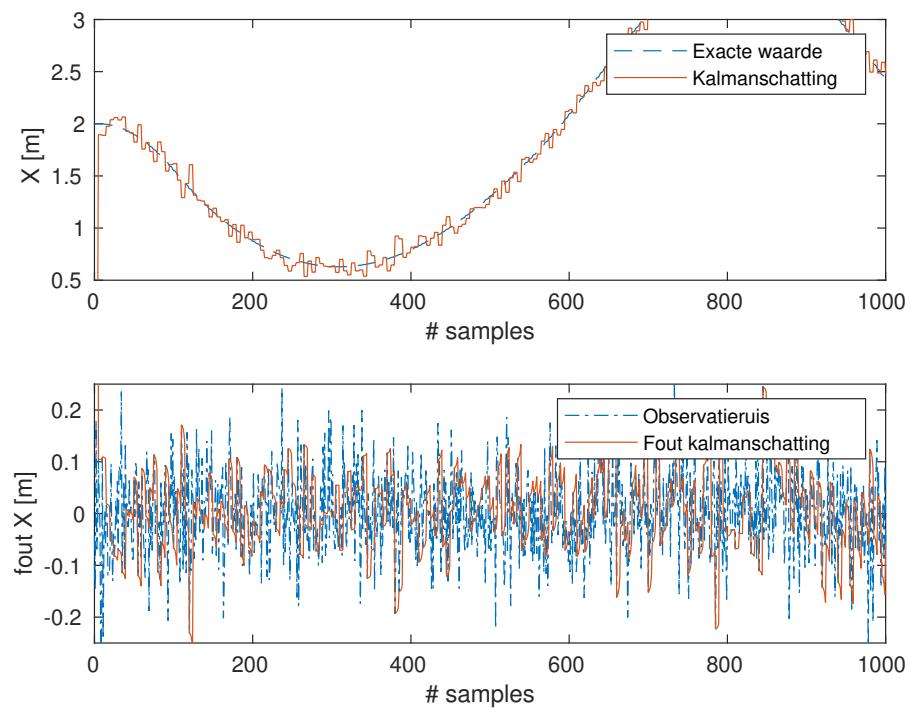
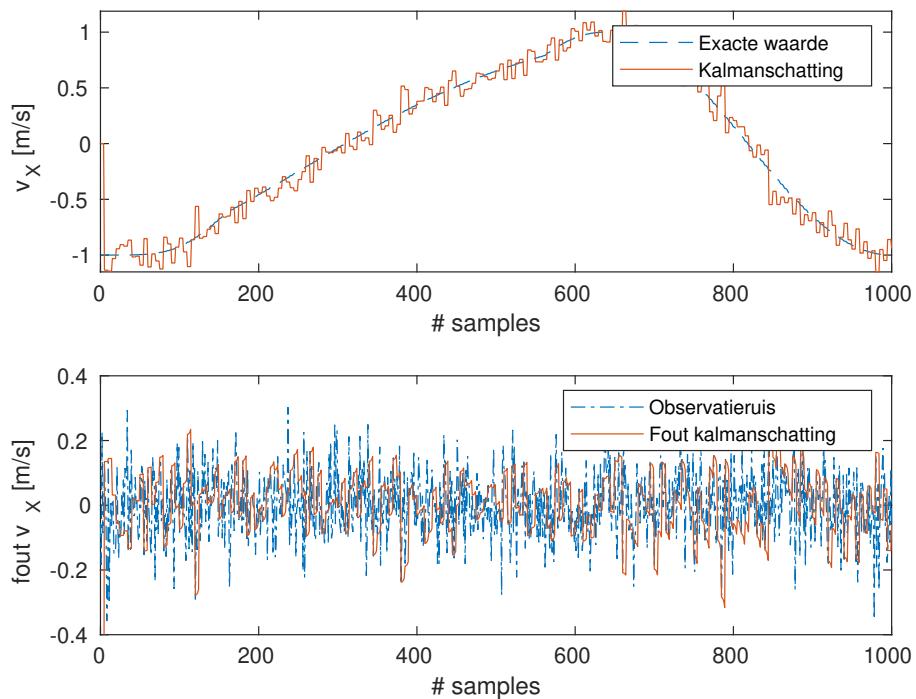
29 s.A = [1 Ts 0 0 0 0;
30      0 1 0 0 0 0;
31      0 0 1 Ts 0 0;
32      0 0 0 1 0 0;
33      0 0 0 0 1 Ts;
34      0 0 0 0 0 1];
35 s.B = [cos(theta_o)*Ts 0;
36      cos(theta_o) 0;
37      sin(theta_o)*Ts 0;
38      sin(theta_o) 0;
39      0 Ts;
40      0 0];
41 s.H = eye(6);
42
43 s.R = [varx 0 0 0 0 0;
44      0 varvx 0 0 0 0;
45      0 0 vary 0 0 0;
46      0 0 0 varvy 0 0;
47      0 0 0 0 vartheta 0;
48      0 0 0 0 0 varomega];
49 s.Q = eye(6);
50 %%%%%% calculate approximation of the next state %%%%%%
51 ret = kalmanf(s);
52 x = ret.x;
53 P = ret.P;
54 y = x;

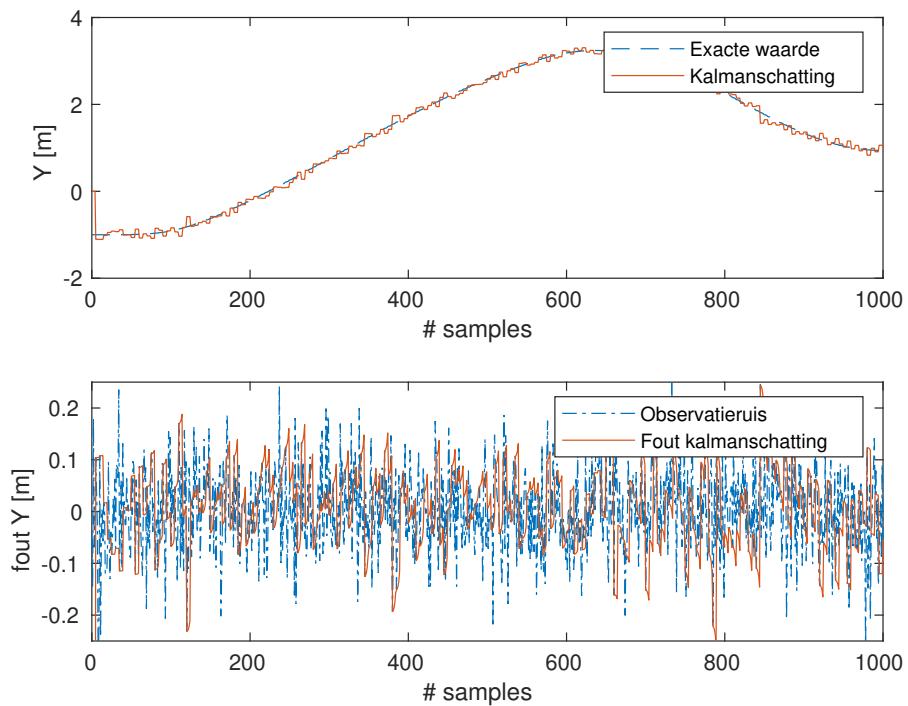
```

Listing 2.9: Matlab code Kalman filter blok

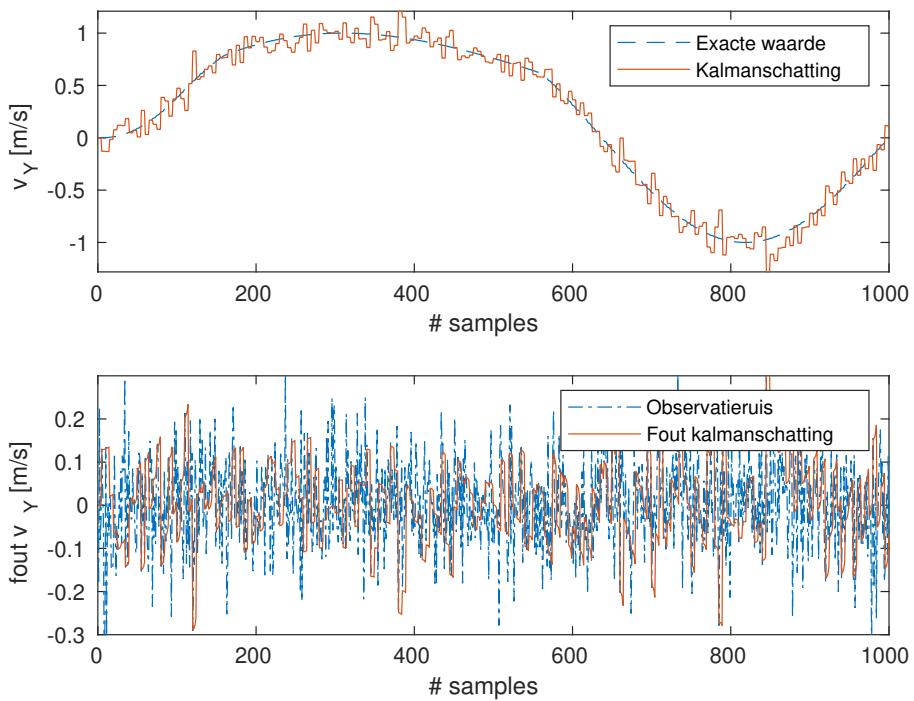
2.5.3 Resultaten

Om de Kalman filter te evalueren worden er per variabele twee plots gemaakt. Enerzijds plot men het geschatte resultaat (de output van het "Kalman filter" blok → "rese" op figuur 2.37) samen met het exacte resultaat (de staat van de robot: de output van het "Plant" blok → "res" op figuur 2.37). In deze plot kan men zien of de filter de exacte waarde wel goed benadert. Anderzijds is de observatieruis (resv-res) en de fout van de schatting (rese-res) weergegeven. Hierbij is het voor een goed resultaat belangrijk dat de fout van de schatting over een groot bereik kleiner is dan de observatieruis. De resultaten voor de volledige staat van de robot zijn te zien in figuur 2.29 tot 2.34.

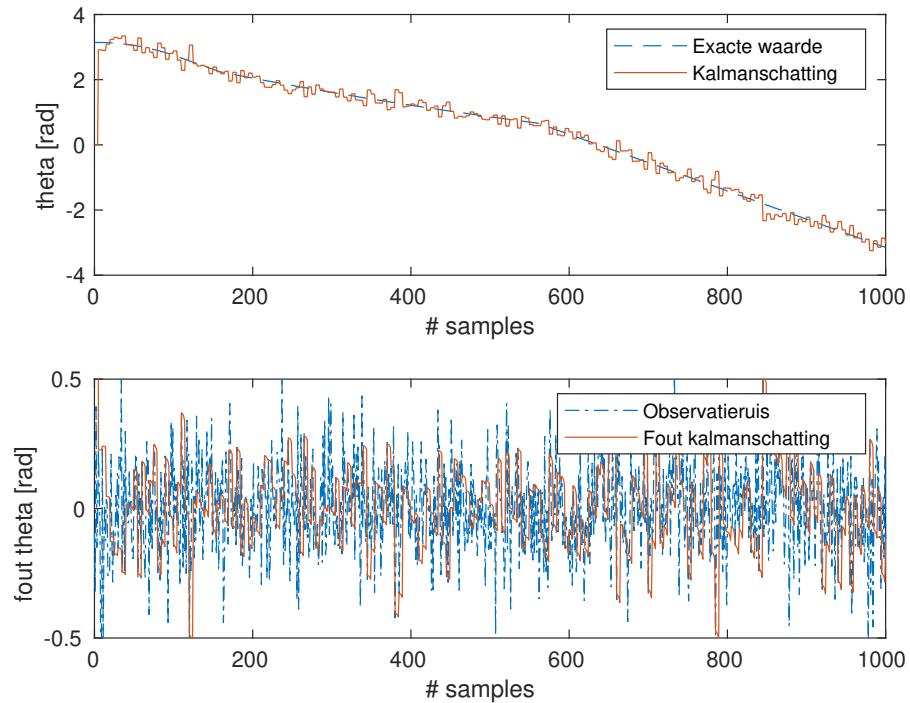
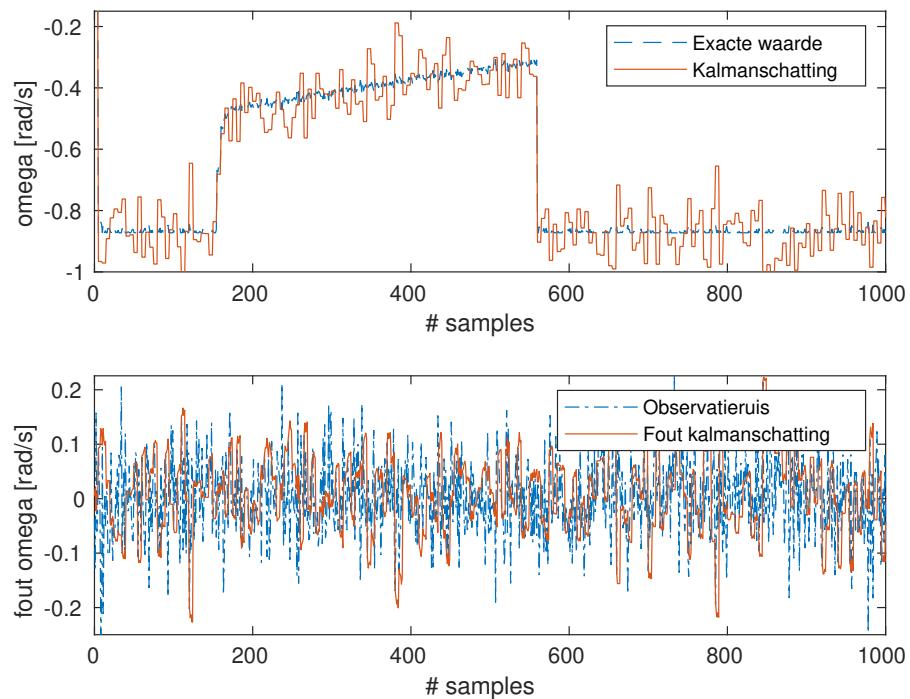
**Figuur 2.29:** Kalman filter responsie voor x **Figuur 2.30:** Kalman filter responsie voor v_x



Figuur 2.31: Kalman filter responsie voor y



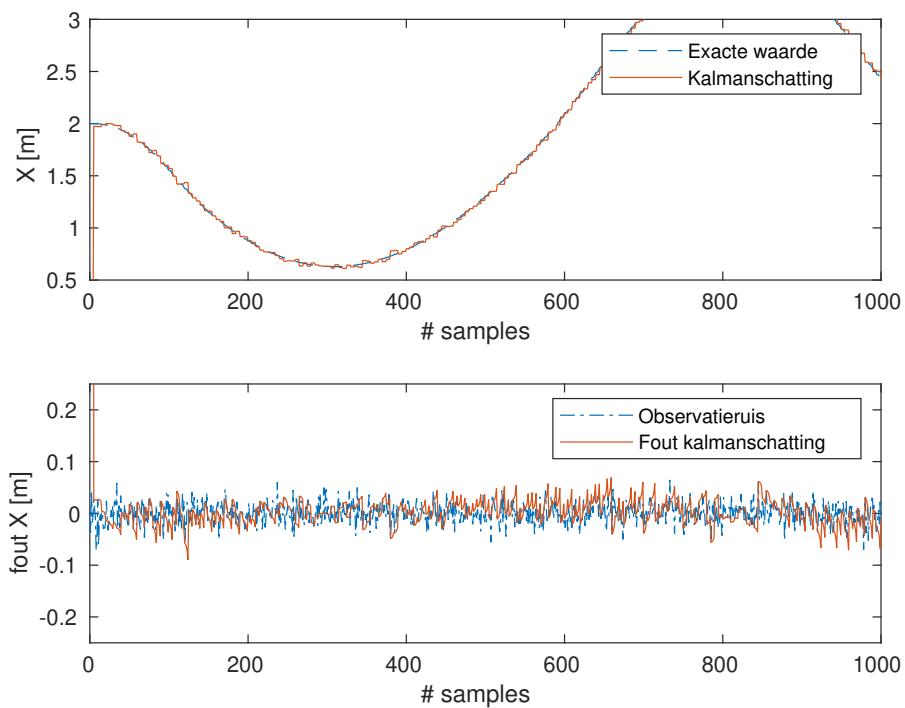
Figuur 2.32: Kalman filter responsie voor v_y

**Figuur 2.33:** Kalman filter responsie voor θ **Figuur 2.34:** Kalman filter responsie voor ω

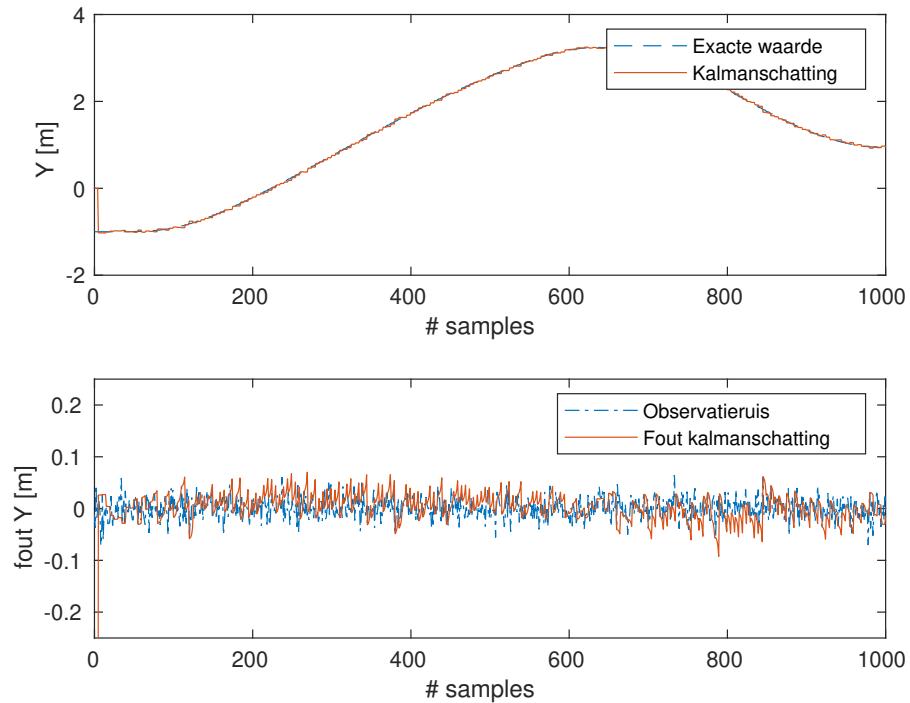
Of de Kalman filter voordeliger is dan de geobserveerde resultaten hangt sterk af van de standaardafwijking. Hierboven werd gekozen voor:

$$\left\{ \begin{array}{l} \sigma_X = 0.08 \\ \sigma_{v_X} = 0.1 \\ \sigma_Y = 0.08 \\ \sigma_{v_Y} = 0.1 \\ \sigma_\theta = 10 \cdot \left(\frac{\pi}{180} \right) \\ \sigma_\omega = 4 \cdot \left(\frac{\pi}{180} \right) \end{array} \right.$$

Stelt men echter vast dat de GPS werkt op een nauwkeurigere stand waardoor $\sigma_X = 0.02$ en $\sigma_Y = 0.02$, dan ziet men voor de variabelen x en y het resultaat in figuur 2.35 en figuur 2.36. Eerst en vooral stelt men vast dat de ruis sterk is afgenoem. Dit spreekt voor zich gezien de volgende stelling over σ . In 68.3% van de gevallen ligt het ruissignaal in het interval $[-\sigma, +\sigma]$, in 95.4% van de gevallen ligt het ruissignaal in het interval $[-2\sigma, +2\sigma]$ en in 99.7% van de gevallen in het interval $[-3\sigma, +3\sigma]$. Ten tweede stelt men vast dat de fout van de kalmanschatting dikwijs groter is dan die van de observatieruis. De vraag is of het wel zinvol is om de Kalman filtering te gebruiken voor de variabelen X en Y wanneer σ dermate klein (en de GPS dus dermate nauwkeurig) is. In deze situatie blijken de geobserveerde sensorwaarden dus nauwkeuriger dan de geschatte waarden door de Kalman filter.



Figuur 2.35: Kalman filter responsie voor x met $\sigma_x = \sigma_y = 0.02$

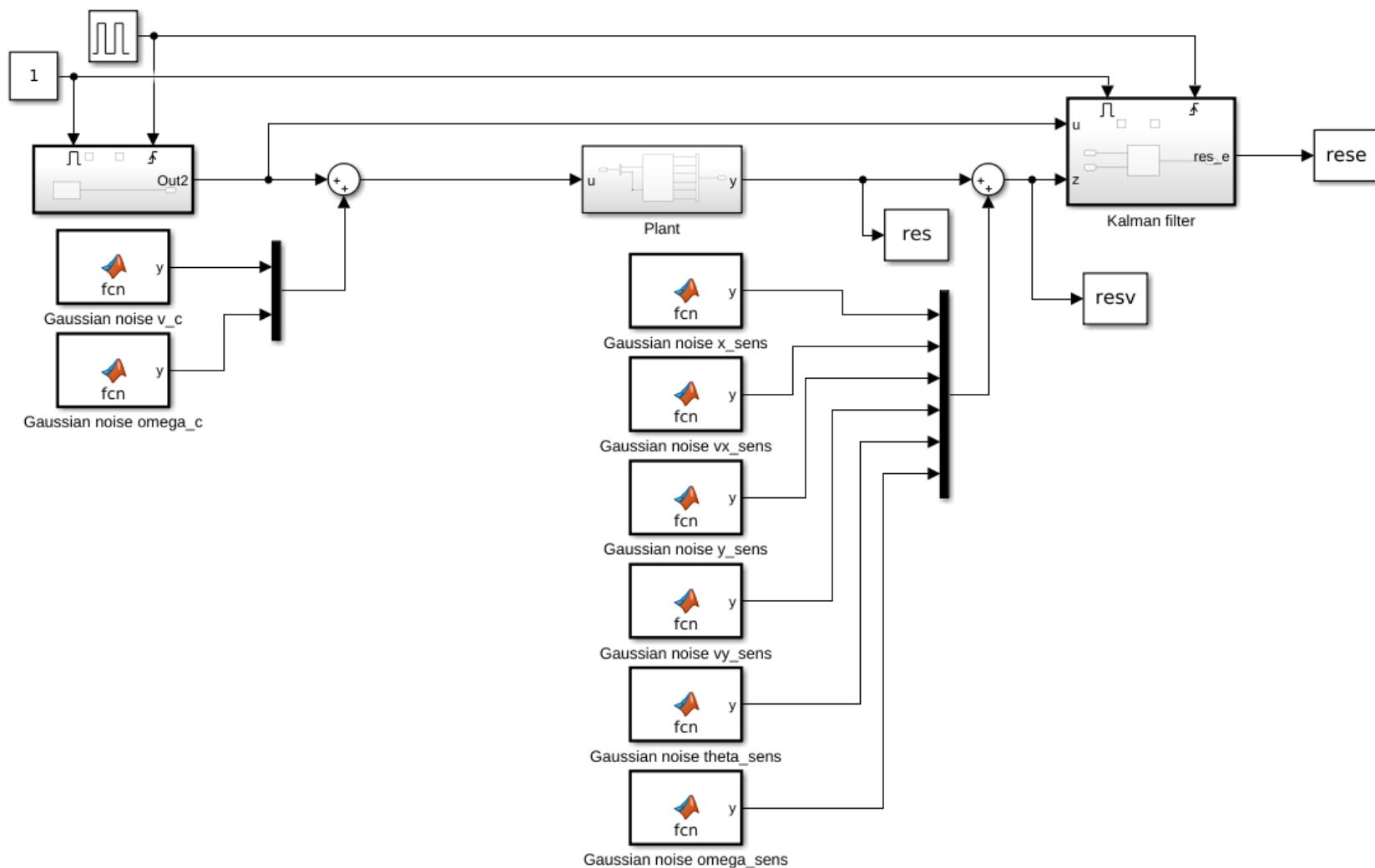


Figuur 2.36: Kalman filter responsie voor y met $\sigma_x = \sigma_y = 0.02$

De covariantiematrix kan dikwijls bepaald worden uit informatie van de datasheets. Voor de GPS zit de standaardafwijking in de noordelijke, de oostelijke en de horizontale richting reeds vervat in het GPGGA formaat van een NMEA sentence. Om de juiste covariantiematrix te bekomen kan in laatste instantie beroep worden gedaan op het Autocovariance Least-Squares (ALS) algoritme, waarmee uit de ruwe data van de sensor de covariantiematrix kan bepaald worden.

2.5.4 Besluit

Kalman filtering biedt de mogelijkheid de staat van de robot te schatten aan de hand van geobserveerde data waarvan de ruis normaal verdeeld is. De filtering heeft tot doel een nauwkeurigere schatting te maken dan de gemeten, geobserveerde variabele. De standaarddeviatie van de te observeren variabele bepaalt de mate dat de schatting doorweegt ten opzichte van de geobserveerde data. Toch kan het gebeuren dat bij een voldoende kleine standaarddeviatie, de sensor data nauwkeuriger is dan de Kalman schatting waardoor het op dat ogenblik interessanter is, voor deze variabele, om met de geobserveerde sensor data te werken in plaats van met de Kalman schatting. Ook is het interessant dat kalman filtering niet steeds een nieuwe observatiewaarde nodig heeft om een schatting te maken. Op die manier kan de updatefrequentie waarop sensor-data wordt gepubliceerd verhoogd worden waardoor de GPS, met een updatefrequentie van 10 Hz, in de praktijk zelfs lager, niet meer de bottleneck is. Ook op tijdstippen dat er geen sensordata vorhanden is kan er nu informatie over de staat van de robot verkregen worden.



Figuur 2.37: Simulatie Kalman filter model Simulink

2.6 Coördinatensystemen

Om tot een goede werking van een controle algoritme te komen is een nauwkeurige GPS lokalisatie nodig. In tabel 2.1 zijn de drie GPS systemen weergegeven met hun respectievelijke nauwkeurigheid.

Systeem	Nauwkeurigheid
Global Positioning System (GPS)	> 5 m
Differential Global Positioning System (DGPS)	≈ 20 cm
Real Time Kinematic (RTK)	≈ cm

Tabel 2.1 GPS systemen met hun nauwkeurigheid

Deze systemen maken gebruik van een coördinatenreferentiesystemen om aan elke positie op de aarde een uniek coördinaat toe te kennen. Het World Geodetic System 1984 (WGS84) komt op enkele centimeters na overeen met het International Terrestrial Reference System (ITRS). Op deze coördinatenreferentiesystemen zit een variatie van enkele centimeters over de jaren heen vanwege de drift van de continenten. Zoals te zien in tabel 2.1 is dit voor GPS niet problematisch, gezien de onnauwkeurigheid hier dusdanig groot is waardoor men deze variatie kan verwaarlozen. Voor meer nauwkeurige systemen, zoals DGPS en RTK-GPS, speelt deze variatie wel een rol. Daarom gaat men over tot European Terrestrial Reference System (ETRS) als coördinatenreferentiesysteem, waar dit probleem wordt verholpen door een vast referentiepunt (basisstation) op het Europese vasteland te nemen.

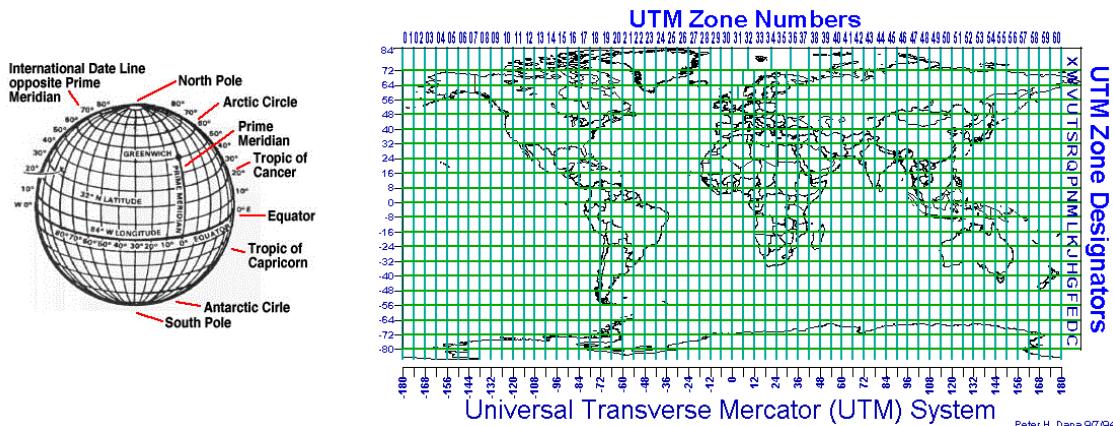
DGPS maakt gebruik van differentiaalcorrecties om zijn nauwkeurigheid op te drijven. Een RTK-GPS heeft een nog grotere positienauwkeurigheid vanwege de Flemish Positioning Service (FL-EPOS)-correcties die door een referentiepaal dichtbij wordt berekend en over het 4G netwerk naar het GPS toestel worden verstuurd.

Met behulp van RTK-GPS is men in staat nauwkeurige GPS posities te ontvangen in een geografisch coördinatenstelsel zoals links weergegeven op figuur 2.38. Om de controlealgoritmen van hierboven en Kalman filtering te kunnen toepassen is echter een (X,Y) coördinatenstelsel vereist. Om van een geografisch coördinatenstelsel over te gaan tot een cartografisch coördinatenstelsel zijn er verschillende projecties mogelijk. Men kan beroep doen op een Lambert projectie of op het Universal Transverse Mercator coordinate system (UTM). Deze laatste wordt rechts weergegeven op figuur 2.38 en zal ook gebruikt worden in deze masterproef omwille van het grote aantal beschikbare *libraries*. UTM bestaat uit transversale mercatorprojecties voor 60 stroken van west naar oost van elk zes lengtegraden breed. Door deze opsplitsing is de vertekening gering [13]. UTM heeft over heel West Europa een afwijking van 400 ppm, dat wil zeggen dat er over een afstand van 1 km een fout van 40 cm optreedt. De Lambert projectie is een Belgische projectie, door het beperkte oppervlak haalt men hierdoor een fout van slechts 10 ppm wat overeenkomt met een fout van 10 cm per km. Echter zijn deze gemaakte fouten steeds hetzelfde, waardoor de grootte ervan geen invloed heeft op de werking van het algoritme.

Veronderstel de volgende situatie:

- UTM wordt gebruikt als coördinatensysteem.
- De robot dient een rechte lijn van 1 km af te rijden op eenzelfde breedtegraad.
- De robot start op deze breedtegraad.

Naargelang waar de robot zich op de projectie bevindt, kan het zijn dat hij na 1 km 40 cm afwijkt van deze breedtegraad. Echter is dit niet zo'n groot probleem gezien de rij hier vlak naast eenzelfde afwijking zal vertonen ten opzichte van deze breedtegraad. Op dit deel van de projectie is de vertekening immers gelijk. Omwille van deze reden is het gerechtvaardigd om voor het UTM systeem te opteren ook al is de fout hier groter dan bij Lambert projectie.



Figuur 2.38: Links: geografisch coördinatenstelsel (longitude-latitude) – Rechts: cartografisch coördinatenstelsel (UTM)

Hoofdstuk 3

Implementatie

3.1 Inleiding

Het algoritme dat gebruik maakte van de SMC-regelaar was tijdens de simulaties stabiel en zocht daar voldoende snel de evenwichtspositie op. Bovendien is de implementatie ervan erg eenvoudig en het aantal te tunen parameters klein. Om deze redenen is ervoor gekozen dit algoritme te implementeren op de robot. Dit in combinatie met Kalman filtering waardoor op frequentere basis een commando naar de motoren kan worden verstuurd en gemeten waarden met een grote variantie worden uitgemiddeld.

Zoals eerder vermeld zal deze implementatie gebeuren in ROS. ROS is een ontwikkelingsplatform voor robot software waarin het mogelijk is hardware- en software functies te abstracteert tot nodes. ROS maakt vervolgens communicatie tussen deze nodes mogelijk. Er is keuze tussen:

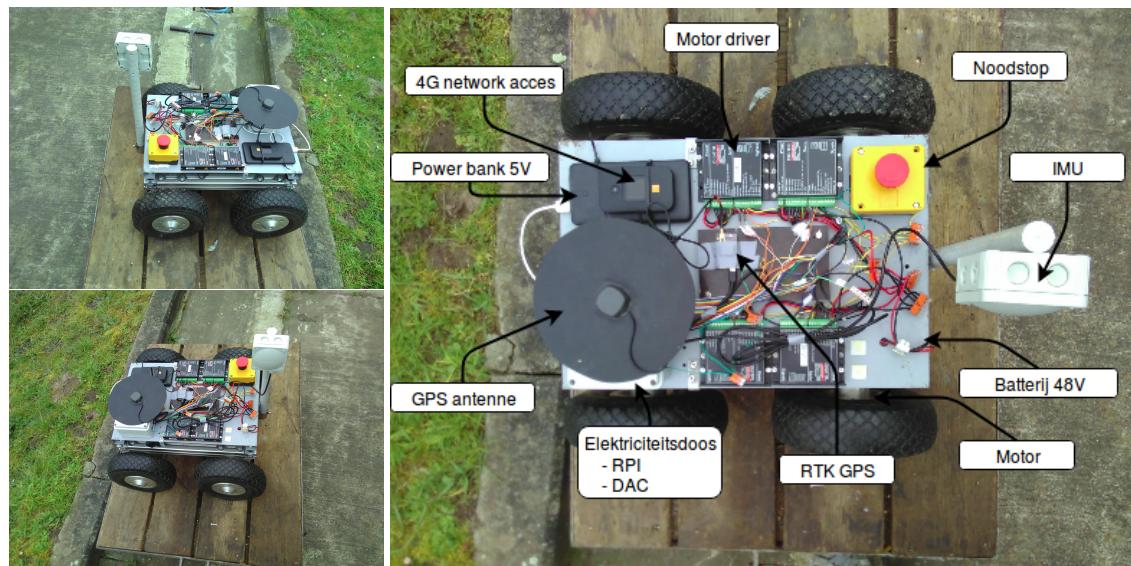
- Publisher - Subscriber: Typisch is dit een sensor die aan een bepaalde frequentie data spant. Op het computersysteem waarop de roscore draait is er dan een node aangemaakt die de socketverbinding met de sensor verzorgt. Vervolgens wordt een message opgevuld aan de hand van de ruwe sensordata. Deze message wordt dan gepubliceerd op een topic. Andere nodes kunnen vervolgens subscriben op deze topic.
- Server - Client: Dit is te vergelijken met een functie oproep over verschillende nodes heen. De client vraagt hierbij aan een andere node, de server, een berekening te doen aan de hand van meegegeven parameters. De server geeft vervolgens de resultaten van de berekening terug.
- ActionServer - ActionClient: Dit is eveneens te vergelijken met een functie oproep over verschillende nodes heen. De client vraagt hierbij aan een andere node, de ActionServer, een actie uit te voeren.

Door deze opbouw is een ROS project eenvoudig uit te breiden en modulair. Bovendien laat het toe bepaalde functionaliteiten van de robot apart te testen.

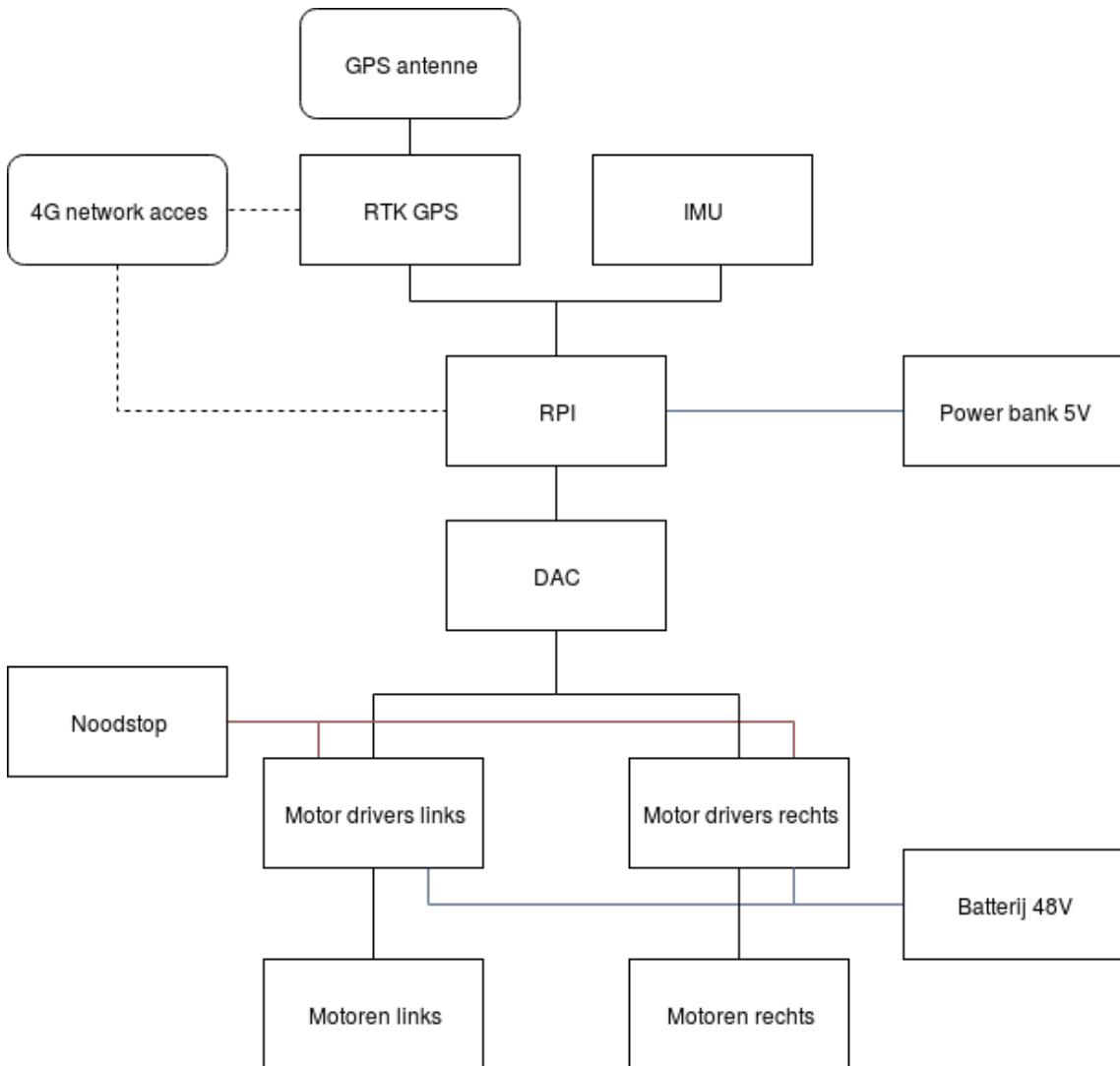
In de subsectie *Software* worden de verschillende ROS nodes overlopen. In de subsectie *Hardware* wordt de hardware implementatie van de robot besproken.

3.2 Hardware

Een afbeelding van de robot waarop de hardware elementen zijn aangeduid, is te zien in figuur 3.1. Zoals in de inleiding vermeld worden de vier wielen elk aangedreven door een tweepolige borstelloze DC motor (type EC-max 40 serie van Maxon motor) die op zijn beurt wordt aangedreven door een motordriver (type 4-Q-EC EC DECV 50/5 305259 ook van Maxon motor). Verder is er op de robot een RTK - GPS gemonteerd van de fabrikant Emlid. Deze is vastgetaped op houten bordje centraal op de robot. Gezien we gebruik maken van de snelheidmetingen van de GPS doet de oriëntatie ertoe. De antenne van de Emlid Reach is gemonteerd op de elektriciteitsdoos links vooraan op de robot. In deze elektriciteitsdoos zit de Raspberry Pi en de LTC1661 DAC. De Razor 9-DOF IMU van Sparkfun is gemonteerd op een zekere hoogte, op een plastic stok, om interferentie zoveel mogelijk te vermijden. Na kalibratie van de IMU moet erop worden gelet dat er geen nieuwe metalen elementen worden gebruikt bij montage, anders zijn de bekomen kalibratiewaarden niet meer van toepassing, meer hierover in sectie 3.3.3. De 48 V lithium magnesium batterij om de motoren en motordrivers aan te drijven bevindt zich onder het robot platform. Een powerbank om de Raspberry Pi te voeden is rechtsboven op het robotplatform te zien. Daarbovenop is het bakje bevestigd dat aangesloten is op 4G netwerk om de RTK gps van FLEPOS-correcties te voorzien. Deze hotspot wordt ook gebruikt om *ssh* naar de Raspberry Pi mogelijk te maken. Een blokdiagram van de hardware implementatie is te zien in figuur 3.2.



Figuur 3.1: Robot hardware implementatie

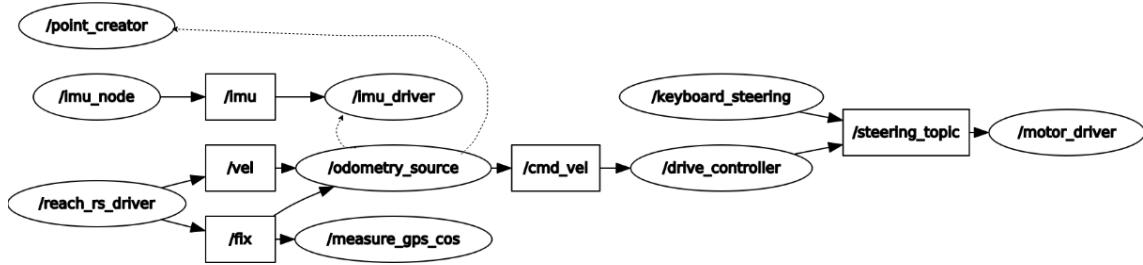


Figuur 3.2: Blokdiagram hardware implementatie

3.3 Software

Een overzicht van alle ROS nodes en de ROS topics zijn te zien in figuur 3.3. De Ros service *Angle.srv* van de *odometry_source* naar de *imu_driver* en de ROS service *GetPoint.srv* van de *odometry_source* naar de *point_creator* zijn op deze figuur weergegeven in stippellijn. Gewone topics zijn weergegeven in volle lijnen. De ROS nodes zullen in wat volgt elk afzonderlijk besproken worden.

Opmerking: De volledige *catkin_ws* met de code van de verschillende nodes is terug te vinden op de githubpagina <https://github.com/axelwillekens/Point-to-Point-Navigation>.



Figuur 3.3: Rosnodes (ovalen), Rostopics (rechthoeken, volle pijlen) en Rosservices (stippellijn pijlen)

3.3.1 reach_rs_driver

Voor de Reach RTK GPS module is er gebruik gemaakt van de *reach_rs.driver* [14]. De gepubliceerde topics zijn hieronder opgeliist.

- $\sim\text{fix}$: The GNSS fix (*sensor_msgs/NavSatFix*)
- $\sim\text{vel}$: Reported velocity (*geometry_msgs/TwistStamped*)
- $\sim\text{time_reference}$: Time reference (unused)
- $/diagnostics$: Diagnostics about the hardware driver (see <http://wiki.ros.org/diagnostics>)

Op de $\sim\text{fix}$ topic worden er *sensor_msgs/NavSatFix* gepubliceerd. Informatie die hieruit wordt gehaald is de latitude en longitude met hun variantie, de status van de GPS en eventueel nog de timestamp waarop de message gepubliceerd is. Uit de *geometry_msgs/TwistStamped*, gepubliceerd door de $\sim\text{vel}$ topic, is de lineaire snelheid van de robot te halen. Snelheidsbepaling door de GNSS module gebeurt op basis van het doppler effect. Voor het volledige formaat van de ROS messages wordt verwezen naar de documentatie van ROS <http://docs.ros.org>.

De informatie, nodig voor de ROS messages die hierboven zijn vermeld, haalt de *reach_rs_driver* node uit de NMEA sentences afkomstig van de GPS. In de GPGGA sentence staat de 3D locatie met hun variantie, nodig om de *nav_msgs/NavSatFix* op te vullen. Uit de GPRMC sentence haalt de node informatie over de snelheid, dewelke wordt gebruikt voor de *geometry_msgs/TwistStamped* [15].

3.3.2 measure_gps_cos

De *measure_gps_cos* node is nodig om de robot te kunnen volgen op de robotTool webapplicatie (figuur 3.9). De node luistert eveneens naar de $\sim\text{fix}$ topic en stuurt de longitude, latitude data over een websocket naar een javascript van de webapplicatie. Op deze manier kan de robot real time gevolgd worden, wat het debuggen sterk vereenvoudigt.

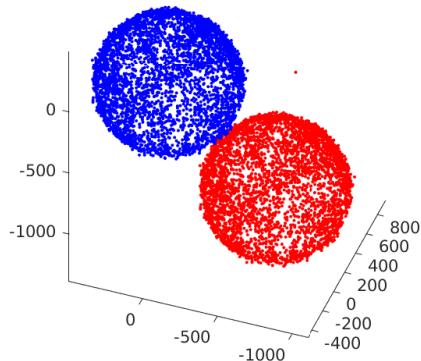
3.3.3 imu_node

Data van de 9DoF Razor IMU M0 op het Sparkfun breakout bordje wordt verwerkt door de *imu_node* [16]. Over deze ROS node is meer informatie terug te vinden op de officiële ROS documentatie. De gepubliceerde topics zijn hieronder opgeliist.

- $\sim imu$: (*sensor_msgs/Imu*)
- $/diagnostics$: Diagnostics about the hardware driver

Vanuit de *sensor_msgs/Imu*, gepubliceerd op de topic $\sim imu$, wordt een quaternion, de hoeksnelheid en de lineaire versnelling verstuurd samen met de overeenkomstige covariantiematrix.

Om een goede werking van de IMU te verzekeren en soft en hard iron errors te vermijden, is een goede kalibratie nodig. Wanneer de sensor niet goed gekalibreerd is kunnen er zich vervelende fenomenen voordoen zoals bijvoorbeeld drift in de yaw. De kalibratie zoals naar verwezen in [16] werd toegepast. In figuur 3.4 is het resultaat te zien van een kalibratie van de magnetometer. Hierbij werd gebruik gemaakt van *Processing 3.5.3*. In het blauw zijn de origineel gemeten magnetometer data weergegeven, in het rood de gecompenseerde data.



Figuur 3.4: Correctie magnetometer data. blauw: originele magnetometer data; rood: gecompenseerde data

3.3.4 imu_driver

De node *imu_driver* ondersteunt de ROS service *Angle.srv*. Het client - server systeem in ROS is een vorm voor communicatie tussen nodes waarin de client op aanvraag data kan verkrijgen van de service. Een andere manier om communicatie tussen ROS nodes te bewerkstelligen is het gebruik van het publisher - subscriber systeem, waarbij een node steeds data op een topic flusht. De *imu_driver* vormt een soort brug tussen de *imu_node* en de hieronder besproken *odometry_source* node. De *imu_driver* node verwerkt de data verkregen van de *imu_node* tot de vorm waarin de

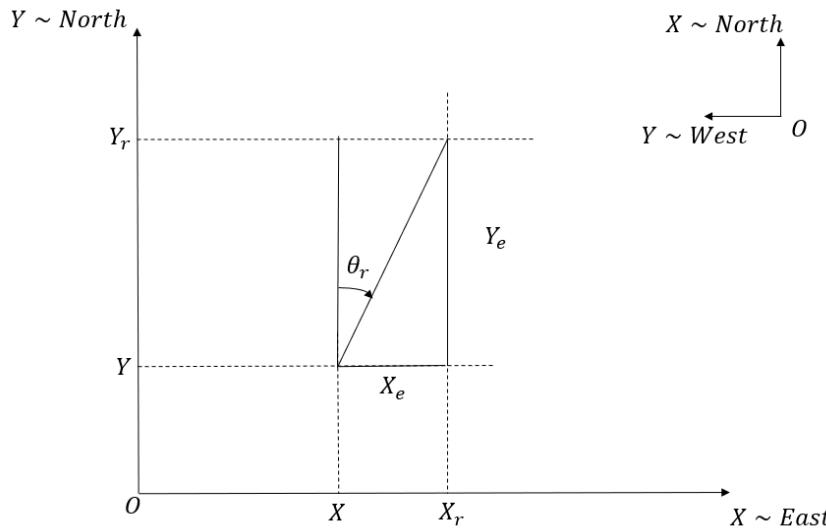
odometry_source node deze nodig heeft en stelt ze ter beschikking in de ROS service *Angle.srv*. Zo is het noodzakelijk voor het algoritme het quaternion in Euler formaat te brengen volgens de onderstaande formule [17].

$$\begin{bmatrix} \phi \\ \psi \\ \theta \end{bmatrix} = \begin{bmatrix} \arctan2(2(q_0q_1 + q_2q_3), 1 - 2(q_1^2 + q_2^2)) \\ \arcsin(2(q_0q_2 + q_3q_1)) \\ \arctan2(2(q_0q_3 + q_1q_2), 1 - 2(q_2^2 + q_3^2)) \end{bmatrix}$$

De yaw θ stelt de oriëntatie van de robot voor ten opzichte van het noorden. Deze is nodig voor het SMC algoritme.

3.3.5 point_creator

De trajectpunten waarop de robot zich oriënteert moeten continu geüpdatet worden. Hiervoor is de node *point_creator* gemaakt. Deze node stelt een service *GetPoint.srv* ter beschikking waaraan de huidige positie en oriëntatie van de robot kunnen meegegeven worden en waaruit het nieuwe trajectpunt wordt bepaald. Het traject van de robot wordt berekend met behulp van de zelf geschreven robotTool, een webtool geprogrammeerd in javascript en python die in staat is op een gebruiksvriendelijke manier robottrajecten te creëren en de robot tijdens de navigatie te volgen. Enkele screenshots van dit programma zijn te zien in figuur 3.9. Voor parcours in andere dan rechthoekige vormen is een script in PyQGIS (*QGIS Desktop 3.4.5*) geschreven. In de node *point_creator* is er speciale aandacht nodig voor het mappen van de (latitude, longitude) - coördinaten naar de (X, Y) - coördinaten. Om de projectie te maken wordt gebruik gemaakt van UTM waarbij België zich in zone 31N bevindt. UTM is in staat de projectie van (latitude, longitude) - coördinaten naar (X: East, Y: North) - coördinaten te doen. Links op figuur 3.5 is af te leiden dat $\theta_r = \text{Bgtan}\left(\frac{X_e}{Y_e}\right)$. Echter is de hoek die we binnen krijgen van de IMU, de yaw, georiënteerd volgens het assenstelsel te zien op figuur 3.5 rechtsboven. Bijgevolg dienen we $\theta_r = -\text{Bgtan}\left(\frac{X_e}{Y_e}\right)$ te kiezen. De hoekfout θ_e is in sectie 2.4.1 gedefinieerd als $\theta_e = \theta_r - \theta$ waarbij θ de werkelijke hoek is waarop de robot staat georiënteerd, bepaald door de IMU. Bijgevolg geldt: $\theta_e = -\text{Bgtan}\left(\frac{X_e}{Y_e}\right) - \theta$.

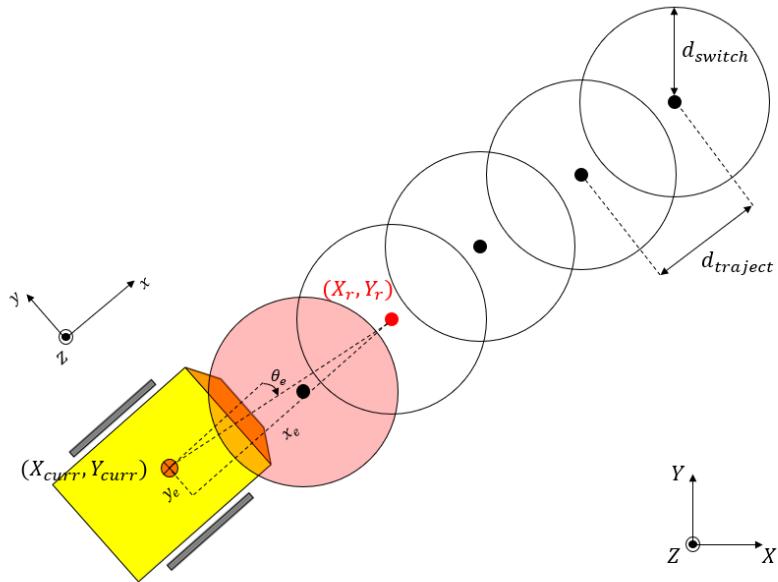


Figuur 3.5: Links: assenstelsel gebruikt bij UTM – rechts assenstelsel gebruikt door IMU

De service *GetPoint.srv* maakt het voor de *odometry_source* mogelijk het trajectpunt op te vragen. Hiervoor maakt de node *point_creator* gebruik van de parameters d_{switch} , $d_{traject}$ en *arrayLength*, die worden gespecificeerd in de *launch file*. d_{switch} en $d_{traject}$ zijn reeds beschreven in sectie 2.4.1, hieronder worden ze nog verder verduidelijkt.

1. $d_{traject}$: de afstand tussen de trajectpunten waar de robot naartoe rijdt. De trajectpunten (X_r, Y_r) staan in een *waypointfile*. Deze file wordt tijdens de uitvoer van het programma deel per deel uitgelezen tot het einde van de file bereikt wordt en de robot wordt gestopt. Hoe dichter de trajectpunten op elkaar liggen, hoe meer ze een lijn benaderen en hoe beter de werking van het algoritme. De waarde van $d_{traject}$ wordt dus bepaald bij het aanmaken van het traject. Bij de creatie van de trajectpunten met PyQGIS, is $d_{traject} \approx 2$ cm. $d_{traject}$ is weergegeven op figuur 3.6.
2. d_{switch} : wanneer de robot zich binnen een straal d_{switch} bevindt ten opzichte van het trajectpunt (X_r, Y_r) , wordt dit punt geüpdate naar het volgende trajectpunt waar de robot naar moet rijden. d_{switch} vindt u eveneens terug in figuur 3.6. d_{switch} heeft een belangrijke impact op de responsiviteit van het algoritme, zoals te zien in figuur 3.6 heeft deze parameter invloed op de grootte van θ_e .
3. *arrayLength*: de responsiviteit van het algoritme wordt grotendeels bepaald door de grootte van de hoekfout. Om dit verband duidelijk te maken wordt er in de node *point_creator* gebruik gemaakt van een vector die opgevuld is met trajectpunten. Het eerste trajectpunt wordt gebruikt om x_e en y_e te berekenen het laatste om θ_e te berekenen. De hoekfout θ_e wordt dus steeds berekend met een trajectpunt verder op het traject dan het punt waarmee x_e en y_e worden berekend. De lengte van de array is een maat voor de uitmiddeling van θ_e en kan ook als parameter worden ingevuld in de *launch file*. Deze parameter samen met d_{switch}

heeft een grote invloed op de responsiviteit van het algoritme. Op figuur 3.6 is hetzelfde trajectpunt gekozen ter bepaling van zowel x_e en y_e als θ_e . De parameter *arrayLength* werd hier dus 1 gekozen.



Figuur 3.6: Illustratie parameters robottraject – (X_{curr}, Y_{curr}) huidige positie van de robot – (X_r, Y_r) trajectpunt ter bepaling van x_e , y_e en θ_e

3.3.6 odometry_source

De *odometry_source* implementeert het SMC algoritme en verwerkt de sensor data met behulp van Kalman filtering. Door Kalman filtering wordt er met een frequentie van 20 Hz data gepubliceerd op de *cmd_vel* topic in plaats van met een frequentie van 10 Hz wat de door de fabrikant vooropgestelde updatefrequentie is van de GPS (deze valt in de praktijk zelfs nog lager uit). Gezien de razor IMU veel sneller data publiceert dan de GPS, worden de IMU data opgevraagd met *Angle.srv* wanneer een Kalman schatting gemaakt dient te worden. Bij elke Kalman schatting wordt de nieuwe snelheid v en de nieuwe hoeksnelheid ω berekend met behulp van het SMC algoritme en gepubliceerd op de *cmd_vel* topic. *GetPoint.srv* van de node *point_creator* maakt het mogelijk het nieuwe trajectpunt op te halen.

We kiezen er hier voor de Kalman filter zoals beschreven in sectie 2.5 te reduceren en de volgende staat te schatten: $x_k = (X_k, v_{X,k}, Y_k, v_{Y,k})^T$. Vergelijking 2.17 en vergelijking 2.20 worden vervolgens:

$$x_k = F_k x_{k-1} + B_k u_k + w_k$$

$$\Leftrightarrow \begin{bmatrix} X_k \\ v_{X,k} \\ Y_k \\ v_{Y,k} \end{bmatrix} = \begin{bmatrix} 1 & T_s & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T_s \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_{k-1} \\ v_{X,k-1} \\ Y_{k-1} \\ v_{Y,k-1} \end{bmatrix} + \begin{bmatrix} \cos(\theta) T_s & 0 \\ \cos(\theta) & 0 \\ \sin(\theta) T_s & 0 \\ \sin(\theta) & 0 \end{bmatrix} \cdot \begin{bmatrix} v_{k,actie} \\ \omega_{k,actie} \end{bmatrix} + w_k$$

$$z_k = H_k x_k + v_k$$

$$\Leftrightarrow \begin{bmatrix} X_{GPS,k} \\ v_{X,GPS,k} \\ Y_{GPS,k} \\ v_{Y,GPS,k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_k \\ v_{X,k} \\ Y_k \\ v_{Y,k} \end{bmatrix} + v_k$$

Voor het algoritme is vooral de positie (X, Y) van belang. θ is bepaald met de IMU en kan ten alle tijden worden opgevraagd. Gezien de nauwkeurigheid ervan is het niet nodig is deze ook te gaan schatten.

3.3.7 drive_controller

De *drive_controller* node luistert naar de topic *cmd_vel* waarop de snelheid v en de hoeksnelheid ω wordt gepubliceerd door de *odometry_source* in een *geometry_msgs/Twist*. De snelheid v en de hoeksnelheid ω worden vervolgens gemapt naar de snelheden v_l en v_r dewelke respectievelijk de snelheid van de linkerwielen en de snelheid van de rechterwielen voorstellen.

Om dit te berekenen is gebruik gemaakt van figuur 3.7. Merk op dat net zoals in figuur 2.2 het Instantaneous Centers of Rotation (ICR) op de y-as van het kleine assenstelsel ligt. Dit is eigen aan een non-holonomic robot waarvan het geometrisch zwaartepunt zich in het massazwaartepunt bevindt [18]. Merken tevens op dat de lineaire snelheidsvector \vec{v} en de hoeksnelheid vector $\vec{\omega}$ onafhankelijk zijn van elkaar. Men kan een willekeurige beweging in een vlak immers steeds opsplitsen in een rotatie en een translatie.

Dan haalt men uit figuur 3.7 dat

$$\omega_z = \frac{v_r}{R+a} = \frac{v_l}{R-a}$$

$$\Leftrightarrow \frac{v_l}{v_r} = \frac{R-a}{R+a} \text{ met } R = \frac{v_x}{\omega_z} \quad (3.1)$$

Uit vergelijking 3.1 vindt men:

$$v_l = \frac{2 \cdot v_x}{\frac{R+a}{R-a} + 1} \text{ met } R = \frac{v_x}{\omega_z}$$

$$v_r = \frac{2 \cdot v_x}{\frac{R-a}{R+a} + 1} \text{ met } R = \frac{v_x}{\omega_z}$$

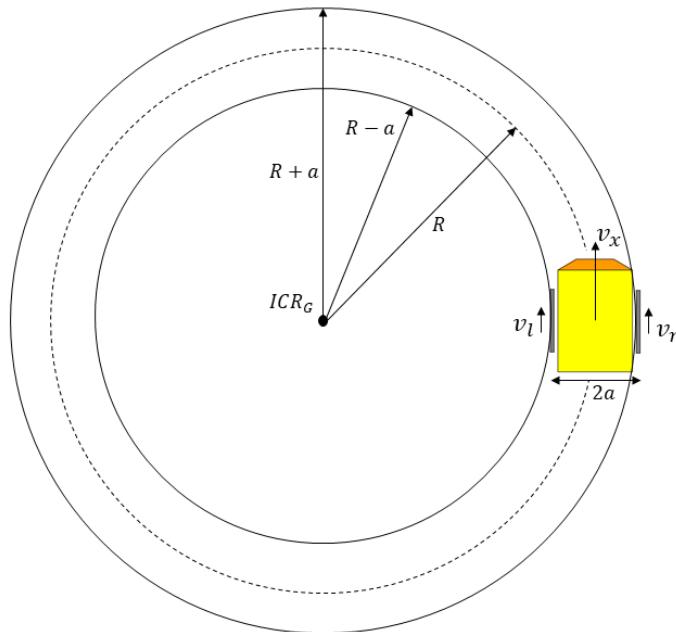
waarbij $\omega_z \circ > 0$ en $\omega_z \circ < 0$

Vervolgens dienen v_l en v_r nog gemapt te worden naar een Digitaal Analoog Convertor (DAC) waarde. Als DAC wordt de LTC1661 gebruikt. Met behulp van een tachometer wordt het verband tussen de binaire DAC waarde en de snelheid van de robot in kaart gebracht. Een tachometer meet de RPM Rotaties Per Minuut (RPM) van het wiel. De diameter van het wiel bedraagt 25 cm, de omtrek is dus 0.25π m. Hieruit kan de snelheid berekend worden:

$$v \left[\frac{m}{s} \right] = \text{omtrek} \left[\frac{m}{360^\circ} \right] \cdot \text{RPM} \left[\frac{360^\circ}{min} \right] \cdot \frac{1}{60} \left[\frac{min}{60s} \right]$$

In tabel 3.1 zijn de metingen weergegeven. Bij elke verhoging van de DAC waarde met 20 werd met de tachometer de RPM opgemeten en de snelheid van het wiel berekend.

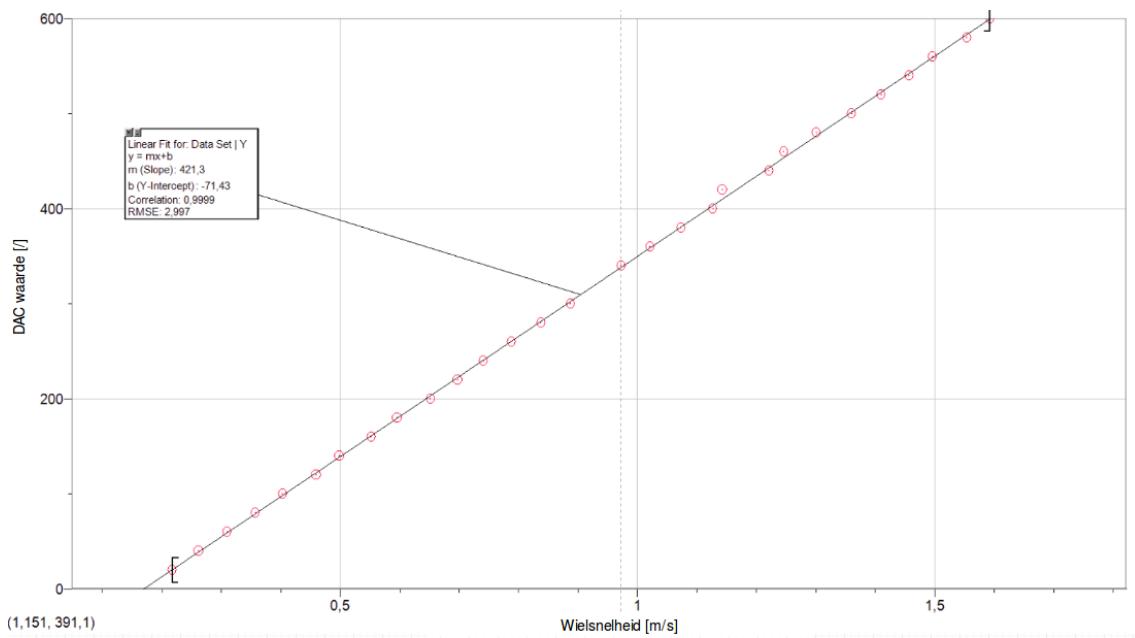
Na het uitzetten van de snelheid $\left[\frac{m}{s} \right]$ in functie van de DAC waarde $[/]$, was een lineaire fit waar te nemen. Deze werd bepaald met behulp van *Logger Pro 3.15 Demo*. De lineaire fit is te zien in figuur 3.8 hierbij zijn $m = 421.3$ en $b = -71.43$ bepaald.



Figuur 3.7: Illustratie ter bepaling van v_l en v_r

DAC waarde	RPM (tachometer)	snelheid $\left[\frac{m}{s} \right]$
20	16.6	0.217
40	20.0	0.262
60	23.7	0.310
80	27.3	0.357
100	30.8	0.403
120	35.1	0.459
140	38.1	0.499
160	42.2	0.552
180	45.5	0.596
200	49.8	0.652
220	53.3	0.698
240	56.6	0.741
260	60.2	0.788
280	64.0	0.838
300	67.8	0.887
340	74.3	0.972
360	78.0	1.021
380	82.0	1.073
400	86.1	1.127
420	87.3	1.142
440	93.3	1.221
460	95.2	1.246
480	99.4	1.301
500	103.9	1.360
520	107.7	1.410
540	111.3	1.457
560	114.3	1.496
580	118.7	1.554
600	121.7	1.593

Tabel 3.1 Meting ter bepaling van het verband tussen de DAC waarde in functie van de snelheid $\left[\frac{m}{s} \right]$



Figuur 3.8: Lineaire fit van de *DACwaarde* [V] in functie van de *Wielsnelheid* [$\frac{m}{s}$] ter bepaling van $m = 421.3$ en $b = -71.43$ ($DACwaarde = m \cdot Wielsnelheid + b$)

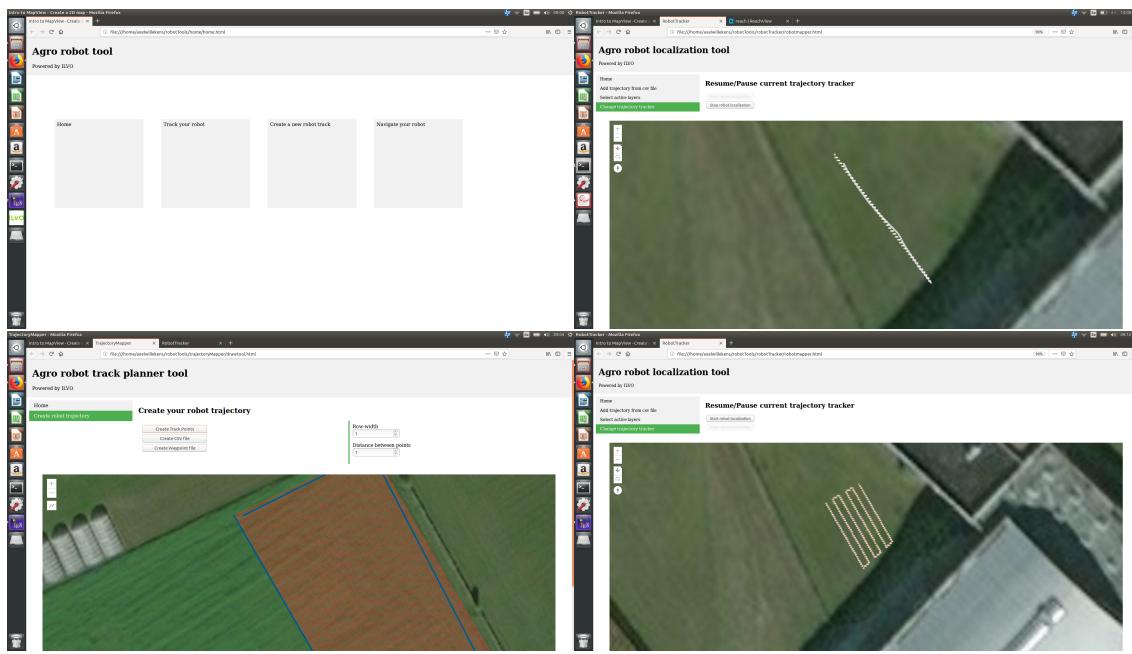
Met behulp van deze fit wordt in de *drive_controller* node de DAC waarde voor zowel het linker als het rechter wiel bepaald. Deze worden vervolgens in een *std_msgs/Int16MultiArray* op de *steering_topic* geplaatst.

3.3.8 motor_driver

De *motor_driver* luistert op de *steering_topic* en zet vervolgens de DAC waarden op de GPIO's van de raspberry pi. Om deze seriële communicatie te verwezenlijken is de Arduino library voor de LTC1661 van Stefan Wallnoefer(Walle86) [19] aangepast naar de hardware van de raspberry pi gebruik makend van de wiringPi library geschreven door Gordon Henderson [20].

3.3.9 keyboard_steering

Node die het mogelijk maakt de robot te sturen indien het algoritme niet actief is. Hierdoor kan de robot via het toetsenbord naar zijn uitgangspositie worden genavigeerd.



Figuur 3.9: RobotTool webapplicatie

Hoofdstuk 4

Evaluatie

4.1 Inleiding

De testen werden uitgevoerd op een oneffen grasveld, dat bovendien in hoogte afloopt. Bijgevolg werden er voldoende onnauwkeurigheden toegevoegd aan het kinematisch model, waardoor deze testen representatief zijn voor een robot in het veld.

Opmerking: De Matlab files, gebruikt in dit hoofdstuk, zijn terug te vinden op de githubpagina <https://github.com/axelwillekens/Point-to-Point-Navigation>. Ook de robotTool webapplicatie en de gebruikte PyQGIS scripts zijn hier terug te vinden.

4.2 Methodiek bepaling fout

4.2.1 Rechtlijnig traject

Om de fout in kaart te brengen wordt bij elke positie die de robot aanneemt de kortste afstand ten opzichte van het traject beschouwd. Voor deze berekening wordt er een rechte r getrokken tussen twee punten van het traject: $P_1(X_1, Y_1)$ en $P_2(X_2, Y_2)$ (figuur 4.1). Het snijpunt van de rechte r_{\perp} en r bepaalt het punt op de rechte r dat het dichtst bij de huidige positie $P_{curr}(X_{curr}, Y_{curr})$ gelegen is. Dit punt noemt hier $P_{close}(X_{close}, Y_{close})$.

De vergelijking van r wordt gegeven door:

$$\begin{aligned} Y - Y_1 &= \frac{Y_2 - Y_1}{X_2 - X_1} \cdot (X - X_1) \\ \text{Noem hierbij } a &= \frac{Y_2 - Y_1}{X_2 - X_1} \\ Y &= a \cdot (X - X_1) + Y_1 \\ Y &= a \cdot X - a \cdot X_1 + Y_1 = a \cdot X + b \quad (4.1) \\ \text{waarbij } a &= \frac{Y_2 - Y_1}{X_2 - X_1} \text{ en } b = -a \cdot X_1 + Y_1 \end{aligned}$$

De vergelijking van r_{\perp} wordt gegeven door:

$$\begin{aligned} Y - Y_{curr} &= -a \cdot (X - X_{curr}) \\ Y &= -a \cdot X + a \cdot X_{curr} + Y_{curr} \end{aligned} \quad (4.2)$$

Combinatie van vergelijking 4.1 en vergelijking 4.2 levert het volgend stelsel:

$$\begin{cases} Y = a \cdot X - a \cdot X_1 + Y_1 \\ Y = -a \cdot X + a \cdot X_{curr} + Y_{curr} \end{cases}$$

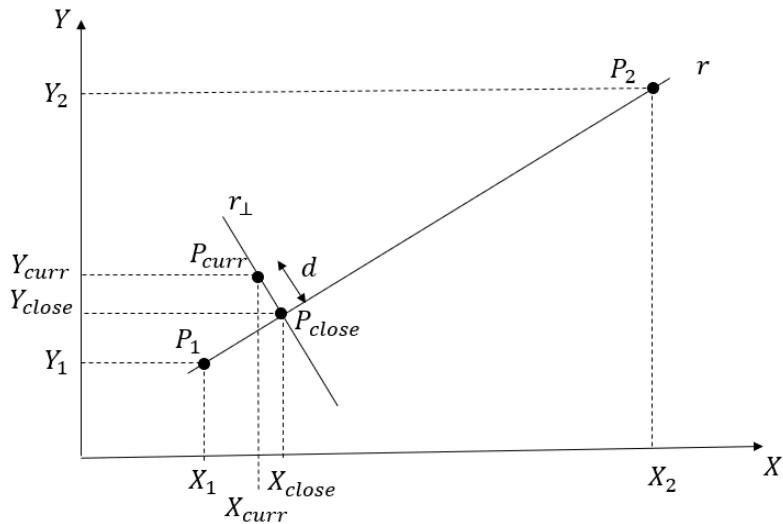
Het bovenstaande stelsel oplossen naar X levert:

$$\begin{aligned} a \cdot X - a \cdot X_1 + Y_1 &= -a \cdot X + a \cdot X_{curr} + Y_{curr} \\ \Leftrightarrow 2 \cdot a \cdot X &= a \cdot (X_{curr} + X_1) + (Y_{curr} - Y_1) \\ \Leftrightarrow X &= \frac{a \cdot (X_{curr} + X_1) + (Y_{curr} - Y_1)}{2 \cdot a} \end{aligned}$$

Dit levert het punt:

$$P_{close} = \left(\frac{a \cdot (X_{curr} + X_1) + (Y_{curr} - Y_1)}{2 \cdot a}, -\frac{a \cdot (X_{curr} + X_1) + (Y_{curr} - Y_1)}{2} + a \cdot X_{curr} + Y_{curr} \right)$$

De fout is bijgevolg $d = \sqrt{(X_{close} - X_{curr})^2 + (Y_{close} - Y_{curr})^2}$



Figuur 4.1: Verduidelijking berekening fout

De Matlab code vindt u terug in listing 4.1. Om te bepalen of de lijn links (positief) of rechts (negatief) van het traject ligt, wordt de conditie $Y_{curr} < a \cdot X_{curr} + b$ gecontroleerd.

```
1 cosfile = 'metingen/line1.9.log';
2 trackfile = 'tracks/track_18mm_line.csv';
3 cosLL = csvread(cosfile);
4 trackLL = csvread(trackfile);

5
6 cosUTM = repmat(cosLL,1);
7 for c=1:size(cosLL,1)
8     [x,y,zone] = ll2utm(cosLL(c,:),31);
9     cosUTM(c,1) = x;
10    cosUTM(c,2) = y;
11 end

12
13 trackUTM = repmat(trackLL,1);
14 for c=1:size(trackLL,1)
15     [x,y,zone] = ll2utm(trackLL(c,:),31);
16     trackUTM(c,1) = x;
17     trackUTM(c,2) = y;
18 end

19
20 x1 = trackUTM(1,1);
21 x2 = trackUTM(10,1);
22 y1 = trackUTM(1,2);
23 y2 = trackUTM(10,2);
24 a = (y2-y1)/(x2-x1);
25 b = -a*x1+y1;

26
27 error = zeros(size(cosUTM,1),1);
28 for c=1:size(cosUTM,1)
29     X_curr = cosUTM(c,1);
30     Y_curr = cosUTM(c,2);
31     x = (a*(X_curr+x1) + (Y_curr - y1))/(2*a);
32     y = a*(X_curr-x) + Y_curr;
33     d = sqrt((x-X_curr)^2 + (y-Y_curr)^2);
34     if Y_curr < a*X_curr+b
35         error(c) = d;
36     else
37         error(c) = -d;
38     end
39 end

40
41 plot(error);
```

```

42 xlabel( '# samples' );
43 ylabel( 'fout [m]' );
44 grid on

```

Listing 4.1: Matlab code ter bepaling van de fout op een rechtlijnig traject

4.2.2 Niet-rechtlijnig traject

Om de fout gemaakt ten opzichte van een niet-rechtlijnig traject te bepalen, wordt iteratief te werk gegaan. De kortste afstand tussen alle trajectpunten en het punt (X_{curr}, Y_{curr}) worden berekend. Het minimum hiervan bepaalt de fout.

De Matlab code vindt u terug in listing 4.2. Hier wordt niet gecontroleerd waar het traject zich ten opzichte van de lijn bevindt, de fout is dus gegeven in absolute waarden.

```

1 cosfile = 'metingen/slinger7.1.log';
2 trackfile = 'tracks/slinger7.csv';
3 cosLL = csvread(cosfile);
4 trackLL = csvread(trackfile);
5
6 cosUTM = repmat(cosLL,1);
7 for c=1:size(cosLL,1)
8     [x,y,zone] = ll2utm(cosLL(c,:),31);
9     cosUTM(c,1) = x;
10    cosUTM(c,2) = y;
11 end
12
13 trackUTM = repmat(trackLL,1);
14 for c=1:size(trackLL,1)
15     [x,y,zone] = ll2utm(trackLL(c,:),31);
16     trackUTM(c,1) = x;
17     trackUTM(c,2) = y;
18 end
19
20 x1 = trackUTM(1,1);
21 x2 = trackUTM(10,1);
22 y1 = trackUTM(1,2);
23 y2 = trackUTM(10,2);
24 a = (y2-y1)/(x2-x1);
25
26 error = zeros(size(cosUTM,1),1);
27 for c=1:size(cosUTM,1)
28     X_curr = cosUTM(c,1);

```

```

29 Y_curr = cosUTM(c,2);
30
31 error(c) = min(sqrt((trackUTM(:,1)-X_curr).^2
32 + (trackUTM(:,2)-Y_curr).^2));
33 end
34
35 plot(error);
36 xlabel('# samples');
37 ylabel('error [m]');
38 grid on

```

Listing 4.2: Matlab code ter bepaling van de fout op een niet-rechtlijnig traject

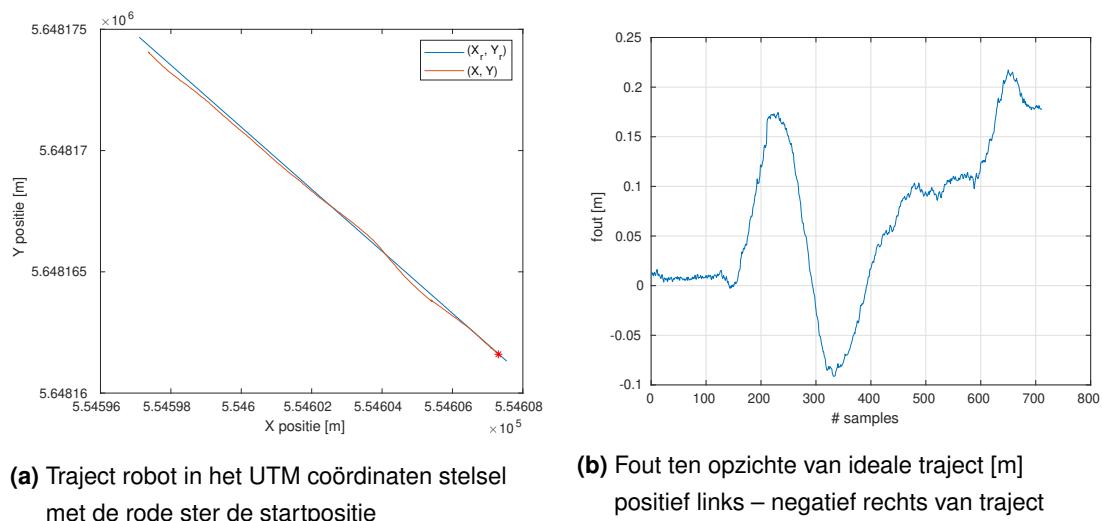
4.3 Evaluatie SMC algoritme

Een eerste evaluatie bestaat erin de fout te bepalen tijdens een rechtlijnig traject. Deze meting is natuurlijk enkel zinvol als de GPS Fixed is. De robot volgt de lijn ook indien de GPS Floaty is, maar gezien er op dat ogenblik een te grote afwijking op de positie is, is het hier niet aangewezen de fout in kaart te brengen. Zoals reeds beschreven in sectie 2.4.1 hangt de grootte van de fout af van onderstaande parameters.

- de parameters k_1 en k_2 . Een juiste keuze van deze parameters is noodzakelijk voor het correcte gedrag van de robot.
- de afstand tussen de trajectpunten $d_{traject}$: de punten waar de robot naar rijdt. Deze afstand kan meegegeven worden bij het creëren van een traject met de robotTool webapplicatie. Echter kan deze afstand op robotTool nog niet klein genoeg worden ingesteld vanwege afrondingsfouten. Daarom is er voor gekozen een traject te maken met PyQGIS, hierbij is een afstand tussen de punten ($d_{traject}$) van minder dan 2 cm mogelijk.
- de afstand d_{switch} . Vanaf het ogenblik dat de robot d_{switch} verwijderd is van het punt waar naartoe gereden wordt, wordt een nieuw trajectpunt aangewezen. Uit de testen werd duidelijk dat de maximale fout gelijk is aan d_{switch} . Rijdt de robot voorbij zijn trajectpunt en komt hij niet in een straal van d_{switch} m rond dit punt, dan komt de robot tot stilstand als gevolg van het algoritme. Echter heeft d_{switch} ook een invloed op de snelheid en het gedrag van de robot.
- updatefrequentie. Door gebruik te maken van Kalman filtering wordt deze constant op 20 Hz gekozen.
- de gewenste lineaire snelheid v_r en de gewenste hoeksnelheid ω_r . ω_r varieert lineair met θ_e door middel van de evenredigheidsconstante k_3 . Dit is een derde parameter die een sterke invloed heeft op de responsiviteit van het systeem.

Zoals in subsectie 3.3.5 uitgelegd, wordt de responsiviteit van het algoritme grotendeels bepaald door de grootte van de hoekfout. Om dit verband duidelijk te maken wordt er in de node *point.creator* gebruik gemaakt van een vector die opgevuld is met trajectpunten. Het eerste trajectpunt wordt gebruikt om x_e en y_e te berekenen het laatste om θ_e te berekenen. De hoekfout θ_e wordt dus steeds berekend met een trajectpunt verder op het traject dan het punt waarmee x_e en y_e worden berekend. De lengte van de array is een maat voor de uitmiddeling van θ_e en kan als parameter worden ingevuld in de launch file. Deze parameter wordt *arrayLength* genaamd en samen met d_{switch} heeft hij een grote invloed op de responsiviteit van het algoritme.

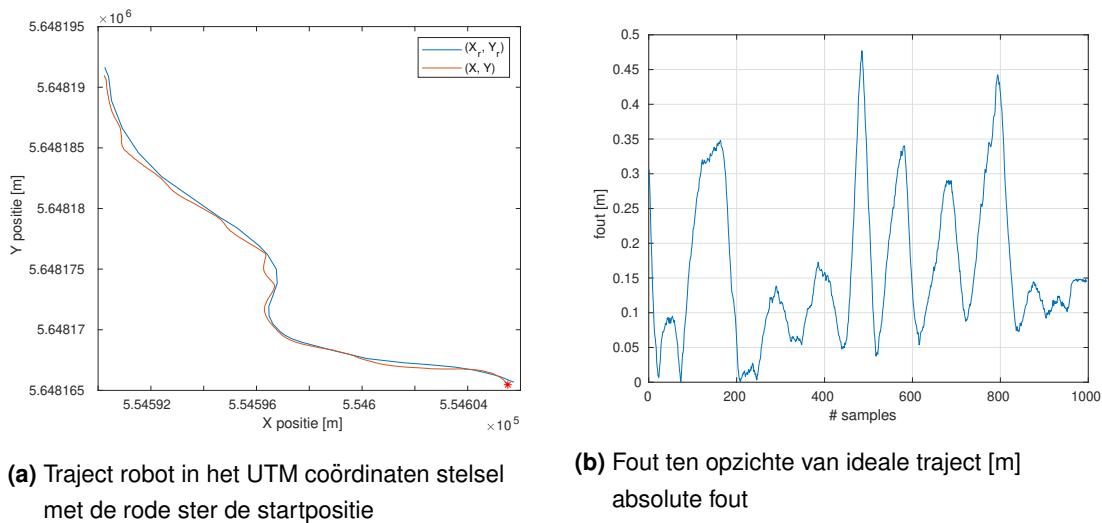
Het resultaat op een rechtlijnig traject is weergegeven in figuur 4.2. Hierbij is de afstand tussen trajectpunten $d_{traject} = 18 \text{ mm}$, $d_{switch} = 50 \text{ cm}$ en $arrayLength = 30$. De (X, Y) posities te zien op figuur 4.2a staan UTM coördinaten.



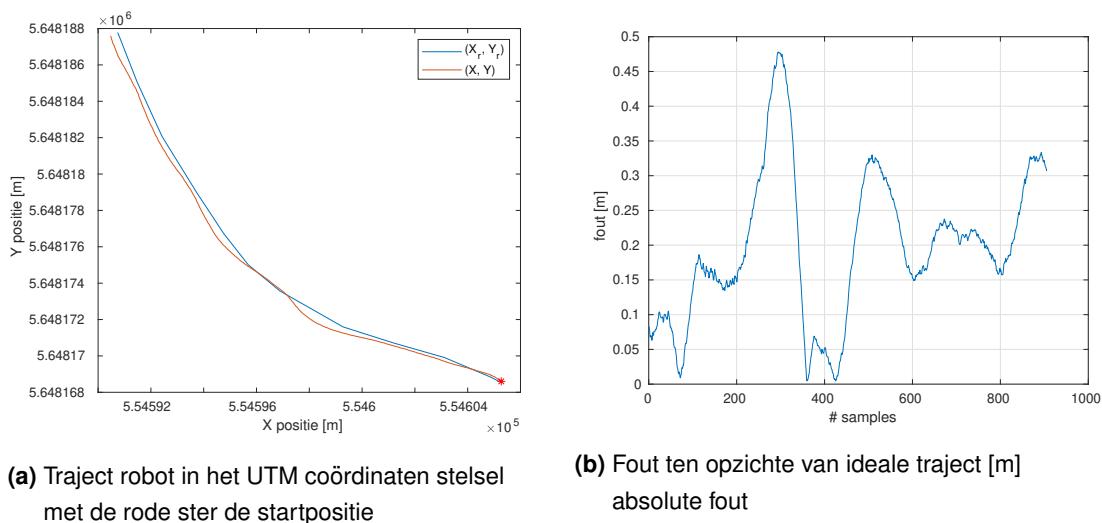
Figuur 4.2: Nauwkeurigheidstest: $d_{traject} = 18 \text{ mm}$, $d_{switch} = 50 \text{ cm}$, $arrayLength = 30$

De beste parameterkeuze gevonden tijdens de testen van deze studie leveren een maximaal gemaakte absolute fout van $\approx 20 \text{ cm}$ op. Dit blijkt uit verschillende testen met verschillende oriëntaties.

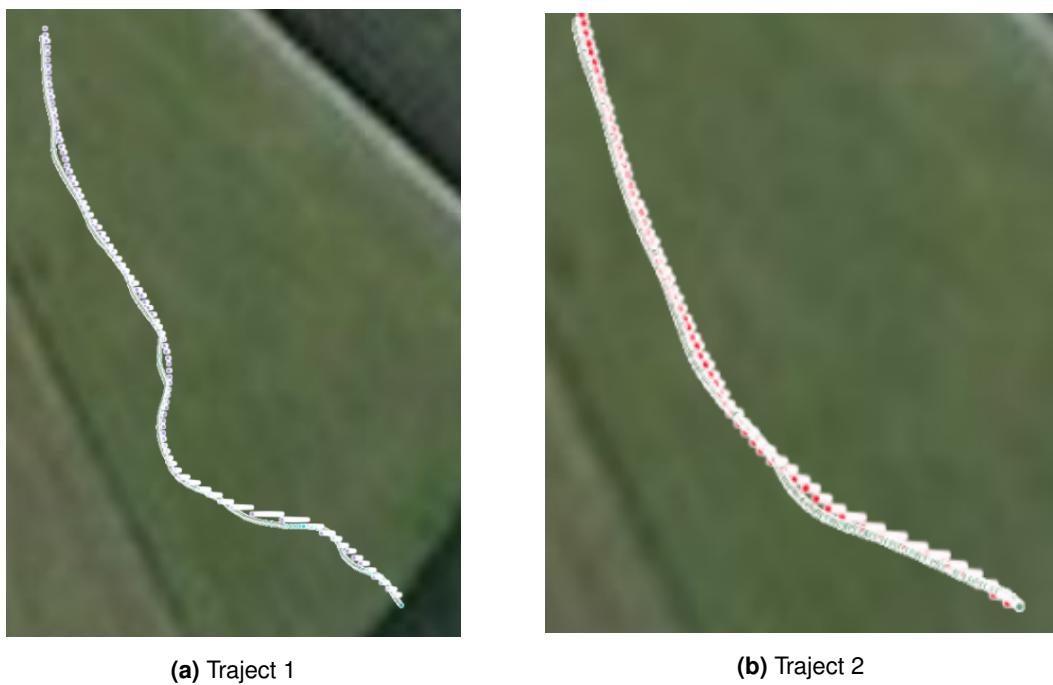
Om de responsiviteit van de robot te testen wordt een niet-rechtlijnig traject afgelegd. Door het wijzigen van de parameters k_1 , k_2 en k_3 is men in staat de responsiviteit te verhogen. Een toename van de responsiviteit heeft wel een afname van nauwkeurigheid tot gevolg. In deze evaluatie zijn twee extra trajecten toegevoegd. In figuur 4.3 en figuur 4.4 is een deel van het afgelegde traject geplot in Matlab en wordt de respectieve fout getoond. In figuur 4.5 is het volledige traject te zien op een screenshot van de robotTool webapplicatie. Zoals te zien op figuur 4.3b en figuur 4.4b, blijft de fout ook op bochtige trajecten binnen de perken. Gezien de implementatie kan men met zekerheid stellen dat wanneer de robot niet stopt de fout kleiner is dan d_{switch} , in dit geval is de gemaakte fout dus < 50 cm. Dit wordt bevestigd door figuur 4.3b en figuur 4.4b.



Figuur 4.3: Responsiviteitstest: Traject1



Figuur 4.4: Responsiviteitstest: Traject2



Figuur 4.5: Responsiviteitstest: printscren robotTool webapplicatie

Hoofdstuk 5

Algemene Conclusie en Future Work

5.1 Algemene Conclusie

In deze masterproef zijn verschillende algoritmen voor autonome point-to-point navigatie onderzocht uitgaande van enkel een GPS en een IMU als sensoren. Het Fuzzy PID trajectory algoritme [5] en het SMC trajectory tracking algoritme [4] zijn in Simulink geïmplementeerd en geëvalueerd. Hieruit bleken beide algoritmen stabiel te zijn. Deze twee algoritmen samen met het Type-2 Fuzzy logic algoritme [6] zouden kunnen dienen voor deze toepassing. Uit de literatuurstudie volgde dat deze algoritmen goed overweg konden met onzekerheden van een niet-lineaire systeem en storingen van buitenaf. Het SMC algoritme oogde erg aantrekkelijk gezien het beperkte aantal te tunen parameters anderzijds is het Fuzzy PID algoritme veel intuïtiever, wel zijn het aantal te tunen parameters hier nog moeilijk te overzien en bovendien was het algoritme minder nauwkeurig dan het SMC algoritme. Vanwege zijn eenvoud en de goede resultaten tijdens simulatie is ervoor gekozen het SMC algoritme te implementeren in ROS op de agrorobot. Dit robotplatform is uitgerust met een RTK-reach GPS en een 9Dof IMU. Elk van de vier wielen wordt aangedreven door een tweopolige borstelloze DC motor bekragtigt door elk zijn motordriver. De motordrivers worden met een DAC aangestuurd. Het navigatie algoritme draait op een Raspberry Pi die zich op de robot bevindt. Het SMC trajectory tracking algoritme is in [4] getest op een RFID sensor mat. In de literatuur zijn er ook projecten te vinden die dit algoritme hebben geïmplementeerd op een tractor. Hierbij wordt soms gebruik gemaakt van andere Lyapunov functies [12].

Als de parameters goed gekozen zijn, haalt de implementatie van het algoritme op rechte lijnen een absolute nauwkeurigheid van ≈ 20 cm. Bij een niet rechtlijnig traject is een andere parameterkeuze noodzakelijk en wordt de onnauwkeurigheid groter, wel nog steeds onder de 50 cm. Ook wordt hier de responsiviteit van de implementatie van dit algoritme duidelijk. Wel is deze responsiviteit niet groot genoeg om richtingsveranderingen van 90° te verwerken. Draaien op de kopakker is met deze implementatie dus niet mogelijk. Dit algoritme is in staat een basis te vormen voor low cost point-to-point navigatie. Een maximale absolute fout van 20 cm is voldoende om zich te verplaatsen tussen gewassen die meestal op 40 tot 50 cm uit elkaar worden geplant. Ook gedraagt deze implementatie zich voorspelbaar in alle mogelijke windrichtingen en het doet zijn werk op oneffen terrein.

5.2 Future Work

Het SMC algoritme kan volstaan om rechte lijnen te volgen in het veld. Verder onderzoek kan nagaan of andere Lyapunov functies de nauwkeurigheid nog verbeteren. Ook kan een andere keuze van kinematisch model mogelijks nog een nauwkeurigheidswinst opleveren.

Er zal een methode moeten bedacht worden om te draaien op de kopakker, gezien het algoritme daar niet toe in staat is.

Bibliografie

- [1] Liangliang Yang Chi Zhang, Noboru Noguchi. Leader-follower system using two robot tractors to improve work efficiency. *Computers and Electronics in Agriculture*, (121):269–281, 2016.
- [2] Claude Lague Chengming Luo, Ahmad Mohsenimanesh. Parallel point-to-point tracking for agricultural wide-span implement carrier (wsic). *Computers and Electronics in Agriculture*, (153):302–312, 2018.
- [3] math24. Method of lyapunov functions. <https://www.math24.net/method-lyapunov-functions/>, 2018. geraadpleegd op 21/10/2018.
- [4] Hoon Lim Jang Myung Lee Jun Ho Lee, Cong Lin. Sliding mode control for trajectory tracking of mobile robot in rfid sensor space. *International Journal of Control, Automation, and Systems*, 7(3):429–435, 2009.
- [5] Shanan Chen Shengqi Yan Qing Xu, Jiangmin Kan. Fuzzy pid based trajectory tracking control of mobile robot and its simulation in simulink. *International Journal of Control and Automation*, 7(8):233–244, 2014.
- [6] Mojtaba Ahmadieh Khansar Erdal Kayacan. *Fuzzy Neural Networks For Real Time Control Applications*. Joe Hayton, 2016.
- [7] Robert L. Bryant. *150 years of mathematics at Washington University in St. Louis*, volume 395. Providence, RI: American Mathematical Society, 2006. Geometry of manifolds with special holonomy: '100 years of holonomy'.
- [8] wiki. Kalman filter. https://en.wikipedia.org/wiki/Kalman_filter, 2018. geraadpleegd op 20/11/2018.
- [9] E. Kayacan, E. Kayacan, H. Ramon, O. Kaynak, and W. Saeys. Towards agrobots: Trajectory control of an autonomous tractor using type-2 fuzzy logic controllers. *IEEE/ASME Transactions on Mechatronics*, 20(1):287–298, Feb 2015.
- [10] Jianhong Liang Chenhao Han Jiao Chen Tianmiao Wang, Yao Wu and Qiteng Zhao. Analysis and experimental kinematics of a skid-steering wheeled robot based on a laser scanner sensor. *Sensors*, (15):9681–9702, 2015.

- [11] M. Salehi and G. Vossoughi. Impedance control of flexible base mobile manipulator using singular perturbation method and sliding mode control law. *International Journal of Control, Automation, and Systems*, 6(3):677–688, 2008.
- [12] B. Thuilot P. Martinet H. Fang, Ruixia Fan. Trajectory tracking control of farm vehicles in presence of sliding. *Robotics and Autonomous Systems*, 54(10):828–839, March 2006.
- [13] Wikipedia. Universele transversale mercatorprojectie. https://nl.wikipedia.org/wiki/Universele_transversale_mercatorprojectie, 2019. geraadpleegd op 8/05/2019.
- [14] enwaytech. reach_rs_ros_driver. https://github.com/enwaytech/reach_rs_ros_driver, Jun 29, 2018. geraadpleegd op 03/03/2019.
- [15] Dale DePriest. Nmea data. <https://www.gpsinformation.org/dale/nmea.htm>, 2015. geraadpleegd op 10/03/2019.
- [16] KristofRobot. razor_imu_9dof. https://github.com/KristofRobot/razor_imu_9dof, Jan 20, 2018. geraadpleegd op 03/03/2019.
- [17] NASA. Euler angles, quaternions and transformation matrices. <https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19770024290.pdf>, 1997. geraadpleegd op 13/04/2019.
- [18] J. Morales S. Pedraza A. Garcia-Cerezo J. L. Martinez, A. Mandow. Approximating kinematics for tracked mobile robots. *The International Journal of Robotics Research*, 24(10):867–878, 2005.
- [19] Stefan Wallnoefer. Ltc1661. <https://github.com/walle86/LTC1661>, 2014. geraadpleegd op 10/03/2019.
- [20] Gordon Henderson. Wiring pi. <http://wiringpi.com/>, 2018. geraadpleegd op 10/03/2019.

Bijlage A

Lyapunov functies

Opm: Deze samenvatting is gebaseerd op [3].

A.1 Definitie

Een functie $V(X)$ die continu afleidbaar is in de omgeving U die de oorsprong omvat. Dan is de functie $V(X)$ een Lyapunov functie voor een autonoom systeem $X' = f(X)$ als en slechts als de volgende voorwaarden zijn voldaan:

1. $V(X) > 0 \quad \forall X \in U \setminus \{0\}$
2. $V(0) = 0$
3. $\frac{dV}{dt} \leq 0 \quad \forall X \in U.$

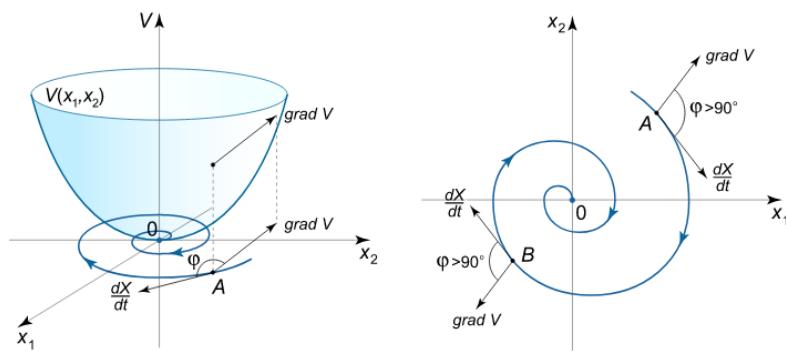
A.2 Voorbeeld

Beschouw een autonoom systeem $X' = f(X)$ of $\begin{cases} \frac{dx_1}{dt} = f_1(x_1, x_2) \\ \frac{dx_2}{dt} = f_2(x_1, x_2) \end{cases}$ met een evenwichtspunt in de oorsprong $X \equiv 0$ of dus in ons vereenvoudigd voorbeeld een evenwichtspunt bij $\begin{cases} x_1 = 0 \\ x_2 = 0 \end{cases}$.

Dan zijn $V(x_1, x_2) = a \cdot x_1^2 + b \cdot x_2^2$ en $V(x_1, x_2) = a \cdot x_1^4 + b \cdot x_2^4$, met $a, b > 0$ zijn mogelijks Lyapunov functies. De eerste en tweede voorwaarde uit de definitie zijn voldaan, maar de derde voorwaarde moet nog nagegaan worden. Leiden men $V(X)$ af naar de tijd, dan krijgt men

$$\frac{dV}{dt} = \frac{\partial V}{\partial x_1} \frac{dx_1}{dt} + \frac{\partial V}{\partial x_2} \frac{dx_2}{dt} = \left(\frac{\partial V}{\partial x_1} \cdot \vec{1}_{x_1} + \frac{\partial V}{\partial x_2} \cdot \vec{1}_{x_2} \right) \cdot \left(\frac{dx_1}{dt} \cdot \vec{1}_{x_1} + \frac{dx_2}{dt} \cdot \vec{1}_{x_2} \right) = \text{grad } V \cdot \frac{dX}{dt}$$

Als $\frac{dV}{dt} \leq 0$, dan is het systeem stabiel, anders is het systeem onstabiel. Dit is eenvoudig te begrijpen uit figuur A.1. Indien $\frac{dV}{dt} > 0$ zou de pijl immers de andere kant uitwijzen en zou de oorsprong (het evenwichtspunt) nooit bereikt worden.



Figuur A.1: Verloop $\frac{dV}{dt}$ voor $V(x_1, x_2) = a \cdot x_1^2 + b \cdot x_2^2$ waarbij $\frac{dV}{dt} < 0$

A.3 Gebruikte stabiliteitscriterium en theorema

- 1. Stabiliteitscriterium met betrekking tot Lyapunov:** Als bij een autonoom systeem, in de omgeving U de nulplossing $X = 0$ geldt en er is een Lyapunov functie $V(X)$, dan is het evenwichtspunt $X = 0$ Lyapunov stabiel.
- 2. Assymptotisch stabiliteitstheorema:** Als bij een autonoom systeem, in de omgeving U de nulplossing $X = 0$ geldt en er is een Lyapunov functie $V(X)$ met een negatieve afgeleide $\frac{dV}{dt} < 0 \quad \forall X \in U \setminus \{0\}$, dan is het evenwichtspunt $X = 0$ assymptotisch stabiel.

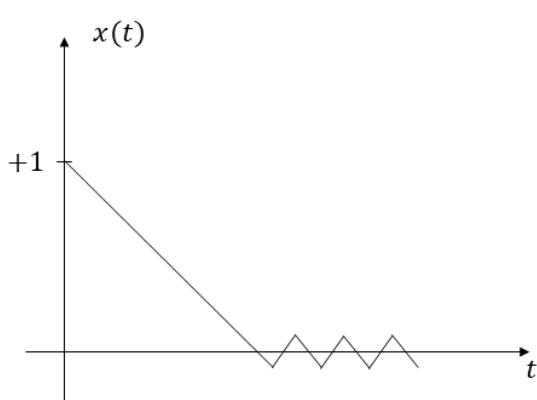
Bijlage B

Sliding Mode Control (SMC)

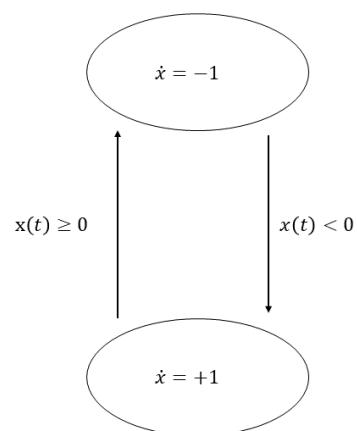
Opm: Deze samenvatting is deels gebaseerd op [6].

B.1 Eenvoudig voorbeeld

Om SMC als controle algoritme te duiden wordt een eenvoudig voorbeeld gegeven. Beschouw een systeem dat zich slechts in twee staten kan bevinden zoals te zien in figuur B.2. In de eerste staat is de afgeleide $\frac{dx(t)}{dt} = \dot{x} = -1$, in de tweede staat geldt $\dot{x} = +1$. Start het systeem op tijdstip $t = 0$ op $x = 1$ en bevindt het zich in de eerste staat, dan neemt x bij toename van t evenveel af. Dit is te zien in figuur B.1. Wanneer $x < 0$ gaat het systeem over naar de tweede staat waarbij $\dot{x} = +1$. Bij verdere toename van t neemt x dus met eenzelfde hoeveelheid toe. Tot $x \geq 0$, dan gaat het systeem terug over naar de eerste staat en bevindt men zich terug in de eerste situatie. Het is duidelijk dat het systeem uiteindelijk zal schommelen rond $x = 0$ wat hier de sliding mode is. De switching mode functie is hier $x(t)$. Merk op dat de afgeleide van de switching mode functie steeds het gedrag in de verschillende staten bepaalt.



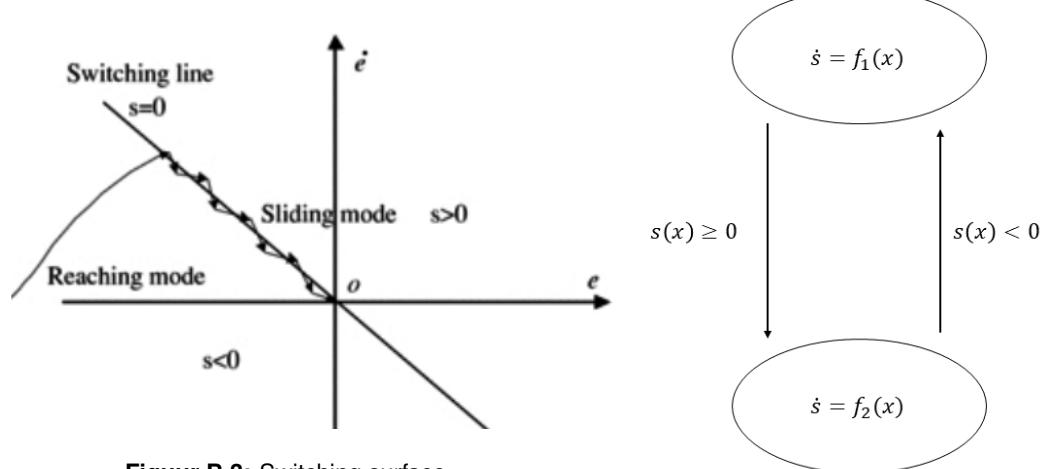
Figuur B.1: vb: Switching surface



Figuur B.2: vb: Staten diagram

B.2 Algemeen

Het eenvoudige voorbeeld van hierboven kan worden uitbreid tot de algemene definitie van SMC. Boven de switching line, te zien in figuur B.4, implementeert het systeem de functie $f_1(x)$ en onder de switching line $f_2(x)$. Beide zijn de afgeleiden van de switching functie s .



Figuur B.3: Switching surface

Figuur B.4: Statendiagram

FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
CAMPUS DE NAYER SINT-KATELIJNE-WAVER
J. De Nayerlaan 5
2860 SINT-KATELIJNE-WAVER, België
tel. + 32 15 31 69 44
iw.denayer@kuleuven.be
www.iw.kuleuven.be

