Faculteit Industriële Ingenieurswetenschappen

master in de industriële wetenschappen: elektromechanica

Masterthesis

Optimaliseren van nauwkeurigheid van een robot-airhockeytafel met machine learning via ensemble trees en neurale netwerken

Ian Cappuyns

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektromechanica

PROMOTOR :
ing. Geert LEEN

PROMOTOR :
Dhr. Kim ROBBENS

Gezamenlijke opleiding UHasselt en KU Leuven

Universiteit Hasselt | Campus Diepenbeek | Faculteit Industriële Ingenieurswetenschappen | Agoralaan Gebouw H - Gebouw B | BE 3590 Diepenbeek

Universiteit Hasselt | Campus Diepenbeek | Agoralaan Gebouw D | BE 3590 Diepenbeek
Universiteit Hasselt | Campus Hasselt | Martelarenlaan 42 | BE 3500 Hasselt

2022
2023

Faculteit Industriële Ingenieurswetenschappen

master in de industriële wetenschappen: elektromechanica

Masterthesis

Optimaliseren van nauwkeurigheid van een robot-airhockeytafel met machine learning via ensemble trees en neurale netwerken

Ian Cappuyns

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektromechanica

PROMOTOR :

ing. Geert LEEN

PROMOTOR :

Dhr. Kim ROBBENS

UHASSELT KU LEUVEN

Woord vooraf

Voor het behalen van het diploma industriële ingenieurswetenschappen bij de gezamenlijke opleiding van de UHasselt en KU Leuven krijgt elke masterstudent de mogelijkheid om zijn opgedane kennis te bewijzen via een masterproef.

Na het behalen van mijn professionele bachelor in juni 2019 ben ik bij Beckhoff Automation BV deeltijds in dienst getreden. Ik kreeg er de mogelijkheid om 70% van een fulltime contract te werken. Dit maakte het voor mij mogelijk om mijn schakel- en masterjaar te voltooien over een periode van vier jaar in plaats van de gebruikelijke twee jaar.

Beckhoff Automation BV heeft mij voorgesteld om als masterproef hun robot-airhockeytafel in combinatie met machine learning te optimaliseren. Dit was voor mij een mooie opportuniteit om zo mijn kennis rond visie, machine learning en meer technologieën met een Beckhoff PLC te verbreden. Tijdens de uitvoering van mijn dagdagelijkse taken als werknemer kom ikzelf niet direct in aanraking met deze technologieën. Dit terwijl machine learning wel een topic is wat mij erg interesseert.

Dit traject had ik vanzelfsprekend niet alleen kunnen afleggen. Graag zou ik dan ook volgende personen in het bijzonder willen bedanken:

Patrick Gielis, general manager van Beckhoff Automation BV wie mij de opportuniteit heeft gegeven om mijn studies en deze masterproef uit te voeren.

Kim Robbens, externe bedrijfspromotor en product specialist automation Beckhoff Automation BV. Hij heeft mij doorheen het jaar begeleid om de masterproef in goede banen te leiden en was mijn eerste aanspreekpunt bij vragen.

Bram Seys, HMI-specialist Beckhoff Automation BV wie mij met het kalibreren van de camera's en in het bijzonder met machine learning heeft bijgestaan met zijn kennis.

ing. Julia Otero Jiménez, vision specialist Beckhoff Automation BV wie mij de kennis van TwinCAT 3 Vision heeft bijgebracht.

Philip Neyens, building automation specialist Beckhoff Automation BV wie mij heeft bijgestaan met hardwarematige vragen rond de airhockeytafel.

ing. Koen Kerkhofs, motion specialist Beckhoff Automation BV wie mij geholpen heeft om de H-robot aan te sturen.

ing. Geert Leen, interne begeleider wie mijn masterproef mee opgevolgd heeft en mij geholpen heeft met het nakijken van mijn scriptie.

prof. dr. ing. Jeroen Lievens, docent MP-seminarie wie mij heeft bijgestaan in het academisch opstellen en schrijven van de scriptie.

Mijn collega's, docenten, ouders en vrienden om mij tijdens dit hele project te steunen en steeds rekening te houden met mijn overvolle agenda door de combinatie werk/studeren.

Bedankt allemaal!

Juni 2023

Ian Cappuyns

Inhoudsopgave

Lijst van tabellen

Lijst van figuren

Verklarende woordenlijst

Begrip	Verklaring
AI	Artificiële Intelligentie
ANN	Artificiële Neurale Netwerken
BML	Binaire representatie van een XML-bestand voor machine learning.
Core	Onderdeel van de CPU of GPU welke berekeningen uitvoert.
CPU	Central Processing Unit
CSV	Comma-Separated Values
CUDA	Compute Unified Device Architecture
CV	Cross Validation
Estimator	Hyperparameter voor ensemble trees die aangeeft uit hoeveel decision trees het model is opgebouwd.
EtherCAT	Ethernet for Control Automation Technology
Feature	Een eigenschap van de trainingsdataset.
FOV	Field Of View
FPS	Frames Per Second
GPU	Graphical Processing Unit
HMI	Human Machine Interface
Hyperparameter	Parameter om het model bij te sturen in het trainings- en evaluatieproces.
IoT	Internet of Things
LightGBM	Light Gradient-Boosting Machine
MAE	Mean Average Error
ML	Machine Learning
MLP	Multilayer perceptron
MSE	Mean Squared Error
NCI	Numerical Control Interpolation
Onderfitting	Het ML-model vertoont grote fouten op de trainingsdata en op de validatiedata.
ONNX	Open Neural Network Exchange
Overfitting	Het ML-model focust zich op de trainingsdata en heeft grote fouten op de validatiedata.
PLC	Programmable Logic Controller
PoE	Power over Ethernet
R2-score	Evaluatiemetriek die weergeeft hoe goed de originele waarden de getrainde regressielijn omvatten.
RAM	Random Access Memory
Regularisatieparameter	Hyperparameter die de complexiteit van het model bestraft.
RMSE	Root Mean Squared Error
SMA	Simple Moving Average
SVC	Support Vector Classification
SVM	Support Vector Machine
SVR	Support Vector Regression
TwinCAT 3	The Windows Control and Automation Technology
VRAM	Video Random Access Memory
XGBoost	eXtreme Gradient Boosting
XML	eXtensible Markup Language

Abstract

Beckhoff Automation BV te Lummen is een verdeler van automatiseringsoplossingen voor elke industrie. Ter illustratie van haar diverse producten werd er een robot-airhockeytafel ontwikkeld. De robot bepaalt wiskundig het traject van de puck met behulp van visie en speelt de puck terug. Het doel van deze masterproef is het optimaliseren van de nauwkeurigheid van dit traject door middel van machine learning.

Bij deze masterproef werden de eisen opgesplitst in twee delen. Vooraleer er aan het machine learning model gewerkt kan worden moeten er eerst data verzameld worden over het traject van de puck aan de hand van camera's. Hierbij moet de locatie van de puck bepaald worden door de combinatie van twee camera's i.p.v. één. Als volgt moet er onderzocht worden hoe de trajectbepaling en de aanvalstechnieken van de puck geoptimaliseerd kunnen worden via machine learning.

Eerst werden de camera's softwarematig gecombineerd in TwinCAT 3. De PLC logt het traject van de puck in CSV-bestanden die kunnen worden gebruikt voor de training van het machine learning model. Python scripts gebruiken deze logs om de modellen te trainen en het traject te voorspellen. Ten slotte werd het beste model in de TwinCAT 3 real-time omgeving geïmplementeerd om de robot aan te sturen. Uit deze masterproef is gebleken dat het best presterende model een neuraal netwerk was. Deze kan de puck op 50 frames in de toekomst voorspellen met een gemiddelde fout kleiner dan 3 cm.

Abstract in English

Beckhoff Automation BV in Lummen is a distributor of automation solutions for every industry. To illustrate its various products, a robotic air hockey table was developed. The robot mathematically determines the trajectory of the puck using vision and plays the puck back. The aim of this master's thesis is to optimise the accuracy of this trajectory using machine learning.

In this master's thesis, the requirements were divided into two parts. Before working on the machine learning model, data must first be collected on the trajectory of the puck using cameras. Here, the location of the puck must be determined by combining two cameras instead of one. Next, research needs to be done how the puck's trajectory and attack techniques can be optimised via machine learning.

First, the cameras were combined software-wise in TwinCAT 3. The PLC logs the trajectory of the puck into CSV files that can be used for training the machine learning model. Python scripts use these logs to train the models and predict the trajectory. Finally, the best model was implemented in the TwinCAT 3 real-time environment to control the robot. This master's thesis showed that the best-performing model was a neural network. It could predict the puck at 50 frames in the future with an average error smaller than 3 cm.

1 Inleiding

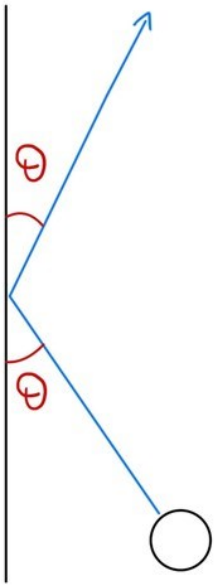
1.1 Situering

Beckhoff Automation is een internationaal bedrijf dat automatiseringsoplossingen biedt voor bijna elke sector. In het hoofdkantoor, gelegen in Duitsland, worden de producten ontworpen en geproduceerd. Beckhoff Automation BV is een dochterbedrijf dat voor de lokale sales en support zorgt voor de regio's België en Luxemburg. In België heeft Beckhoff Automation twee vestigingen: de hoofdvestiging in Lummen, waar ook de masterproef wordt uitgevoerd, en een sales/support-vestiging in Zwevegem.

Om de automatiseringstechnologieën van Beckhoff aan de klant te tonen is er enkele jaren geleden een robot-airhockeytafel ontwikkeld in samenwerking met studenten. Onderstaande figuur 1 toont de airhockeytafel tijdens de Indumation-beurs te Kortrijk in 2019.

Figuur 1: Airhockeytafel op Indumation 2019 [1]

De bedoeling van deze airhockeytafel is dat de speler met een puck via een pusher speelt naar de overkant met de intentie om te scoren. Aan de overkant wordt de pusher van de robot bestuurd door een H-robot die zich verdedigt. Boven de tafel staat een camera opgesteld om de positie van de puck te bepalen. Op dit ogenblik wordt de aankomende positie van de puck bepaald door twee parameters die de camera waarneemt, namelijk de snelheid en de aankomende hoek van de puck (Figuur 2).



Figuur 2: Puck trajectopname 2019

De H-robot met de pusher wordt dan aangestuurd naar de berekende positie om de puck weg te duwen. Ondertussen is de installatie hardwarematig uitgebreid met een extra camera en aangepaste belichting. De camera's hangen nog niet op dezelfde hoogte en zijn nog niet correct gepositioneerd, hierdoor is de mechanische opstelling van de tafel niet volledig in orde. Figuur 3 toont de opstelling van de airhockeytafel voor de aanvang van de masterproef. Door tijdsgebrek is er nog geen software geschreven om de machine met de nieuwe opstelling te laten werken.

Figuur 3: Airhockeytafel aanvang masterproef 2022 [2]

1.2 Probleemstelling

Het huidige probleem is dat de berekende positie van de puck niet accuraat genoeg is. De positie van de puck wordt in de voorgaande opstelling op basis van slechts twee parameters berekend, namelijk de hoek en de snelheid. Dit terwijl er ook nog andere parameters een effect hebben op de baan van de puck, zoals bijv. de wrijving op de baan, de versnelling, de hoeksnelheid, de hoekversnelling, enz. van de puck. Als de puck met een andere hoeksnelheid tegen de rand van de tafel komt, kaatst deze immers onder een andere hoek weg. Hierdoor komt de pusher van de H-robot soms te laat en/of op de verkeerde positie, waardoor er gescoord wordt in het doel van de robot. Deze extra parameters werden in de vorige iteratie van de airhockeytafel niet uitgevoerd omdat ze tot een complexe wiskundige formule zouden leiden. Deze formule bevat immers verschillende onbekenden zoals de wrijvingsfactor van de puck op het veld, de luchtwrijving, het effect van de hoeksnelheid, … die moeilijk te bepalen zijn en niet eenvoudig in een formule om te zetten zijn.

Een bijkomend probleem van de voorgaande opstelling is dat de robot enkel probeert te verdedigen. Hij valt niet actief aan om punten te scoren. De robot zal dus zelf geen punten scoren tenzij het een toevalstreffer zou zijn.

1.3 Doelstellingen

Het doel van de masterproef is om de airhockeytafel sneller, accurater en aanvallend te maken. Als de robot de pucks van de tegenstander voor 90 % kan stoppen en actief kan scoren in het doel van de tegenstander kan van een succesvol project gesproken worden.

In plaats van een complexe wiskundige formule samen te stellen om het pad van de airhockey puck te bepalen moet er een onderzoek gedaan worden om de robot te verbeteren door middel van machine learning. Hiermee kan de complexe formule vervangen worden door een dataset waaruit de machine leert om het pad van de puck te bepalen.

Om een breder beeld te krijgen over het airhockeyspeelveld wordt er in de huidige iteratie gebruik gemaakt van twee camera's. De FOV (field-of-view) van deze twee camera's moet softwarematig gecombineerd worden om de puck over meer dan 95 % van het speelveld te kunnen volgen.

Deze doelen van de robot zouden gehaald moeten worden tegen het evenement in september. Zo kunnen met deze demo de mogelijkheden van Beckhoff getoond worden. Dit op vlak van machine learning, visie en motion waarbij deze technologieën draaien op een Beckhoff controller.

1.4 Methode

Om de machinelearningdata te verzamelen moet de robot-airhockeytafel in eerste instantie in orde gebracht worden. Er worden nieuwe houders voor de camera's ontwikkeld zodat ze op een gelijke manier gemonteerd kunnen worden op het frame. In een volgende stap worden de camera's gekalibreerd. Na het toepassen van visiealgoritmes kunnen de positie van de puck en andere factoren bepaald worden. De parameters worden gelogd om ze later te gebruiken bij het machinelearningmodel.

Na de kalibratie is er een onderzoek nodig naar welke parameters een invloed hebben op het traject van de puck en welke verschillende machinelearningalgoritmes geschikt zijn.

Aan de hand van de gelogde datasets moet de robot getraind worden met de gekozen machinelearningalgoritmes. Zo kan de robot op een nauwkeurige manier de positie van de puck bepalen gebaseerd op voorgaande getrainde resultaten. Als eenmaal het correcte traject van de puck bepaald is moet een ander machinelearningmodel de beste aanvalstactiek aan de robot meegeven.

Als de geprefereerde methode(s) via de bronnenstudie gevonden zijn worden deze getraind met behulp van de programmeertaal Python. Deze trainingsmodellen kunnen dan geïntegreerd worden in TwinCAT 3, de real-time omgeving van de Beckhoff-PLC. In de PLC (Programmable Logic Controller) moet daarna het model geïmplementeerd worden. Hiermee wordt de positie van de puck, verkregen door de camera's, in het model doorlopen en krijgt de H-robot een signaal naar waar hij zich moet bewegen.

Na het uitwerken van het resultaat wordt er gekeken hoe de robot reageert. Indien de robot nog afwijkt van zijn optimale positie moet het trainingsmodel of de dataset aangepast worden. Zo worden er nieuwe iteraties gemaakt via machine learning om de optimale baan te vinden voor de pusher van de robot.

2 Literatuurstudie

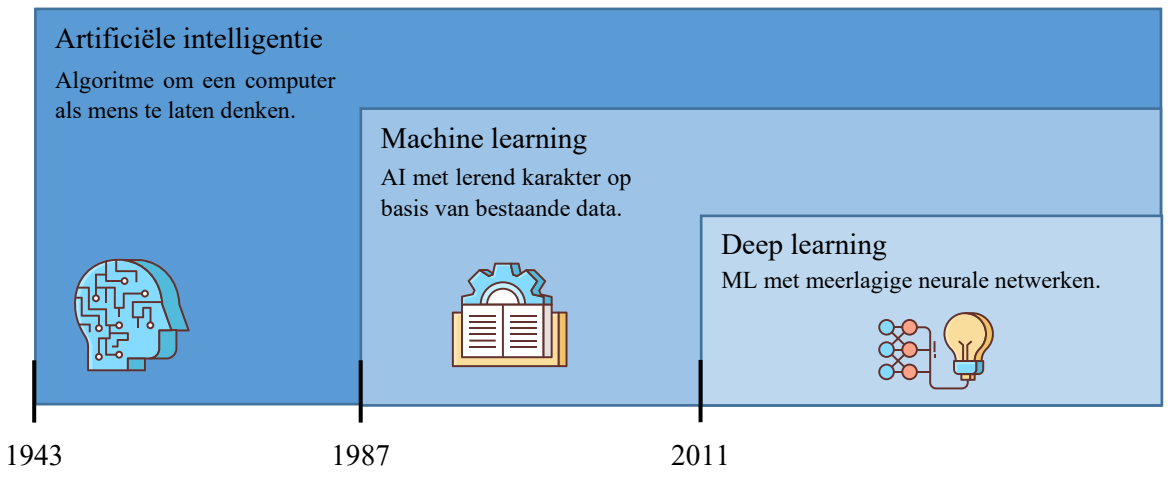
Vooraleer er met de ML-implementatie gestart kon worden is er een literatuurstudie uitgevoerd over de verschillende ML-topics. In dit hoofdstuk is de studie opgesplitst in drie paragrafen. Bij het eerste deel wordt er een introductie gegeven aan het concept ML. In het tweede deel is er een overzicht gemaakt van verschillende ML-modellen. De werking van elk onderzocht model wordt hierbij verduidelijkt. En als laatste komen er verschillende onderzoeken van andere airhockeytafels aan bod met een gelijkaardige probleemstelling. Hier wordt er een vergelijking gemaakt met deze masterproef alsook met de concepten die in dit onderzoek gebruikt zijn.

2.1 Machine learning introductie

ML is een begrip wat vaak in dezelfde context gebruikt wordt als artificiële intelligentie (AI) volgens [3] terwijl deze niet exact hetzelfde zijn. AI kan beschreven worden als een algoritme wat geprogrammeerd is om een computer te laten gedragen als een mens. Om een beter onderscheid te maken tussen de verschillende soorten AI worden deze onderstaand even verduidelijkt:

- De klassieke AI, ook wel *Good Old-Fashioned AI* of symbolische AI genoemd, zoals beschreven in [4]. Hier worden AI-algoritmes met verschillende eindige mogelijkheden of regels opgesteld om een probleem op te lossen. Het algoritme wordt berekend met behulp van verschillende IF en THEN-statements om zo het gewenste doel te bereiken. Via de klassieke AI kan men bijvoorbeeld aan een computer schaakregels meegeven om te winnen in een bepaald aantal zetten. Industriële voorbeelden van klassieke AI zijn robotsystemen of computer visie.
- ML daarentegen is een subgroep van AI waarbij het algoritme kan leren zonder hier expliciet voor geprogrammeerd te zijn. Een algoritme/model kan getraind worden op bestaande data. Het model zoekt dan zelf wiskundig verbanden in de voorgelegde data om zijn voorspellingen te uit te voeren [3].
- *Deep learning* is op zijn beurt een onderdeel van ML. Met het woord “deep” wordt er gerefereerd naar modellen met een diepgaande lagenstructuur. Dit wil zeggen dat de computer vertrekkend van de ingaande data langs verschillende lagen moet passeren om een predictie te maken van de uitgang. Deze modellen zijn origineel ontstaan door het proberen na te bootsen van de neuronenstructuur in het menselijk brein. Hierdoor worden deze modellen ook wel neurale netwerken genoemd [3].

Figuur 4 geeft de historie weer van artificiële intelligentie wanneer elke techniek ontstaan is.



Figuur 4: Geschiedenis van artificiële intelligentie [5]

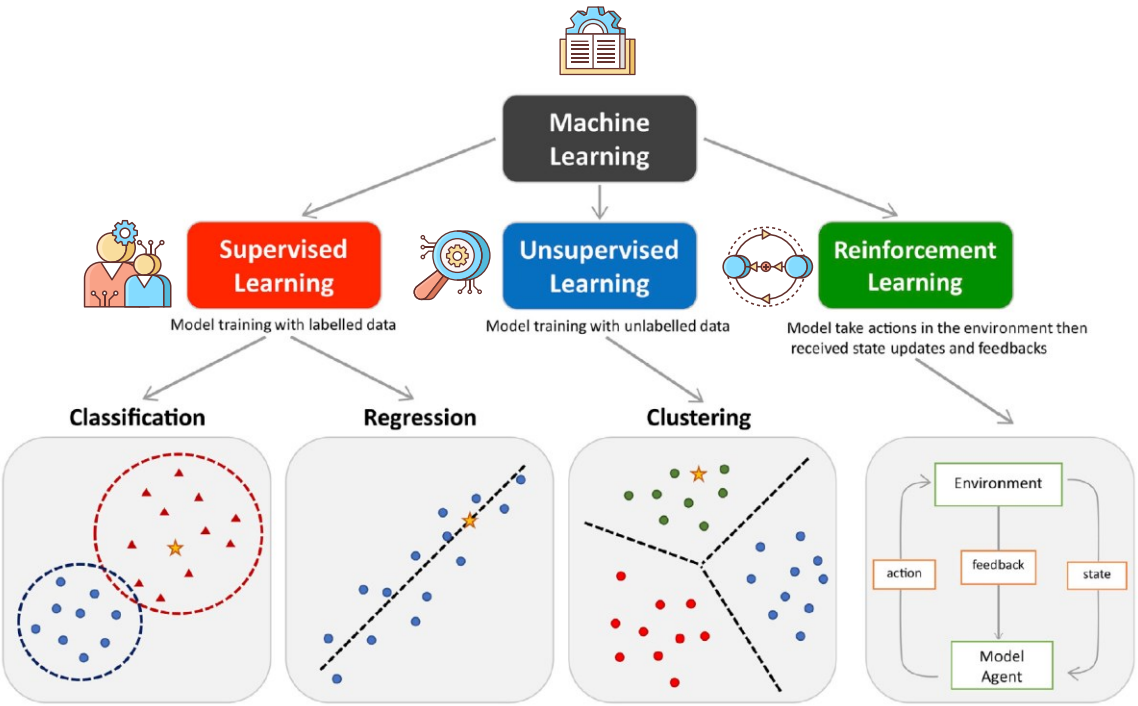
2.1.1 Soorten problemen

De problemen waarvoor ML gebruikt wordt kunnen opgesplitst worden in categorieën [6]:

- Classificatie: Bij classificatie is er een probleem waarbij de data voorspeld moeten worden onder welke groep/klasse deze toebehoren. Er wordt aan het model meegegeven onder welke gedefinieerde klasse de data kunnen vallen. De ingaande data kunnen bijvoorbeeld foto's zijn van honden en katten waarop het model getraind wordt. Het algoritme moet dan voorspellen of een nieuwe foto behoort tot de klasse hond of kat.
- *Clustering*: Deze probleemstelling is gelijkaardig aan classificatie. Er worden patronen gezocht in de data. De data worden dan opgesplitst in verschillende clusters waarbij de data, dewelke het meest gemeenschappelijk zijn, in éénzelfde cluster worden omvat. Het verschil met classificatie is dat bij clustering de hoeveelheid clusters niet bekend is. Het model kan honden- en kattenfoto's onder elkaar onderscheiden maar het model is niet geleerd dat dit honden-en kattenfoto's zijn. Er worden zelf verbanden gezocht tussen de foto's.
- Predictie: Als laatste probleemstelling zijn er de predicties. Hier worden de modellen getraind op historische data om zo een voorspelling op nieuwe data te maken. Een voorbeeld hiervan kan toegepast worden op huizenprijzen. Afhankelijk van de eigenschappen (*features*) van het huis, zoals de oppervlakte van het huis, aantal badkamers, aantal slaapkamers, … wordt er een model getraind. Met nieuwe data kan er dan een nieuwe prijsvoorspelling gemaakt worden.

2.1.2 Machine learning typen

De voornaamste ML-typen kunnen opgesplitst worden in drie methodieken. Namelijk *supervised learning*, *unsupervised learning* en *reinforcement learning* [7]. Met Figuur 5 wordt er een overzicht weergegeven over de verschillende technieken



Figuur 5: Voornaamste machine learning typen [7]. De gekleurde bollen en driehoeken representeren de trainingsdata, de sterren stellen de door het model voorspelde data voor

1. Supervised learning modellen worden getraind waarbij de getrainde data gelabeld zijn. De labels zijn de fundamentele waarheid dewelke overeenstemmen met de features van de getrainde dataset. Het getrainde model wordt dan gebruikt om nieuwe input data mee te geven om zo een label te voorspellen. Supervised learning kent 2 mogelijke problemen nl. classificatie en regressie:

    - Classificatie is het voorspellen bij welke groep/klasse de data toebehoren zoals eerder aangehaald in 2.1.1. Het algoritme wordt getraind op de features van de dataset en moet met nieuwe niet-gelabelde data de klasse voorspellen.
    - Regressie is het trainen en voorspellen van een continue uitkomst. De voorspelling van huizenprijzen is bijvoorbeeld een regressieprobleem zoals vermeldt in 2.1.1. De labels van deze dataset zijn de prijzen van de huizen. Het model berekent een regressielijn welke zo goed mogelijk de trainingsdataset benaderd. Bij het invoegen van eigenschappen van een onbekend huis op het getrainde model maakt deze een prijsvoorspelling aan de hand van deze regressielijn.

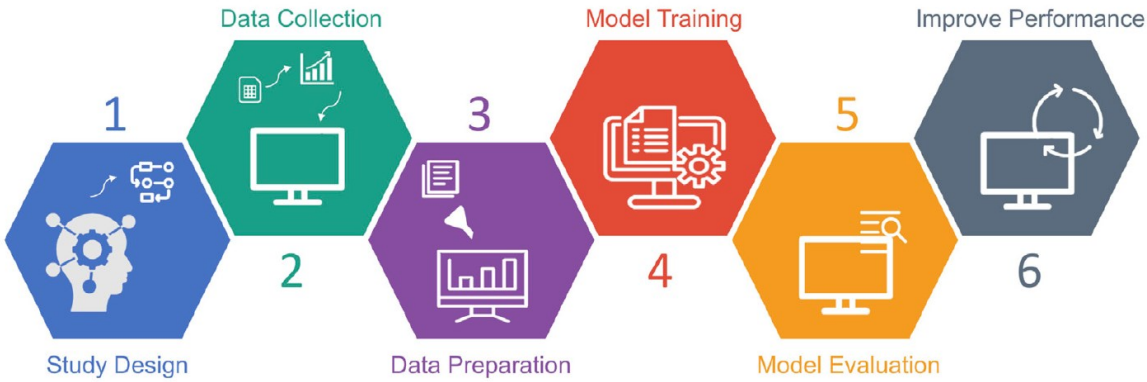
2. Unsupervised learning maakt het mogelijk om met niet-gelabelde data een model te trainen. Hierbij zoekt het model zelf naar patronen in de data. De meest voorkomende methode van unsupervised learning is de methode waarbij de data zich opsplitsen in verschillende clusters. Deze methode wordt dan ook wel clustering genoemd, zie 2.1.1. Een andere welbekende methode is de anomalie detectie. Hierbij moet het algoritme dan zelf zoeken wat er niet normaal is aan de data. Stel de getallenreeks 1, 1, 2, 3, 5, 8, 13, 20, 33, 53. Bij deze reeks hoort het getal 20 niet omdat deze niet de som is van de twee voorgaande getallen [8].

3. Reinforcement learning geeft de mogelijkheid om modellen te ontwikkelen die kunnen leren uit hun eigen succes of falen zoals beschreven wordt in [3]. Het verschil met supervised learning is dat deze modellen geen meester nodig hebben om het model te begeleiden. Bij deze modellen is er een agent welke de status ontvangt van de huidige situatie. Aan de hand van deze status onderneemt de agent een actie. Afhankelijk hoe goed of slecht de actie was krijgt de agent een beloning of een bestraffing. Als voorbeeld hierbij kan een schaakspel genomen worden:

    - De supervised learning aanpak van dit probleem verloopt als volgt. Bij schaken kan de omgeving het bord zijn waarbij twee schaakmeesters het tegen elkaar opnemen. De computer doet een actie waarvan hij denkt dat deze correct is. Hierna wordt het algoritme beloond of gestraft als deze zet wel of niet overeenstemt met de zet die de schaakmeester uitgevoerd zou hebben. Deze “correcte zet” is het label van deze dataset. Het nadeel van deze aanpak is dat er een relatief weinig aantal gelabelde zetten zijn tussen de beste schaakmeesters (10^8) terwijl er veel meer mogelijke schaakzetten zijn (10^{40}) [3].
    - Een alternatieve methode zou zijn om het probleem op te lossen via reinforcement learning. Twee schaakcomputers kunnen tegen elkaar opgezet worden waarbij enkel de schaakregels aan het model worden meegegeven. Als beloning krijgt de computer bijvoorbeeld bij winst 1, bij een gelijkspel + ½ en bij verlies 0. De computer weet op voorhand niet waarom hij een beloning krijgt en wordt geleidelijk aan slimmer. Hij zal het winnen ook linken aan een beloning. Met deze methode kan de computer veel meer data verzamelen dan de gekende data van de beste schaakmeesters. Hiermee leert hij slimmer worden dan de beste en wordt hijzelf de kampioen in het schaken [3].

2.1.3 Machine learning workflow

Elk machine learning probleem volgt een workflow om vertrekkend van data een resultaat te bekomen. De workflow dewelke ook in deze masterproef gevolgd is kan opgesplitst worden in verschillende methodologische stappen [7]. In de volgende subparagrafen worden respectievelijk de stappen van de workflow getoond. Figuur 6 geeft hiervan een visuele voorstelling weer.



Figuur 6: Machine learning workflow [7]

2.1.3.1 Onderzoeksopzet

In deze eerste fase wordt er onderzoek verricht naar de beschikbare ML-methodes om het probleem op te lossen (paragraaf 2.1 en 2.2) en wordt er gekeken naar gelijkaardig onderzoek (paragraaf 2.3). Na het onderzoek wordt er een procedure opgemaakt met daarin in grote lijnen de aanpak voor de volgende uit te voeren stappen.

2.1.3.2 Data verzamelen

Als volgt worden de ruwe data verzameld van de applicatie waarop het ML-model getraind wordt. Deze data kunnen zelf geregistreerd worden of via externe bronnen verzameld worden.

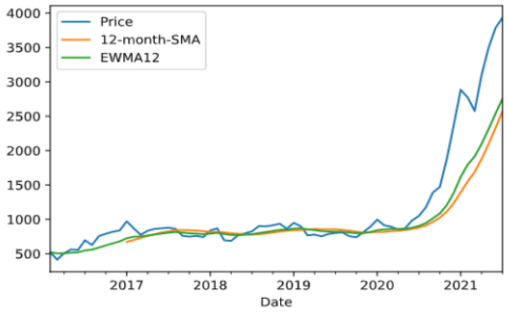
2.1.3.3 Data voorbereiden

Nu de ruwe data ter beschikking zijn kunnen deze bewerkt worden om de features zo optimaal mogelijk te maken voor het aanleren van het ML-model. In volgende subparagrafen worden verschillende data pre-processing technieken aangehaald waaraan er tijdens deze masterproef ook aandacht is besteed.

2.1.3.3.a Feature engineering

Hier worden van bestaande features nieuwe features gemaakt. Een voorbeeld van een tijdsgerelateerde feature wordt voorgesteld in [9]. Dit is de *Simple Moving Average* (SMA). Hier wordt het gemiddelde genomen van de beschikbare data over een gekozen periode.

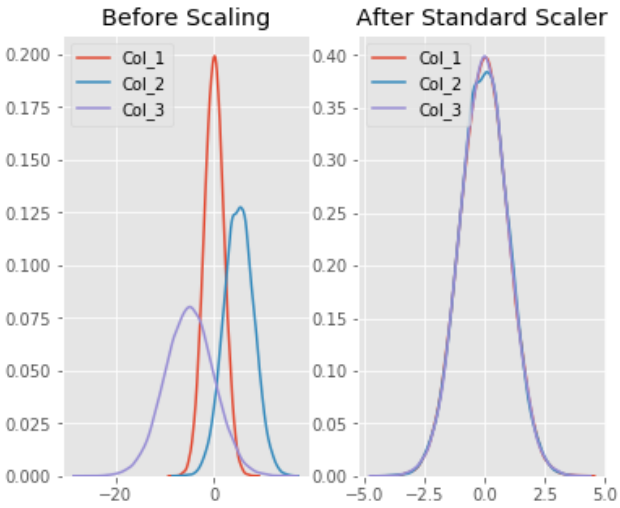
Figuur 7 toont hiervan een voorbeeld met in het blauw de prijswijziging en in het oranje de SMA over de laatste 12 maanden.



Figuur 7: Simple Moving Average voorbeeld[9]

2.1.3.3.b Standaardiseren van de data

Elke feature van de data kan geschaald worden zodat deze gestandaardiseerd is. Het model houdt dan met elke feature evenveel rekening en focust zich dan niet op bepaalde features die hogere waarden hebben. Een bekende ML-bibliotheek, scikit-learn, heeft hiervoor standaard functies ingebouwd [10]. De features van de data worden hierbij beschouwd als een normaal gedistributeerd proces. Hierbij worden de data gecentreerd tot een gemiddelde van 0 en een standaardafwijking van 1. Bij figuur 8 wordt er een vergelijking weergegeven tussen niet geschaalde en geschaalde data.



Figuur 8: Schalen van data[11]

2.1.3.3.c Train, validatie en test split van de data

De data worden opgesplitst in 3 batchen zoals beschreven in [5]. Afhankelijk van hoeveel data er ter beschikking zijn, is deze verhouding groter of kleiner. Indien er veel data beschikbaar zijn kan de split er als volgt er uitzien: 80 % train, 10 % validatie en 10 % test. Met de train set wordt het model getraind. Via de validatie set wordt de performantie van het model geëvalueerd en worden *hyperparameters* bijgestuurd. Het model met de beste resultaten wordt dan met die hyperparameters getest op de onafhankelijke test set.

2.1.3.4 Model trainen

De data zijn voorbereid en in deze fase kan het model getraind worden. Elk model leert op een verschillende manier. Bij deze masterproef is er gekozen om het probleem als een supervised regressieprobleem te behandelen. De gebruikte modellen en hun leerfase worden aangehaald in paragraaf 2.2. Voordat een algoritme getraind wordt moeten er hyperparameters aan elk algoritme worden meegegeven. Deze hyperparameters bepalen dan ook hoe het model getraind wordt op de trainingsdata. De hyperparameters kunnen gaan van de grootte van de graad in een polynomische regressie tot het aantal neuronen in een neuraal netwerk.

Het trainen van een algoritme heeft als doel de kostfunctie zo laag mogelijk te maken. Met de gekozen hyperparameters worden er wiskundige berekeningen uitgevoerd. Het algoritme wordt bestraft als de kost stijgt, hierdoor wordt deze bij elke iteratie beter. Het type kostfunctie is afhankelijk van het model.

2.1.3.4.a Regularisatie

Een hyperparameter die in veel modellen gebruikt wordt is de regularisatieparameter. Volgens [3] is dit een positieve parameter die de relatie bevat tussen de kostfunctie en de complexiteit van de hypothese of van het model. Door deze correct in te stellen wordt een complexe hypothese bestraft waardoor het model beter inspeelt op de data.

2.1.3.5 Model evaluatie

Na het trainen wordt er een evaluatie uitgevoerd op het model. Er wordt nagegaan hoe het model presteert op niet getrainde data. Als volgt kunnen er uit deze resultaten conclusies getrokken worden om te kijken hoe het model bijgestuurd moet worden.

2.1.3.5.a Evaluatiemetrieken

Er zijn verschillende metrieken om te valideren of de voorspellingen zijn zoals verwacht. Onderstaand zijn er enkele metrieken opgesomd voor regressieproblemen welke gebruikt zijn in deze masterproef [12]:

1. MAE (Mean Average Error), de absolute fout tussen het verschil van de de originele waarde en de voorspelde waarde.
2. MSE (Mean Squared Error), de kwadratische fout tussen het verschil van de de originele waarde en de voorspelde waarde. De uitschieters hebben met deze metriek meer effect doordat de fout gekwadradeerd wordt.
3. RMSE (Root Mean Squared Error), dit is de vierkantswortel van MSE. Deze metriek is makkelijker te lezen omdat de MSE een gekwadrateerde waarde is.
4. R2-score, deze metriek geeft weer hoe goed de originele waarden de getrainde regressielijn omvatten. De waarde hiervan kan zijn tussen 0 en 1. Hoe hoger de waarde hoe beter de getrainde regressielijn de waarden evenaart.

Deze metrieken kunnen als volgt uitgedrukt worden:

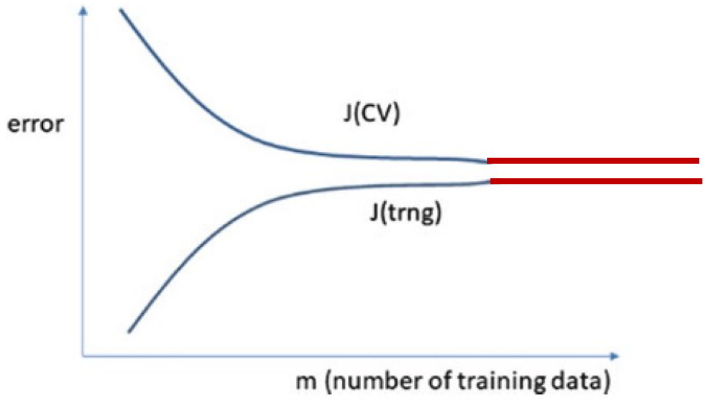
$$MAE = \frac{1}{m}\sum_{i=1}^{m}|X_i-Y_i| \;;\; MSE = \frac{1}{m}\sum_{i=1}^{m}(X_i-Y_i)^2 \;;\; RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(X_i-Y_i)^2} \;;\; R^2 = 1 - \frac{\sum_{i=1}^{m}(X_i-Y_i)^2}{\sum_{i=1}^{m}(\overline{Y}-Y_i)^2}$$

Met m = het aantal elementen; X_i = de voorspelde waarde bij i; Y_i = de actuele waarde bij i; $\overline{Y}$ = het gemiddelde van de actuele waarden.

2.1.3.5.b Bias vs Variance

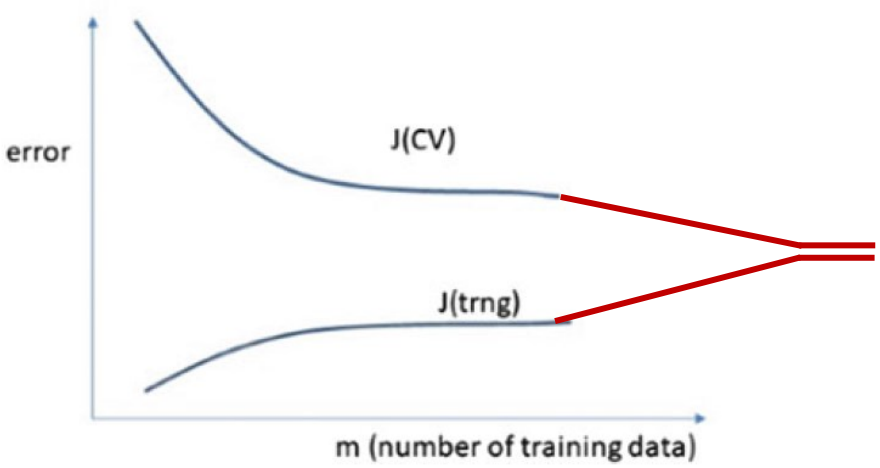
Deze evaluatiemetrieken worden zowel getest op de trainingsset als op de *cross-validation* set. De resultaten tussen de verschillende sets kunnen wijzen op een *bias*/*underfitting* of *variance*/*overfitting* probleem [13]:

- Bij een **high bias** probleem voorspelt het model zowel een grote fout op de cross-validation set alsook op de trainingsset. Figuur 9 toont aan hoe de cross-validation kostfunctie zijn error (J(CV)) en de trainingskostfunctie zijn error (J(trng)) evolueren als er meer trainingsdata beschikbaar worden gemaakt om het model te trainen. De fout van de voorspellingen convergeert na een tijd waardoor deze niet meer veranderen na het toevoegen van nieuwe trainingsdata. Meer trainingsdata hebben dus geen effect meer bij een onderfittingsprobleem doordat de fout van de cross-validation set niet onder dit bepaalde niveau zakt. Het model moet eerst op andere manieren aangepakt worden. In deze figuur wordt in het rood weergegeven wat het gevolg is als er nog meer trainingsdata worden toegevoegd.



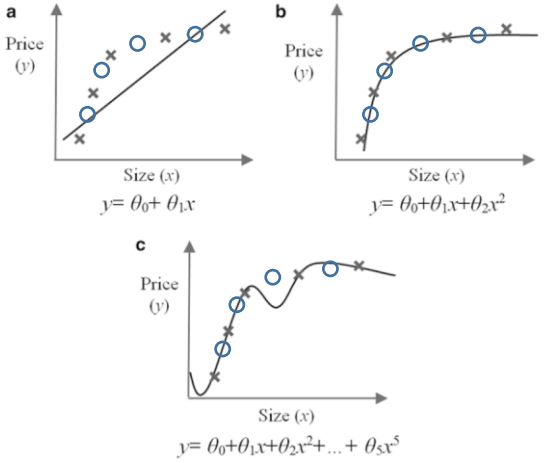
Figuur 9: High bias probleem [13]

- Een **high-variance** probleem daarentegen heeft wel ook een grote fout op de cross-validation set maar heeft een lage fout op de trainingsset. Dit fenomeen wordt ook wel overfitting genoemd omdat het model zich te veel focust op de trainingsset. Figuur 10 geeft een voorbeeld weer van de evolutie van een high variance probleem als er meer trainingsdata worden toegevoegd. De cross-validation kostfunctie (J(CV)) en de trainingskostfunctie (J(trng)) kunnen in deze figuur terugvonden worden met hun bijpassende error. Zoals er opgemerkt kan worden heeft bij dit probleem het toevoegen van meer trainingsdata wel een positief effect. Als er meer trainingsdata worden toegevoegd zal het model zich beter generalizeren over de data. In het rood op deze figuur is er gevisualiseerd hoe de cross-validation fout verlaagt en trainingsfout verhoogt bij het toevoegen van meer data.



Figuur 10: High variance probleem [13]

Figuur 11 toont een voorbeeld van de voorspelling van een huizenprijs waar deze problemen worden aangehaald. De x-as representeert de features, in dit geval de grootte van het huis, en de y-as representeert de prijs van het huis. De kruisjes zijn de trainingsdata en de cirkels de cross-validation data. De geplotte lijn is de regressielijn die het model heeft voorspeld a.d.h.v. de trainingsdata. Deze is simpel bij het underfitting probleem en te complex bij het overfitting probleem.



Figuur 11: Huizenprijzen voorspelling bij drie hypotheses underfitting (a), goede fit (b), overfitting (c) [14]

Om het model in beide situaties te verbeteren kunnen volgende parameters aangepast worden [13]:

- High bias
  - Het aantal features verhogen.
  - De regularisatie parameter verlagen zodat er meer features overblijven.
- High variance
  - Het aantal features verlagen.
  - De regularisatie parameter verhogen zodat het model minder complex wordt met minder features.
  - Trainen met meer data.

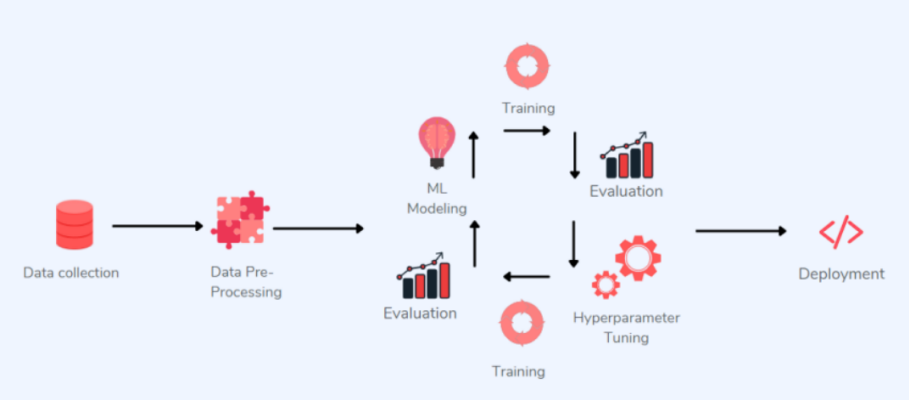
2.1.3.6 Performantie verbeteren

Als laatste stap wordt de performantie van het model nog verbeterd zodat het model optimaal is voordat dit in de machine geïmplementeerd wordt. Er bestaan verschillende methodes om een ML-model te verbeteren.

2.1.3.6.a Model-centric vs data-centric

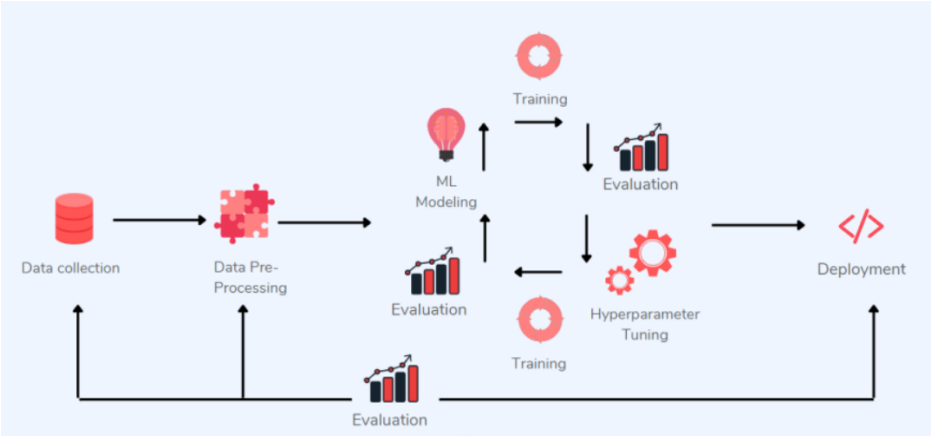
In deze masterproef wordt er een combinatie uitgevoerd van de *model-centric* en *data-centric* methode [15]:

- Model-centric: Met deze methode worden de prestaties van het model verbeterd door het model zelf te tunen. Dit door middel van *hyperparameter tuning*. Hiermee worden de parameters van het model aangepast en wordt het model opnieuw getraind en geëvalueerd. Deze cyclus blijft zich herhalen totdat de evaluatie is zoals verwacht. Hierna wordt het model geïmplementeerd in de toepassing. Een voorbeeld van een hyperparamter kan het aantal neuronen van een neuraal netwerk zijn. Met deze parameter wordt de complexiteit van dit model aangepast. Figuur 12 toont de workflow van deze methode.



Figuur 12: Model-centric workflow [16]

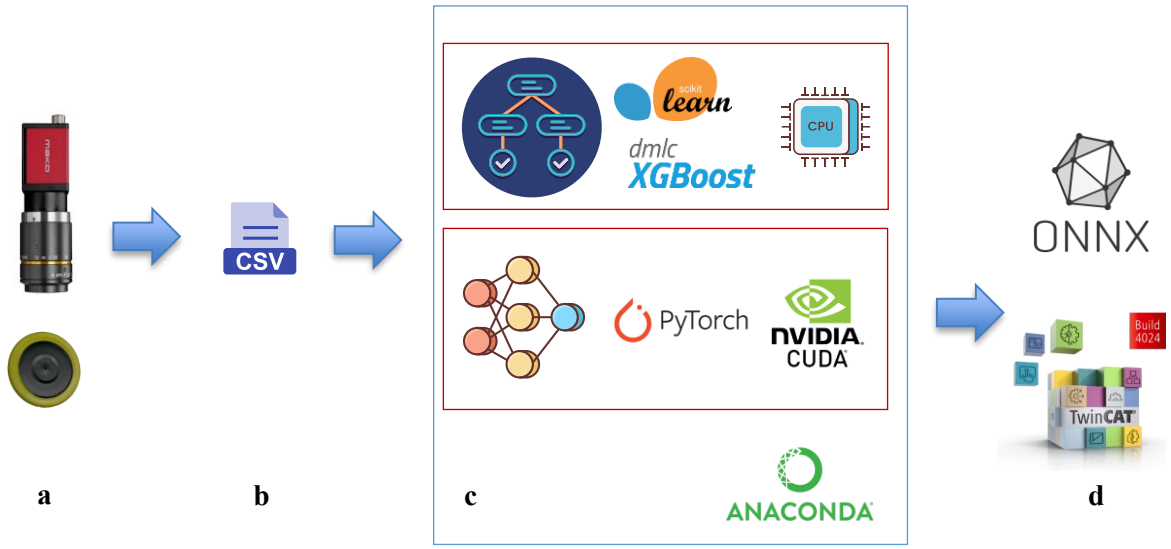
- Data-centric: Een andere methode is niet alleen het model verbeteren maar ook de inkomende data. Hierbij kan de ruwe data aangepast worden door bijvoorbeeld extra data te generen i.p.v. enkel op de geleverde data te vertrouwen. In deze data pre-processing fase kunnen er op de bestaande data features ook nog feature engineering toegepast worden. Na het aanpassen van deze methodes wordt het model opnieuw getraind en geëvalueerd. Figuur 13 geeft de workflow weer van een combinatie van de data-centric en model-centric methode.



Figuur 13: Data-centric i.c.m. Model-centric workflow [16]

2.1.3.7 Workflow robot-airhockeytafel

Figuur 14 toont de verschillende stappen van de ML-workflow die gevolgd werden na de literatuurstudie. Figuur 14a+b hebben betrekking op de stappen van de dataverzameling. Deze worden besproken in hoofdstuk 3. Bij figuur 14c wordt in de Pythonomgeving zowel de data-preprocessing uitgevoerd alsook worden er verschillende ML-modellen op deze data getraind en geëvalueerd (Zie hoofdstuk 4). En als laatste wordt in figuur 14d het model in TwinCAT 3 geïmporteerd om te gebruiken in de real-time omgeving. Meer hierover is terug te vinden in hoofdstuk 5.



Figuur 14: ML-workflow robot-airhockeytafel

2.2 Supervised learning regressie modellen

In deze masterproef is er gekozen om de robot-airhockeytafel als een supervised learning regressie probleem te behandelen zoals later besproken wordt in paragraaf 2.3.3. Er bestaan veel bronnen met verschillende technieken over hoe dit probleem opgelost kan worden met of zonder ML. In deze paragraaf worden de verschillende supervised learning regressie ML-algoritmes aangehaald met hun achterliggende werking en beperkingen.

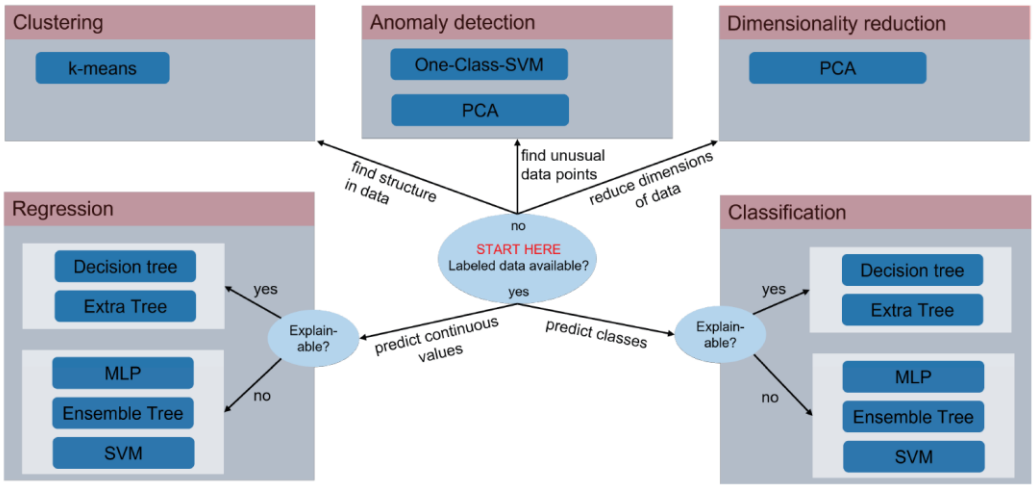
2.2.1 Ondersteuning ML in TwinCAT 3

TwinCAT 3 is de automatiseringssoftware van Beckhoff Automation, hun ML-toolbox "TF3800 TwinCAT 3 Machine Learning Inference Engine" is nog maar een paar jaar op de markt. Deze ondersteunt al veel modellen maar zeker nog niet allemaal. De tot nu toe ondersteunde modellen kan men terugvinden in tabel 1.

Tabel 1: Ondersteunde ML-modellen met Beckhoff [17]

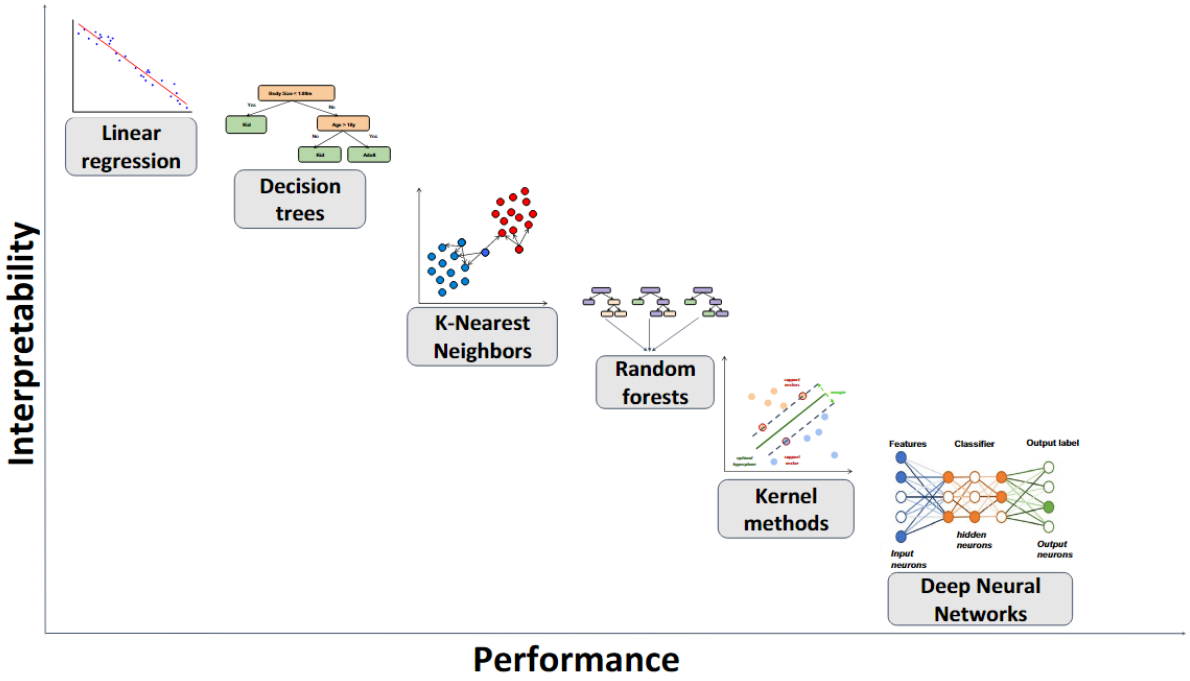
Model type	License	Available from setup version
Support vector machine [▶ 27] (SVM)	TF3800	3.1.42.0
Principal Component Analysis [▶ 34] (PCA)	TF3800	3.1.57.0
k-Means [▶ 33]	TF3800	3.1.57.0
Random Forest [▶ 43]	TF3800	3.1.58.0
Multi-layer perceptron [▶ 21] (MLP)	TF3810	3.1.42.0
Decision Tree [▶ 36]	TF3800	3.1.62.0
Extra Tree [▶ 38]	TF3800	3.1.62.0
Extra Trees [▶ 41]	TF3800	3.1.62.0
Gradient Boosting [▶ 46]	TF3800	3.1.62.0
Hist Gradient Boosting [▶ 48]	TF3800	3.1.62.0
XGBoost [▶ 50]	TF3800	3.1.62.0
LightGBM [▶ 54]	TF3800	3.1.62.0

Het is niet mogelijk om bij al deze modellen een supervised regressieprobleem op te lossen. Figuur 15 toont een visuele voorstelling waarbij er via enkele keuzes het best passend, door TwinCAT 3 ondersteunde model gekozen kan worden voor het probleem. De mogelijke opties in deze masterproef zijn *decision tree*, *ensemble tree*, SVM (Support Vector Machine) en MLP (Multilayer perceptron) neuraal netwerk,



Figuur 15: TwinCAT 3 AI model cheat sheet [17]

De regressiemodellen worden hierbij ook opgesplitst in 2 groepen naar gelang hun interpreteerbaarheid. Sommige modellen zijn van aard potentieel performanter terwijl andere beter verstaanbaar zijn door de mens waarom een bepaalde beslissing is genomen. In figuur 16 is het gedrag van de meest voorkomende modellen voorgesteld.



Figuur 16: Vergelijking potentiële performantie en interpreteerbaarheid van ML-modellen [18]

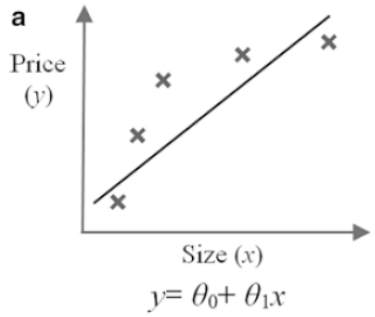
In de volgende subparagrafen wordt de werking van de ondersteunde modellen aangehaald alsook van enkele niet-ondersteunde modellen waar ook onderzoek op uitgevoerd is. Deze modellen hebben allemaal hun eigen eigenschappen en leren elk op hun eigen manier.

2.2.2 Lineaire regressie

Om kennis te maken met de supervised learning regressie methodes is er eerst onderzoek gedaan naar de eenvoudigste vormen van regressie met ML. Dit zijn namelijk de lineaire en polynomische regressie ook al zijn deze niet ondersteund door TwinCAT 3. Bij deze modellen leert het model een regressielijn op basis van een lineair of polynomisch karakter.

2.2.2.1 Hypothese

De voorspelde waarden van een ML-model kunnen ook als een hypothese omschreven worden. Stel er is data beschikbaar over verschillende huizenprijzen (y) welke overeenstemmen met de grootte van het huis (x). Figuur 17 stelt deze data voor in een grafiek waarbij de kruisjes de voorstelling zijn van de huizenprijzen bij een bepaalde grootte van een huis. Er kan een lineair verband waargenomen worden met deze datapunten. Dit verband is hier afgebeeld met de geplotte regressielijn. Deze regressielijn wordt hier de hypothese genoemd. De formule van deze hypothese is als volgt: $h_\theta(x) = \theta_0+\theta_1 x$. Dit model heeft twee parameters namelijk θ_0, deze stelt hier het snijpunt voor met de y-as en θ_1, de richtingscoëfficiënt van de hypothese. Deze waarden worden gekozen zodanig dat $h_\theta(x)$ zich zo dicht mogelijk bij de trainingsdata bevindt. [19].



Figuur 17: Huizenprijzen lineaire regressie [14]

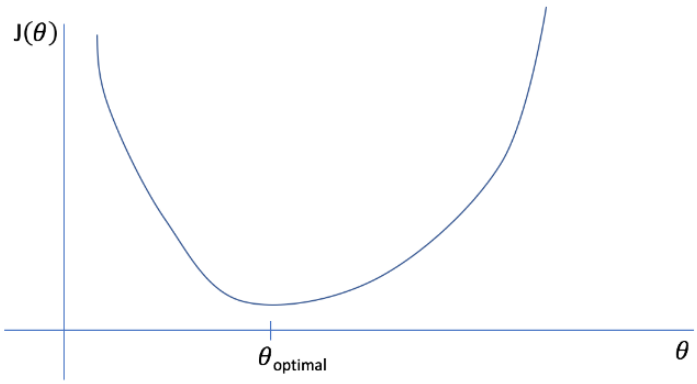
2.2.2.2 Kostfunctie

Om de optimale θ_j parameters te vinden wordt er een kostfunctie opgesteld. De kostfunctie bepaalt wat de kost is van de huidige hypothese. Of anders verwoordt deze geeft een indicatie hoe de hypothese presteert t.o.v. de originele data. De kostfunctie van lineaire regressie is als volgt [19]:

$$J(\theta_0, \ldots, \theta_j) = \frac{1}{m}\sum_{i=1}^{m} (h_\theta(x^{(i)})\text{-}y^{(i)})^2$$

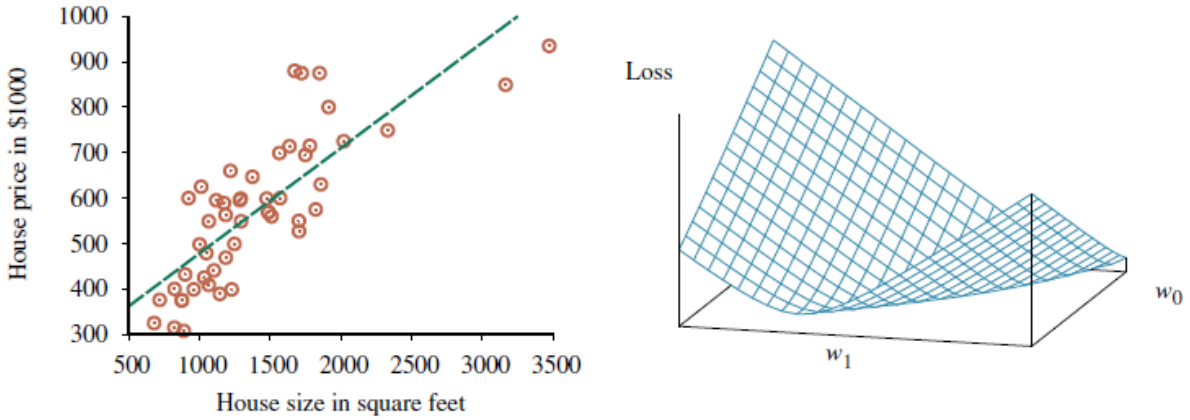
Hier wordt de error voorgesteld door de waarde van de regressielijn ($h_\theta(x^{(i)})$) af te trekken met de reële waarde ($y^{(i)}$). Deze fout wordt gekwadrateerd en uitgevoerd voor elk element van de trainingsset door de som hiervan te nemen. De letter m staat voor het aantal datapunten in de trainingsset. Als laatste wordt er gedeeld door het aantal datapunten zodat de kostfunctie niet altijd groter wordt als er meer datapunten beschikbaar zijn. Nu is de kostfunctie compleet, deze bepaalt de kost van een bepaalde hypothese voor de gekozen parameters θ_j. De letter j staat hier voor het aantal parameters. Het doel is om deze kostfunctie zo laag mogelijk te maken, als deze 0 zou zijn betekent het dat de hypothese perfect overeenkomt met de reële waarden [19]. Deze kostfunctie komt overeen met de MSE-evaluatiemetriek eerder aangehaald in subparagraaf 2.1.3.5.a.

Figuur 18 toont de grafiek van de kostfunctie voor één van deze parameters, deze is parabolisch van aard. Er is een minimum kost waar de θ dus optimaal is.



Figuur 18: Plot kostfunctie met één parameter [19]

Als er 2 parameters zijn zoals in het vorig voorbeeld met de huizenprijzen wordt de kostfunctie afgebeeld d.m.v. een convex (Figuur 19). Hier zijn θ_0 en θ_1 afgebeeld als w0 en w1. De convexe figuur heeft ook een minimum bij een bepaalde waarde van θ_0 en θ_1, dit is de optimale hypothese.



Figuur 19: Huizenprijzen lineaire regressie met convexe kostfunctie [3]

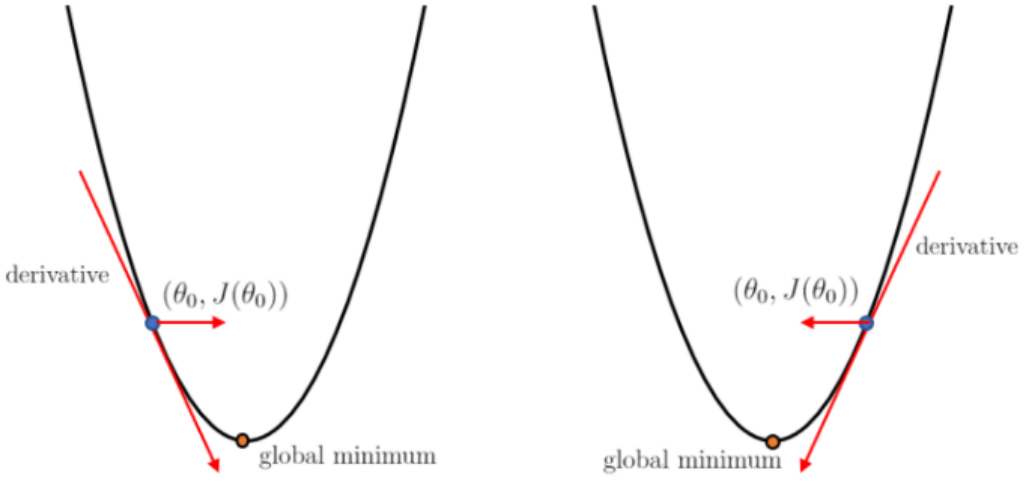
2.2.2.3 Gradient descent

Als volgt moet er een algoritme ontwikkeld worden welke het minimum van de kostfunctie opzoekt. Een voorbeeld hiervan is *gradient descent*. Gradient descent start met willekeurige waardes voor de parameters. Hierna worden de parameters iteratief geüpdatet tot als de kostfunctie vervolgens zijn minimum bereikt. Het algoritme voert volgende functie uit tot als deze convergeert [19]:

$$\theta_j = \theta_j - \alpha \frac{\delta}{\delta\theta_j} J(\theta_0, \dots, \theta_n)$$

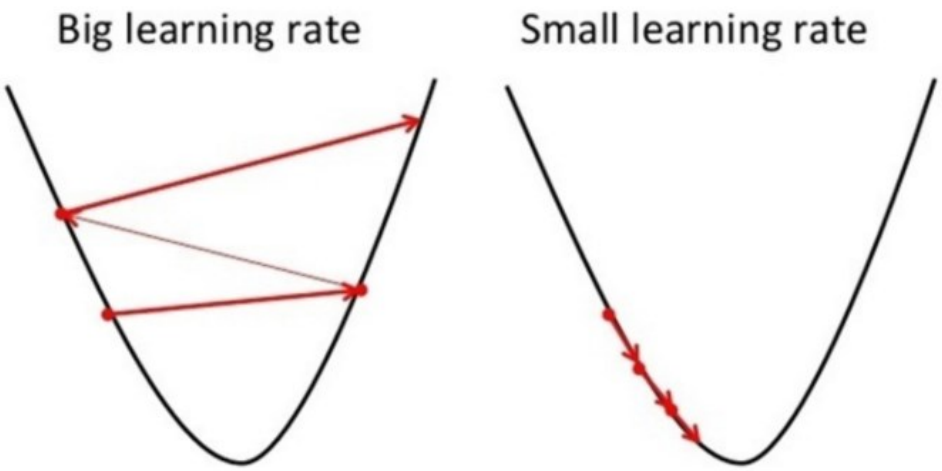
Met α de leersnelheid en j het aantal parameters

De formule vertrekt van de huidige waarde van parameter θ_j. Deze wordt afgetrokken met de afgeleide van de huidige waarde van de kostfunctie en vermenigvuldigd met de leersnelheid. Dit wordt eerst uitgevoerd voor alle parameters waarbij elke nieuwe waarde in een tijdelijke variabele opgeslagen [19] spatie voor wordt. Na het uitvoeren van deze formule op alle parameters worden de huidige waarden van de parameters vervangen door hun nieuwe waarden. Hierna herhaalt de formule zich terug tot deze convergeert. Het doel van deze functie is altijd het globale minimum zoeken van de functie. Figuur 20 toont in het blauw een bepaalde waarde voor de kostfunctie bij θ_0. Na het uitvoeren van gradient descent blijkt dat de linkse figuur naar rechts moet convergeren en de rechtse figuur naar links moet convergeren om tot het globaal minimum te geraken [19].



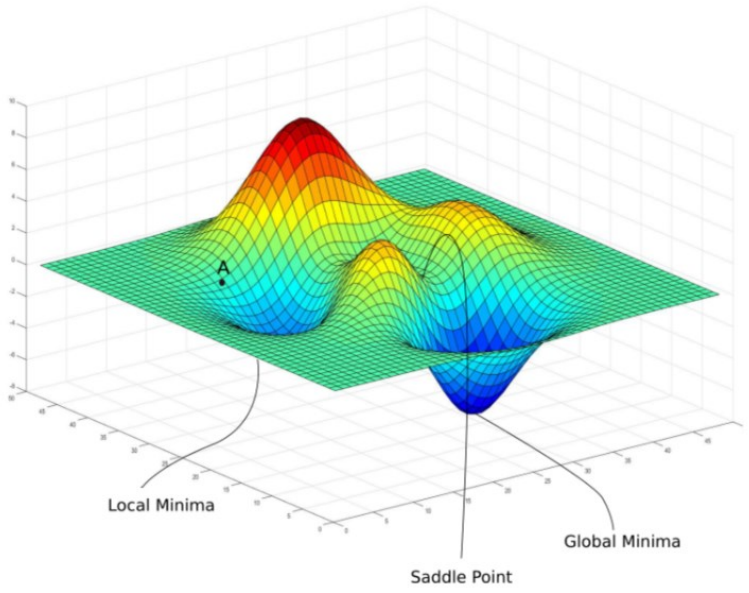
Figuur 20: Gradient descent [19]

Het is belangrijk om met de α waarde rekening te houden. Indien deze te hoog is kan er niet geconvergeerd worden, anderzijds als deze te laag is duurt het veel langer om te convergeren [19]. Figuur 21 toont het verschil aan tussen verschillende leersnelheden [19].



Figuur 21: Leersnelheid gradient descent [19]

Een bijkomend probleem bij deze aanpak kan zijn dat er in de convexe functie niet één maar meerdere minima zijn. Figuur 22 geeft hiervan een voorbeeld weer. Als er vanaf punt A vertrokken wordt kan het zijn dat deze enkel convergeert tot het lokaal minimum dat zich het dichtstbij bevindt. Zo denkt het model dat ze de beste parameters gevonden heeft maar het globaal minimum is niet gevonden met de beste parameters [19]. Hieruit kan geconcludeerd worden dat niet altijd het beste model gevonden kan worden met deze methode.



Figuur 22: Lokaal minimum versus globaal minimum [19]

2.2.2.4 Regularisatieparameter

Om modellen beter te regulariseren op de data kan er de regularisatieparameter worden toegepast zoals eerder aangehaald in 2.1.3.4.a. Voor lineaire regressie is dit de λ parameter, deze wordt toegevoegd aan de kostfunctie waar deze afhankelijk is van de parameters θ. De regularisatieparameter heeft een effect op alle parameters buiten θ_0. De nieuwe kostenfunctie ziet er nu als volgt uit [20]:

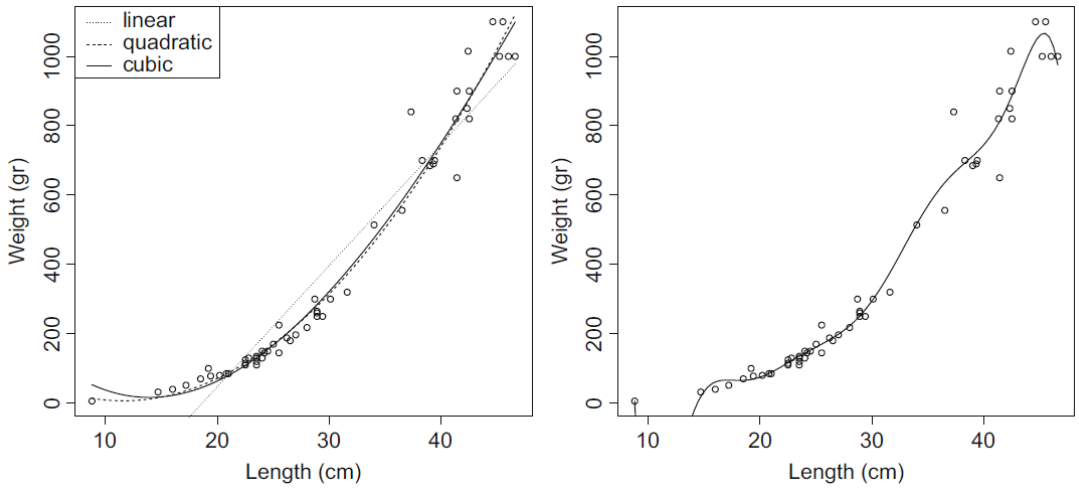
$$J(\theta_0, \ldots, \theta_n) = \frac{1}{m}\left(\sum_{i=1}^{m} (h_\theta(x^{(i)})\text{-}y^{(i)})^2 + \lambda \sum_{j=1}^{n} (\theta_j)^2\right)$$

Het eerste deel van de kostfunctie blijft gelijk maar bij het 2de deel wordt de regularisatie toegevoegd. Een grote λ zorgt ervoor dat de rechterzijde van de formule een groter effect heeft op de kost. De θ_j parameters worden dan zo klein mogelijk gemaakt. Dit komt doordat de θ_j gekwadrateerd worden in de functie. Bij het toepassen van gradient descent moet dan een gekwadrateerde waarde zo klein mogelijk gehouden worden om de kost zo laag mogelijk te houden. Het model wordt met deze aanpassing minder complex, dit is goed om overfitting te voorkomen. Bij een kleine λ daarentegen wordt de kost vooral bepaald door de linkerzijde van de functie. De features hebben dan meer effect op het model waardoor dit complexer wordt. Dit is een gewenst effect als het model een onderfittingsgedrag vertoont [20].

Deze parameter wordt bij het tunen van het model aangepast om een goede waarde te bekomen. Hierom wordt dit ook wel een hyperparameter genoemd. Een goede waarde voor λ is op voorhand moeilijk te bepalen. Er kunnen verschillende positieve waarden als mogelijkheden geprobeerd worden [20]. Volgende λ waarden kunnen een goed vertrekpunt zijn: 0,03; 0,1; 0; 1; 3. Zo kunnen de eerste modellen getraind worden met verschillende stappen van λ om te zien welk effect deze geven aan de resultaten.

2.2.2.5 Polynomische regressie

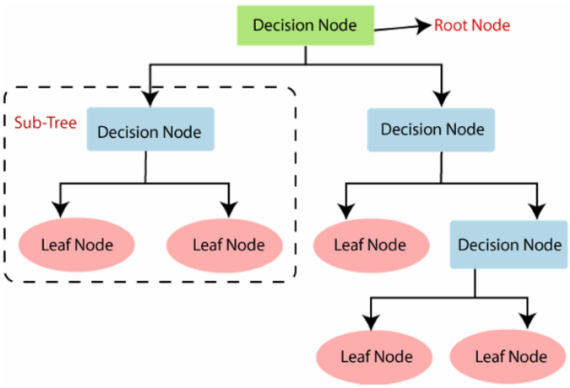
Polynomische regressie bouwt verder op de principes van lineaire regressie. In tegenstelling van lineaire regressie worden er features tot een bepaalde macht verheft. Hiermee is het mogelijk om complexere hypotheses te vormen. Bij figuur 23 worden er 4 verschillende regressielijnen geplot om dezelfde data te evenaren. Op de linkse figuur wordt de lineaire, kwadratische en kubieke regressielijnen getoond. Op de rechtste figuur wordt de regressielijn met een macht tot de 10de getoond. De regressielijn rechts heeft natuurlijk de beste fit op de trainingsdata maar de andere regressielijnen doen het beter bij niet geziene data.



Figuur 23: Gewicht versus lengte bij baars uit het Laengelmavesi meer [21]

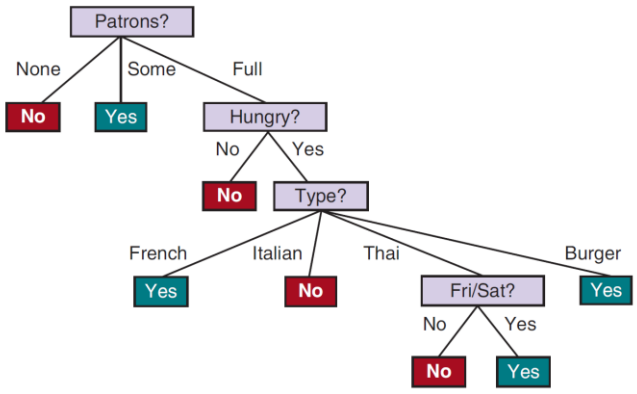
2.2.3 Decision tree

Decision tree is een makkelijk interpreteerbaar model net zoals lineaire regressie. Bij een *decision tree* moeten er in een boomstructuur verschillende IF-statements worden uitgevoerd op de features. Het model leert de belangen van de features op de trainingsdata. Na het trainen vertrekt het model van een eerste beslissing of *root node*, deze beslissing is de belangrijkste. Na de eerste beslissing komt het model terecht bij een volgende *decision node* of bij een *leaf node*. De leaf node is de voorspelling die het model meegeeft aan de gebruiker [22]. Figuur 24 toont de opstelling van een decision tree met zijn verschillende eigenschappen.

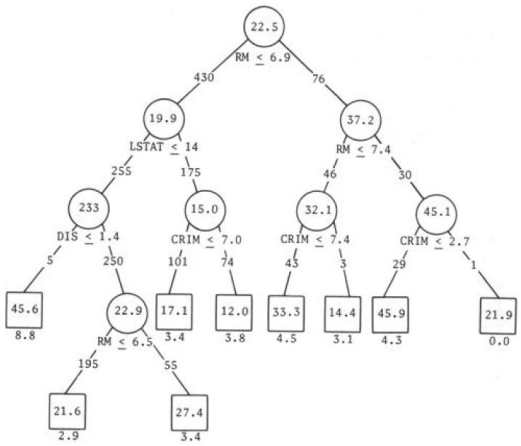


Figuur 24: Decision tree [23]

Dit algoritme kan toegepast worden op zowel classificatie modellen als op regressie modellen. Bij classificatie decision trees zijn de leaf nodes een waarde zoals “waar” of “niet waar” of andere discrete categorieën. Voor regressie decision trees zijn de leaf nodes altijd een numerieke waarde [3]. Op Figuur 25 is er een decision tree voorgesteld a.d.h.v. de vraag of er als klant gewacht gaat worden bij een restaurant. Het model heeft uit trainingsdata geleerd wanneer klanten bereid zijn om te wachten voor een tafel en wanneer niet. Figuur 26 daarentegen stelt een voorspelling van de huizenprijs in Boston voor. Hier maakt het regressiemodel beslissingen naar gelang het misdaadcijfer, als het land gelegen is bij de Charles River, het aantal kamers van de woning, enz.



Figuur 25: Decision tree classificatie voorbeeld [3]



Figuur 26: Decision tree regressie voorbeeld [24]

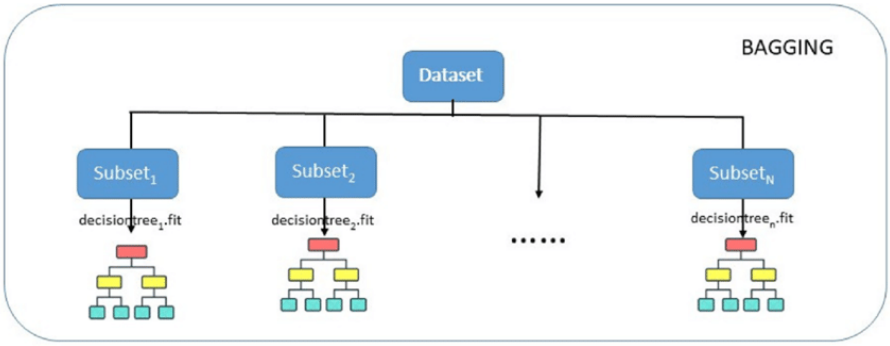
Het decision tree model heeft als eigenschap snel een overfitting gedrag te vormen rond de trainingsdata wat ervoor zorgt dat dit geen optie is voor complexere modellen. Als uitbreiding op de decision trees is er onderzoek uitgevoerd op dit algoritme en zijn de ensemble tree algoritmes ontstaan [22].

2.2.4 Ensemble tree

Ensemble tree is een model gebaseerd op verschillende decision trees. Voor regressieproblemen kan er een onderscheid gemaakt worden tussen *bagging* trees en *boosting* trees [3]. Door het combineren van meerdere decision trees heeft het model minder overfitting gedrag maar wordt de interpreteerbaarheid moeilijker.

2.2.4.1 Bagging

Bij deze methode wordt de data opgesplitst in n-aantal subsets. Dit wordt gedaan door middel van *Bootstrap AGGregatING* (bagging). Deze methode splitst de rijen op in data met vervanging en maakt n-aantal nieuwe datasets van de originele trainingsdataset. Met vervanging wil zeggen dat dezelfde rij meermaals gebruikt kan worden in éénzelfde subset [22]. Na het splitsen worden deze parallel en individueel getraind. Het model neemt dan de voorspellingen van alle decision trees samen als een gemiddelde welke dan de uiteindelijke voorspelling vormt [25]. Decision trees op hun eigen vormen een onstabiel karakter waarbij er bij de kleinste wijziging van de trainingsdata een hele andere beslissingsboom gevormd kan worden. Met bagging wordt deze variatie verzacht en vormt het model een objectief resultaat waar het model de belangrijkste features zoekt in elke subset [3]. Figuur 27 toont een voorbeeld van een bagging ensemble tree implementatie.



Figuur 27: Ensemble tree bagging voorbeeld [25]

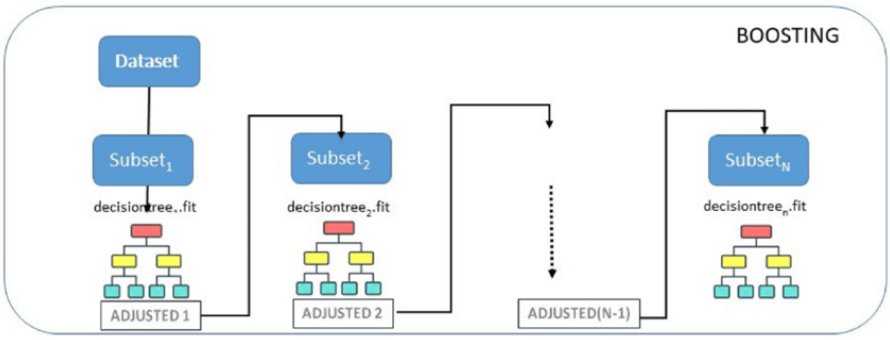
2.2.4.1.a Random forest

Vertrekkend van de bagging workflow zijn er diverse variaties hierop ontwikkeld. Bagging zorgt voor een beter resultaat dan een enkele decision tree. Maar als de data een feature bevat met een grote invloed op het model is het waarschijnlijk dat deze een grote invloed heeft op elke decision tree. Om de variatie nog meer te verkleinen is er het *random forest* model ontwikkeld. Het doel van random forest is om het aantal features te verdelen i.p.v. de trainingsdata op te splitsen. Elke decision tree krijgt dan een bepaalde variatie van de features van de originele trainingsset. Het punt waar de tree zich opsplitst (decision node) wordt telkens bepaald door het lokaal optimum te berekenen. Welke features elke decision tree krijgt wordt willekeurig verdeeld. Hiervan komt de naam random forest [3].

Een bijkomende variatie op de random forest methode is om de data niet via de bagging methode te splitsen maar om de data willekeurig zonder vervanging te splitsen. Dit houdt in dat in elke decision tree de samples uniek zijn. Hierna worden de features terug willekeurig verdeeld over de subsets. De opsplitsing van elke decision tree in een nieuwe decision node gebeurt willekeurig. Door deze willekeurig te maken is het meer waarschijnlijk dat elke decision tree uniek is. Deze techniek is dus meer willekeurig van aard, daarom dat dit de *extra randomized trees* (ExtraTrees) methode wordt genoemd [3].

2.2.4.2 Boosting

Een alternatieve methode voor ensemble tree is de boosting aanpak. Hierbij wordt de dataset terug opgesplitst in meerdere subsets welke meegegeven worden aan meerdere decision trees. Het verschil is dat deze nu niet meer parallel uitgevoerd worden. Na het resultaat van één decision tree wordt de volgende decision tree verbeterd door een feedback terugkoppeling [25]. Figuur 28 visualiseert de workflow van deze aanpak.



Figuur 28: Ensemble tree boosting voorbeeld [25]

2.2.4.2.a Gradient boosting

Net zoals bij bagging bestaan er verschillende boosting methodes voor ensemble tree. Gradient boosting is een ensemble tree boosting techniek welke het mogelijk maakt om een tabel met gestructureerde data te laten leren via gradient descent zoals aangehaald in 2.2.2.3 [3]. Een nadeel van deze techniek is dat het lang kan duren om dit model te trainen door de vele rijen aan data. Bij boosting technieken kunnen de volgende decision tree pas getraind worden als de vorige zijn voorspelling heeft gemaakt. Het is dus niet mogelijk om deze parallel te trainen zoals bij de bagging methode.

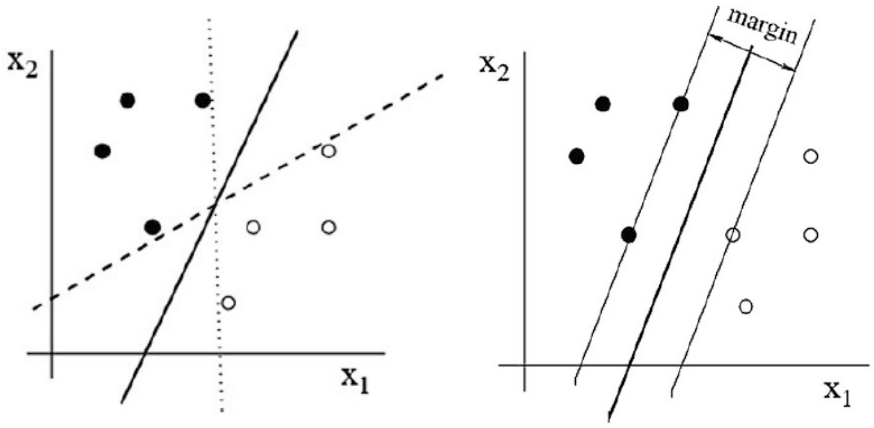
Om dit proces te optimaliseren worden technieken toegepast zoals het discretiseren (*binning*) van de inkomende data. Hierbij wordt de continue waarde van de trainingsdata vereenvoudigd naar enkele honderden continue waarden. Deze methode wordt histogram gradient descent genoemd [26]. Bekende algoritmes rond deze methode zijn Light Gradient-Boosting Machines (LightGBM) [27] en Extreme Gradient Boosting (XGBoost) [28].

2.2.5 SVM

SVM is een bekend algoritme. Deze technologie wordt voor classificatie problemen gebruikt en noemt men SVC (Support Vector Classification). In deze masterproef is de tegenhanger SVR (Support Vector Regressor) onderzocht voor regressie problemen.

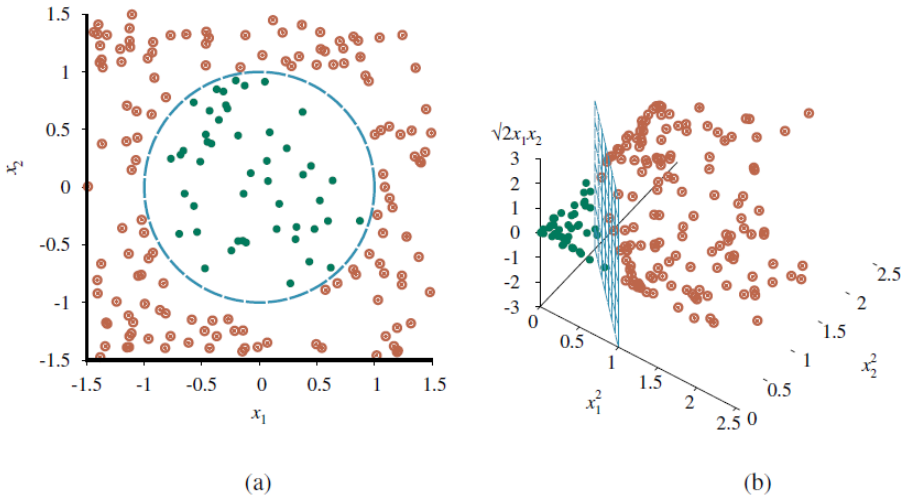
2.2.5.1 SVC

Met dit algoritme wordt er een beslissingsgrens of *hyperplane* gevormd welke de klassen van elkaar splitst. Er is bewezen dat de hyperplane met de meeste marge tussen beide klassen het beste presteert met toekomstige, ongeziene data. Figuur 29 toont een SVM-classificatie toepassing waarbij de volle bollen worden gescheiden van de lege bollen. Aan de linkse kant van de figuur zijn er verschillende hyperplanes berekend en aan de rechtse kant van de figuur wordt de beste hyperplane getoond met de meeste marge. De punten van beide klassen die op de marge lijn liggen worden ook wel de support vectors genoemd vandaar de naam SVM [29].



Figuur 29: SVM hyperplane en marge [29]

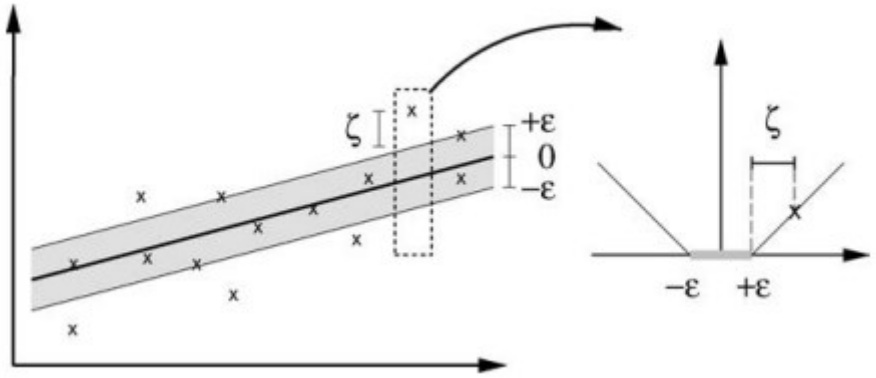
Standaard creëert een SVM een lineaire hyperplane. Het is mogelijk om voor niet-lineaire problemen deze beslissingsgrens tot een hogere orde te maken. Dit wordt mogelijk gemaakt door een *kernel* toe te passen. Hierbij worden de huidige features vervangen door nieuwe features. De kernel functie zoekt met deze nieuwe onbekende features een mogelijkheid om de data op te splitsen. Figuur 30.a toont de opgesplitste data met de originele features door middel van de kernel functie. In figuur 30.b is deze kernel functie bepaald door nieuwe features te zoeken en de data te splitsen van elkaar in een hogere dimensie [3].



Figuur 30: SVM met kernel functie [3]

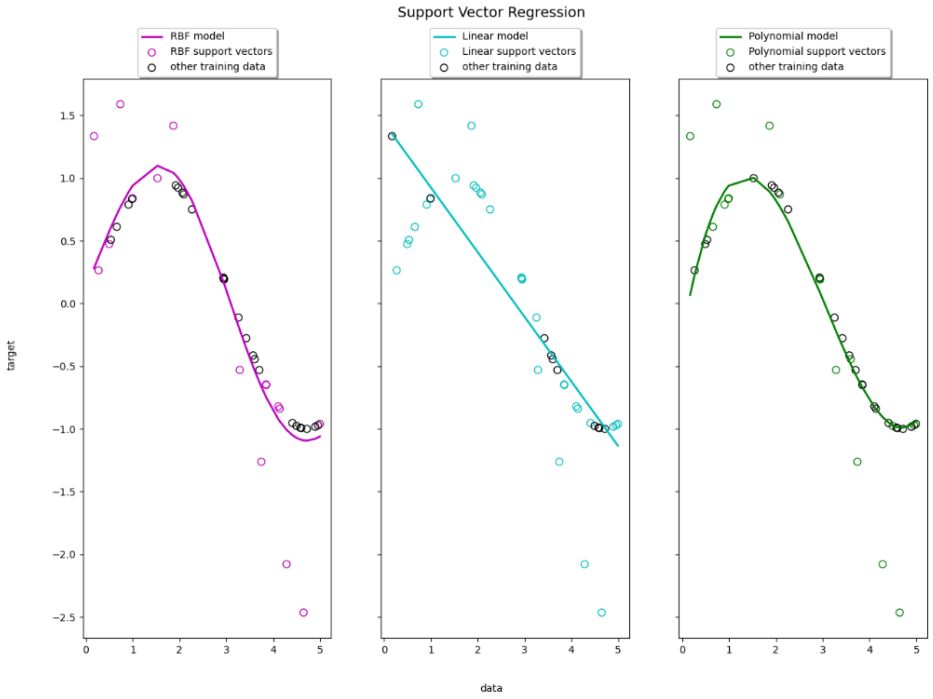
2.2.5.2 SVR

Het werkingsprincipe van SVR is gelijkaardig aan de SVC-werking. Bij de SVC worden de klassen via de hyperplane van elkaar onderscheiden en met de SVR wordt er via de hyperplane een regressielijn gevormd die het best bij de trainingsdata past. De marge met dit model wordt aangeduid met de ε parameter. Alle waarden die afwijken van de vooropgestelde marge tellen mee voor de kostfunctie. Het doel van de SVR is om een functie te ontwikkelen waarbij de waarden zo veel mogelijk tussen de marges +ε en -ε vallen. Figuur 31 geeft een voorbeeld weer van een hyperplane en zijn marges. De kost wordt hier aangeduid met ξ [30].



Figuur 31: SVR met marge en kostfunctie [30]

Het is ook mogelijk dat er complexere regressielijnen gevormd kunnen worden van hogere orde opnieuw met behulp van kernels (Figuur 32).

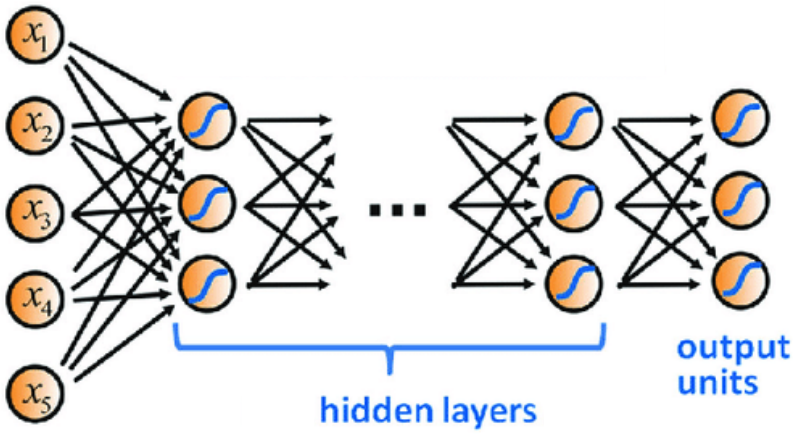


Figuur 32: SVR met kernel functie [31]

2.2.6 Artificieel neuraal netwerk

ANN (Artificiële neurale netwerken) is een veel gebruikte technologie om complexe problemen op te lossen. Zoals aangehaald in 2.1, is deze technologie ontstaan door het nabootsen van de neuronen van het menselijk brein. Er bestaan verschillende methodes om neurale netwerken toe te passen maar in deze masterproef is de focus gezet op de MLP-methode.

De basis van deze neurale netwerken kunnen opgesplitst worden in 3 delen volgens [32]. Namelijk de *input layer*, *hidden layer* en *output layer*. Al deze lagen bestaan uit een gekozen aantal neuronen. De input layer is de ingaande neuronen/features welke meegegeven worden aan het model. De hidden layers zijn verschillende neuronen waar het neuraal netwerk berekeningen mee uitvoert. Er wordt over een MLP-model gesproken als het model bestaat uit meerdere hidden layers. De output layer is het aantal waarden/neuronen dat het model moest voorspellen. Figuur 33 geeft de voorstelling weer van een MLP neuraal netwerk mee met de 3 verschillende soorten lagen.



Figuur 33: Voorstelling MLP neuraal netwerk [33]

Elke neuron vertrekkend vanaf de hidden layer neemt 3 soorten parameters op waarmee deze de berekeningen maakt om de volgende laag te voorspellen. Dit zijn de waarden, gewichten en bias van de vorige laag [34].

2.2.6.1 Kostfunctie

De kostfunctie voor het MLP-algoritme wordt voorgesteld in [35]:

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + \left(1 - y_k^{(i)}\right) \log\left(1 - (h_\Theta(x^{(i)}))_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{a=1}^{s_l}\sum_{b=1}^{s_{l+1}} \left(\Theta_{b,a}^{(l)}\right)^2$$

Hierbij is

- i gelijk aan een van de training samples m.
- k gelijk aan een van de K output neuronen.
- l gelijk aan een van de lagen L-1. L is het aantal lagen maar voor l wordt de output laag niet meegeteld.
- s_l gelijk aan het aantal neuronen in de laag l exclusief de bias neuron.
- a gelijk aan een van de neuronen van s_l in laag l.
- b gelijk aan een van de neuronen van s_{l+1} in laag l.
- Θ gelijk aan de matrix van de gewichten voor de functie op te stellen van de huidige laag l naar de laag l+1

Met deze kostfunctie wordt net zoals bij de lineaire regressie de kost van het model berekend bij een bepaalde voorspelling. De regularisatie term is ook al in deze functie toegevoegd als laatste term. Het opzet is om deze kost terug zo klein mogelijk te maken om betere voorspellingen te maken. Om deze te minimaliseren kan er terug het gradient descent algoritme worden toegepast tot de fout convergeert maar nu met een aangepaste formule voor het MLP-netwerk [35]:

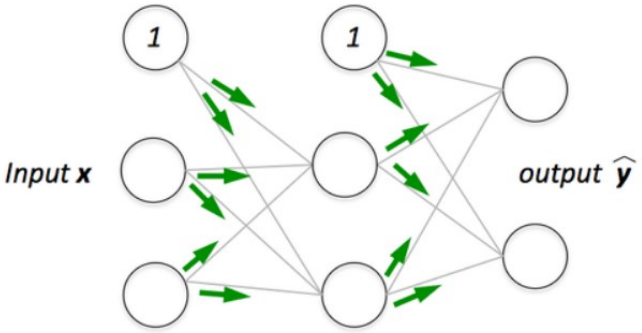
$$\Theta_{b,a}^{(l)} = \Theta_{b,a}^{(l)} - \alpha \frac{\delta}{\delta \Theta_{b,a}^{(l)}} J(\Theta)$$

Hierbij is

- a gelijk aan een van de neuronen van s_l in laag l.
- b gelijk aan een van de neuronen van s_{l+1} in laag l.
- l gelijk aan een van de lagen L-1. De output laag wordt niet meegeteld voor de matrix van gewichten Θ. Dit komt omdat dit de laatste laag is en geen gewicht heeft naar de volgende laag.
- α gelijk aan de leersnelheid.

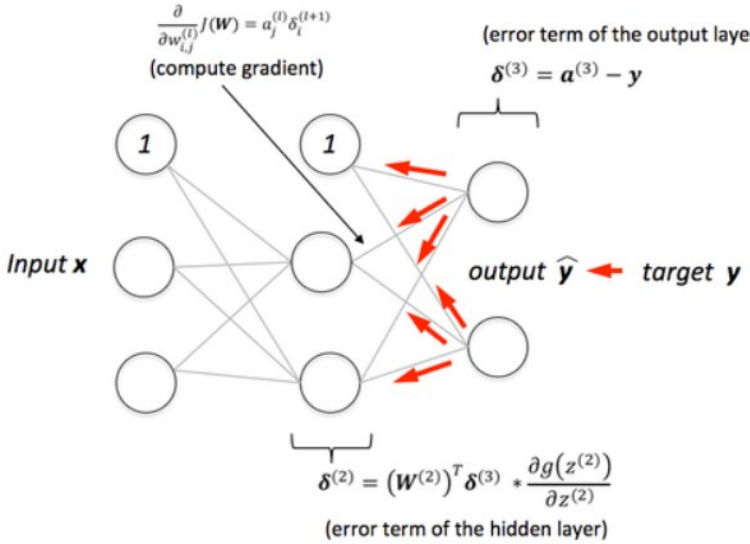
2.2.6.2 Forward vs backward propagation

Met *forward propagation* worden er voorspellingen gemaakt met een neuraal netwerk. Het model voert de wiskundige berekeningen uit vertrekkend vanaf de input layer waarna deze doorstroomt naar de hidden layer en eindigt bij de output layer. Hier vertrekt het model vanaf de ingang en eindigt deze bij de uitgang ook een voorwaarts systeem genoemd [32]. Figuur 34 geeft een visuele voorstelling van forward propagation.



Figuur 34: Forward propagation [36]

Na de forward propagation wordt het model achterwaarts uitgevoerd ook wel *backward propagation* genoemd. Via deze methode wordt er in de tegenovergestelde manier tewerk gegaan. Er wordt per laag (input laag niet meegerekend) de fout berekend op de effectieve waarden. Hiermee wordt het model gecorrigeerd waarna er opnieuw een forward propagation wordt uitgevoerd met een nieuwe voorspelling [35]. Figuur 35 visualiseert de backward propagation methode.



Figuur 35: Backward propagation [36]

2.2.6.3 Workflow

Om een MLP-netwerk op te stellen zodat dit kan leren kunnen er volgende stappen gevolgd worden [32].

1. Inkomende data aan het model meegeven.
2. Initialiseer de gewichten van de lagen willekeurig.
3. Forward propagation uitvoeren om de hypothese te verkrijgen voor de features.
4. De voorspelde waarden worden vergeleken met de labels.
5. De fout tussen de labels en de voorspelde waarde wordt gebruikt om de gewichten het model aan te passen.
6. Deze gewichten worden via backward propagation aangepast doorheen het model voor elke neuron.

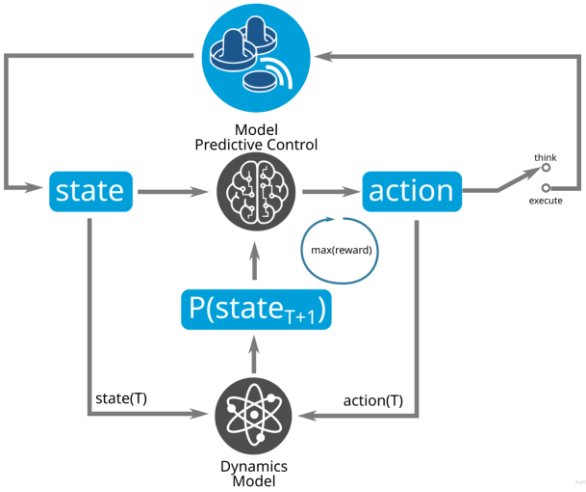
Deze 6 stappen blijven herhaald tot de kostfunctie convergeert.

2.3 Applicaties met robot-airhockeytafel

In deze paragraaf worden de bevindingen verzameld van andere onderzoeken rond een gelijkaardige robot-airhockeytafel probleemstelling.

2.3.1 Reinforcement learning

In [37] wordt er door middel van neurale netwerken een reinforcement learning model (zie paragraaf 2.1.2) opgesteld op een robot-airhockeytafel. Deze tafel wordt d.m.v. de reinforcement learning techniek Q-learning getraind en voorspeld. Met [38] worden er verschillende cases voorgesteld hoe de agent via reinforcement learning kan leren. Zo wordt het model beloond als de agent scoort of als deze een botsing detecteert tussen de mover van de robot en de puck. Het model wordt op zijn beurt bestraft als er gescoord wordt in de goal van de robot of als de puck stilstaat. Figuur 36 toont een schema hoe het reinforcement model de voorspelling maakt. Hierbij vertrekt het model van een huidige status, het model maakt hierop een voorspelling en geeft een actie mee aan de robot.



Figuur 36: Robot-airhockeytafel met reinforcement learning voorspelling [38]

Na de literatuurstudie is in de masterproef ervoor gekozen om niet verder te gaan met reinforcement learning. De implementatie hiervan i.c.m. een Beckhoff controller zou meer tijd vereisen dan het opgelegde tijdsframe. Enkele uitdagingen om reinforcement learning in een toekomstig onderzoek met TwinCAT 3 te implementeren zijn op volgende pagina vernoemd.

- De camera's zijn gekalibreerd en geconfigureerd via de TwinCAT 3 software. Hier moet er een methode ontwikkeld worden om deze buiten de real-time omgeving te gebruiken. Dit met dezelfde configuratie om ze in een andere programmeertaal te trainen. Een bekende programmeertaal voor reinforcement learning is Python.
- Een reinforcement learning model van een robot-airhockey kost veel iteraties aan traingingstijd. Het zou veel te intensief zijn om elk traject manueel te trainen via reinforcement learning. Er bestaan open-source modellen zoals [39] met een simulatieomgeving van een airhockeytafel maar deze moet dan ook aangepast worden aan de fysieke eigenschappen van de in deze masterproef besproken airhockeytafel.
- Het implementeren van een reinforcement model in TwinCAT 3 staat op dit moment nog niet beschreven op de website van Beckhoff Automation. De mogelijkheid van het te implementeren reinforcement model in de real-time omgeving zou hiervoor nog verder onderzocht moeten worden.

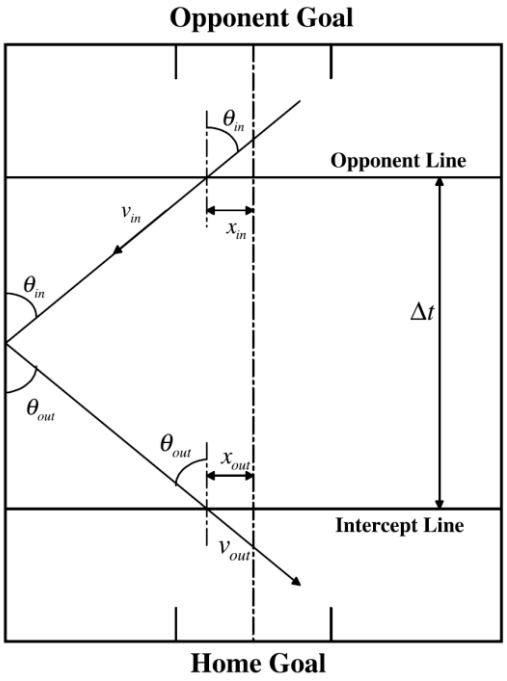
2.3.2 Classificatie aanpak

Bij classificatie zijn er ook verschillende modellen rond dit topic te vinden. In [40] wordt er voorgesteld om via een neuraal netwerk classificatie voorspellingen uit te voeren waarbij de robot de puck moet onderscheppen. De x-posities van de robot worden in de vooropgestelde klassen in de x-as opgesplitst zodat alle posities van de robot gemaakt kunnen worden. Het model is dan getraind om te voorspellen in welke klasse de puck bij de robot terechtkomt. Na verder onderzoek is er besloten om het volledige traject te bepalen via ML als een regressieprobleem i.p.v. een classificatieprobleem.

Een ander type classificatie probleem kan zijn het bepalen hoe de robot moet reageren afhankelijk van de puck zijn traject. Bij [41] zijn er verschillende technieken aangeduid die niet als een classificatie probleem worden beschouwd in dit onderzoek maar die wel als basis voor dit probleem gebruikt kunnen worden. De robot bepaalt afhankelijk van het traject van de puck wat hij moet doen: verdedigen, aanvallen of niets doen. Wegens tijdsgebrek zijn de specifieke aanvalstechnieken nog niet geïmplementeerd in deze masterproef. Dit wordt in een toekomstig onderzoek verder opgenomen.

2.3.3 MLP-regressie aanpak

Een ander onderzoek [42] gebruikt MLP-modellen om regressie voorspellingen in de toekomst te maken. Hierbij wordt er een figuurlijke lijn getrokken zowel aan de tegenstander- als aan de robotzijde. Deze lijnen hebben een vaste y-waarde. Het model maakt een voorspelling waar de puck bij de robotlijn uitkomt als de puck zich bij de tegenstanderlijn bevindt. Als input neemt het model drie parameters van de puck op bij de tegenstanderlijn: de x-positie, de snelheid en de ingaande hoek. Hierna maakt het model een voorspelling van vier parameters van de puck op de robotlijn: de X-positie, de snelheid, de ingaande hoek alsook het tijdstip tussen de tegenstander- en de robotlijn. Figuur 37 geeft de opstelling van dit onderzoek weer. Voor elk van deze labels is er een apart MLP-model aangemaakt, zo kunnen deze apart verbeterd worden. De modellen worden volgens de supervised learning methode getraind op 3000 samples. Hiervoor zijn deze trajecten eerst manueel uitgevoerd op de tafel waarbij de features van de tegenstanderlijn gekoppeld zijn aan de labels van de robotlijn. Dit waren trajecten zonder botsing (1500 samples), met één botsing (1500 samples) of met gemende data (750 samples met één botsing en 750 samples zonder botsing). Hun resultaten zijn hier getraind rekening houdend met de MSE-evaluatiemetriek. Het beste resultaat kan bekomen worden als er geen botsing is, het slechtste resultaat wordt verkregen bij de gemende data. De MSE-evaluatiewaarde voor de onderscheppingslocatie is respectievelijk 0,2830m en 2,8184m.



Figuur 37: Robot-airhockey MLP-model volgens [42]

Tijdens deze masterproef is er voornamelijk inspiratie genomen uit dit onderzoek. Het nadeel van dit onderzoek is dat het model niet getraind geweest is op data waarbij de puck meer als een keer botst tegen de rand. In de robot-airhockeytafel van deze masterproef is er gekozen om het volledig toekomstig traject van de puck te voorspellen vanaf een bepaald aantal frames in de toekomst. Als de puck meerdere keren botst tegen de rand gaat deze hierop ook een voorspelling kunnen maken. Parameters zoals de x-positie, de y-positie, de snelheid en de hoek zijn ook opgenomen in het onderzoek. De figuurlijke lijnen worden niet gebruikt voor het voorspellen en het trainen van het model maar worden op een andere manier toegepast om de robot aan te sturen, zie paragraaf 5.2.3.

3 TwinCAT 3 data verzameling

Na het onderzoek kan er begonnen worden met de volgende stap van de ML-workflow, het verzamelen van data. Deze wordt verzameld door gebruik te maken van camera's i.c.m. TwinCAT 3, de automatseringssoftware van Beckhoff Automation. Dit hoofdstuk geeft een introductie van TwinCAT 3 en overloopt verder de opzet om de data te verzamelen.

3.1 TwinCAT 3 opzet

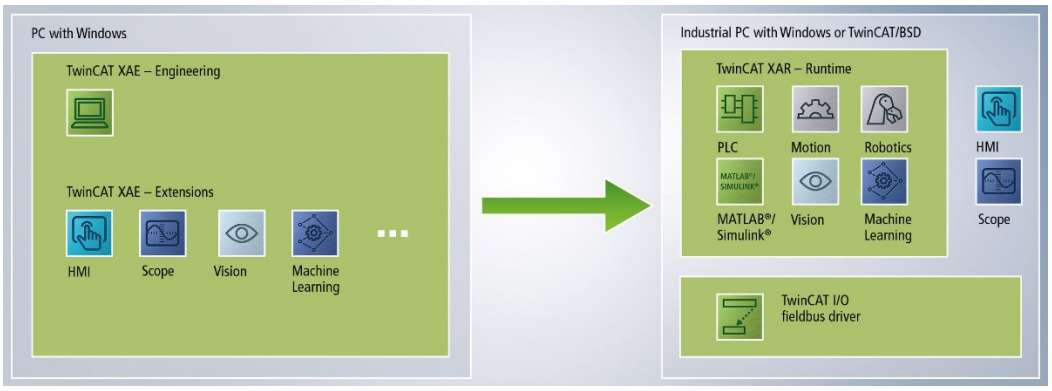
De ideologie van Beckhoff Automation is om een PC-gebaseerde oplossing te bieden voor automatiseringsproblemen. TwinCAT 3 staat hier centraal, dit is de software waarmee de PC-controllers van Beckhoff Automation geprogrammeerd en geconfigureerd kunnen worden. TwinCAT 3 staat voor "The Windows Control and Automation Technology" waarbij 3 wijst op de derde generatie van de software. Met TwinCAT 3 is het mogelijk om verschillende industriële technieken te combineren in een softwarepakket. Hiermee is het mogelijk om een moderne HMI te ontwikkelen, visie applicaties te configureren, PLC-programma's te schrijven, motoren aan te sturen en meer.

3.1.1 Architectuur

De architectuur hiervan wordt de eXtended Automation Architecture (XAA) genoemd. Deze kan opgesplitst worden in twee pakketten:

1. eXtended Automation Engineering (XAE): De XAE is de ontwikkel/engineering omgeving van TwinCAT 3. Deze maakt het mogelijk om de applicatie te configureren en te programmeren. In Visual Studio wordt de XAE geïntegreerd. Dit om met een geïntegreerde ontwikkelomgeving (IDE) oplossingen met TwinCAT 3 en andere toepassingen te ontwerpen.
2. exTended Automation Runtime (XAR): De XAR is de runtime van TwinCAT 3. Deze is geschikt om de programma's uit te voeren via de real-time omgeving. Bijkomend aan het uitvoeren van real-time applicaties zoals de PLC, visie, ML, … heeft elk XAR-systeem ook zijn eigen besturingssysteem. Dit kan Windows of TwinCAT/BSD zijn de gebruiker heeft ook de mogelijkheid om zijn eigen applicatie hierop uit te voeren [43].

Met de XAE-omgeving wordt standaard de XAR-omgeving mee geïnstalleerd. Het is mogelijk om een Beckhoff IPC op Windows zowel als runtime- als ontwikkelomgeving te gebruiken. Een andere optie is om een externe pc te gebruiken als engineering omgeving en de Beckhoff PLC met de runtime. Om de communicatie tussen de XAE en XAR laag te maken wordt er het Automation Device Specification (ADS) protocol van TwinCAT gebruikt (Figuur 38).



Figuur 38: TwinCAT 3 XAE en XAR [43]

3.1.2 Hardware

Voor elke TwinCAT 3 oplossing is er een bijbehorende Beckhoff controller vereist. Om al de verschillende processen waaruit de airhockeytafel bestaat, zoals visie, motion, ML, PLC, enz., goed te kunnen uitvoeren is er een krachtige controller nodig. Voor deze applicatie is er dan ook voor een embedded CX2072 controller gekozen. Dit is een 2,1 GHz controller met 12 cores die gemonteerd kan worden op een DIN-rail. 12 cores is het grootst aantal beschikbare cores dat afzonderlijk gebruikt kan worden voor verschillende doeleinden. Verder heeft de controller als besturingssysteem Windows 10 dat op een aparte core wordt uitgevoerd. Zo heeft elke camera zijn eigen core, de motion taak heeft zijn eigen core, …. Later met een HMI-uitbreiding en een cloud uitbreiding is de controller ook toekomstbestendig door zijn groot aantal cores.

In figuur 39 kan de huidige opstelling van de CX2072 in de robot-airhockeytafel getroffen worden. Links is de controller opgesteld en rechts is de AX8000 servo drive gemonteerd voor het aansturen van de motoren. Op de servo drive wordt er later nog dieper ingegaan bij onderdeel 5.2.

Figuur 39: CX2072 Beckhoff controller

Aan de linkerzijde van de CPU zijn er drie RJ45 kabels verbonden. Uiterst links aangeduid in het blauw is de poort die gebruikt wordt om de controller te programmeren. De twee kabels boven de DVI-poort aangeduid in het rood zijn twee Gigabit poorten die gebruikt worden voor de camera's.

3.2 Data verzamelen

In deze paragraaf wordt de workflow van de data verzameling van de robot-airhockeytafel aangetoond. Hierbij wordt er gestart met het mechanisch in orde brengen van de camera's. Als volgt worden ze gekalibreerd en is via de PLC met behulp van TwinCAT 3 de software geschreven om de puck te detecteren via de camera's. Als laatste is er een procedure opgesteld om met de PLC het traject van de puck te loggen zodat dit later gebruikt kan worden voor het trainen.

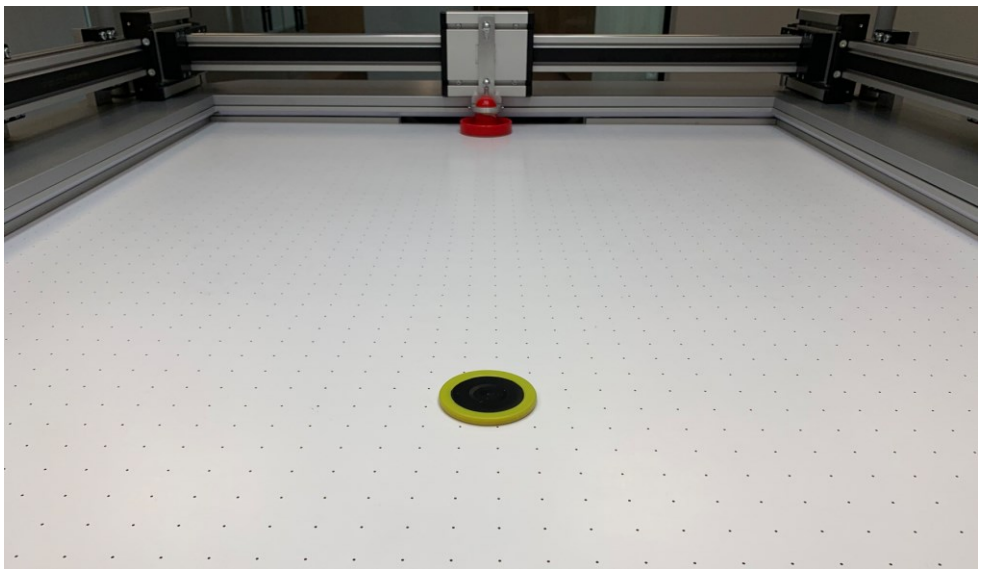
3.2.1 Visie opzet

Ter beschikking zijn er twee Mako G-030B camera's (Figuur 42). Deze camera's ondersteunen het "GiGE Vision" protocol. Dit is een industrieel protocol dat gebruikt wordt voor hoge performante camera's. Om de camera's met de TwinCAT Vision architectuur van Beckhoff Automation te verbinden is dit protocol noodzakelijk. Deze standaard mag niet verward worden met normale GiGE camera's welke niet compatibel zijn met TwinCAT 3 [44].

Enkele eigenschappen van deze camera's die later ook nog aan bod komen zijn hieronder terug te vinden [45]:

- De camera's kunnen 309 frames per seconde (FPS) verwerken.
- De pixel resolutie van deze camera's bestaat respectievelijk uit 644 pixels in de hoogte en 484 pixels in de breedte.
- De camera's kunnen enkel monochroom beelden verwerken. Dit betekent dat deze geen kleuren kunnen onderscheiden. Ze kunnen enkel de verschillende grijswaarden van objecten detecteren.
- De camera's hebben de mogelijkheid om gevoed te worden via de RJ45 kabel met PoE (Power over Ethernet).

Het speelveld van de airhockeytafel is wit en de aanwezige pucks zijn geel. De grijswaarden tussen beide is dus niet zo groot waardoor deze standaard niet gedetecteerd kunnen worden. Als oplossing is de middelste cirkel van de puck zwart gemaakt. Hierdoor is er een groter contrast tussen de grijswaarde van het speelveld en deze van de puck (Figuur 40).

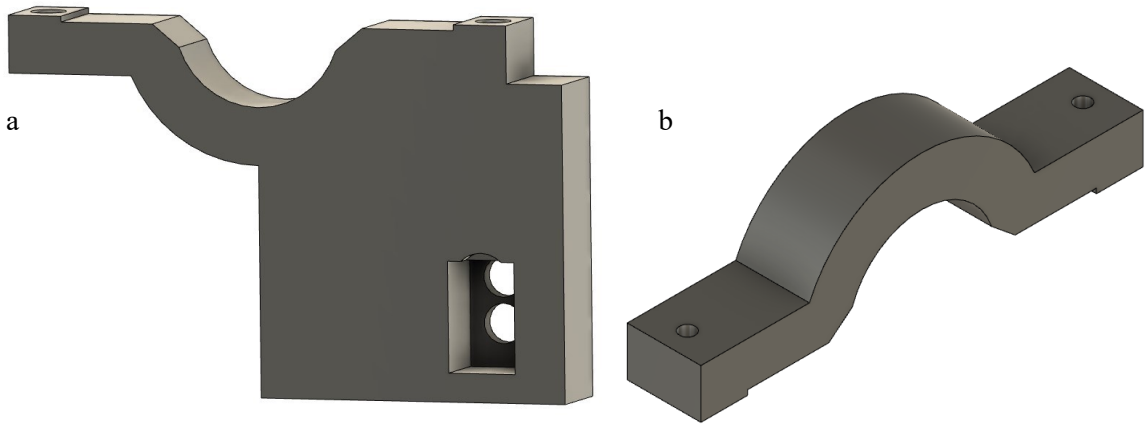


Figuur 40: Airhockeypuck met zwarte aanduiding op airhockeytafel

3.2.1.1 Mechanische opstelling

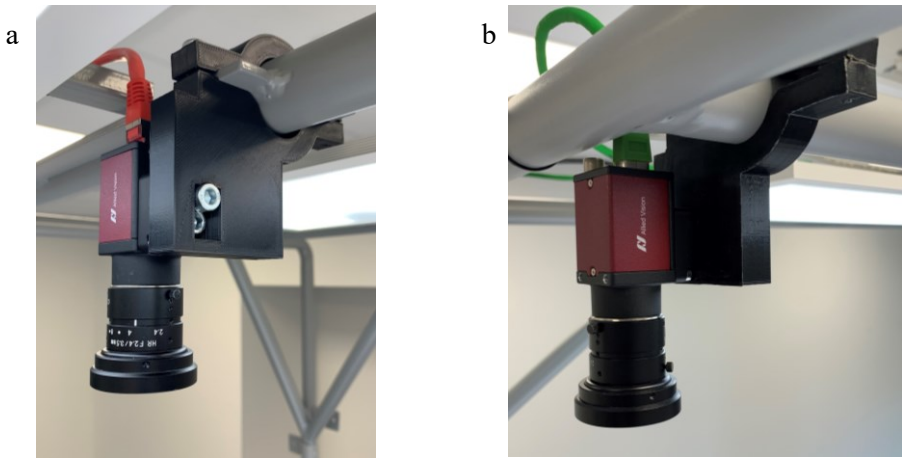
Bij aanvang van de masterproef waren de camera's van de airhockeytafel zoals bij 1.1 niet optimaal geplaatst. Ze waren bevestigd met een aluminiumplaat op het gelaste frame op een plaats waar ze het makkelijkst te monteren waren en niet op een plaats waar ze het grootst mogelijk gezichtsveld hadden. Om de optimale opstelling te bepalen werd er rekening gehouden met twee factoren. Als eerste is de positie van de robotcamera bepaald zodat deze nog juist de hoeken van het speelveld kan zien. Indien er een foute voorspelling gemaakt wordt waardoor de puck aan de robotzijde blijft liggen moet de robot de puck hierna kunnen wegspelen. Als tweede is de positie van de spelerscamera bepaald om ervoor te zorgen dat er één overlap is tussen beide camera's met minimaal de grootte van een puck. Deze overlap zorgt later voor een makkelijke kalibratie van de puckpositie.

De enige mogelijkheid die rekening kan houden met de twee factoren is om de camera's rond de stalen buis in het midden te monteren zodat de camera's nog steeds zicht hebben op de volledige breedte van het speelveld. Om ervoor te zorgen dat de camera's op deze cirkelvorm gemonteerd kunnen worden zijn er verschillende 3D-modellen getekend. Er is gekozen om twee bevestigingen te ontwikkelen voor elke camera. Een deel voor onder de buis waaraan de camera bevestigd wordt (Figuur 41a) en een tweede deel dat boven de buis komt (Figuur 41b). De twee delen worden aan elkaar gemonteerd en vormen een klembevestiging. Deze twee bevestigingen werden met behulp van een 3D-printer vervaardigd.



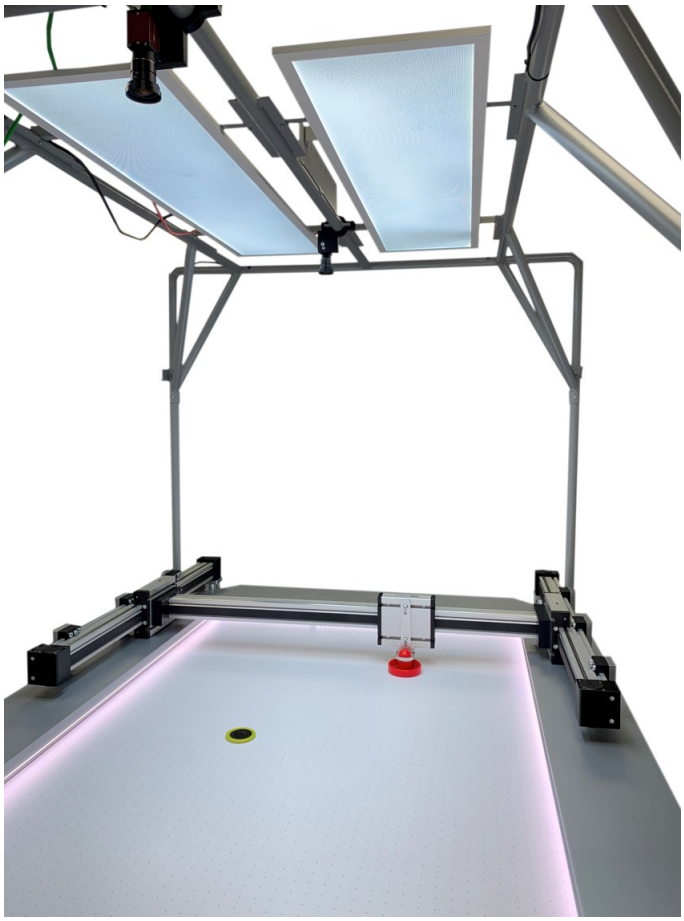
Figuur 41: 3D-modellen camera bevestiging

Het resultaat van de geplaatste camera's op het frame is terug te vinden op figuur 42.



Figuur 42: Geplaatste camera's met de 3D-geprinte klembevestiging voor de spelerscamera (a) en de robotcamera (b)

Figuur 43 geeft een totaal beeld weer van de robot-airhockeytafel met de nieuwe opstelling van de camera's. Het speelveld van de airhockeytafel heeft een breedte van 1075mm en een lengte van 2200mm.



Figuur 43: Robot-airhockeytafel met nieuwe camerasteunen

3.2.1.2 Configuratie

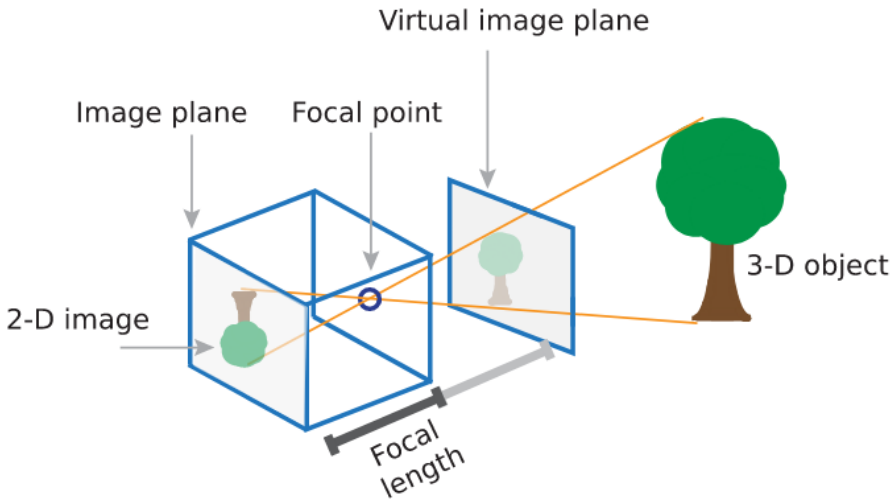
Volgende parameters zijn gewijzigd in de configuratie van de camera vooraleer er gestart werd met de kalibratie:

- De FPS worden ingesteld op 303FPS i.p.v. 309FPS zodat de PLC-cyclustijd later voor de visie taak als een makkelijk veelvoud hiervan ingesteld kan worden. Om deze FPS met de camera's te bereiken moest het maximaal aantal bytes (124.000.000bytes ofwel 0,992gigabits) ingesteld worden dewelke gestreamd worden door de camera's. Beide camera's waren verbonden via één PoE switch naar één Gigabit poort op de CX2072 Beckhoff PLC. Dit maakte het niet mogelijk om deze bytestream te ondersteunen waardoor er een tweede PoE switch is bijgeplaatst zodat elke camera zijn eigen Gigabit poort ter beschikking heeft op de controller.
- ROI (Region of interest) is de regio op het beeld van de camera waar er interesse aan gehecht wordt. De regio voor deze camera's kunnen maximaal een grootte hebben van 644 pixels in de hoogte en 484 pixels in de breedte, zie 3.2.1. Deze wordt zodanig geconfigureerd dat enkel het speelveld van de tafel te zien is. Indien dit groter wordt ingesteld is er meer bandbreedte nodig en wordt de puck gezocht op een groter beeld dan nodig is wat voor extra niet bruikbare PLC-executietijd zorgt.

3.2.1.3 Kalibratie

Na de mechanische opzet en het initialiseren van de camera's in TwinCAT 3 moeten deze gekalibreerd worden. Een niet-gekalibreerde camera heeft verschillende vervormingen en heeft niet de mogelijkheid om de X en Y positie in het beeld te bepalen.

De camera's gebruiken het pinhole principe (Figuur 44). Het 3D-object in de wereld wordt door lichtstralen opgenomen en voorgesteld als een 2D-beeld. Dit beeld komt aan op het brandpunt/uiteinde van de lens. Als volgt wordt het beeld verder geprojecteerd op de module binnenin de camera die dit licht opvangt. De afstand tussen het brandpunt en de lichtdetectiemodule wordt de brandpuntafstand genoemd. Door de lichtinval wordt het beeld verticaal geïnverteerd [46].



Figuur 44: Pinhole camera principe [46]

De omvorming van de 3D-omgeving naar de 2D-omgeving wordt beschreven met een cameramatrix. Het type lens dat er gebruikt wordt brengt een bepaalde vervorming mee in het beeld. Deze vervormingen kunnen opgesplitst worden in radiale en tangentiële vervormingen. Bij radiale vervormingen buigen de lichtstralen aan de randen van de lens meer dan het optisch centrum (Figuur 45a). Tangentiële vervorming daarentegen treedt op wanneer de lens van de camera niet perfect parallel staat met het beeldvlak van de camera (Figuur 45b) [46].

Figuur 45: Radiale versus tangentiële vervorming [46]

Deze vervormingen worden berekend als vervormingscoëfficiënten waarna deze worden toegepast op de cameramatrix. Dit proces wordt het berekenen van de intrinsieke matrix genoemd. Als de radiale en tangentiële fouten zo goed mogelijk gecompenseerd zijn worden de rotatie- en de translatievector berekend. De rotatiematrix beschrijft de rotatie van de camera t.o.v. een wereldcoördinatensysteem. Met de translatie vector wordt de translatie van de camera beschreven t.o.v. het wereldcoördinatensysteem. Deze twee vormen de extrinsieke berekening van de camera en maken het mogelijk om een 3D-punt (X,Y,Z) in wereldcoördinaten om te vormen naar een 2D-punt (X,Y) in beeldcoördinaten.

De gecombineerde formule van de cameramatrix kan als volgt worden opgesteld: P = K * [R*t]. Waarbij de termen van de formule voor volgende begrippen staan [46]:

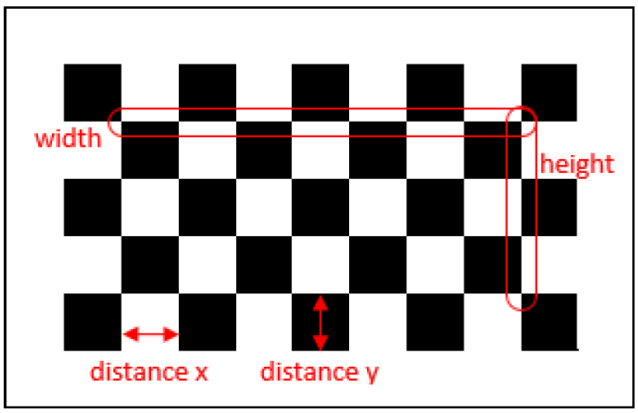
- P = cameramatrix;
- K = intrinsieke matrix aangepast met de vervormingscoëfficiënten;
- R = extrinsieke rotatiematrix;
- T = extrinsieke translatiematrix

Bij TwinCAT Vision kan de kalibratie uitgevoerd worden met een configuratietool. De stappen om deze kalibratie uit te voeren kunnen opgesplitst worden in 4 stappen.

3.2.1.3.a Kalibratiepatroon definiëren

In de eerste stap moet het kalibratiepatroon gedefinieerd worden. Zowel voor de intrinsieke als de extrinsieke parameters moeten met een patroon door de camera opgemeten worden. Een bekend patroon dat ook gebruikt is voor deze camera's is het schaakbordpatroon (Figuur 46). Hierbij moeten volgende parameters worden meegegeven aan de kalibratie tool [44]:

- Breedte: Het aantal kruisingen tussen de witte en zwarte velden in de breedte.
- Hoogte: Het aantal kruisingen tussen de witte en zwarte velden in de hoogte.
- Afstand x: De afstand van elk veld volgens de x-as.
- Afstand y: De afstand van elk veld volgens de y-as.



Figuur 46: Schaakbordpatroon voor kalibratie [44]

Dit patroon wordt op een bepaalde manier gemaakt zodanig dat er geen fouten in het kalibratiepatroon zelf zitten. De hoogte van dit patroon is ook belangrijk. Deze moet even hoog zijn als het te meten object, in dit geval een airhockeypuck. Indien hier geen rekening mee gehouden wordt zullen de berekeningen niet overal kloppen.

3.2.1.3.b Kalibratie foto's maken

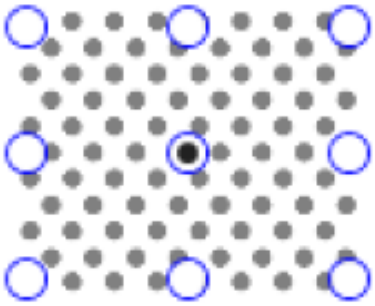
Bij de tweede stap moeten er verschillende foto's gemaakt worden met de camera waarbij het kalibratiepatroon aanwezig is op de airhockeytafel. Het kalibratiepatroon mag hierbij over heel het speelveld verplaatst worden dus overal waar de puck tijdens het spel kan komen te liggen. Best kan het patroon zo veel mogelijk in de hoeken en randen van de tafel geplaatst worden om de intrinsieke matrix zo optimaal mogelijk te bepalen. De foto's kunnen live gemaakt worden of dit kan met behulp van vooraf opgeslagen afbeeldingen [44].

3.2.1.3.c Intrinsieke parameters

Als volgt worden de intrinsieke parameters berekend, dit zijn de cameramatrix en de vervormingscoëfficiënten. TwinCAT 3 geeft met de projectiefout aan hoe goed de parameters berekend zijn a.d.h.v. de foto's. Een goede waarde ligt tussen 0 en 1, waarbij 0 een perfecte kalibratie aangeeft. Een waarde hierbuiten betekent dat er een fout is met de berekening en de foto's nagekeken moeten worden.

3.2.1.3.d Extrinsieke parameters

Als laatste zijn de extrinsieke parameters bepaald. Hier wordt er één van de gemaakte foto's genomen waarop het XY-assenstelsel toegepast wordt. Waar het nulpunt komt te liggen in deze foto kan gekozen worden uit de hoeken of het middelpunt van het patroon (Figuur 47).



Figuur 47: Extrinsieke oorsprong in het midden [44]

Nadat alle parameters voor de kalibratie zijn bepaald worden deze opgeslagen in het geheugen van de camera.

3.2.1.4 Puck detectie programmatie

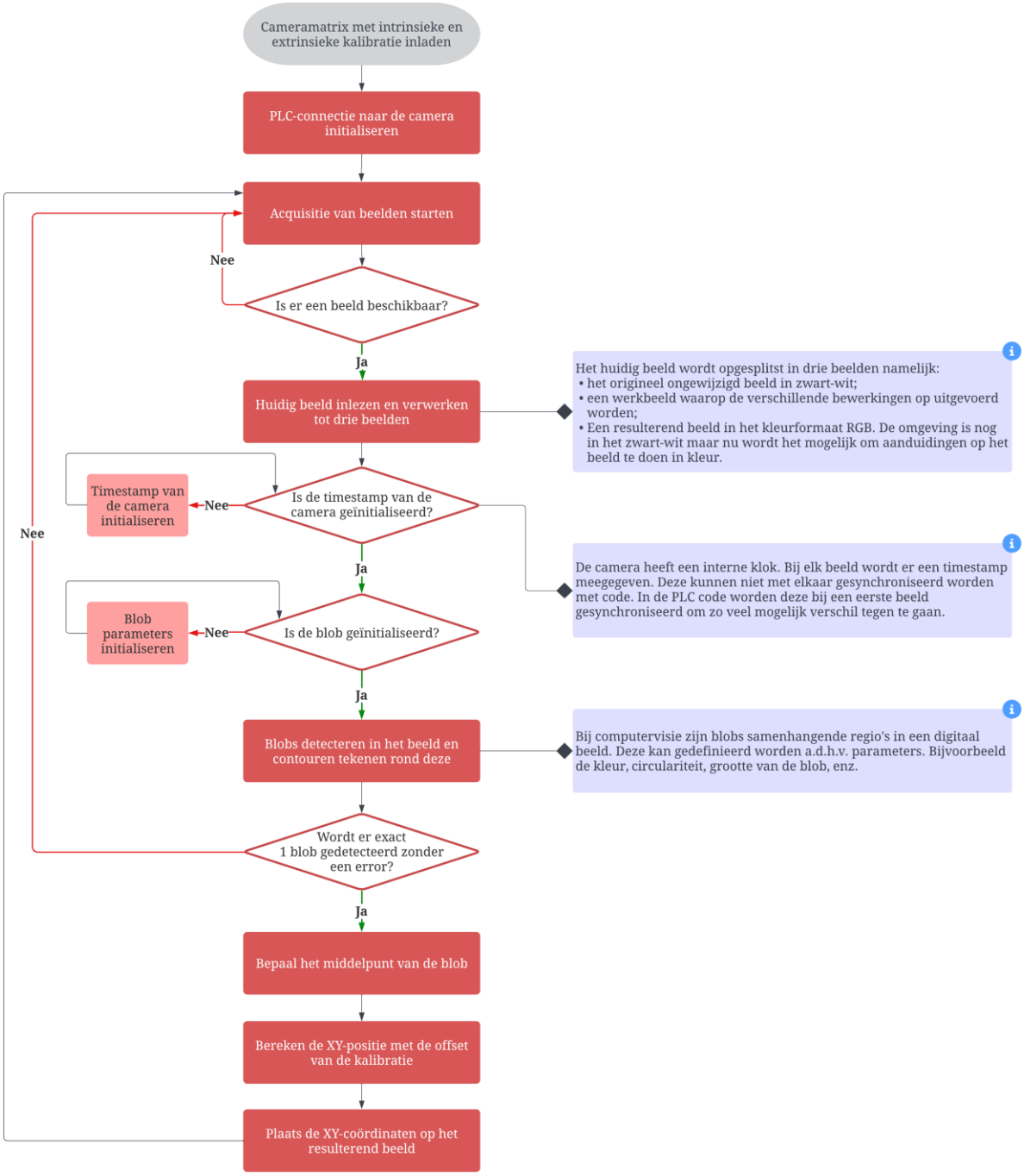
De kalibratie van de camera's is voltooid waardoor er begonnen kan worden met de PLC-programmatie om de puck te detecteren.

3.2.1.4.a Visie PLC-taak

Vooraleer er een code wordt geschreven zijn de PLC-taken ingesteld die de visiecode moet uitvoeren. De camera's werken op 303 FPS wat overeenkomt met 3,3 milliseconden per frame. Volgens het Nyquist-Shannon sample theorema in [47] moet de sample tijd waarop een signaal wordt gesampled minstens tweemaal de frequentietijd zijn van het signaal. In dit geval moet de sampletijd minstens 1,65 ms zijn. Met de CX2072 controller is het mogelijk om de base time van een taak tot 50 µs nauwkeurig te verkrijgen. Voor elke camera is er een aparte taak aangemaakt met één cyclustijd van 1,65 ms. Met de huidige instellingen gebruiken respectievelijk de robot- en speler -taak gemiddeld 370 µs als de puck gedetecteerd wordt en 250 µs als er geen puck wordt gedetecteerd.

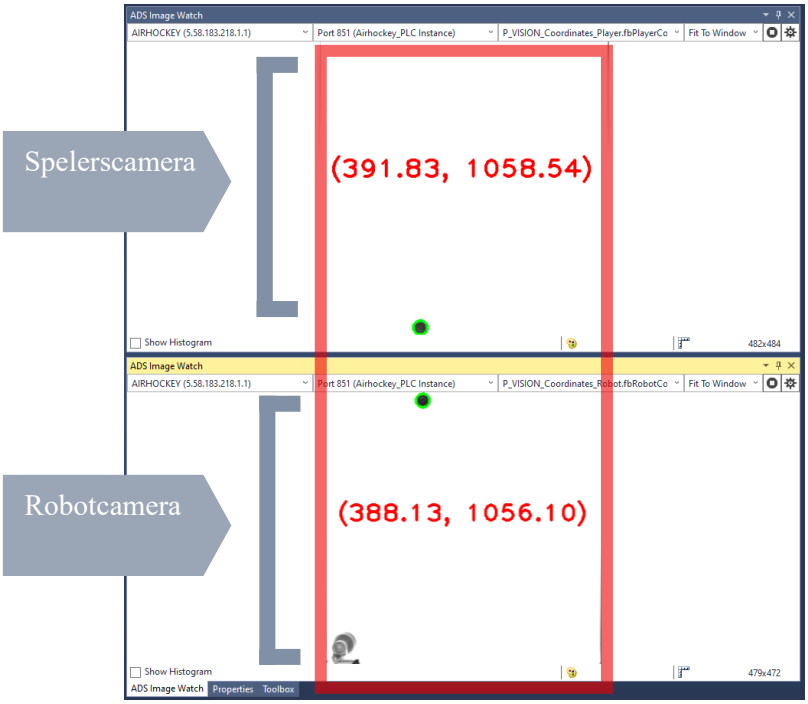
3.2.1.4.b Detectieprocedure

Zowel de camera bij de robot als bij de speler doorlopen dezelfde procedure om de puck te detecteren. In volgende flowchart op figuur 48 wordt het principe van de detectieprocedure verduidelijkt.



Figuur 48: Flowchart werking puckdetectie

Als eindresultaat wordt er voor elke PLC-cyclus de X- en Y-positie in het 2D beeld bepaald voor elke camera. Daarna worden deze posities ook visueel weergegeven op het resulterend beeld. Figuur 49 toont dit resulterend beeld waarbij de puck zich bevindt in de overlap tussen beide camera's. De berekende X- en Y-positie zijn hier aangeduid in millimeter voor elke camera. Hier kan opgemerkt worden dat de berekende posities van beide camera's niet perfect overeenkomen, deze verschillen enkele millimeters. Bij een bepaalde punt op deze overlaplijn zijn deze perfect op elkaar afgestemd met een offset. Maar doordat beide camera's niet exact dezelfde kalibratie hebben verschillen deze waarden altijd van elkaar. Dit zorgt door een fout waarbij het model ook zijn best moet doen om deze te leren.



Figuur 49 Puckdetectie in TwinCAT 3 op overlappositie

3.2.2 Puck traject loggen

De puck zijn X- en Y-positie kunnen met de PLC gedetecteerd worden. Als volgt wordt er een programma geschreven om deze positie en andere features van de puck te loggen. Dit moet een formaat zijn wat later ingelezen kan worden in de programmeertaal Python om het model te trainen. Er is gekozen om via PLC de data te verwerken in een CSV-bestand.

3.2.2.1 CSV-logbestand programmatie

Een CSV-bestand bestaat uit *comma-seperated values*, dit is een bestand waarbij de waarden gescheiden staan van elkaar door komma's of andere tekens. De waarden staan in een tabelvorm opgesplitst met rijen en kolommen. Het is een lichtgewicht bestandsformaat wat eenvoudig te implementeren is. Om te beginnen zijn de features die gelogd zullen worden gedefinieerd. Elke feature vormt later een kolom in het CSV-bestand en elke frame stelt een nieuwe rij voor. Na enkele iteraties aan testen zijn volgende features gekozen:

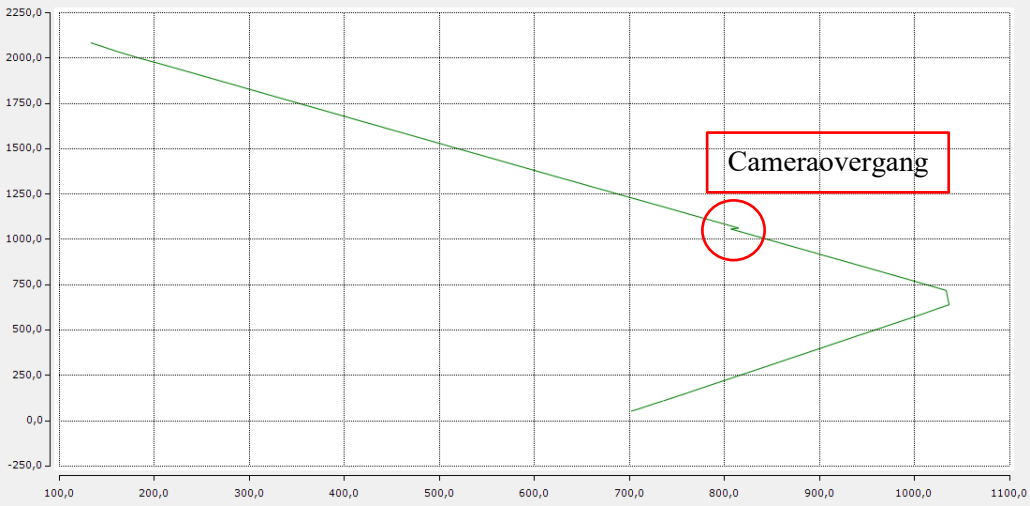
1. X- en Y-positie van de puck in mm.
2. Tijdsverschil tussen de huidige en vorige frame.
3. Snelheid V in mm/s. Deze wordt berekend door het afstandsverschil van het huidige frame en het vorige frame te delen door het tijdsverschil tussen deze frames.
4. Richtingscoëfficiënt, deze geeft de richting aan van de puck.
5. Hoek, deze geeft de richting van de puck aan in graden welke is afgeleid van de richtingscoëfficiënt.
6. Inverse X- en Y-positie. Dit is de X-positie tot aan de rechterkant van de airhockeytafel en de Y-positie tot aan de bovenkant van de airhockeytafel.

Rond deze features is volgende logprocedure geprogrammeerd om een CSV-bestand te creëren dat de data van één traject logt:

1. Eerst wordt bij elke cyclus gecontroleerd of er een verschil wordt vastgesteld in de huidige en de vorige positie van de puck. De puck zijn X- en Y-posities worden gefilterd op 3mm, dus enkel als er een verandering van 3mm is wordt deze gedetecteerd. Bij de visiedetectie verschilt de positie van de puck bij elk frame ook al ligt deze stil. Ze zijn nooit perfect hetzelfde.
2. Als volgt wordt er geobserveerd wanneer de pucksnelheid groter dan 3000mm/s wordt.
3. Vanaf dat er een deceleratie gedetecteerd wordt, start het loggen van de features naar een array in de PLC met maximaal 3000 rijen. Deze werkt volgens het FIFO (First In First Out) principe. Om een enkel traject van de puck te loggen zijn 3000 frames meer dan voldoende.
4. Bij het niet meer detecteren van de puck stopt het loggen naar deze array.
5. De array is dan gevuld met het aantal beschikbare waarden waarna het creëren van het CSV-bestand van dit traject wordt gestart. Hierbij wordt rij per rij gevuld met de data van de array.
6. Als laatste wordt de array terug leeggemaakt en wordt er terug vanaf stap 1 begonnen om het volgend traject te loggen.

3.2.2.2 Traject evaluatie

Er is de mogelijkheid om diverse trajecten op te stellen: verschillende snelheden, verschillende beginposities, verschillende aanvalshoeken en zo meer. In deze masterproef is er gefocust op data vertrekkend van de rechterkant van de speler waarbij de puck aan de linkerkant van de airhockeytafel botst. Dit met verschillende snelheden en hoeken. Figuur 50 toont een voorbeeld van een traject dat geplot is met de scope functionaliteit in TwinCAT 3. Deze grafiek toont op de X-as de positie van de puck in de breedte van de tafel en op de Y-as de lengte van de tafel. Bij dit traject vertrekt de puck vanaf de speler en eindigt deze in de goal van de robot. Hierbij maakt de puck één botsing tegen de linkerrand van de tafel vertrekkende vanuit het perspectief van de speler. Tabel 2 toont de eerste vijf frames aan data voor dit traject in het CSV-bestand.



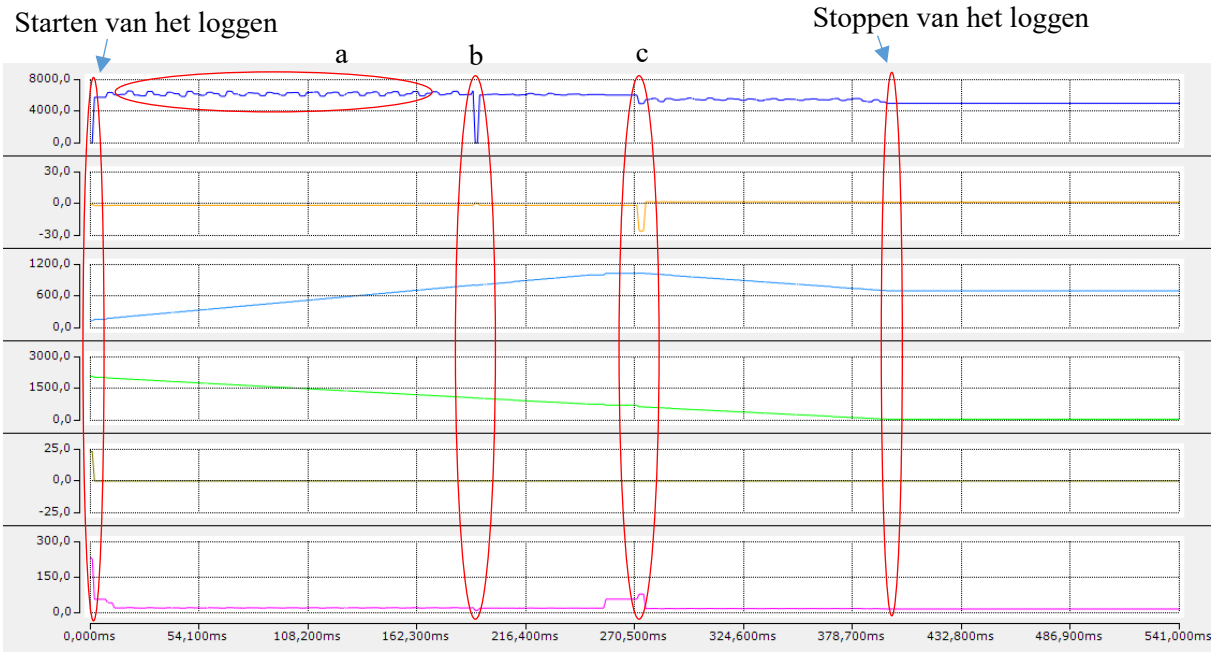
Figuur 50: XY-grafiek traject puckpositie met één botsing

Tabel 2: Logging van het traject met één botsing in een CSV-bestand voor de eerste vijf frames

Frame	Tijdsverschil [s]	X [mm]	Y [mm]	V [mm/s]	Richtingscoëfficiënt []	Hoek [°]	Inverse X [mm]	Inverse Y [mm]
1	3,30E-03	195,42	1982,84	6140,90	-1,41	-54,72	879,58	217,16
2	3,30E-03	207,10	1966,32	6567,41	-1,51	-56,50	867,90	233,68
3	3,29E-03	219,05	1948,28	6053,91	-1,46	-55,51	855,95	251,72
4	3,30E-03	230,34	1931,83	5959,64	-1,48	-55,87	844,66	268,17
5	3,30E-03	241,36	1915,58	6510,46	-1,51	-56,55	833,64	284,42

Het overgangspunt van de spelerscamera naar de robotcamera en vice versa vindt plaats tussen Y = 1050mm en Y = 1075mm. Dit punt kan ook duidelijk op figuur 50 vastgesteld worden doordat er op dat punt een verspringing in de X-waarden is. Het overgangseffect van de ene camera naar de andere camera brengt verschillende problemen met zich mee op de trainingsdata voor het ML-model. Om deze te verduidelijken is in figuur 51 een zes YT-grafiek geplot waarbij zes verschillende waarden van ditzelfde traject zijn gelogd. Een YT-grafiek is een grafiek waar er een bepaalde waarde geplot wordt over een bepaalde tijdsspanne. De zes waarden beginnend vanaf de eerste YT-grafiek bovenaan zijn de volgende: snelheid V in mm/s, richtingscoëfficiënt, X-positie puck, Y-positie puck, tijdsverschil tussen twee frames, afstandsverschil tussen twee frames.

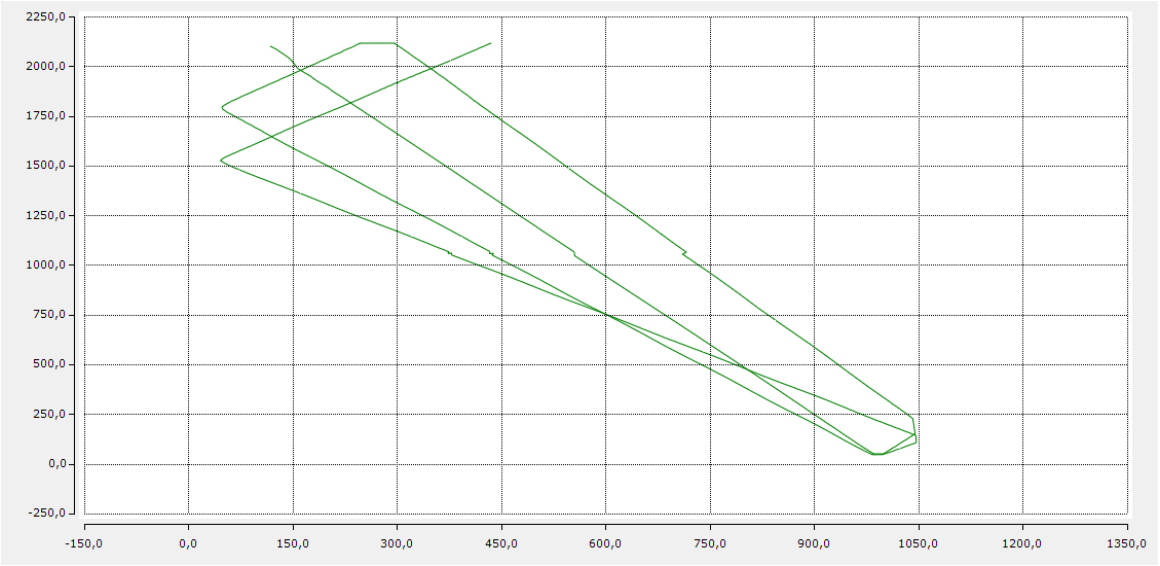
Op de grafiek kunnen volgende problemen aangeduid worden die niet in het PLC-programma uitgefilterd zijn:



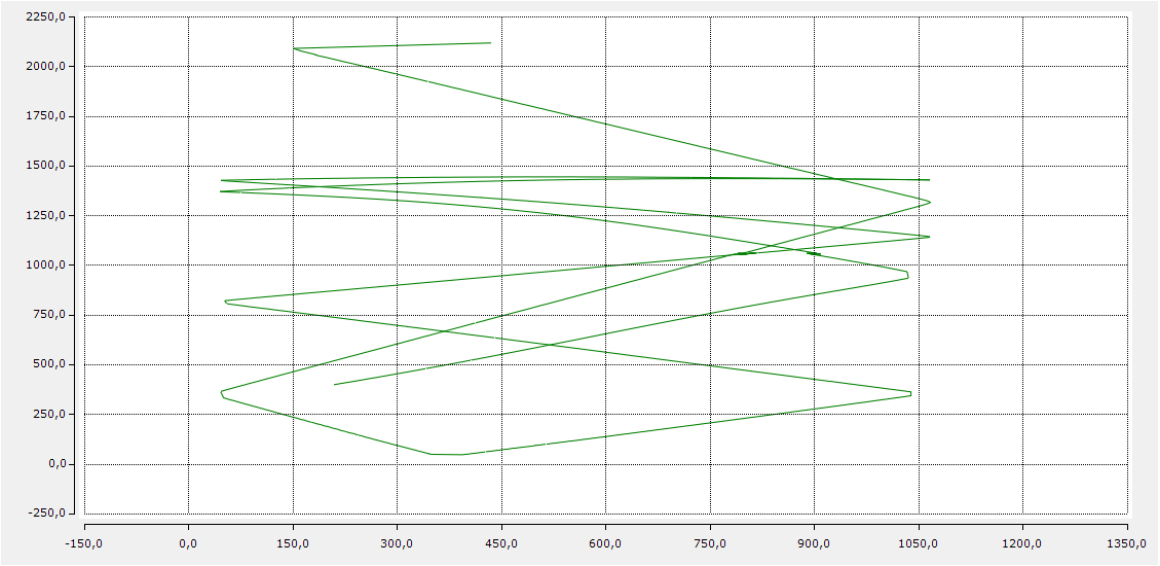
Figuur 51: YT-grafieken traject puckpositie met één botsing

a) De snelheid oscilleert tussen bepaalde waarden. De oorzaak van dit fenomeen is het wisselende afstandsverschil tussen twee frames. Bijvoorbeeld ook al beweegt de puck in een positieve richting dan kan het zijn dat de blob-detectie de puck in een negatieve richting detecteert ook al is er de filter van 3mm ingevoerd. De zwarte cirkel van de airhockeypuck is voor de camera ook niet altijd even groot waardoor het middelpunt ook anders valt. Hierdoor lijkt het of de puck achteruit beweegt waardoor de snelheid daalt.
b) De snelheid wordt plotseling 0mm/s. Dit is te verklaren doordat hier de overgangsperiode plaatsvindt. Het tijdsverschil tussen beide frames is hier negatief. Beide camera's zijn wel gesynchroniseerd bij de eerste PLC-cyclus maar de tijdstippen die zij meegeven aan elke frame zijn nog altijd gebaseerd op de interne klok van de camera.
c) De richtingscoëfficiënt en de snelheid dalen sterk. Indien de puck voor enkele frames niet gevonden wordt zorgt dit voor een groot tijdsverschil tussen de frames op het ogenblik dat de camera de puck opnieuw detecteert.

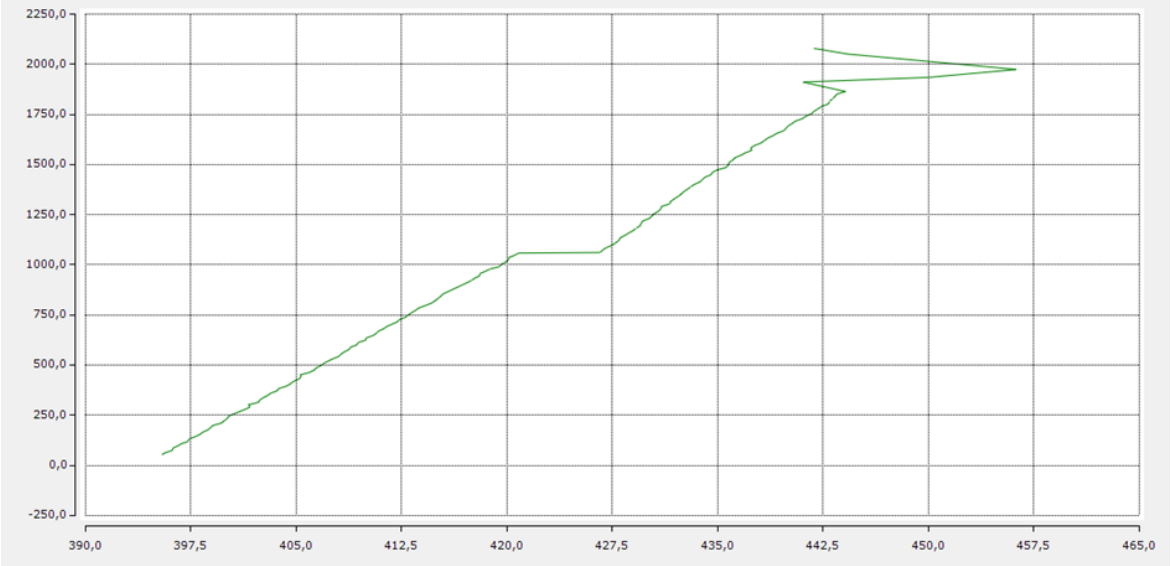
Deze problemen worden bij de fase van de data-preprocessing in hoofdstuk 4.2 zo goed mogelijk afgevlakt met feature engineering. Een van de gebruikte methodes is het berekenen van een SMA van de waarden (zie 2.1.3.3.a). De variatie van deze problemen hebben zo minder effect op het trainen van het model. In de volgende figuren 52 tot en met 54 zijn nog drie andere XY-grafieken opgesteld die de variaties tonen van diverse trajecten waarop het model ook getraind zal worden. Let hierbij wel op dat de X-assen niet overal dezelfde waarden hebben.



Figuur 52: XY-grafiek alternatief traject puck met meerdere botsingen in één hoek



Figuur 53: XY-grafiek alternatief traject puck met zigzag patroon



Figuur 54: XY-grafiek alternatief traject puck rechtstreeks naar de goal van de robot

4 Machine learning omgeving

In dit hoofdstuk wordt het trainen en evalueren van de verschillende ML-modellen besproken. Hierbij worden de verschillende stappen aangehaald beginnend met het pre-processen van de trainingsdata. Als volgt worden met behulp van de programmeertaal Python de ML-modellen getraind. Tot slot wordt elk model geëvalueerd op zijn prestaties. De beste modellen worden verder gebruikt om verbeteringen op uit te voeren. Welk model hieruit het meest nauwkeurigst is wordt geïntegreerd in de TwinCAT 3 implementatie in hoofdstuk 5.

4.1 Python omgeving

Om de data te pre-processen, de modellen te trainen en te evalueren is er de programmeertaal Python. De combinatie van Python met verschillende bibliotheken voor ML maken het mogelijk om eenvoudig een ML-workflow te programmeren. In deze paragraaf worden de componenten beschreven die gebruikt worden om de Python omgeving op te stellen.

4.1.1 Anaconda omgeving

Bij een gewone Python installatie wordt deze programmeertaal op de computer waarop ML wordt toegepast geïnstalleerd onder een bepaalde versie met pakketten. Voor ML zijn er veel Python pakketten nodig maar sommige pakketten zijn enkel compatibel met andere van een specifieke versie. Hier komt Anaconda mee in het verhaal, Anaconda is een distributie van programmeertalen zoals Python om pakketbeheer eenvoudiger te maken. Deze software wordt vaak gebruikt voor datawetenschappelijke doeleinden zoals ML. Via Anaconda kan er een Python omgeving worden aangemaakt voor een specifiek project met de gekozen Python versie en de bijhorende pakketten. Indien er een andere versie van Python of pakketten nodig zijn kan er een nieuwe Python omgeving gecreëerd worden met deze configuratie. Er is ook een mogelijkheid om ondersteuning voor de configuratie met alle Python pakketten binnen Anaconda te exporteren. Dit maakt het eenvoudig om dezelfde omgeving op een andere computer te gebruiken. Bijvoorbeeld een computer welke performanter is om het ML-model te trainen.

4.1.2 Jupyter Notebook

Python code wordt normaal in een .py bestand geprogrammeerd. Dit is een bestand waar de code in het bestand volledig wordt uitgevoerd bij aanvang van de executie. Het nadeel hiervan is dat deze aanpak niet dynamisch is voor ML. Indien een ML-model bijgestuurd moet worden met een aanpassing van een hyperparameter moet heel dit bestand opnieuw uitgevoerd worden. Als dit een grote trainingsdataset is moet deze altijd opnieuw ingeladen worden. Jupyter Notebook geeft hiervoor een oplossing. Met deze toepassing kunnen er notebooks gemaakt worden van de gekozen programmeertaal. Bij een notebook is het mogelijk om codecellen aan te maken. Elke cel kan apart uitgevoerd worden van elkaar, het hele bestand moet dus niet opnieuw uitgevoerd worden bij elke executie. Dit maakt het mogelijk om eenmaal de trainingsdata in te laden en daarna een hyperparameter of ander aanpassingen op het model uit te voeren. Om Jupyter Notebook meer interactief te maken is er gekozen om deze te implementeren in de VS Code IDE. Een van de voordelen is dat VS Code een ingebouwd versiebeheer Git heeft. Met Git kunnen de wijzigingen in de code bijgehouden worden onder verschillende commits. Indien nodig kan er teruggegaan worden naar een specifieke commit. De code kan hiermee ook naar een online platform GitHub gepubliceerd worden zodat de code altijd op meerdere plaatsen bewaard wordt.

4.2 Data pre-processing

De data zijn verzameld, er kan nu begonnen worden met de data pre-processing fase. Bij deze fase worden de CSV-bestanden van TwinCAT 3 als trainings- en testdata ingelezen in de Pythonomgeving. Hierna worden de data aangepast zodat het model zo optimaal mogelijk kan leren.

4.2.1 Procedure omzetten CSV-bestanden naar train- en testset

Volgende subparagrafen overlopen de verschillende stappen tijdens deze procedure.

4.2.1.1 Leeg dataframe creëren

Met de pandas bibliotheek van Python is het mogelijk om de dataframes te manipuleren. Dit is een tweedimensionale tabelstructuur met rijen en kolommen. Hier wordt er een leeg dataframe gecreëerd waar al de trajecten later in verzameld zullen worden.

4.2.1.2 CSV-bestand met traject inlezen

Als volgt wordt het eerste CSV-bestand met het traject van de puck ingeladen als een dataframe. Elke waarde in elke rij wordt van elkaar onderscheiden met zijn onderscheidingsteken. De trajecten zijn verzameld in een map waarbij er over elk traject geïtereerd wordt.

4.2.1.3 Feature engineering

Met feature engineering is er gekeken hoe de data kan worden aangepast om de fouten van 3.2.2.2 minder effect te laten hebben op het model. Met Python zijn er verschillende functies mogelijk voor deze toepassing. Maar deze functies zijn niet allemaal mogelijk in het TwinCAT 3 PLC-programma, daar moet rekening mee gehouden worden. Er is gekozen om op elke feature buiten de timestamp een SMA van 10 en 20 toe te passen. Hierbij worden de vorige 10- en 20-waarden van elke feature gebruikt om nieuwe features te generen met een gemiddelde van deze. Hierdoor hebben de eerste 20 rijen geen waarde bij de SMA20-functie en worden ze later in dit traject verwijderd. Voor de SMA10-functie zijn dit dus 7 extra features en voor de SMA20-functie ook een extra 7 features. Als resultaat zijn dit dus 22 (7 + 7 + 7 +1) features.

4.2.1.4 Labelen van de data

Om de toekomst te voorspellen gebruik makend van supervised learning moeten de trainings- en validatiedata gelabeld zijn. Bij dit model is er gekozen om een bepaald aantal frames in de toekomst te voorspellen. Als label voor de data is er beslist om twee labels te gebruiken, namelijk de X- en Y-positie. Deze labels worden verkregen door de X- en Y-positie van de features te verschuiven naar de labelkolom met het aantal gekozen frames in de toekomst als verschuivingsfactor. Om de robot nog voldoende reactietijd te geven is er gekozen om het model 50 frames in de toekomst te laten voorspellen. De robot heeft dan minimaal 0,233s tijd om te reageren. Dit is berekend door 70 frames (50 in de toekomst + 20 voorgaande van de SMA) te vermenigvuldigen met de ingestelde FPS van de camera, namelijk 3,3 milliseconden. De data van de features wordt dan overgeplaatst naar de labelkolommen waarbij deze 50 rijen verschoven zijn. De eerste frame/rij heeft dan als label de X- en Y-positie van frame 50. Dit zorgt ervoor dat de laatste 50 rijen van een dataframe geen labelwaarde hebben.

4.2.1.5 Lege rijen verwijderen

Zowel de eerste 20 rijen (feature engineering) als de laatste 50 rijen (labelen van data) missen data. Deze rijen worden verwijderd van het traject.

4.2.1.6 Trajecten combineren

Hierna wordt het traject gecombineerd met de lege dataframe van stap 1. Dit proces herhaalt zich tot als alle CSV-bestanden in de map zijn doorlopen. Na de combinatie van de 1252 trajecten bestaat dit dataframe uit 522710 rijen/frames met 24 kolommen. 22 kolommen zijn features en de andere 2 zijn de labels.

4.2.1.7 Train- en validatie-split

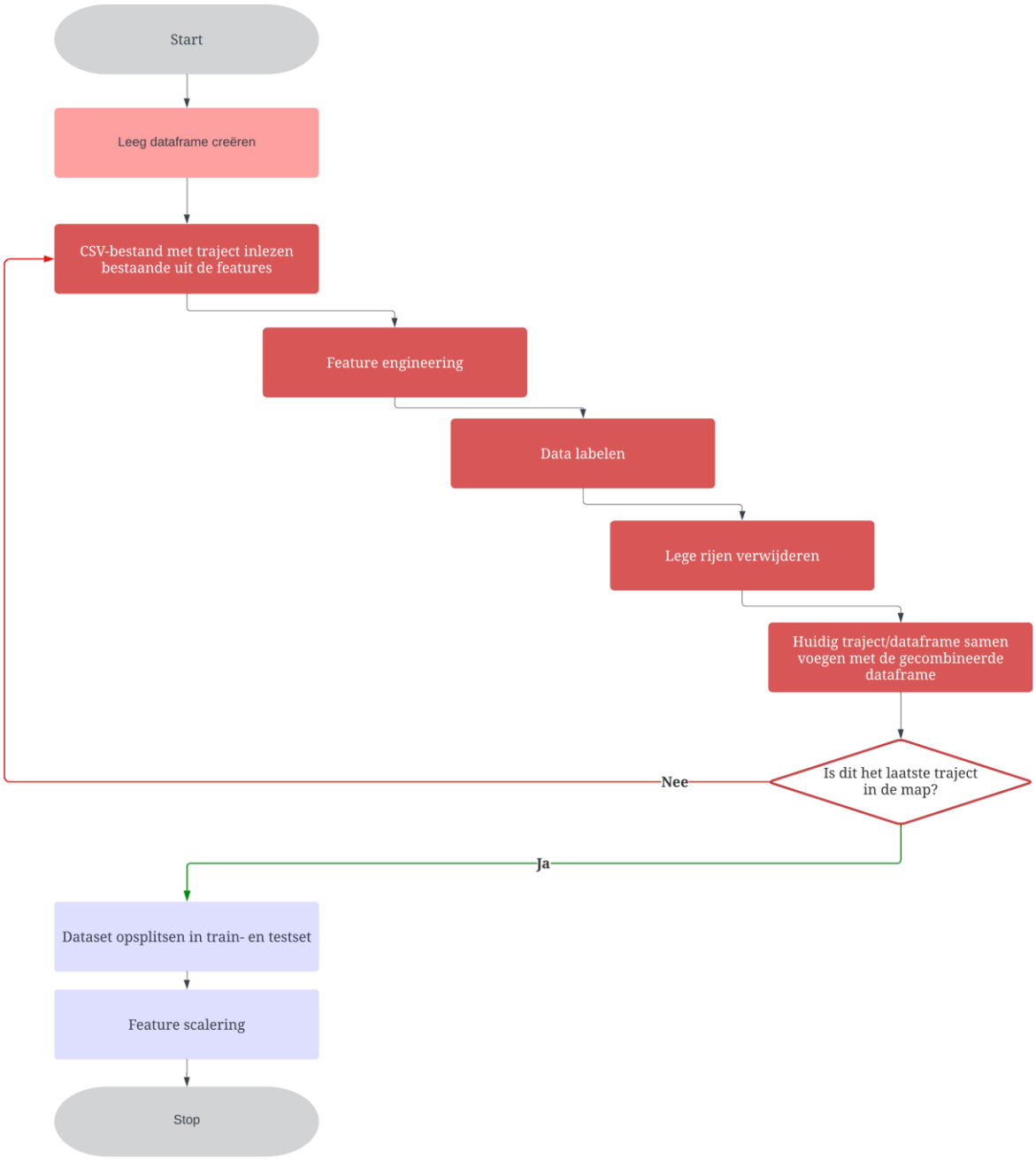
Daarna is de gecombineerde dataframe opgesplitst in een trainings- en validatieset. De trainingsset bestaat uit 80% van de data en de validatieset uit 20%. Om het model zelf het patroon beter te laten zoeken zijn de data hier ook gemengd. Zo is er ook geen plotse overgang van een traject op het volgend traject. In 2.1.3.3.c is er vermeldt dat de data opgesplitst wordt in drie delen namelijk een trainingsset, een validatieset en een testset. Als testset wordt er later bij de evaluatie één traject gebruikt dat onafhankelijk gelogd is van de originele data.

4.2.1.8 Schalering

Als laatste is er op de sets een standard schalering met behulp van de scikit-learn bibliotheek toegepast zodat elke feature evenveel effect heeft op het model (zie 2.1.3.3.b).

4.2.1.9 Overzicht

Figuur 55 op volgende pagina toont de schematische voorstelling van de gevolgde procedure.



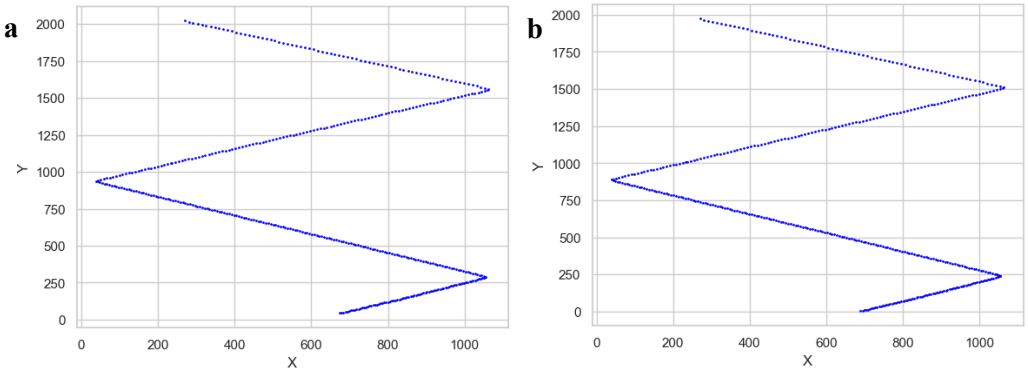
Figuur 55: Flowchart procedure CSV-bestanden naar trainings- en validatieset

4.2.2 Data generatie

Er zijn veel verschillende soorten trajecten mogelijk bij een airhockeytafel. Om deze trajecten te leren heeft het model dus veel data nodig. De trajecten kunnen zelf gemaakt worden door manueel te spelen maar dit zou dagen duren om een degelijke dataset te bekomen met duizenden trajecten. Als alternatief is er gekozen om meer data te verkrijgen door nieuwe data te genereren vertrekkende vanuit de al gecreëerde CSV-bestanden. Om deze nieuwe data te generen is volgende procedure gevolgd:

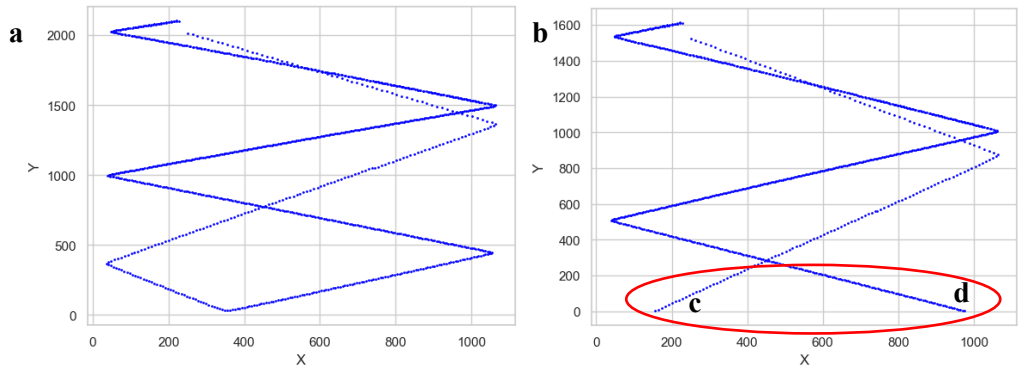
1. Verplaats al de Y-waarden in het origineel bestand met 1mm naar onder. De inverse Y-positie moet dan ook met 1mm verhoogd worden.
2. Verwijder alle datapunten waar de Y-positie onder 0mm valt. Dit is buiten het speelveld.
3. Creëer een nieuw CSV-bestand met daarin de nieuwe data.
4. Herhaal deze stappen totdat er minder dan 200 rijen aan data zijn. Zo zijn er nog genoeg datapunten om het pad van de puck te zien. Ofwel wordt er gestopt als hetzelfde origineel CSV-bestand meer dan 50 keer geïtereerd is. Dit zorgt ervoor dat dezelfde data maar maximaal 50 keer terugkomt.

In figuur 56a staan de originele data en figuur 56b toont de gegenereerde data van enkele iteraties later.



Figuur 56: Datageneratie zonder terugkaatsing met (a) de originele data en met (b) de data 50mm naar beneden verschoven

Er moet wel rekening gehouden worden dat bovenstaande procedure niet op elk CSV-bestand uitgevoerd kan worden. Deze stappen mogen enkel uitgevoerd worden op trajecten waar de puck niet terugkaatst naar de kant van de speler. Figuur 57a toont een voorbeeld van een traject met botsing en figuur 57b geeft het gegenereerd traject weer van deze data na 50 iteraties. Hier kan opgemerkt worden dat er zich een gat van data vormt aan de robotzijde (rode aanduiding) omdat alle punten onder 0mm gewist worden. Bij geen terugkaatsing naar de speler is dit geen probleem maar in dit geval lijkt het traject direct van punt c naar punt d te springen wat niet logisch is. Hieruit zou het model een foutief patroon kunnen leren.



Figuur 57: Datageneratie met terugkaatsing met (a) de originele data en met (b) de data 50mm naar beneden verschoven

4.3 Modellen trainen en evalueren

Na de data-preprocessing fase zijn de data klaar om getraind te worden door ML-modellen. In deze paragraaf worden de data getraind op de verschillende ML-modellen aangehaald in 2.2. Deze worden geëvalueerd volgens de evaluatiemetrieken MAE, MSE, RMSE en de R2-score volgens 2.1.3.5.a. Aan de hand van hun prestaties op de validatiesets worden er verschillende hyperparameters toegepast. Voor elk model wordt er enkel het model met de beste prestaties en maximaal twee andere modellen met hun hyperparameters beschreven om het verschil te tonen. De modellen met de beste prestaties van elke techniek worden verzameld in een overzicht in de laatste subparagraaf 4.2.1.9. Zo wordt er een evaluatie gemaakt tussen de verschillende technieken i.p.v. alle mogelijke combinaties met hyperparameters met één techniek. Het beste model hieruit wordt dan later geïmporteerd in TwinCAT 3 bij hoofdstuk 5.

Er is gekozen om een apart model voor zowel de voorspelling van de X-positie als van de Y-positie aan te maken. Hierdoor is het mogelijk om de modellen beter af te stellen in tegenstelling tot één model dat zowel de X- als de Y-positie voorspeld. De lineaire, polynomische en SVR-modellen worden getraind om de trainingsdata zo goed mogelijk te benaderen. Alle andere modellen zijn getraind om de MSE-fout te minimaliseren. Het is belangrijker om minder grote uitschieters in de voorspellingen te verkrijgen dan te opteren voor een betere MAE-score die nog grote uitschieters heeft in zijn resultaten. De grote uitschieters veroorzaken bij de robot een heel gevoelig systeem.

Zo goed als alle modellen zijn getraind met de ML-bibliotheek scikit-learn buiten de LightGBM, de XGBoost en het PyTorch neuraal netwerk. De meeste bibliotheken hebben opties om CPU-versnelling of GPU-versnelling te gebruiken wat het trainingsproces met een grote factor kan versnellen. Eerst werden deze modellen op een laptop uitgevoerd maar om de modellen nog sneller te trainen en te evalueren is er overgeschakeld naar een desktop pc. De Anaconda Pythonomgeving kon makkelijk overgeplaatst worden zoals aangehaald in 4.1.1. Dit zorgt op zijn beurt weer voor een extra prestatieverbetering. In tabel 3 staan de laptop en desktop pc met hun verschillende specificaties met betrekking tot het trainen van een ML-model weergegeven.

Tabel 3: Specificaties laptop versus desktop pc voor ML-training

	CPU				**RAM**	**GPU**		
	Type	**Cores**	**Basis frequentie**	**Boost frequentie**	**Capaciteit**	**Type**	**CUDA cores**	**VRAM**
Laptop	Intel Core i7-8850H	6 cores en 12 threads	2,6 GHz	4,3 GHz	16 GB	NVIDIA MX130	384	2 GB
Desktop pc	AMD Ryzen 5 2600X	6 cores en 12 threads	3,6 GHz	4,2 GHz	16 GB	NVIDIA RTX 3060 Ti	4864	8 GB

Onderstaand is de betekenis van deze specificaties voor het leren van een ML-model terug te vinden:

- De CPU (Central Processing Unit) of processor van de pc die wordt gebruikt om ML-modellen te trainen heeft volgende eigenschappen:
    - Type: Het type geeft aan welk model van CPU er is gebruikt voor de trainingsfase. Bij de laptop is dit een CPU die compact is terwijl de grotere CPU van de desktop pc een betere koeling heeft.
    - Cores: Beide CPU's hebben 6 fysieke cores. Door *hyperthreading* bestaan er 2 *threads*/werkers per core. Sommige modellen maken het mogelijk om het model te trainen op meerdere threads wat de trainingssnelheid verbetert.
    - Basis frequentie: Dit is de frequentie waarop de CPU taken uitvoert als de processor niet zwaar belast is.

    - Boost frequentie: Dit is de frequentie waarop de CPU taken uitvoert als de processor wel zwaar belast is. Bijvoorbeeld bij het trainen van een ML-model. Deze frequentie is hoger bij de laptop maar omdat deze daar in een compact formaat is houdt de laptop deze frequentie minder lang vol. De CPU vertoont hier *thermal throttling* waarbij de CPU op een lagere snelheid werkt om niet oververhit te raken.

- Het RAM-geheugen (Random Access Memory) is het werkgeheugen van een pc. De dataset bestaat uit 522710 rijen met 24 kolommen aan data met het type float64. Dit datatype gebruikt 8bytes per element wat uitkomt op (522710 x 24 x 8bytes) 100.360.320bytes of 95,7MB. Deze data worden opgeslagen in een dataframe dat bewaard wordt in het RAM-geheugen van de pc. Dit geheugen moet dus voldoende groot zijn om deze dataset op te kunnen slaan. Voor sommige ML-modellen moet de dataset nog verder gemanipuleerd worden in een andere dataset wat op zijn beurt ook nog extra geheugen kost. 16GB RAM-geheugen is meer dan voldoende om deze applicatie te trainen met de gekozen dataset en om nog ruimte over te houden voor het besturingssysteem. Maar indien er bijvoorbeeld 100 keer meer trainingsdata worden gevoed aan de dataframe kan het model beter presteren al zorgt dit wel dat er dan 10GB aan RAM nodig is.

- De GPU (Graphical Processing Unit) of grafische kaart heeft ook de mogelijkheid om ML-modellen te trainen.
    - Type: De toepassingen om ML-modellen te trainen op een GPU worden gedomineerd door NVIDIA met hun CUDA (Compute Unified Device Architecture) parallelle computertechnologie. Zowel de laptop als de desktop pc heeft een NVIDIA GPU met de mogelijkheid om CUDA-cores te gebruiken voor het trainen.
    - CUDA: Het aantal CUDA-cores geeft de indicatie weer hoe snel een ML-model getraind kan worden. De desktop kan hetzelfde model meer dan 10x sneller uitvoeren dan de laptop, dit is ook te zien in het aantal CUDA-cores.
    - VRAM (Video Random Access Memory): De GPUs hebben VRAM ter beschikking dit is het werkgeheugen van de GPU. ML-modellen getraind op de GPU gebruiken dit geheugen t.o.v. het RAM-geheugen. Dit moet ook voldoende groot zijn om de dataframe in te laden.

In tabel 4 zijn de trainingstijden vergeleken van de laptop en desktop pc voor het random forest model en het PyTorch model met de beste prestaties die later in de volgende paragrafen aan bod komen.

Tabel 4: Vergelijking trainingstijd laptop versus desktop pc

	Hyperparameters	**Laptop**	**Desktop pc**
Random Forest (1 CPU-thread)	Diepte X = 20; Diepte Y = 20 en 50 estimators	3m27s	2m5s
Random Forest (12 CPU-threads)	Diepte X = 20; Diepte Y = 20 en 50 estimators	1m	25,4s
PyTorch (GPU)	Twee hidden layers met X = 10x10 en Y = 8x10 neuronen op 20000 iteraties	33m10s	3m25

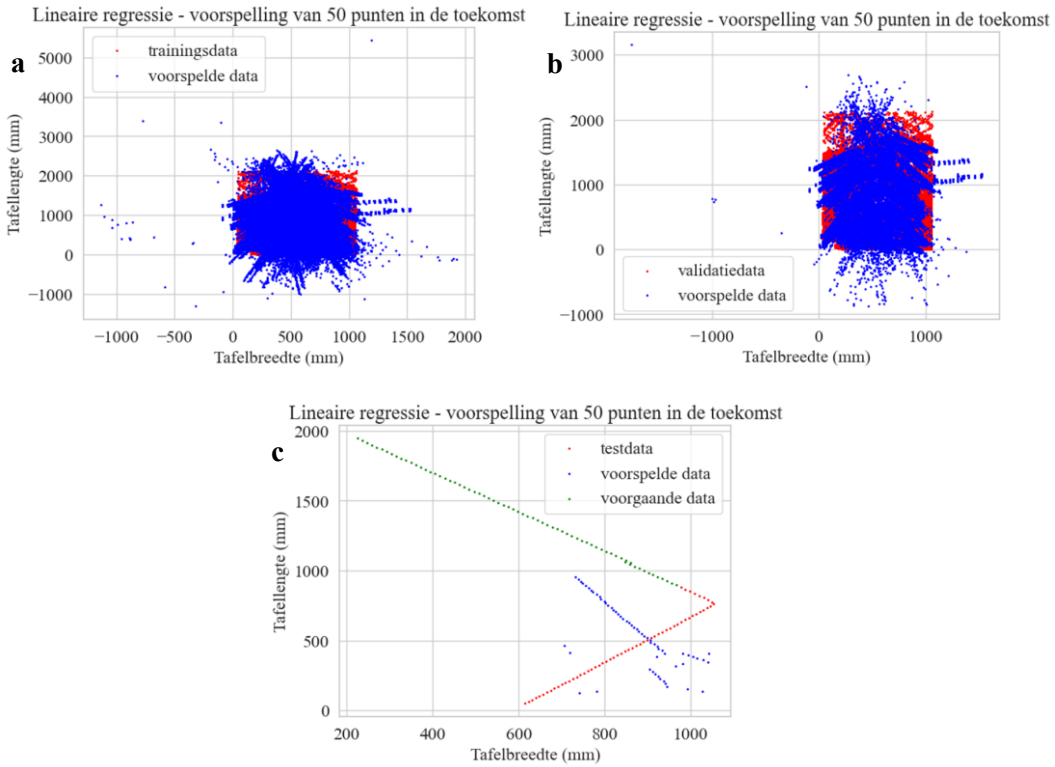
4.3.1 Lineaire regressie

Het eenvoudigste model is de lineaire regressie. Dit model wordt uitgevoerd d.m.v. de scikit-learn bibliotheek met de Ridge-functie. Deze functie probeert een lineair verband te zoeken in de data. Hier is er één hyperparameter beschikbaar, namelijk alpha maar deze heeft geen effect op de prestaties van het model. In tabel 5 worden de resultaten van elke evaluatiemetriek weergegeven voor het X- en Y-model bij zowel de trainings-, de validatie- en de testdata.

Tabel 5: Lineaire regressie prestaties

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	122,84	20,12	123,29	20,34	183,19	66,98
MSE [mm²]	24729,83	2670,52	24952,94	2792,74	42557,92	5010,53
RMSE [mm]	157,26	51,68	157,96	52,85	206,3	70,79
R2-score	0,713	0,987	0,708	0,987	-1,304	0,916

Het X-model heeft de slechtste resultaten met de grootste fouten. Hierbij heeft de onafhankelijke set zelfs een negatieve R2-score. Het Y-model daarentegen heeft al veel betere resultaten. Hier kan het model het lineair karakter van de puck beter terugvinden in de data. Figuur 58 toont een spreidingsdiagram van de verschillende sets met hun voorspelde punten. Oranje zijn de trainingsdata, de validatiedata of testdata en blauw zijn de voorspelde data. In het onafhankelijk traject of testtraject zijn ook de voorgaande data van het traject in het groen aangeduid. Dit zijn de 70 voorgaande punten vóór er een eerste voorspelling wordt uitgevoerd. Deze 70 punten bestaan uit de eerste 20 punten die nodig zijn om de SMA20-waarden te creëren gevolgd door de 50 punten voor de toekomstvoorspelling.



Figuur 58: Spreidingsdiagram lineaire regressie met (a) trainingsdata, (b) validatiedata en (c) testdata

4.3.1.1 Polynomische regressie

Een variant op de lineaire regressie is de polynomische regressie die ook wordt uitgevoerd met scikit-learn. Bij deze methode is de graad van de regressielijn instelbaar als hyperparameter. Nu zoekt het model een regressielijn waarbij de hoogste macht in de functie overeenkomt met de ingestelde hyperparameter. In tabel 6 worden de resultaten getoond voor een maximale graad van twee en in tabel 7 de resultaten met een maximale graad van drie. Een vierdegraadsfunctie is niet mogelijk op de gebruikte hardware omdat het model een RAM-capaciteit van 46,6GB nodig had om de array te trainen. Op de trainingscomputer is maar 16GB ter beschikking.

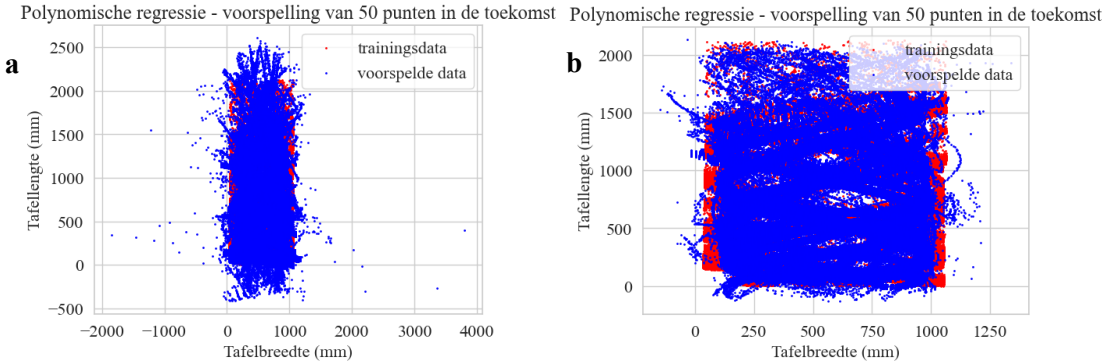
Tabel 6: Polynomische regressie prestaties 2de graad

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	78,46	15,023	78,73	15,28	83,31	102,67
MSE [mm²]	10051,86	1420,28	12302,64	1645	9190,9	14850,37
RMSE [mm]	100,26	37,69	110,92	40,56	95,87	121,86
R2-score	0,883	0,993	0,856	0,992	0,502	0,75

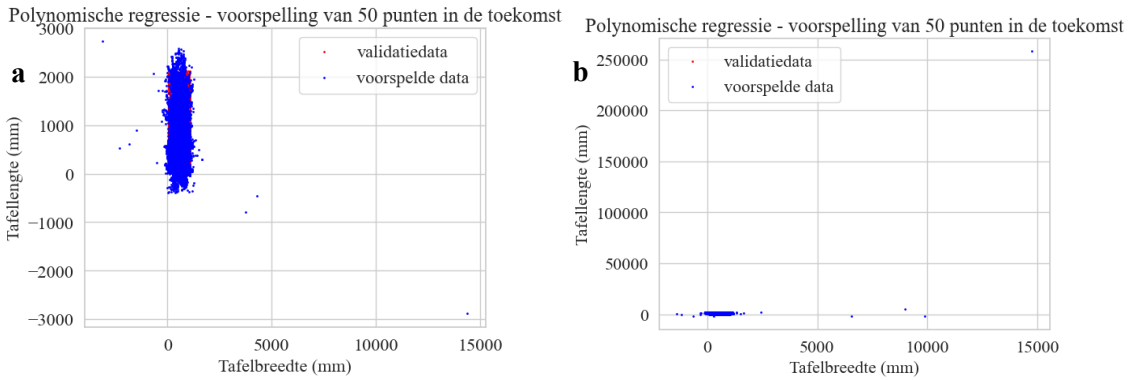
Tabel 7: Polynomische regressie prestaties 3de graad

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	28,38	11,33	28,98	14,23	120,95	102,83
MSE [mm²]	1765,55	459,6	5842,71	636630,98	49155,59	24365,47
RMSE [mm]	42,02	21,44	76,44	797,89	221,71	156,09
R2-score	0,979	0,998	0,932	-2,079	-1,661	0,589

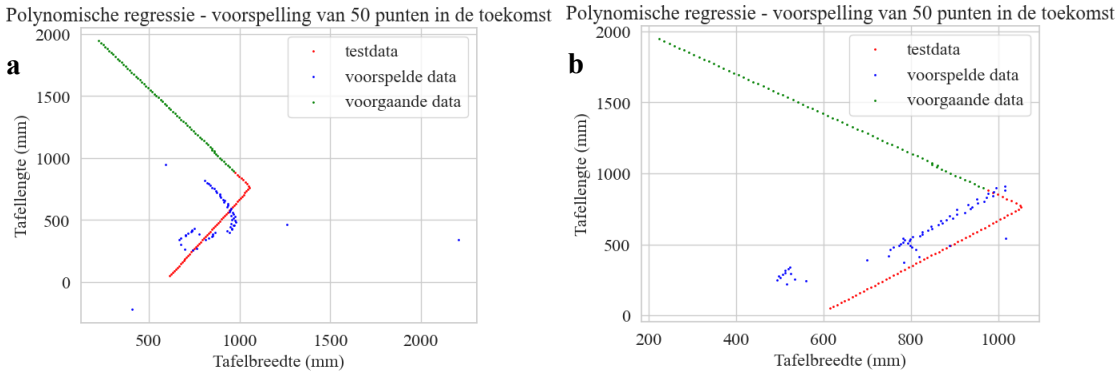
De resultaten van het model op de trainingsdata zijn positief gestegen. Dit komt doordat het model nu complexere functies kan maken en beter aanleunt bij de trainingsdata. De resultaten op de validatie- en testdata en dan vooral de MSE-fout, zijn in het algemeen toegenomen. Dit komt doordat het model een overfittingsgedrag vertoont. Het model heeft in het algemeen betere MAE-waarden maar heeft meer uitschieters in de validatie- en testdata die vooral voor problemen zorgen. Figuur 59 tot en met 61 tonen de spreidingsdiagrammen van deze datasets voor zowel de tweedegraadsfunctie als de derdegraadsfunctie.



Figuur 59: Spreidingsdiagram polynomische regressie met trainingsdata voor (a) de tweedegraadsfunctie en voor (b) de derdegraadsfunctie



Figuur 60: Spreidingsdiagram polynomische regressie met validatiedata voor (a) de tweedegraadsfunctie en voor (b) de derdegraadsfunctie



Figuur 61: Spreidingsdiagram polynomische regressie met testdata voor (a) de tweedegraadsfunctie en voor (b) de derdegraadsfunctie

4.3.2 Decision tree

Als volgend model is de decision tree getest. De belangrijkste hyperparameter voor de decision tree is de diepte van de boomstructuur. Deze bepaalt hoe complex het model zichzelf kan trainen om de MSE-fout te minimaliseren. Er werd gestart met een grote waarde voor het aantal nodes in de diepte, nl. 20. De prestaties van dit model kunnen geconsulteerd worden in tabel 8.

Tabel 8: Decision tree prestaties met diepte 20

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	2,72	1,25	3,95	3,05	72,82	126,12
MSE [mm²]	119,12	6,4	386,65	364,18	9258,56	25146,12
RMSE [mm]	10,91	2,53	19,66	19,08	96,22	158,58
R2-score	0,999	0,999	0,996	0,998	0,499	0,577

Dit model presteert heel goed op de trainings- en validatiedata maar presteert niet zo goed op de testdata. Het model heeft de paden van zijn trainingsdataset vanbuiten geleerd en toont dus overfittingsgedrag. Omdat de validatiedata hierop gebaseerd is heeft deze gelijkaardige resultaten. Vervolgens is er een minder complex model gekozen met als diepte 10 waarvan de resultaten in tabel 9 staan opgesomd.

Tabel 9: Decision tree prestaties met diepte 10

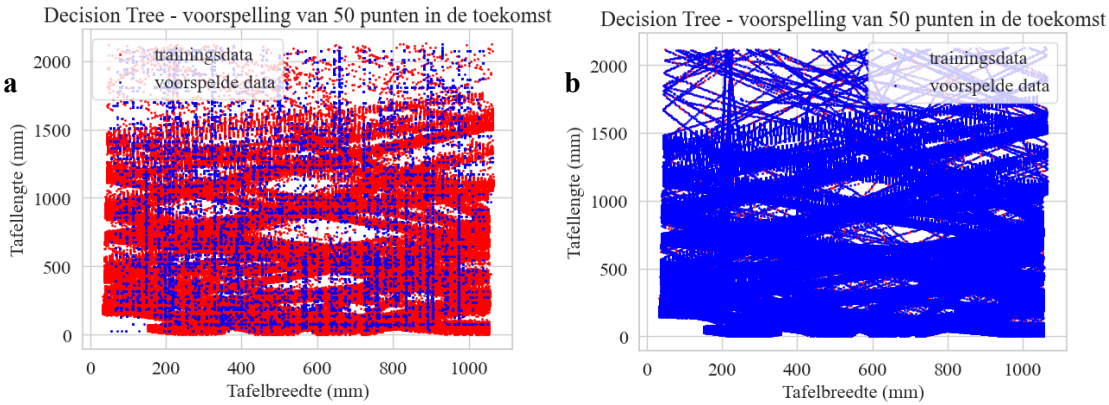
	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	39,85	30,5	40,1	31,35	155,6	198,89
MSE [mm²]	3922,93	2669,66	3988,5	3026,67	28087,95	67345,7
RMSE [mm]	62,63	51,669	63,15	55,02	167,595	259,51
R2-score	0,954	0,987	0,954	0,985	-0,521	-0,134

De resultaten van dit model zijn niet beter. Met een decision tree heeft het model niet de mogelijkheid om met weinig diepte ongeziene trajecten te voorspellen. Omdat het model beter presteert bij hogere dieptes is er een test gedaan om geen restricties op de diepte van het model te plaatsen. Tabel 10 toont de prestatie van het model bij een diepte van 54 voor het X-model en een diepte van 39 voor het Y-model.

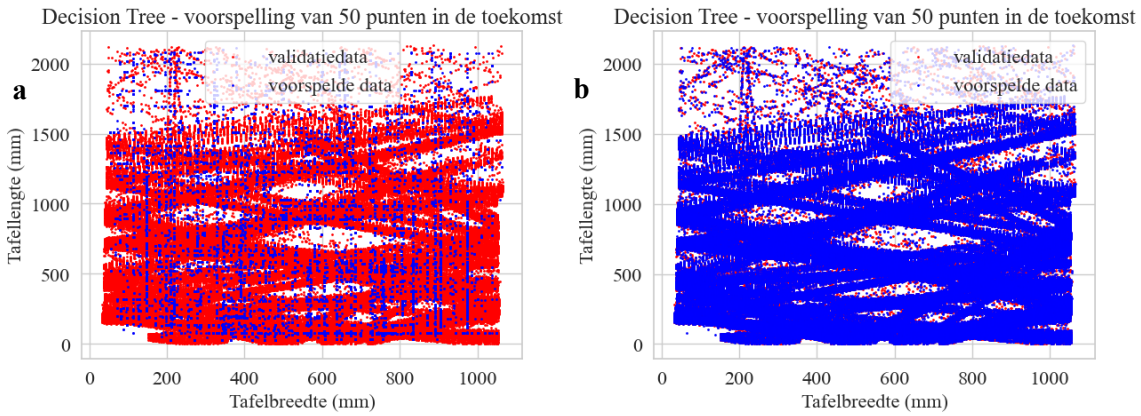
Tabel 10: Decision tree prestaties met diepte X = 54 en diepte Y = 39

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	1,86E-13	5,93E-10	1,11	2,33	76,68	131,03
MSE [mm²]	7,93E-26	6,32E-15	176,78	474,16	8492,17	23575,25
RMSE [mm]	2,82E-13	7,95E-08	13,3	21,78	92,15	153,54
R2-score	1	1	0,998	0,998	0,54	0,603

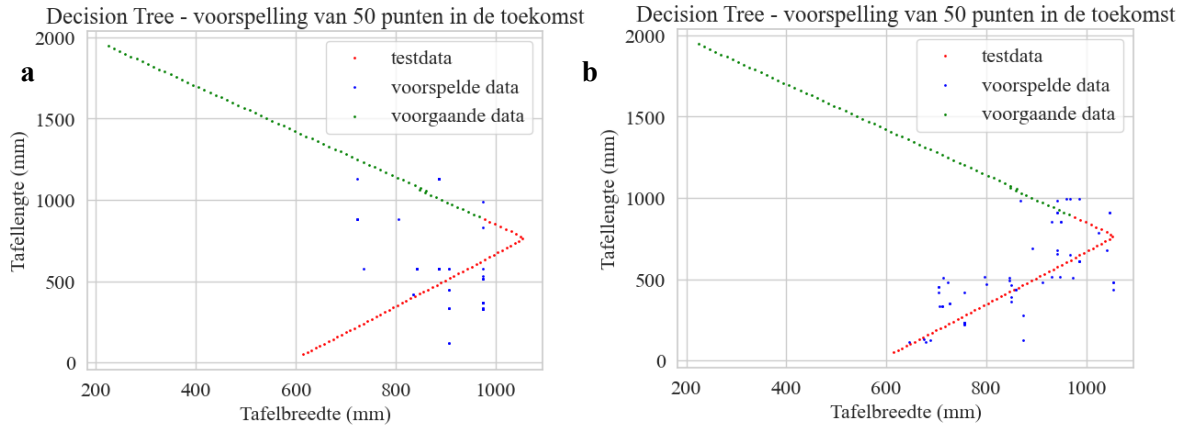
Bij deze resultaten kan er opgemerkt worden dat de trainingsdata zo goed als perfect worden voorspeld door het model. Er is wel een R2-score van 1 wat direct aan overfitting doet denken. Daarnaast zijn de andere evaluatiemetrieken zeer minimaal. De testdata heeft ongeveer dezelfde resultaten als het model met een diepte van 20. Figuur 62 tot en met 64 tonen de resultaten in de spreidingsdiagrammen van het model met een diepte van 10 met daarnaast de resultaten van het model met ongelimiteerde diepte. Hier kan het onderfittingsgedrag bij de diepte van 10 en het overfittingsgedrag bij de ongelimiteerde diepte vastgesteld worden.



Figuur 62: Spreidingsdiagram decision tree met trainingsdata (a) met diepte 10 en (b) met ongelimiteerde diepte



Figuur 63: Spreidingsdiagram decision tree met validatiedata (a) met diepte 10 en (b) met ongelimiteerde diepte



Figuur 64: Spreidingsdiagram decision tree met testdata (a) met diepte 10 en (b) met ongelimiteerde diepte

4.3.3 Ensemble tree

Diverse decision trees worden gecombineerd om verschillende ensemble trees te ontwikkelen. Het overfittingsgedrag bij het decision tree model met ongelimiteerde diepte is ook merkbaar bij de ensemble trees. Daarom worden er bij ensemble trees ook geen modellen met ongelimiteerd diepte getest. Een hyperparameter die bij elke ensemble tree model voorkomt is het aantal *estimators*. Dit zijn het aantal decision trees of schattingen die nodig zijn om het ensemble tree model op te bouwen. Ook een hyperparameter welke bij al deze modellen voorkomt is het aantal features per estimator. Volgende testresultaten zijn getraind waarbij het maximumaantal features per estimator is beperkt tot de vierkantswortel van het aantal features. Dit is de standaardinstelling, deze wijzigen had niet veel effect op de evaluatiemetrieken.

Alle gebruikte ensemble tree modellen buiten de gradient boosting kunnen met de CPU versneld worden door gebruik te maken van alle beschikbare CPU-cores i.p.v. één enkele CPU-core.

4.3.3.1 Random Forest

Als eerste model is het random forest model getest. Tabel 11 tot en met 13 tonen de resultaten van dit model met enkele hyperparameters.

Tabel 11: Random forest prestaties met diepte = 8 en estimators = 50

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	40,54	28,88	40,7	29,24	82	227,38
MSE [mm²]	3743,06	2595,98	3779,15	2728,52	9721,64	62439,06
RMSE [mm]	61,18	50,95	61,47	52,235	98,6	249,88
R2-score	0,956	0,987	0,956	0,986	0,474	-0,051

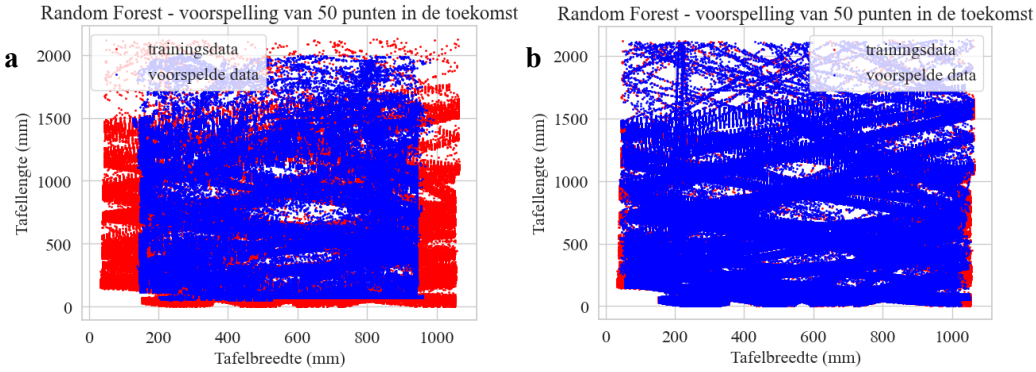
Tabel 12: Random forest prestaties met diepte = 20 en estimators = 50

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	3,04	1,11	3,55	1,93	46,1	111,77
MSE [mm²]	70,26	11,74	108,23	56,06	2739,17	22377,2
RMSE [mm]	8,38	3,43	10,4	7,49	52,34	149,59
R2-score	0,999	0,999	0,999	0,999	0,852	0,623

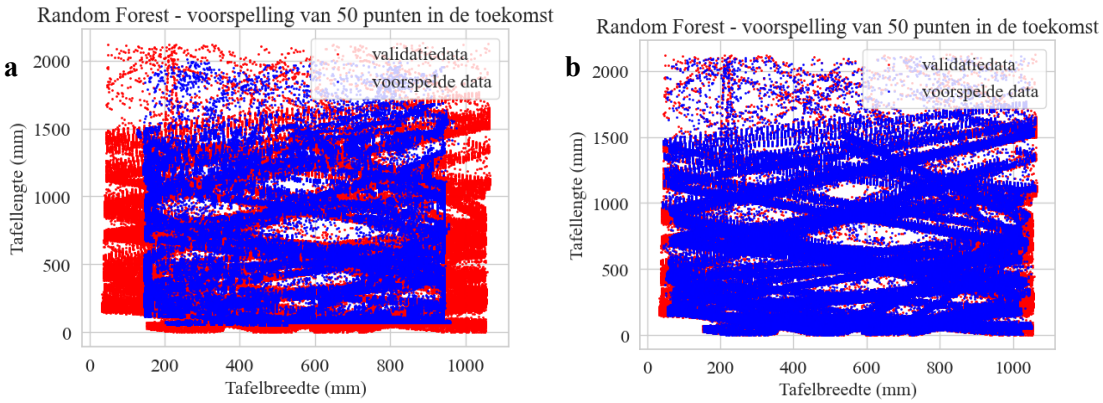
Tabel 13: Random forest prestaties met diepte = 20 en estimators = 25

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	3,08	1,2	3,62	2,065	47,14	125,48
MSE [mm²]	68,56	15,9	109,29	68,47	2622,77	28874,08
RMSE [mm]	8,28	3,99	10,45	8,28	51,21	169,92
R2-score	0,999	0,999	0,999	0,999	0,858	0,514

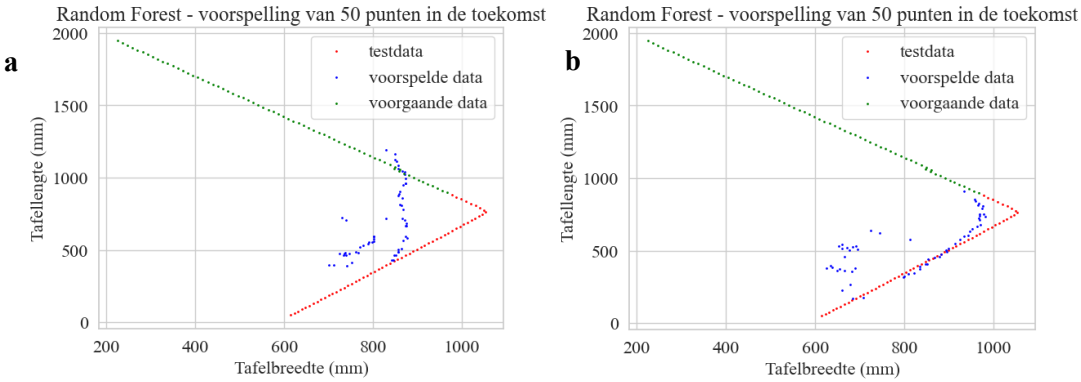
Bij deze resultaten kan er geconcludeerd worden dat het beste model de grootste diepte en het grootste aantal estimators heeft. Dit model is wel beter dan een enkelvoudige decision tree model met dezelfde diepte maar de trainingsdata hiervan vertonen nog steeds overfitting. Het aantal estimators verlagen heeft een negatief effect op de prestaties van het model. Figuur 65 tot en met 67 tonende resultaten aan in spreidingsdiagrammen voor het model met diepte 8 en voor het model met diepte 50, beide met 50 estimators. Hier kan een onderfittingsgedrag bij de diepte van 8 vastgesteld worden. Doordat dit model niet complex kan leren heeft deze niet de mogelijkheid om de relatie te leggen met de hoeken en randen van de tafel waar de puck botst. In de figuren is het airhockeyveld voor de voorspelde waarden kleiner dan de werkelijke waarden. In het andere model kan het overfittingsgedrag vastgesteld worden.



Figuur 65: Spreidingsdiagram random forest met trainingsdata (a) met diepte 8 en (b) met diepte 20, beide met 50 estimators



Figuur 66: Spreidingsdiagram random forest met validatiedata (a) met diepte 8 en (b) met diepte 20, beide met 50 estimators



Figuur 67: Spreidingsdiagram random forest met testdata (a) met diepte 8 en (b) met diepte 20, beide met 50 estimators

4.3.3.2 ExtraTrees

ExtraTrees lijken op het random forest model maar zijn nog meer willekeurig van aard (zie 2.2.4.1.a). Tabel 14 en 15 tonen de resultaten van de modellen met respectievelijk een diepte van 8 en een diepte van 20. Ook hier hebben beide modellen 50 estimators.

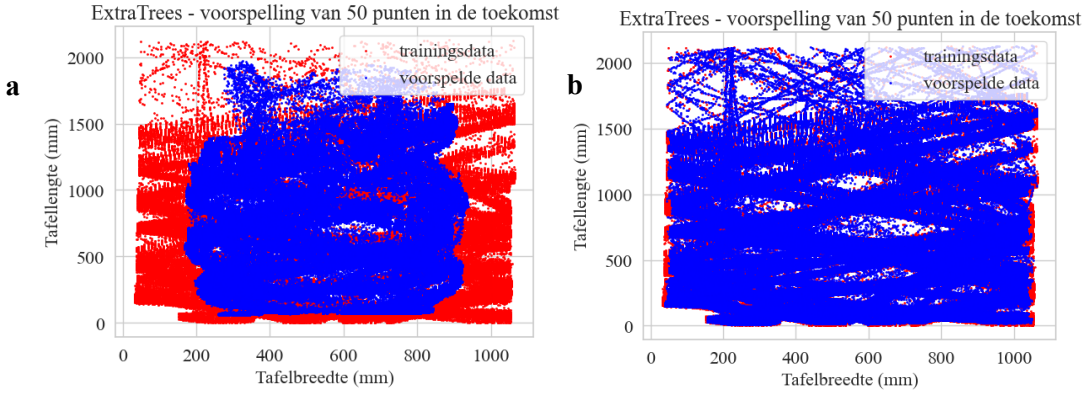
Tabel 14: ExtraTrees prestaties met diepte = 8 en estimators = 50

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	79,89	70,47	79,9	70,56	131,16	431,16
MSE [mm²]	11085,7	11213,88	11093,2	11314,46	22214,76	190487,94
RMSE [mm]	105,29	105,9	105,32	106,37	149,05	436,45
R2-score	0,871	0,946	0,871	0,945	-0,203	-2,206

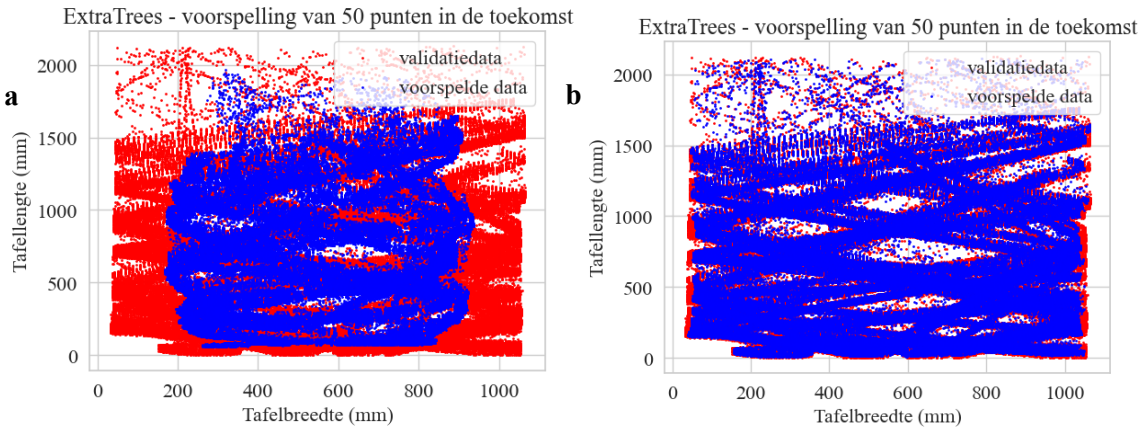
Tabel 15: ExtraTrees prestaties met diepte = 20 en estimators = 50

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	4,77	5,21	5,26	5,85	54,88	121,55
MSE [mm²]	54,66	63,98	82,5	110,91	3769,1	29824,61
RMSE [mm]	7,39	8	9,08	10,53	61,39	172,7
R2-score	0,999	0,999	0,999	0,999	0,796	0,498

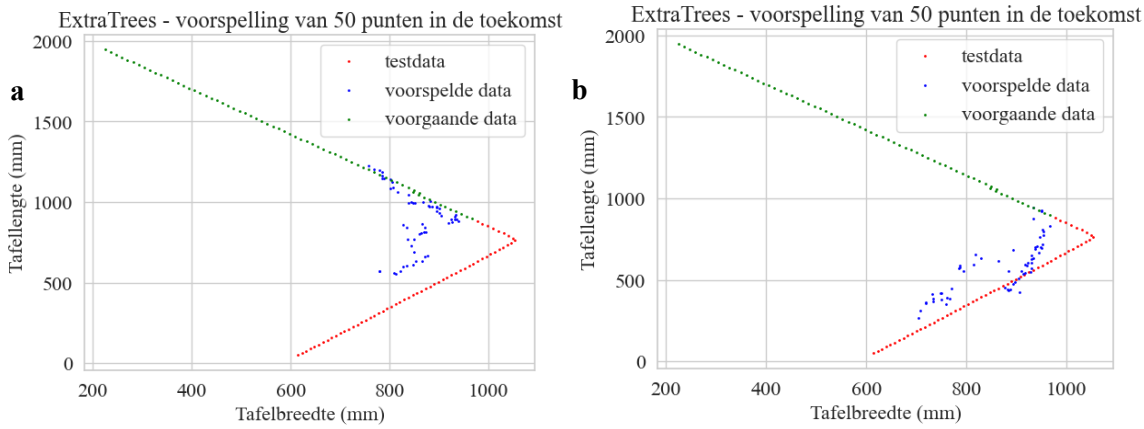
In vergelijking met het random forest model waarbij er ook 50 estimators en een diepte van 8 gebruikt werden presteert het ExtraTrees model minder goed in alle metrieken. Met dezelfde instellingen als het beste random forest model presteert het ExtraTrees model dan weer wel beter maar nog altijd minder dan het equivalent random forest model. In de figuren Figuur 68 tot 70 kan dit ook waargenomen worden.



Figuur 68: Spreidingsdiagram ExtraTrees met trainingsdata (a) met diepte 8 en (b) met diepte 20, beide met 50 estimators



Figuur 69: Spreidingsdiagram ExtraTrees met validatiedata (a) met diepte 8 en (b) met diepte 20, beide met 50 estimators



Figuur 70: Spreidingsdiagram ExtraTrees met testdata (a) met diepte 8 en (b) met diepte 20, beide met 50 estimators

4.3.3.3 Gradient boosting

Het gradient boosting model is het eerste ensemble tree model waarbij de boosting-techniek wordt gebruikt. Dit model heeft veel gelijkaardige hyperparameters met de vorige modellen. Om te beginnen is het model getraind met een diepte van 20 en met 50 estimators zoals de vorige modellen (Tabel 16).

Tabel 16: Gradient boosting prestaties met diepte = 20 en estimators = 50

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	1,33	1,97	1,96	2,75	35,86	113,05
MSE [mm²]	2,67	5,5	32,21	50,88	1754,69	23766,8
RMSE [mm]	1,63	2,34	5,68	7,13	41,89	154,164
R2-score	0,999	0,999	0,999	0,999	0,905	0,599

Dit model heeft net zoals alle vorige ensemble trees modellen een overfittingsgedrag op de originele trainingsdata. Maar de R2-score is voor het eerst groter dan 0,9 bij een van de twee labels van de testdata. Als volgt is de diepte verlaagd om te kijken of het model minder overfitting vertoont op de data (Tabel 17). Het aantal estimators is wel behouden.

Tabel 17: Gradient boosting prestaties met diepte = 10 en estimators = 50

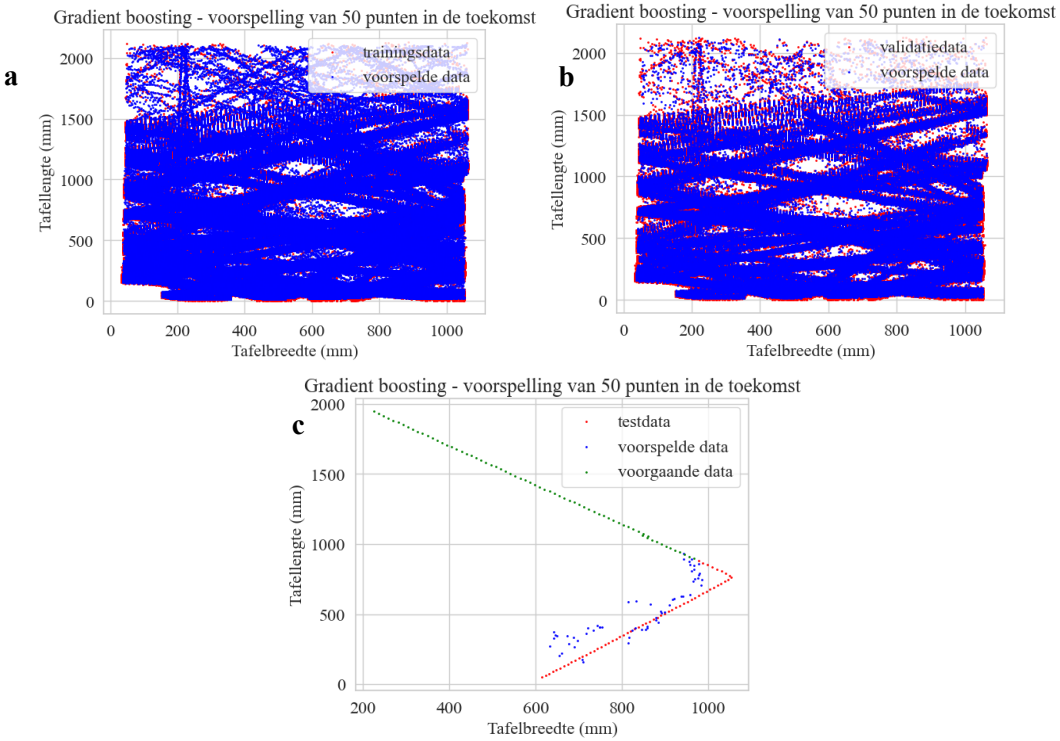
	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	6,61	5,71	7,21	6,42	43,38	83,3
MSE [mm²]	134,8	73,17	195,33	139,14	2655,31	12107,08
RMSE [mm]	11,61	8,55	13,97	11,8	51,53	110,03
R2-score	0,998	0,999	0,998	0,999	0,856	0,796

Deze diepte in het model zorgt ervoor dat het X-label minder goed presteert maar dat het Y-label daarentegen wel beter presteert. Door het verlagen van de diepte kan het model de punten aan de randen niet meer correct leren. Als beste model is er gekozen om een combinatie van beide hyperparameters te nemen. Voor het X-model is er een diepte van 20 gebruikt en voor het Y-model is er een diepte van 10 gekozen. In tabel 18 zijn de best verkregen resultaten met gradient boosting weergegeven.

Tabel 18: Gradient boosting prestaties met diepte X = 20, diepte Y = 10 en estimators = 50

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	1,33	5,71	1,96	6,42	35,86	83,3
MSE [mm²]	2,67	73,17	32,21	139,14	1754,69	12107,08
RMSE [mm]	1,63	8,55	5,68	11,8	41,89	110,03
R2-score	0,999	0,999	0,999	0,999	0,905	0,796

In figuur 71 zijn de punten van de trainings-, validatie en testdata uitgezet met hun voorspelde waarden voor het beste model. Dit met diepte X = 20 en diepte Y = 10.



Figuur 71: Spreidingsdiagram gradient boosting met (a) trainingsdata, (b) validatiedata en (c) testdata

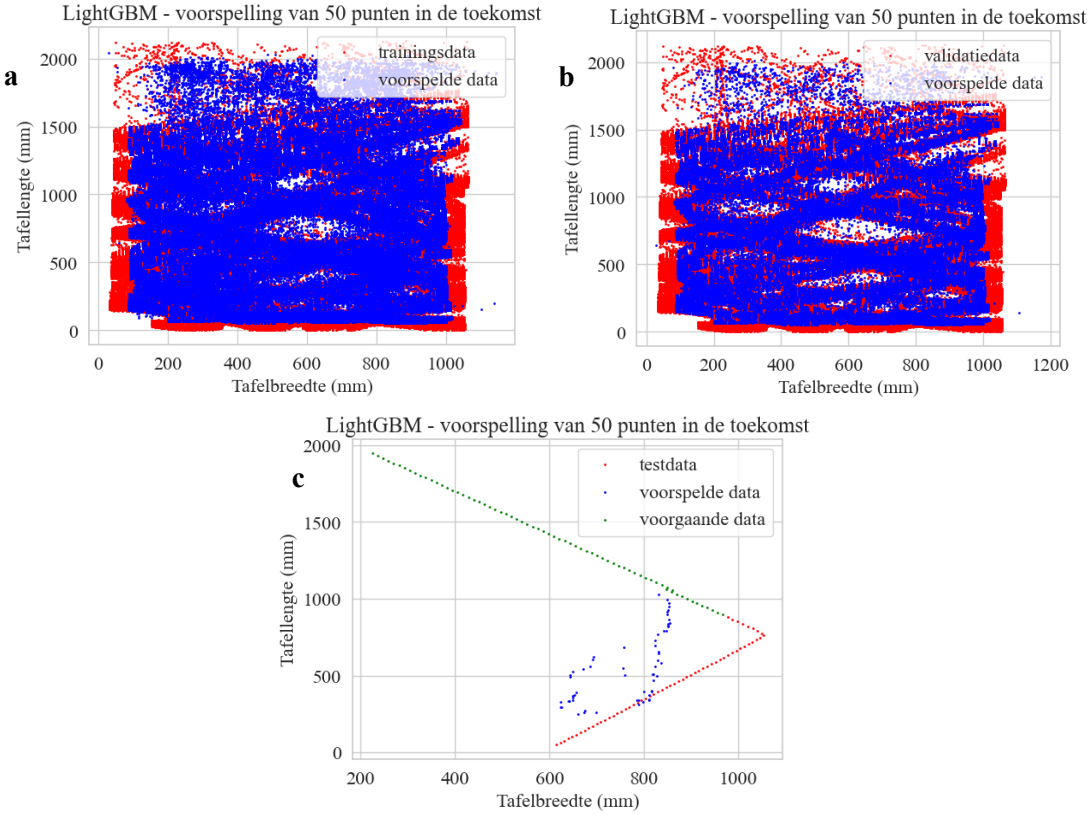
4.3.3.4 LightGBM

Het volgende getrainde boosting model is het LightGBM model. Dit model wordt getraind met zijn eigen bibliotheek. Het model kan heel snel getraind worden op enkele seconden maar is ook veel minder nauwkeurig dan de andere modellen. Tabel 19 toont een overzicht van de beste waarden die voor dit model gevonden zijn, de hyperparameters X en Y met diepte van 20 en 50 estimators.

Tabel 19: LightGBM prestaties met diepte = 20 en estimators = 50

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	24,55	19,61	24,77	19,92	100,1	131,3
MSE [mm²]	1567,45	1249,25	1602,56	1343,76	12923,5	23451,86
RMSE [mm]	39,6	35,34	40,03	36,66	113,68	153,14
R2-score	0,982	0,994	0,981	0,994	0,3	0,605

In figuur 72 zijn de punten van de trainings-, validatie en testdata uitgezet met hun voorspelde waarden voor het beste model. Dit met diepte van 20 voor beide modellen.



Figuur 72: Spreidingsdiagram LightGBM met (a) trainingsdata, (b) validatiedata en (c) testdata

4.3.3.5 XGBoost

Het laatste geteste ensemble tree model is het XGBoost model. Dit model wordt ook getraind met zijn eigen bibliotheek maar heeft de mogelijkheid om via GPU of CPU versneld te worden. De GPU-acceleratie is bij het testen ongeveer gelijk aan de CPU-acceleratie waardoor in deze masterproef enkel de CPU-acceleratie gebruikt is. Bij dit model is ook de regularisatieparameter lambda gebruikt om overfitting te voorkomen.

Bij het eerst getrainde model zijn dezelfde hyperparameters gebruikt als bij de vorige modellen, namelijk een maximale diepte van 20 voor X en Y alsook 50 estimators en is er nog geen regularisatie toegepast (Tabel 20).

Tabel 20: XGBoost prestaties met diepte = 20 en estimators = 50

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	0,1	6,85	1,01	7,66	86,87	67,47
MSE [mm²]	0,05	80,27	51,03	155,51	12129,73	7683,47
RMSE [mm]	0,22	8,96	7,14	12,47	110,14	87,66
R2-score	0,999	0,999	0,999	0,999	0,343	0,871

Dit model vertoont duidelijk een overfittingsgedrag bij de X-waarden. De fouten van de X-waarden zijn heel klein bij de trainingsdata. De Y-waarden zijn iets hoger en vertonen minder overfittingsgedrag.

Vervolgens is het model getest met verschillende maximale dieptes. Uit deze testen is gebleken dat het beste model een diepte van 8 had bij X en een diepte van 10 bij Y. Deze worden getoond in tabel 21.

Tabel 21: XGBoost prestaties met diepte X = 8, diepte Y = 10 en estimators = 50

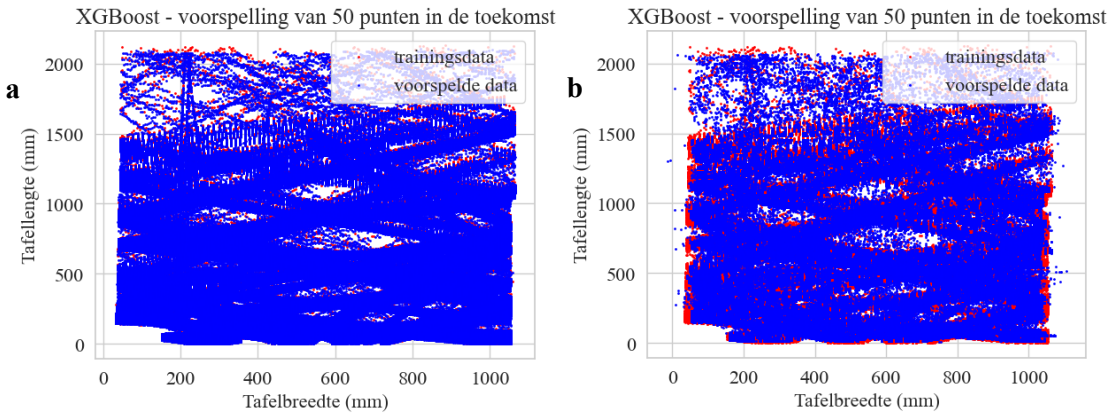
	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	16,75	80,77	17,08	80,99	72,93	82,52
MSE [mm²]	684,76	9648,89	753,77	9758,34	7405	9833,36
RMSE [mm]	26,17	98,23	27,45	98,78	86,05	99,16
R2-score	0,992	0,953	0,991	0,953	0,599	0,834

Als laatste is het model nog verbeterd door de regularisatieparameter lambda toe te voegen. Bij lambda = 0,01 zijn de beste resultaten gevonden (Tabel 22).

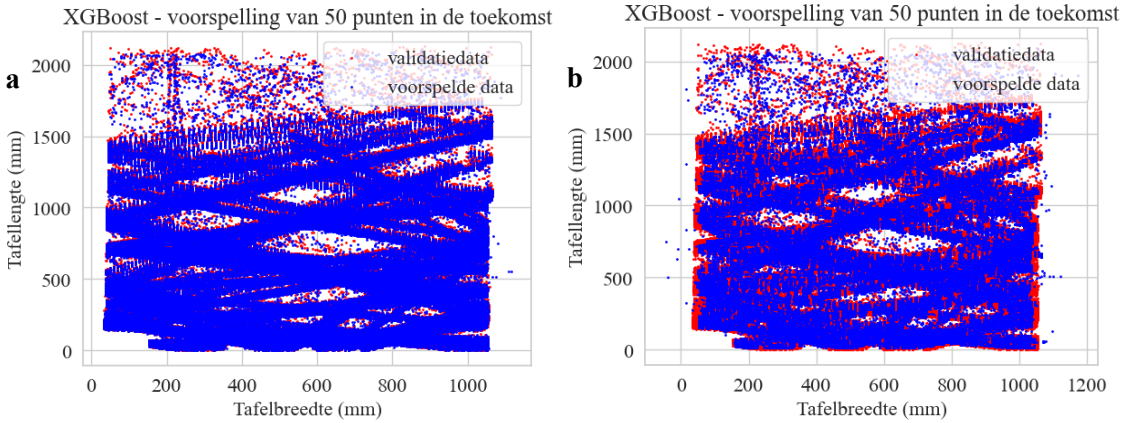
Tabel 22: XGBoost prestaties met diepte X = 8, diepte Y = 10, estimators = 50 en regularisatieparameter = 0,01

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	9,99	15,18	10,53	15,65	39,06	66,96
MSE [mm²]	238,57	368,64	309,65	435,42	2229,65	6816,93
RMSE [mm]	15,45	19,2	17,6	20,87	47,22	82,57
R2-score	0,997	0,998	0,996	0,998	0,879	0,885

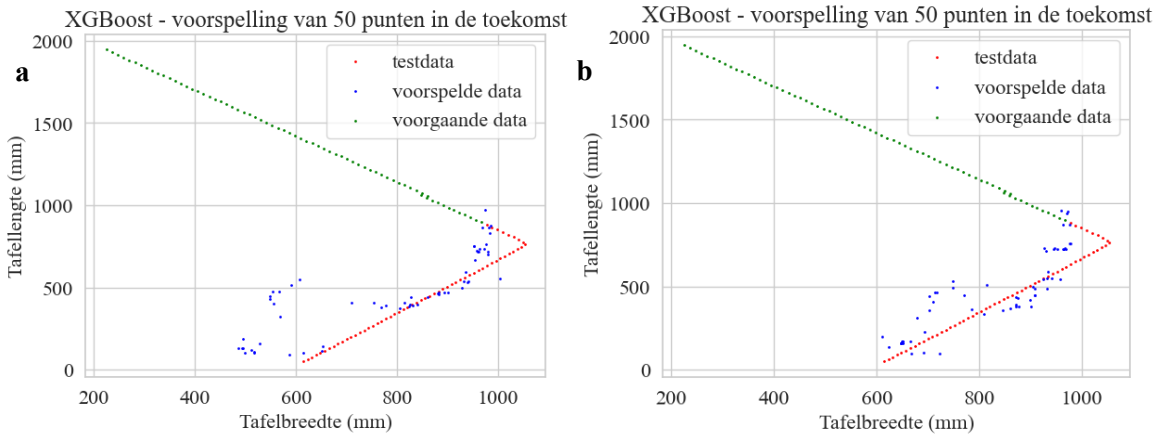
Om een vergelijking te maken tussen de originele hyperparameters en de parameters van het beste model zijn de trainings-, validatie- en testdata van deze modellen vergeleken in Figuur 73 tot en met 75.



Figuur 73: Spreidingsdiagram XGBoost met trainingsdata (a) met diepte X = 20 en diepte Y = 20 en (b) met diepte X = 8, diepte Y = 10, lambda = 0,01 en beide met 50 estimators



Figuur 74: Spreidingsdiagram XGBoost met validatiedata (a) met diepte X = 20 en diepte Y = 20 en (b) met diepte X = 8, diepte Y = 10, lambda = 0,01 en beide met 50 estimators



Figuur 75: Spreidingsdiagram XGBoost met testdata (a) met diepte X = 20 en diepte Y = 20 en (b) met diepte X = 8, diepte Y = 10, lambda = 0,01 en beide met 50 estimators

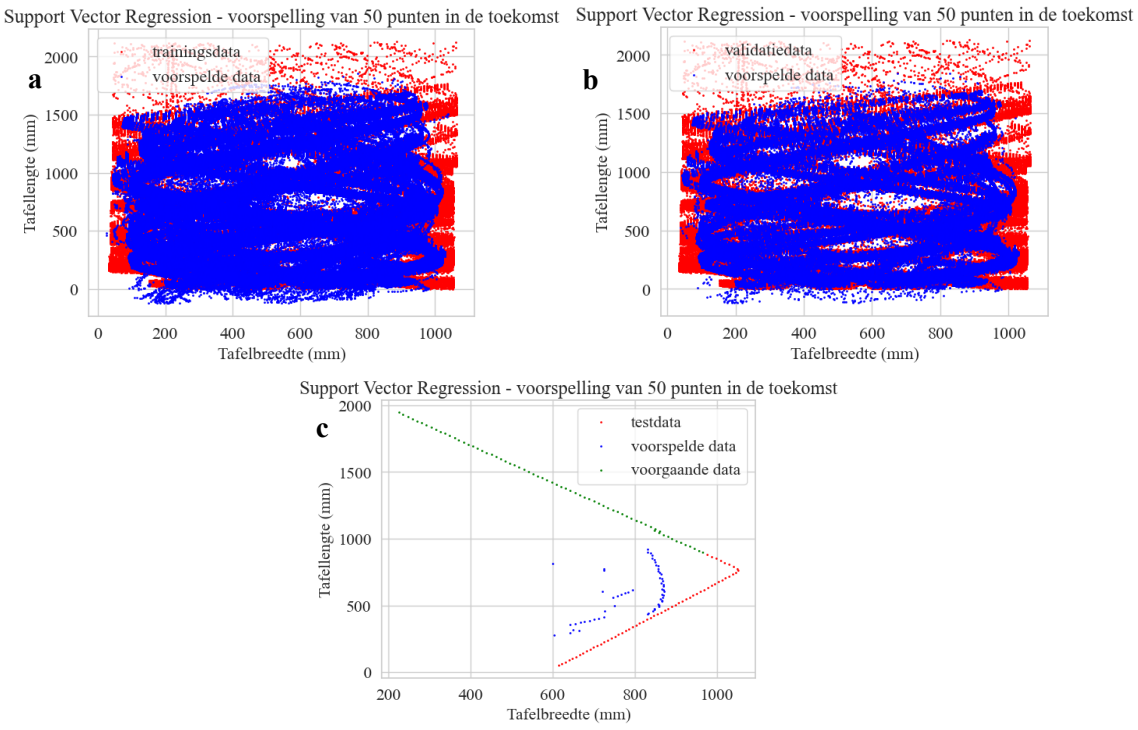
4.3.4 SVM

Naast de lineaire regressie, de decision tree en de ensemble tree is er ook het SVM-regressiemodel of kortweg SVR-model. Dit model heeft verschillende hyperparameters die mogelijk zijn. In het getrainde model zijn er drie parameters ingesteld namelijk: de kernel, de regularisatieparameter c en de epsilon waarde. De SVR heeft als groot nadeel dat dit model lange tijd nodig heeft om te trainen met de trainingsdataset. Het trainen met dezelfde dataset als alle andere modellen duurt hier 15u35m16s terwijl de andere modellen hiervoor slechts minuten of seconden nodig hebben. Ook de duur van het voorspellen neemt veel meer tijd in beslag. De voorspelling van de X- en Y-waarden voor de validatieset duurt bij het SVR-model 2u45m42s terwijl dit bij alle andere modellen slechts enkele seconden duurt. Dit komt omdat bij dit model de voorspeltijd sterk afhankelijk is van de data waarop deze de voorspelling moet uitvoeren. Bij dit model zijn eerst de standaardinstellingen genomen waarvan de resultaten in tabel 23 getoond worden. De standaardinstellingen zijn rbf kernel (radial basis function), regularisatieparameter c = 1 en epsilon ξ gelijk aan 0,1.

Tabel 23: SVR-prestaties met kernel = rbf, c = 1 en epsilon = 0,1

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	39,08	28,06	39,18	28,44	71,49	132,08
MSE [mm²]	5323,15	9813,53	5363,34	10073,14	9821,52	34423,99
RMSE [mm]	72,96	99,06	73,24	100,37	99,1	185,54
R2-score	0,938	0,952	0,937	0,951	0,468	0,421

De resultaten van dit model zijn niet goed in vergelijking met de andere modellen die eveneens maar een kortere trainingstijd nodig hebben. Omdat ook de trainings- en voorspeltijd bij dit model zo tijdrovend is werd er gekozen om niet meer verder te testen met andere hyperparameters op dit model. Figuur 76 toont een spreidingsdiagram van de verschillende sets met hun voorspelde punten.



Figuur 76: Spreidingsdiagram Support Vector Regression met (a) trainingsdata, (b) validatiedata en (c) testdata

4.3.5 Artificieel neuraal netwerk

Het laatste model dat getraind is op de data is het artificieel neuraal netwerk. Hierbij is eerst het neuraal netwerk getraind met de scikit-learn bibliotheek die op de CPU traint. Maar deze functie heeft niet de mogelijkheid om meer dan één CPU-core te benutten tijdens het trainen. Om dit proces te versnellen is er overgeschakeld naar de PyTorch bibliotheek die het mogelijk maakt om het model te trainen met GPU-acceleratie. PyTorch maakt gebruik van de CUDA-cores op een NVIDIA grafische kaart.

4.3.5.1 Scikit-learn met CPU

Om via een MLP neuraal netwerk een regressie uitgang te voorspellen is dit model eerst via de scikit-learn bibliotheek eenvoudig opgesteld. Als basis heeft dit model wel veel mogelijke hyperparameters. In deze masterproef zijn nu het optimaal aantal hidden layers en de hoeveelheid neuronen per hidden layer bepaald als hyperparameter. Standaard wordt op het model al een leersnelheid meegegeven van 0,001 welke voldoende laag is om de gradient descent zo optimaal mogelijk te berekenen. Bij alle modellen wordt er ook een limiet van 500 iteraties vastgelegd, dit is het aantal iteraties dat het model zijn eigen mag corrigeren. Als eerste model is er met een simpel model gestart met één hidden layer met 10 neuronen (Tabel 24).

Tabel 24: MLP scikit-learn prestaties met één hidden layer met 10 neuronen

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	16,61	12,58	16,65	12,81	48,19	60,77
MSE [mm²]	849,3	694,06	860,03	747,6	3311,23	7053,76
RMSE [mm]	29,14	26,35	29,33	27,34	57,54	83,99
R2-score	0,99	0,997	0,989	0,996	0,821	0,881

Het model geeft na het trainen al positievere resultaten voor de testdata dan de meeste andere modellen. Vervolgens is er een model getest met twee hidden layers waarvan, elke hidden layer uit 8 neuronen bestaat (Tabel 25).

Tabel 25: MLP scikit-learn prestaties met twee hidden layers met elk 8 neuronen

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	13,14	6,76	13,26	6,88	41,23	36,34
MSE [mm²]	609,07	183,44	597,56	209,67	2874,07	1906,3
RMSE [mm]	24,68	13,54	24,45	14,48	53,61	43,66
R2-score	0,993	0,999	0,993	0,999	0,844	0,968

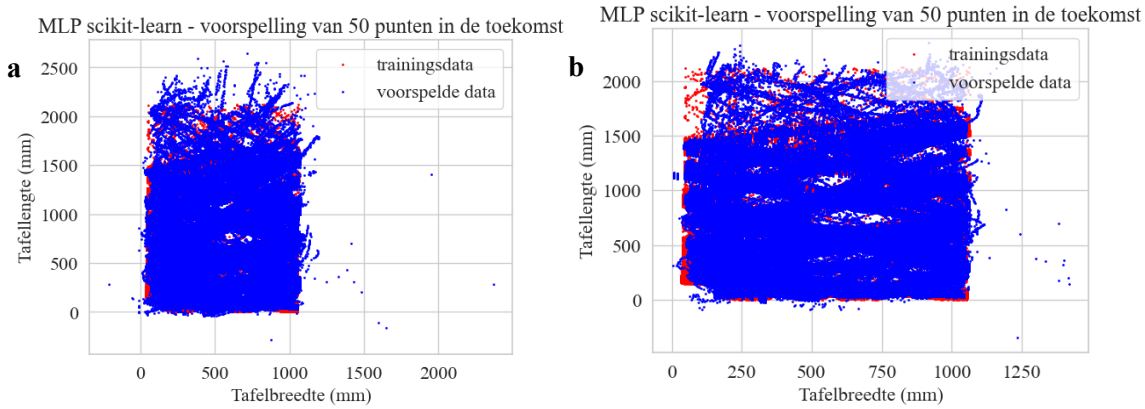
Om het model nog verder te optimaliseren zijn er het aantal neuronen nog verhoogd naar 10 per hidden layer (Tabel 26).

Tabel 26: MLP scikit-learn prestaties met twee hidden layers met elk 10 neuronen

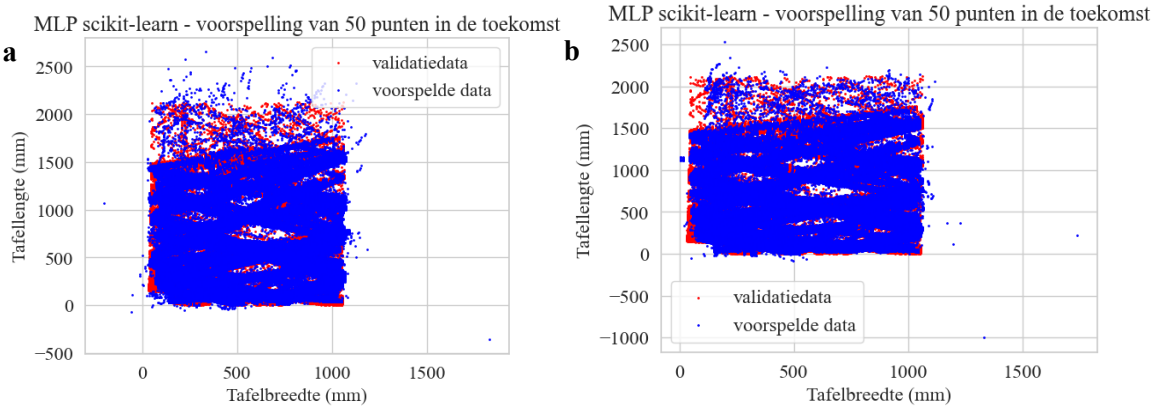
	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	11,3	9,028	11,43	9,168	38,95	33,06
MSE [mm²]	480,21	319,41	519,38	357,11	2395,35	1507,74
RMSE [mm]	21,91	17,87	22,79	18,9	48,94	38,83
R2-score	0,994	0,998	0,994	0,998	0,87	0,975

Dit model presteert het beste op de testdata in vergelijking met de andere hyperparameterinstellingen. Het nadeel van dit model is dan weer dat de trainingstijden veel hoger liggen en zeker als er met nog grotere neuronen en hidden layers gewerkt wordt. De trainingstijd van het beste model hierboven is 11m33s. Dit is al veel minder dan het SVR-model maar de trainingstijd van de andere modellen ligt onder de minuut.

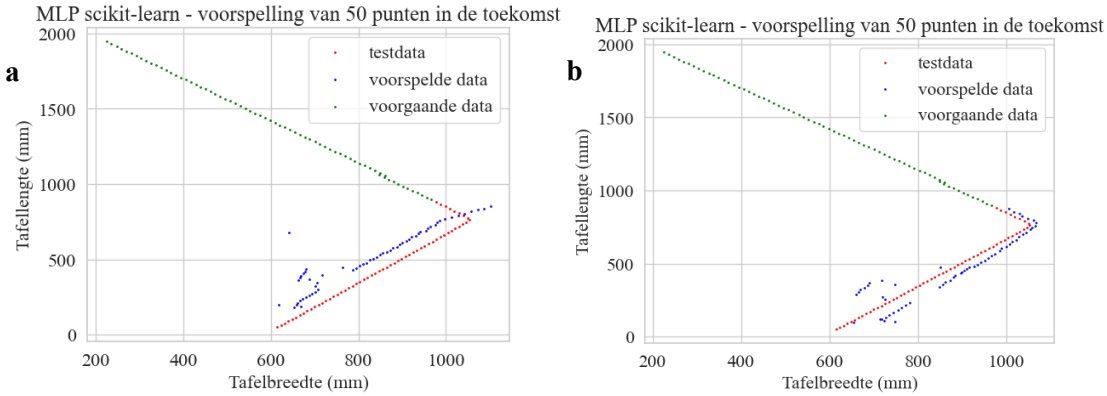
Figuur 77 tot en met 79 tonen de trainings-, validatie- en testdata van deze modellen. Bij de testdata kunnen de betere prestaties van het 10x10 hidden layer model goed worden vastgesteld. Hier wordt het traject bijna perfect gevolgd over de reële data en wordt de hoek ook bij benadering juist voorspeld. Het loopt hier pas fout als de robot een voorspelling moet maken rekening houdend met de verspringing van spelerscamera naar de robotcamera.



Figuur 77: Spreidingsdiagram MLP scikit-learn met trainingsdata (a) met één hidden layer met 10 neuronen en (b) met twee hidden layers van elk 10 neuronen



Figuur 78: Spreidingsdiagram MLP scikit-learn met validatiedata (a) met één hidden layer met 10 neuronen en (b) met twee hidden layers van elk 10 neuronen



Figuur 79: Spreidingsdiagram MLP scikit-learn met testdata (a) met één hidden layer met 10 neuronen en (b) met twee hidden layers van elk 10 neuronen

4.3.5.2 PyTorch met GPU

Om het MLP-model te versnellen is er gebruik gemaakt van de PyTorch bibliotheek. Deze maakt het mogelijk om zoals eerder aangehaald neurale netwerken via GPU te accelereren. Bij deze bibliotheek zijn er gelijkaardige hyperparameters beschikbaar aan het MLP-model van scikit-learn. In deze masterproef is er gekozen om het aantal hidden layers, het aantal neuronen per hidden layer en het aantal iteraties dat het model moet trainen in te stellen als hyperparameters. Standaard staat bij dit model ook al de leersnelheid alpha ingesteld op 0,001; deze is laag genoeg om de minima op te zoeken.

Allereerst zijn de beste parameters van het MLP-model getraind via de scikit-learn bibliotheek. Deze zijn getraind met twee hidden layers met elk tien neuronen. Ook hier is er ingesteld dat het model moet stoppen met trainen na 500 iteraties. Tabel 27 toont de resultaten van het MLP-model van PyTorch getraind met dezelfde hyperparameters.

Tabel 27: MLP PyTorch prestaties met 500 trainingsiteraties, twee hidden layers met elk 10 neuronen

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	293,21	466,51	292,35	446,58	332,41	130,58
MSE [mm²]	128844,58	292544,15	130046,85	296011,52	117137,14	23854,91
RMSE [mm]	358,95	540,87	360,62	544,07	342,25	154,45
R2-score	-0,497	-0,418	-0,517	-0,431	-5,34	0,598

Dit model moet maar enkele seconden trainen maar presteert veel slechter. Dit komt omdat de optimalisatie per iteratie van PyTorch veel minder is dan bij het scikit-learn model. Als volgt is er ontdekt dat 20000 iteraties een goede maatstaf is om de volgende PyTorch modellen te trainen. Tabel 28 bevat de resultaten van het model met dezelfde hyperparameters maar dan getraind tot 20000 iteraties.

Tabel 28: MLP PyTorch prestaties met 20000 trainingsiteraties, twee hidden layers met elk 10 neuronen

	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	11,48	9,5	11,63	9,64	24,93	39,2
MSE [mm²]	468,8	417,07	486,14	442,43	1200,34	3204,8
RMSE [mm]	21,65	20,42	22,05	21,03	3465	56,61
R2-score	0,995	0,998	0,994	0,998	0,935	0,946

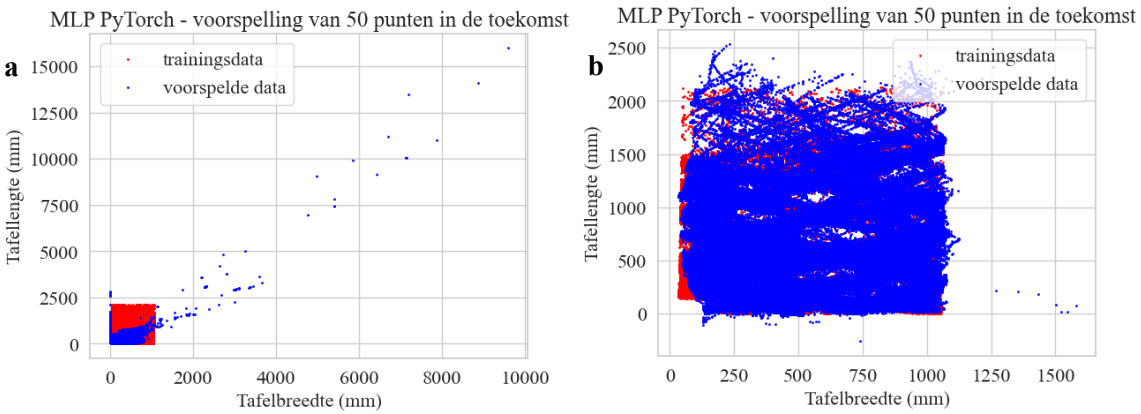
Deze resultaten op de testdata zijn nu beter dan het scikit-learn model. Om het model nog verder te verbeteren en minder complex te maken is er een hyperparameteraanpassing uitgevoerd op het Y-model. Voor de eerste hidden layer is het aantal neuronen van het Y-model aangepast van 10 naar 8 (Tabel 29).

Tabel 29: MLP PyTorch prestaties met 20000 iteraties, twee hidden layers met X = 10x10 neuronen en Y = 8x10 neuronen

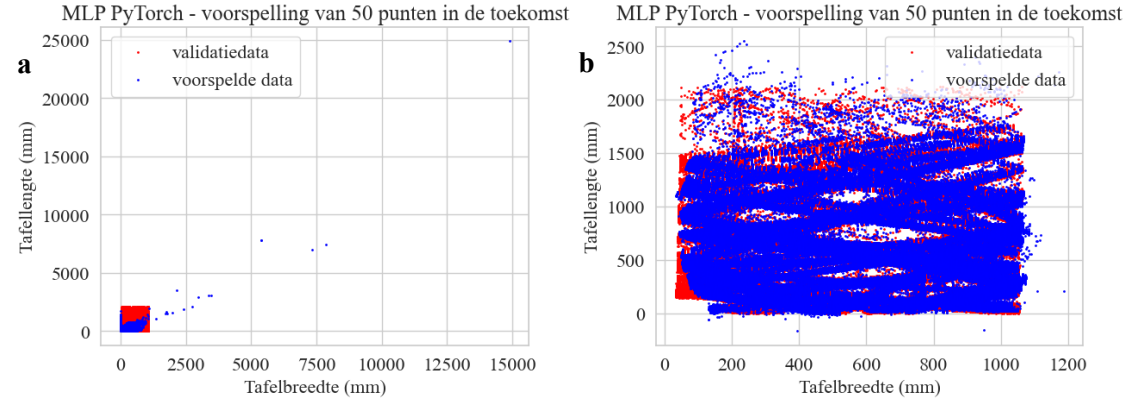
	Trainingsdata		**Validatiedata**		**Testdata**	
	X	**Y**	**X**	**Y**	**X**	**Y**
MAE [mm]	11,58	11,41	11,68	11,54	20,92	28,1
MSE [mm²]	490,37	473,83	509,31	516	737,13	2050,19
RMSE [mm]	22,14	21,77	22,57	22,72	27,15	45,28
R2-score	0,994	0,998	0,994	0,998	0,96	0,965

Met deze hyperparameteraanpassing zijn de fouten op de testdata voor de Y-waarden nog kleiner geworden. Ook kan er hier vastgesteld worden dat de fouten op de testdata voor de X-waarden eveneens verbeterd zijn. Dit is te verklaren doordat de kostfunctie meerdere minima kent en het model niet het globaal minimum (zie paragraaf 2.2.2.3) had gevonden in de vorige trainingsfase. Voor beide modellen is de MAE-fout onder de 30mm waardoor dit het beste model is van alle modellen. Dit model wordt later dan ook gebruikt in de TwinCAT 3 implementatie in hoofdstuk 5.

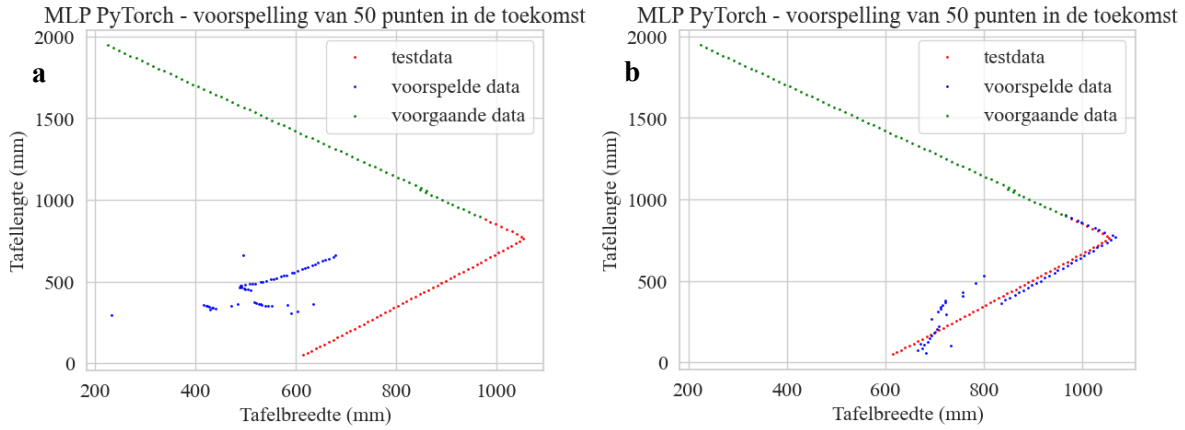
Figuur 80 tot en met 82 tonen de trainings-, validatie- en testdata van deze modellen voor het model van tabel 27 en het beste model uit tabel 29.



Figuur 80: Spreidingsdiagram MLP PyTorch met trainingsdata (a) met twee hidden layers met elk 10 neuronen op 500 iteraties en (b) met twee hidden layers waaruit X = 10x10 en Y = 8x10 neuronen en 20000 iteraties

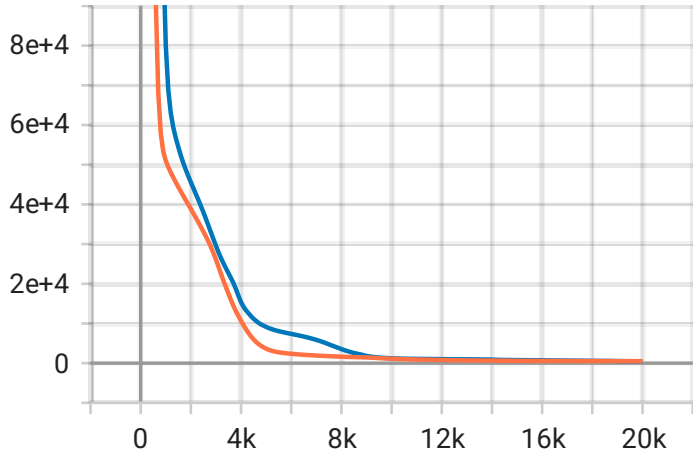


Figuur 81: Spreidingsdiagram MLP PyTorch met validatiedata (a) met twee hidden layers met elk 10 neuronen op 500 iteraties en (b) met twee hidden layers waaruit X = 10x10 en Y = 8x10 neuronen op 20000 iteraties

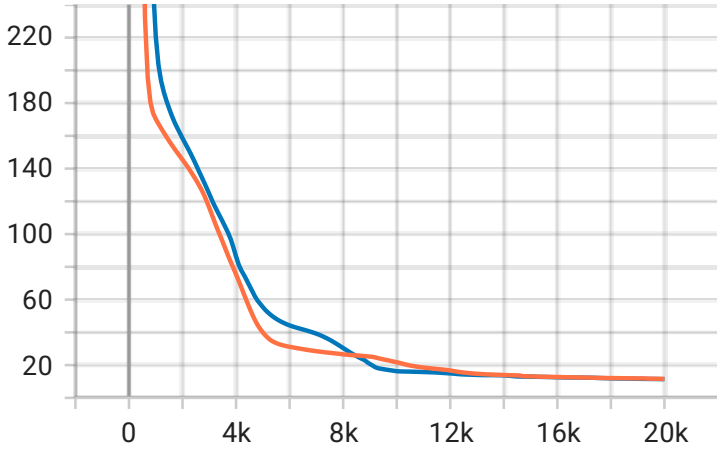


Figuur 82: Spreidingsdiagram MLP PyTorch met testdata (a) met twee hidden layers met elk 10 neuronen op 500 iteraties en (b) met twee hidden layers waaruit X = 10x10 en Y = 8x10 neuronen op 20000 iteraties

Via PyTorch is het ook mogelijk om met de Pythonbibliotheek “Tensorboard” het verbeterproces op de evaluatiemetrieken te loggen over de verschillende trainingsiteraties. In volgende figuren is het verbeterproces op de evaluatiemetrieken zichtbaar voor het beste model uit tabel 29. In het oranje is het X-model aangeduid en in het blauw is het Y-model geplot. Figuur 83 toont de verbetering van het model op de MSE-fout. Dit is de evaluatiemetriek welke het model zo klein mogelijk moest maken. Figuur 84 visualiseert de MAE-fout na de trainingsiteraties. De X-as stelt het aantal trainingsiteraties voor en in de Y-as wordt de fout van de evaluatiemetriek weergegeven.



Figuur 83: MLP PyTorch MSE-fout over 20000 iteraties, twee hidden layers met X = 10x10 neuronen (oranje) en Y = 8x10 neuronen (blauw)



Figuur 84: MLP PyTorch MAE-fout over 20000 iteraties, twee hidden layers met X = 10x10 neuronen (oranje) en Y = 8x10 neuronen (blauw)

4.3.6 Conclusie

In volgende tabellen zijn de hyperparameters met de beste resultaten op de testdata van alle modellen opgesomd. Dit zijn twee tabellen, één met een overzicht voor het voorspellen van het X-label (Tabel 30), een tweede met een overzicht voor het voorspellen van het Y-label (Tabel 31).

Tabel 30: Overzicht prestaties ML-modellen op het X-label

	Hyperparameters	**MAE [mm]**	**MSE [mm²]**	**RMSE [mm]**	**R2-score**
Lineaire regressie	Standaard	183,19	42557,92	206,3	-1,304
Polynomische regressie	2de graad	83,31	9190,9	95,87	0,502
Decision tree	Diepte X = 54	76,68	8492,17	92,15	0,54
Random forest	Diepte X = 20 en 50 estimators	46,1	2739,17	52,34	0,852
ExtraTrees	Diepte X = 20 en 50 estimators	54,88	3769,1	61,39	0,796
Gradient boosting	Diepte X = 20 en 50 estimators	35,86	1754,69	41,89	0,905
LightGBM	Diepte X = 20 en 50 estimators	100,1	12923,5	113,68	0,3
XGBoost	Diepte X = 8; 50 estimators en lambda = 0,01	39,06	2229,65	47,22	0,879
SVR	Kernel = rbf; c = 1 en epsilon = 0,1	71,49	9821,52	99,1	0,468
MLP scikit-learn	Twee hidden layers van 10 neuronen elk	38,95	2395,35	48,94	0,87
MLP PyTorch	Twee hidden layers met X = 10x10 neuronen op 20000 iteraties	20,92	737,13	27,15	0,96

Tabel 31: Overzicht prestaties ML-modellen op het Y-label

	Hyperparameters	**MAE [mm]**	**MSE [mm²]**	**RMSE [mm]**	**R2-score**
Lineaire regressie	Standaard	66,98	5010,53	70,79	0,916
Polynomische regressie	2de graad	102,67	14850,37	121,86	0,75
Decision tree	Diepte Y = 39	131,03	23575,25	153,54	0,603
Random forest	Diepte Y = 20 en 50 estimators	111,77	22377,2	149,59	0,623
ExtraTrees	Diepte Y = 20 en 50 estimators	121,55	29824,61	172,7	0,498
Gradient boosting	Diepte Y = 10 en 50 estimators	83,3	12107,08	110,03	0,796
LightGBM	Diepte Y = 20 en 50 estimators	131,3	23451,86	153,14	0,605
XGBoost	Diepte Y = 10; 50 estimators en lambda = 0,01	66,96	6816,93	82,57	0,885
SVR	Kernel = rbf; c = 1 en epsilon = 0,1	132,08	34423,99	185,54	0,421
MLP scikit-learn	Twee hidden layers van 10 neuronen elk	33,06	1507,74	38,83	0,975
MLP PyTorch	Twee hidden layers met Y = 8x10 neuronen op 20000 iteraties	28,1	2050,19	45,28	0,965

Het MLP-model van PyTorch presteert het meest optimaal voor zowel het X- als het Y-model. Deze modellen worden in hoofdstuk 5 in TwinCAT 3 geïmporteerd om ze dan in de real-time omgeving te gebruiken.

In de derde tabel 32 wordt de vergelijking van de trainingsduur van de verschillende ML-modellen getoond. Dit is de gecombineerde tijd van het X- en Y-model.

Tabel 32: Overzicht trainingstijd X- en Y-label ML-modellen

	Hyperparameters	**Trainingstijd**
Lineaire regressie	Standaard	0,7s
Polynomische regressie	2de graad	10,2s
Decision tree	Diepte X = 54; Diepte Y = 39	4,4s
Random forest	Diepte X = 20; Diepte Y = 20 en 50 estimators	25,4s
ExtraTrees	Diepte X = 20; Diepte Y = 20 en 50 estimators	13,7s
Gradient boosting	Diepte X = 20; Diepte Y = 10 en 50 estimators	2m20s
LightGBM	Diepte X = 20; Diepte Y = 20, 50 estimators	5s
XGBoost	Diepte X = 8; Diepte Y = 10; 50 estimators en lambda = 0,01	27,6s
SVR	Kernel = rbf, c = 1 en epsilon = 0,1	15u35m16s
MLP scikit-learn	Twee hidden layers van 10 neuronen elk voor X en Y	11m33s
MLP PyTorch	Twee hidden layers met X = 10x10 en Y = 8x10 neuronen op 20000 iteraties	3m25

5 TwinCAT 3 machine learning integratie

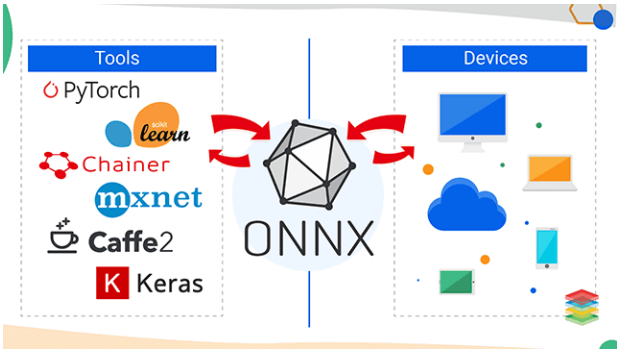
Als laatste stap in de ML-workflow moet het model geïmplementeerd worden in de TwinCAT 3 omgeving. In dit hoofdstuk wordt de implementatie hiervan besproken samen met de programmatie van de robot.

5.1 Python-model naar real-time omgeving

Om het Python-model in TwinCAT 3 te kunnen gebruiken moet dit eerst geconverteerd worden naar een compatibel formaat. Daarna moeten alle data-preprocessing stappen ook uitgevoerd worden in het PLC-programma.

5.1.1 ONNX-bestandsformaat

Het getrainde Python-model moet eerst geëxporteerd worden als een ONNX-bestandsformaat (Open Neural Network Exchange). De getrainde modellen van een bepaald *framework* worden omgezet naar een gestandaardiseerd ONNX-formaat. Dit ONNX-formaat kan dan op zijn beurt geïmporteerd worden in een ander framework dat ONNX-formaten kan herkennen. Dit maakt het mogelijk dat de data-analisten voor het trainen van het ML-model niet één bepaald framework moeten gebruiken maar enkel een framework dat de applicatie van het ONNX-formaat ondersteund. Zij kunnen het ONNX-formaat doorgeven aan de automatiseringsexpert die dan enkel dit model moet inladen in TwinCAT 3. Figuur 85 toont een voorstelling van enkele ondersteunde ML-frameworks die geconverteerd kunnen worden naar een ONNX-formaat om zo in de runtime van de applicatie te gebruiken [17].



Figuur 85: Workflow ONNX-bestandsformaat [48]

Het ONNX-formaat kan eenvoudig in Python gecreëerd worden via de ONNX Pythonbibliotheek of via de eigen bibliotheek van het framework. Het ONNX-formaat kan niet rechtstreeks in de ML-runtime van TwinCAT 3 uitgevoerd worden. Deze moet eerst nog geconverteerd worden naar een door TwinCAT 3 ondersteund formaat. Hier zijn er twee formaten mogelijk namelijk een XML- of een BML-formaat. Beide bestanden bevatten alle informatie van het ML-model. Het verschil tussen de XML- en de BML-bestanden t.o.v. het ONNX-bestand is dat het XML-bestand TwinCAT-specifieke eigenschappen bevat. Het XML-bestand is een bestand dat achteraf nog leesbaar is. Tijdens de engineeringfase kan dit formaat met alle info over het model nog bekeken worden met een XML-lezer. Het BML-bestand daarentegen is een binair formaat wat achteraf niet meer leesbaar is. Dit bestandsformaat heeft als voordeel dat dit minder groot is en dus sneller in de ML-runtime van TwinCAT 3 geladen kan worden. Het is mogelijk om een XML-bestand naar een BML-bestand te converteren maar niet in de omgekeerde richting. Deze bestandsformaten kunnen op drie manieren aangemaakt worden: Met de grafische tool in TwinCAT 3, via de command line interface (CLI) of in Python met de Beckhoff toolbox bibliotheek [17].

5.1.2 Voorspelling in TwinCAT 3

Om de voorspelling in TwinCAT 3 uit te voeren moeten eerst alle data-preprocessing features die in Python zijn toegepast ook in het PLC-programma geïmplementeerd worden:

- De SMA-functionaliteit is in Python ontwikkeld door gebruik te maken van verschillende standaardbibliotheken. In het PLC-programma is deze functie manueel geprogrammeerd.
- De schalering van de features moet ook toegepast worden op al de 22 features in het PLC-programma vooraleer deze door het ML-model worden ingelezen. De schalering is in Python geëxporteerd als een CSV-bestand met voor elke waarde de schaleringparameters. Bij deze modellen is er de standaardschalering gebruikt die twee parameters aanmaakt voor elke feature, namelijk het gemiddelde en de standaarddeviatie. In het PLC-programma wordt onderstaande formule uitgevoerd om de geschaalde feature te bepalen:
  Geschaalde feature = (niet-geschaalde feature – gemiddelde van de feature)/standaarddeviatie van de feature

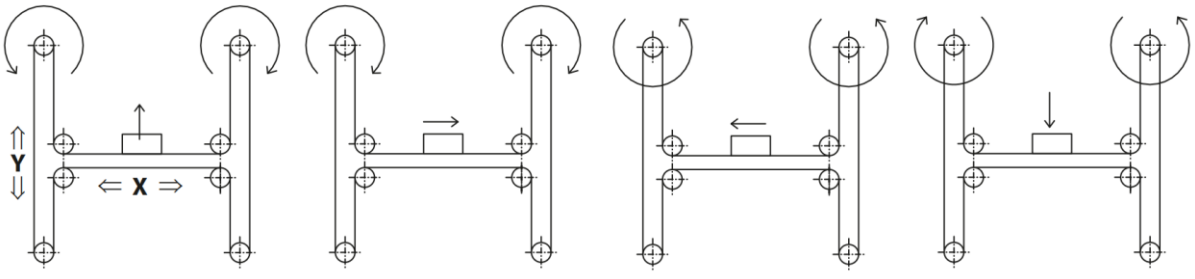
Nadat de features in orde zijn gebracht worden in de X- en Y-positie voorspelling de ML-modellen geïnitialiseerd door het eerder gecreëerde XML-bestand van de ONNX-conversie mee te geven. Daarna worden het aantal features en de te voorspellen labels verstrekt aan de modellen. Als volgt worden de geschaalde features aan de ingang van de predictie functieblok geplaatst. En in de laatste stap wordt de predictie gemaakt. Deze laatste stap wordt bij elke cyclus éénmaal uitgevoerd totdat er een error komt of totdat er een nieuw XML- of BML-model wordt ingeladen.

5.2 H-robot

De airhockeytafel is uitgerust met een H-robot, vandaar de naam robot-airhockeytafel. Aan deze robot is een pusher geconnecteerd. Deze wordt aangestuurd om naar de voorspelde positie van het ML-model te bewegen om zo de puck terug te kunnen spelen. In volgende subparagrafen wordt de hardware besproken alsook de programmatie hiervan.

5.2.1 Hardware

De H-robot is een positioneringssysteem aangedreven door één roterende riem. Hierbij zijn er twee statische Y-assen en één bewegende X-as. Op de X-as is een component bevestigd dat de positie van de robot voorstelt. Dit punt kan naar boven, onder, links of rechts bewegen. Op het uiteinde van elke Y-as is er een aansluitingsbevestiging waar de riemen aangedreven worden door een motor. Aan de hand van de draairichting van de motoren beweegt de positie van de H-robot in een bepaalde richting. Figuur 86 geeft de werking weer van de H-robot. Hierop staan de twee aansluitpunten van de motoren aangeduid met een draairichting. In het midden wordt er aangeduid hoe de XY-positie verandert als de motoren in een bepaalde richting bewegen [49]. De gebruikte H-robot is van het type ELZU 60 van Bahr.

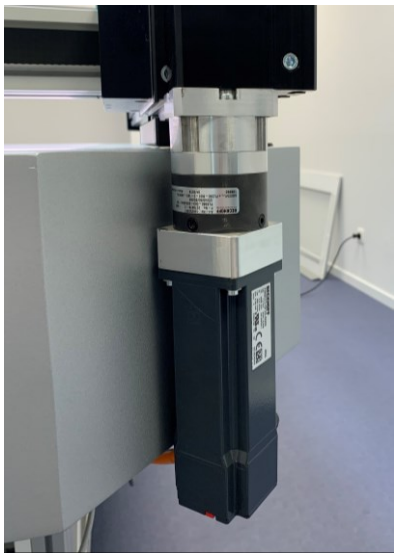


Figuur 86: H-robot werking [49]

Om deze H-robot aan te sturen was de airhockeytafel al voor deze masterproef uitgerust met Beckhoff motoren, reductiekasten en was er al een drive geïnstalleerd:

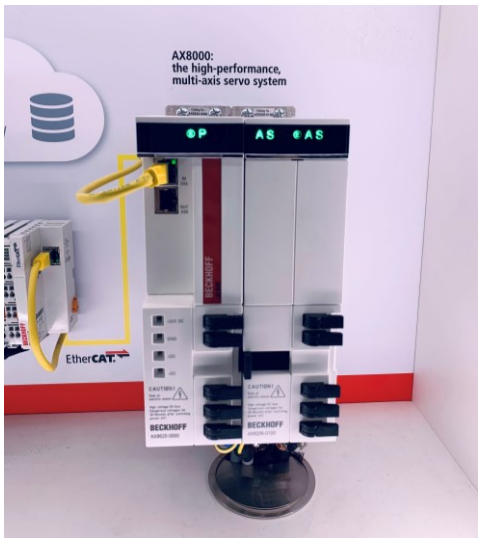
- Als motoren zijn er de AM8023-0F20 synchrone servomotoren geïnstalleerd. Dit zijn permanent magneet motoren met een driefasige synchroonmotor. De motoren hebben een nominaal toerental van 9000toeren/min en een nominaal koppel van 0,9Nm.
- Aan de motoren zijn er compacte planetaire tandwielkasten gekoppeld met een overbrengingsverhouding van 3. Deze zijn van het type AG2250-+PLE60-M01-3-1B1-AM812x.

Figuur 87 toont de combinatie van de motor en de tandwielkast aangesloten op de Y-as van de H-robot.



Figuur 87: Beckhoff motor + tandwielkast combinatie

- Om deze motoren aan te sturen is er gebruik gemaakt van het AX8000 drive-systeem. Dit is een drive systeem ontwikkeld door Beckhoff. Het systeem vertrekt vanuit één voedingsmodule waarna er bijkomende modules geplaatst worden. Deze modules kunnen bestaan uit één-as, twee-assen, een condensator of uit andere modules. In de airhockeytafel bestaat het geconfigureerd systeem uit een AX8620 voedingsmodule gevolgd met een AX8206 twee-assen module (Figuur 88). Deze modules zijn via de veldbus EtherCAT (Ethernet for Control Automation Technology) van Beckhoff verbonden met de CX2072 controller.



Figuur 88: Beckhoff drive combinatie AX8620 + AX8206

5.2.1.1 Motoren van 400V- naar 230V-aansluiting

De robot-airhockeytafel werkte bij aanvang van de masterproef op 400V met een driefasige stekker om zo de maximale snelheid met de motoren te verkrijgen. In het lokaal waar nu onderzoek op de airhockeytafel is uitgevoerd is er geen 400V-aansluiting aanwezig. Er is dan via de Motion Designer tool van Beckhoff nagegaan hoeveel snelheid er op de motoren verloren gaat als de drive met 230V aangestuurd zou worden i.p.v. 400V. Met de originele bekabeling van 400V bereikt de motor zijn nominaal toerental van 9000toeren/min zoals eerder gespecifieerd (Figuur 89).

Figuur 89: Motor op 400V in de Motion Designer

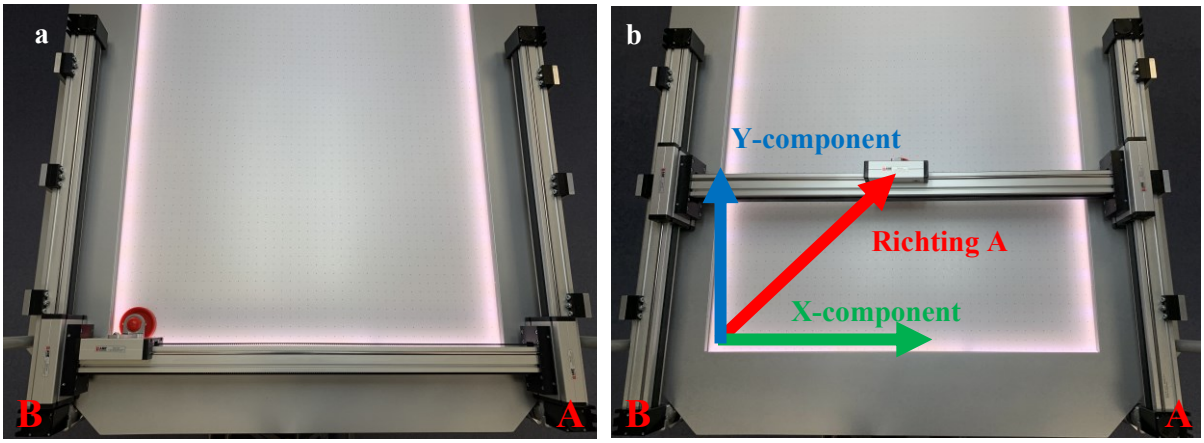
Als de spanning naar de motor wordt aangepast naar 230V heeft de motor een nominaal toerental van 8000toeren/min (Figuur 90).

Figuur 90: Motor op 230V in de Motion Designer

Het toerentalverschil tussen beide spanningen is minimaal waardoor er besloten is om de 400V-stekker om te vormen naar een 230V-aansluiting. Bij deze snelheden is de belangrijkste factor eerder de versnelling van de motoren zijn i.p.v. de effectieve snelheid. Dit maakt dat de robot-airhockeytafel ook breder inzetbaar is nu er geen 400V-aansluiting meer nodig is.

5.2.1.2 Berekening maximale snelheid X- en Y-richting

Het toerental van 8000toeren/min geeft niet de inschatting weer van hoe snel de robot kan bewegen in de X- en Y-richting maar is de snelheid waarbij de motor A in de richting A of motor B in de richting B beweegt. Figuur 91a toont de robot in zijn oorsprongspositie waar de motoren A en B onderaan zijn aangeduid. Als enkel motor A positief wordt aangestuurd (d.w.z. draairichting rechtsom van bovenaf waargenomen) beweegt de pusher schuin op het veld (Figuur 91b). De schuine snelheid van dit traject is hierop aangeduid in het rood. In het blauw en groen zijn de X- en Y-snelheid aangeduid van dit traject. Om de pusher in een specifieke X- of Y-richting te bewegen buiten deze diagonale lijn moeten beide motoren worden aangestuurd.



Figuur 91: H-robot bepaling X- en Y-component waar bij (a) de robot in zijn oorsprong is opgesteld en waar bij (b) de robot bewogen is in de positieve richting van motor A

In de volgende berekening wordt er uiteengezet hoe de maximale snelheid van de X- en Y-richting is bekomen vertrekkend vanaf de nominale motorsnelheid.

1. Eerst wordt het aantal toeren/min omgezet naar toeren/s: (8000toeren/min)/60 = 133,33toeren/s.
2. Hierna wordt dit gedeeld door de overbrengingsverhouding: (133,33toeren/s)/3 = 44,44toeren/s.
3. Als volgt wordt het aantal toeren/s omgezet naar mm/s. Hiervoor moet er bepaald worden wat de omtrek van de as is. Dit is in dit geval 130mm, wat overeenstemt met 130mm per omwenteling. De berekening is dan: 44,44toeren/s x 130mm = 5777,77mm/s.
4. Zoals eerder werd aangehaald beweegt de pusher schuin indien de motor A of de motor B alleen roteert. Om de X- en Y-component van dit traject te verkrijgen moet de rotatiesnelheid gedeeld worden door $\sqrt{2}$, dit wordt dan (5777,77mm/s)/$\sqrt{2}$ = 4085,5mm/s.

Uit de berekening is gebleken dat 4085,5mm/s de maximale snelheid is waarmee de H-robot in de X- of Y-richting kan bewegen.

5.2.2 Configuratie

In TwinCAT 3 is eerst de H-robot geconfigureerd waarbij volgende stappen chronologisch gevolgd zijn om deze op punt te stellen:

1. De eerste configuratiestap is het aan elkaar koppelen van de motoren A en B dat ervoor zorgt dat de robot bewegingen kan maken door een X- en/of Y-positie aan te sturen. Aan de drive zijn twee reële assen gekoppeld, namelijk motor A en B. Verder zijn er twee virtuele assen aangemaakt namelijk motor X en Y waarmee later de positie van de robot aangestuurd wordt. In TwinCAT 3 is er de mogelijkheid om kinematische modellen te maken. Deze kinematische modellen maken het mogelijk om de bewegingen van een bepaalde motorconfiguratie om te vormen naar een gedefinieerd model. Één van de beschikbare transformaties in TwinCAT 3 is de H-bot-transformatie die op de H-robot toegepast kan worden. Hierbij worden de bewegingen van A en B getransformeerd tot de virtuele assen X en Y. Als deze transformatie ingeschakeld wordt is het mogelijk om een X- en Y-beweging aan te sturen zodat de motoren A en B hun nodige bewegingen kunnen uitvoeren.
2. Als volgt is er een NCI-kanaal (Numerical Control Interpolation) aangemaakt op de X- en Y-as. Het NCI-kanaal maakt het mogelijk om op deze assen een G-code uit te voeren. G-code zijn één of meerdere coderegels met instructies die door de robot uitgevoerd worden. Een voorbeeld hiervan is de G-coderegel “G01 X100 Y200 F4000”:
    - G maakt duidelijk welke soort beweging de robot moet uitvoeren met deze regel en 01 staat voor een lineaire beweging.
    - X geeft mee naar welke X-positie de robot moet bewegen, in dit geval is dit 100mm.
    - Y geeft mee naar welke Y-positie de robot moet bewegen, in dit geval is dit 200mm.
    - F duidt de snelheid aan waaraan deze beweging uitgevoerd wordt. In dit voorbeeld is dit 4000mm/s.

    Met deze G-code instructies wordt de robot later aangestuurd om de puck te onderscheppen.
3. Om de robot juist aan te sturen is het nulpunt van de airhockeytafel aangeleerd. Er zijn limieten geplaatst op hoe ver de robot mag bewegen in de X- en Y-richting zodat deze niet tegen de rand botst.
4. Als laatste is de maximale snelheid en versnelling in de robot geconfigureerd. De berekende maximale snelheid werd afgerond op 4000mm/s en de versnelling werd ingesteld op 0,3s. Na deze versnelling moet de robot de snelheid bereiken waarop hij was aangestuurd. Een verlaging van de snelheid zorgt voor schokkende bewegingen van de airhockeytafel en is dus niet aan te raden.

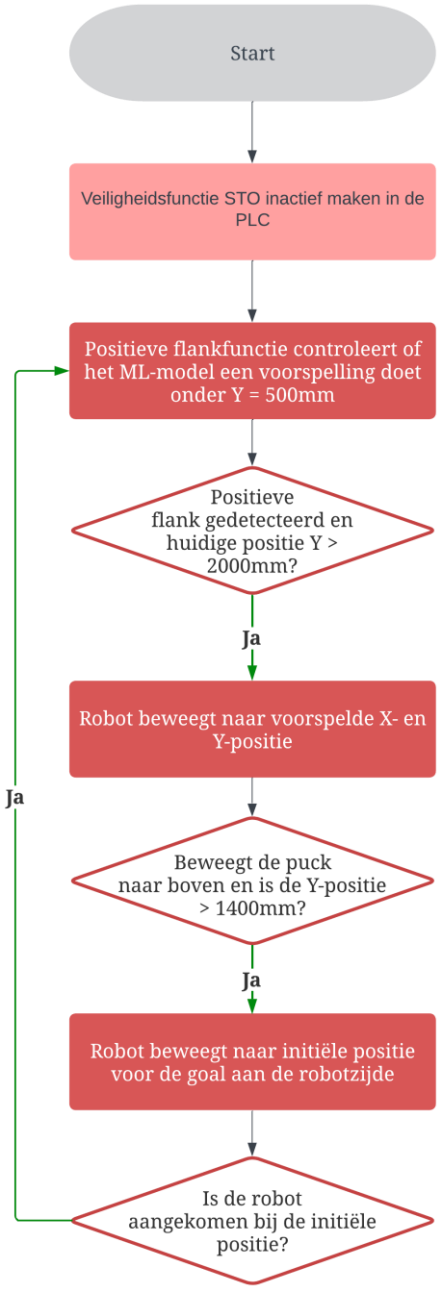
5.2.3 Programmatie

Na de configuratie wordt de robot geprogrammeerd om de puck te onderscheppen. Hierbij wordt de volgende procedure gevolgd:

1. De airhockeytafel is uitgerust met twee noodstoppen en een drive die de mogelijkheid heeft om veiligheidsfuncties op de motoren uit te voeren. Eerst moeten beide noodstoppen die geconnecteerd zijn aan de PLC niet actief zijn. Daarna moet de STO (Safe Torque Off) veiligheidsfunctie van de drive inactief gemaakt worden door deze te resetten en te bevestigen in de PLC-code.
2. Als volgt wacht de robot op een positieve flank als er door het ML-model een Y-positie onder 500mm voorspeld wordt en de huidige Y-positie onder 2000mm is. Dit zijn twee figuurlijke lijnen aangemaakt op het speelveld zoals bij een gelijkaardig onderzoek in [42] (zie paragraaf 2.3.3) ook was toegepast.

3. Indien aan deze voorwaarden voldaan zijn beweegt de pusher naar deze voorspelde X- en Y-positie. Er wordt pas bewogen als de puck onder de 2000mm is, dit is 200mm onder het begin van het speelveld van de speler. Zo wordt er voorkomen dat er geen voorspelling gemaakt wordt bij de kleinste beweging. Ook is er een vaste waarde aangeduid waaronder de puck wordt voorspeld vooraleer de robot beweegt. Deze is niet op de maximale Y-afstand van de robot geplaatst zodat deze extra tijd heeft om zijn voorspelling uit te kunnen voeren.
4. Hierna wordt er gecontroleerd of de puck naar boven aan het bewegen is en of de puck zijn Y-positie groter is dan 1400mm. Indien dit het geval is wordt de robot verplaatst naar de initiële positie bij het doelpunt van de robotzijde. Deze stap wordt niet uitgevoerd als de robot al een beweging aan het maken is of als deze al op zijn initiële positie staat.
5. Nu wordt de cyclus van de robot terug herhaald vertrekkend vanaf stap 2.

In figuur 92 wordt bovenstaande workflow gevisualiseerd.



Figuur 92: Workflow programmatie robot

5.2.4 Evaluatie robot verdediging

De robot kan nu de puck onderscheppen met de voorspelling van het ML-model. De robot heeft de mogelijkheid om te scoren maar de aanvalstechnieken zijn nog niet geprogrammeerd. Als de robot scoort is dit eerder een toevalstreffer. Het model is hier vooral getraind op trajecten waarbij de puck beweegt vertrekkend vanaf de rechterkant van de speler naar de linkerkant van de airhockeytafel. Indien dit traject bij het spelen gevolgd wordt kan de robot de puck onderscheppen met een nauwkeurigheid van circa 50%. Als de puck een traject aflegt in een andere richting dan daalt de nauwkeurigheid. Dit is minder nauwkeurig dan vooropgesteld werd maar is te verklaren door variaties in de data die er momenteel nog niet volledig uitgehaald zijn zoals bijvoorbeeld de overgang tussen de twee camera's die nog niet optimaal is. De timestamps en de X- en de Y-positie in de data worden hierdoor nog te sterk beïnvloed. De robot is dus momenteel minder nauwkeurig dan vroeger met de wiskundige bepaling met één camera. Hier was de nauwkeurigheid circa 80%. Er kunnen dus zeker nog optimalisaties toegepast worden die besproken worden in 6.2.

6 Besluit

In het laatste hoofdstuk wordt er teruggeblikt op de originele doelstellingen en de bekomen resultaten. Verder worden er mogelijke verbeteringen voor de toekomst aangehaald.

6.1 Terugblik op de doelstellingen

In deze masterproef is er ML-onderzoek uitgevoerd op de robot-airhockeytafel met als doel de trajectbepaling van de puck nauwkeuriger te maken. Hier zijn alle stappen van de ML-workflow aanbod gekomen. Eerst is er onderzoek uitgevoerd op verschillende ML-technieken en andere gelijkaardige onderzoeken. Via de camera's en de PLC is er dan een procedure opgemaakt om data te verzamelen. Deze data zijn dan verder geoptimaliseerd met diverse data pre-processing stappen. Hierna zijn de onderzochte modellen getraind en geëvalueerd met diverse hyperparameters. Het beste model is dan in TwinCAT 3 geïmporteerd. Hierna is dan de H-robot geprogrammeerd om de puck te onderscheppen met het getrainde model. Het resultaat na het uitvoeren met de ML-workflow is dat de positie van de puck 50 frames in de toekomst voorspeld kunnen worden met een nauwkeurigheid van 3cm rond de puck. Dit is voldoende klein om het traject van de puck te voorspellen en te raken met de pusher.

De originele doelstellingen van deze masterproef waren dat de robot de puck met minstens 90% nauwkeurigheid moest kunnen terugspelen en ook effectief moest kunnen aanvallen naar de speler. Zoals gebleken is bij de evaluatie van de robot zijn de resultaten nog niet optimaal daar de robot nu nog maar met 50% nauwkeurigheid de puck kan onderscheppen. Door het uitwerken van de vooropgestelde ML-workflow en door tijdsgebrek was het niet mogelijk om de robot aanvalspatronen aan te leren tijdens deze masterproef. De robot kan nu enkel verdedigen. Momenteel zijn de resultaten ook nog niet optimaal genoeg om de airhockeytafel als demo door Beckhoff Automation BV in te zetten. Aansluitend op deze masterproef zijn aan de applicatie dus nog verschillende optimalisaties mogelijk die in de volgende paragraaf kort besproken worden.

6.2 Toekomstige optimalisatie

Onderstaand zijn enkele optimalisaties opgesomd welke nog verder onderzocht en uitgevoerd kunnen worden op de robot-airhockeytafel.

Onderstaande optimalisaties kunnen nog verder onderzocht en uitgevoerd worden op de robot-airhockeytafel.

- Visie optimalisatie:
    - Beckhoff Automation heeft zelf camera's en lenzen ontwikkeld die de huidige camera's op de airhockeytafel kunnen vervangen. Deze worden op de markt verwacht tegen het einde van 2023. Hiermee kan Beckhoff Automation BV met de robot-airhockeytafel de nieuwe uitbreiding van hun visie gamma aantonen. Alsook hebben deze camera's de mogelijkheid om via EtherCAT te communiceren. Dit zou het mogelijk maken om de timestamps van de camera's volledig te synchroniseren met de PLC. Hiermee detecteert de PLC geen grote sprongen meer in de timestamps als de puck van de ene camera naar de andere overgaat.

- Door de tweede camera is het speelveld waarop de puck gedetecteerd kan worden uitgebreid. De huidige LED-belichting kan 75% van het speelveld belichten terwijl beide camera's nu echter samen 95% van het speelveld kunnen detecteren. Het verschil van 20% aan de spelerszijde wordt momenteel niet belicht door de huidige LED-belichting van de airhockeytafel maar door externe belichting. Zonder de externe belichting is het veld te donker en wordt de puck niet meer gedetecteerd. Het is dus aan te raden om de extra zone ook bijkomend te belichten.

- Model optimalisatie:
  Na het onderzoek is er opgemerkt dat een neuraal netwerk de beste prestaties geeft om het traject van de puck te voorspellen. Verdere optimalisatietechnieken zouden kunnen worden toegepast op dit model:
  - Een eerste mogelijkheid is om meer data te creëren door bijkomende verschillende soorten trajecten uit te loggen. Nu is er vooral gefocust geweest op trajecten vertrekkend vanaf de rechterkant van de speler die dan naar links speelt. Er kunnen dan diverse andere trajecten toegevoegd worden aan de trainingsdata zoals bijvoorbeeld het traject vertrekkende links van de speler of vertrekkende van de robotzijde of ….
  - Er is in deze masterproef geopteerd om te focussen op het model die de X- en Y-positie voorspeld voor 50 frames in de toekomst. Er zijn ook modellen getest met andere hoeveelheden frames in de toekomst. Bij een kleiner aantal frames in de toekomst zijn de prestaties beter en bij een groter aantal zijn deze prestaties slechter. Er kan een combinatie gemaakt worden van een voorspelling van 50 frames in de toekomst als initiële voorspelling. Na deze voorspelling kan deze bijvoorbeeld bijgestuurd worden met een ander model welk getraind is op 20 frames in de toekomst.
  - Nu wordt elk punt van het traject van de puck voorspeld wat voor veel variatie in de data kan zorgen. Als alternatief kan de werkwijze van [42] (zie paragraaf 2.3.3) verder onderzocht worden. Hierbij wordt enkel de X-positie voorspeld op een specifieke Y-positie.
  - Als laatste mogelijkheid kan er gekeken worden naar de reinforcement learning applicaties vermeldt in 2.3.1. Hier moet dan eerst de mogelijkheid van reinforcement learning met Beckhoff hardware en software verder onderzocht worden.

- Aanvalstechnieken:
  Om de robot te voorzien van aanvalstechnieken kan het onderzoek van [41] (vermeldt in 2.3.2) verder opgenomen worden. Hierbij worden er door middel van classificatie modellen bepaald a.d.h.v. het traject van de puck of de robot moet aanvallen, verdedigen of niets doen. Hierop volgend moet de robot dan nog aangestuurd worden hoe hij moet aanvallen naar het doel van de speler.

- Visualisatie:
  Om de robot-airhockeytafel visueel aantrekkelijker te maken met nog meer TwinCAT 3 functionaliteiten moet er nog een visualisatie ontwikkeld worden. Dit is mogelijk met een HMI (Human Machine Interface) waarbij het voorspelde en het reële traject op een bedieningspaneel worden voorgesteld. Hierop kan ook een puntensysteem getoond worden wat de huidige score aantoont van de speler en de robot. Ook zijn er IoT-functionaliteiten (Internet of Things) beschikbaar waarbij de PLC met de cloud verbonden kan worden. Zo kunnen verschillende parameters van de applicatie via het internet beschikbaar gesteld worden.

Na de bijkomende uitvoering van deze verdere onderzoeken kunnen alle vooropgestelde doelstellingen volledig gehaald worden. Dit maakt het mogelijk voor Beckhoff Automation BV om de robot-airhockeytafel als demo in te zetten op evenementen om zo de verschillende Beckhoff technologieën te tonen aan de potentiële kopers.

Referentielijst

[1] Beckhoff Automation BV, “Beckhoff Automation BV.” 2019.

[2] Beckhoff Automation BV, “Beckhoff Automation BV.” 2022.

[3] S. J. (Stuart J. Russell, P. $ Norvig, and M.-W. Chang, *Artificial intelligence: a modern approach*, Fourth edition. in Pearson series in artificial intelligence. Harlow: Pearson, 2022.

[4] M. A. Boden, “GOFAI,” in *The Cambridge Handbook of Artificial Intelligence*, K. Frankish and W. M. Ramsey, Eds., Cambridge: Cambridge University Press, 2014, pp. 89–107. doi: DOI: 10.1017/CBO9781139046855.007.

[5] S. Russell and P. Norvig, *Artificial intelligence: A modern approach, global edition*, 4th ed. London, England: Pearson Education, 2021.

[6] G. Rebala, A. Ravi, and S. Churiwala, “Machine Learning Definition and Basics,” in *An Introduction to Machine Learning*, G. Rebala, A. Ravi, and S. Churiwala, Eds., Cham: Springer International Publishing, 2019, pp. 1–17. doi: 10.1007/978-3-030-15729-6_1.

[7] J. Peng, E. C. Jury, P. Dönnes, and C. Ciurtin, “Machine Learning Techniques for Personalised Medicine Approaches in Immune-Mediated Chronic Inflammatory Diseases: Applications and Challenges,” *Front Pharmacol*, vol. 12, Sep. 2021, doi: 10.3389/fphar.2021.720694.

[8] G. Rebala, A. Ravi, and S. Churiwala, “Anomaly Detection,” in *An Introduction to Machine Learning*, G. Rebala, A. Ravi, and S. Churiwala, Eds., Cham: Springer International Publishing, 2019, pp. 153–167. doi: 10.1007/978-3-030-15729-6_13.

[9] K. Koyuncu and L. Tavacioğlu, “Forecasting shanghai containerized freight index by using time series models,” *Mar. Sci. Technol. Bull.*, pp. 426–434, 2021, doi: 10.33714/masteb.1024663.

[10] O. Kramer, “Scikit-Learn,” in *Machine Learning for Evolution Strategies*, O. Kramer, Ed., Cham: Springer International Publishing, 2016, pp. 45–53. doi: 10.1007/978-3-319-33383-0_5.

[11] M. Fuchs, “Feature Scaling with Scikit-Learn,” Aug. 31, 2019. https://michael-fuchs-python.netlify.app/2019/08/31/feature-scaling-with-scikit-learn/#standard-scaler (accessed May 22, 2023).

[12] T. Chai and R. R. Draxler, “Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature,” *Geosci Model Dev*, vol. 7, pp. 1247–1250, Jun. 2014, doi: 10.5194/gmd-7-1247-2014.

[13] G. Rebala, A. Ravi, and S. Churiwala, “Testing the Algorithm and the Network,” in *An Introduction to Machine Learning*, G. Rebala, A. Ravi, and S. Churiwala, Eds., Cham: Springer International Publishing, 2019, pp. 95–102. doi: 10.1007/978-3-030-15729-6_8.

[14] Y. Baştanlar and M. Özuysal, “Introduction to Machine Learning,” in *miRNomics: MicroRNA Biology and Computational Analysis*, M. Yousef and J. Allmer, Eds., Totowa, NJ: Humana Press, 2014, pp. 105–128. doi: 10.1007/978-1-62703-748-8_7.

[15] M. H. Jarrahi, A. Memariani, and S. Guha, “The Principles of Data-Centric AI (DCAI),” 2022, doi: 10.48550/arxiv.2211.14611.

[16] H. Patel, “Data-Centric Approach vs Model-Centric Approach in Machine Learning,” Mar. 27, 2023. https://neptune.ai/blog/data-centric-vs-model-centric-machine-learning (accessed May 22, 2023).

[17] Beckhoff Automation GmbH & Co. KG, “TwinCAT 3 | Machine Learning- und Neural Network Inference Engine,” *Version: 1.6.2*, 2023. https://download.beckhoff.com/download/Document/automation/twincat3/TF38x0_TC3_ML_NN_Inference_Engine_EN.pdf (accessed May 20, 2023).

[18] S. Badillo *et al.*, “An Introduction to Machine Learning,” *Clin Pharmacol Ther*, vol. 107, no. 4, pp. 871–885, 2020, doi: https://doi.org/10.1002/cpt.1796.

[19] W. Ngaw, “Linear Regression,” Apr. 07, 2019. https://wngaw.github.io/linear-regression/#linear-regression-with-scikit-learn (accessed May 29, 2023).

[20] G. Rebala, A. Ravi, and S. Churiwala, “Improving Further,” in *An Introduction to Machine Learning*, G. Rebala, A. Ravi, and S. Churiwala, Eds., Cham: Springer International Publishing, 2019, pp. 41–56. doi: 10.1007/978-3-030-15729-6_4.

[21] D. Forsyth, “Regression,” in *Applied Machine Learning*, D. Forsyth, Ed., Cham: Springer International Publishing, 2019, pp. 205–244. doi: 10.1007/978-3-030-18114-7_10.

[22] G. Rebala, A. Ravi, and S. Churiwala, “Random Forests,” in *An Introduction to Machine Learning*, G. Rebala, A. Ravi, and S. Churiwala, Eds., Cham: Springer International Publishing, 2019, pp. 77–94. doi: 10.1007/978-3-030-15729-6_7.

[23] N. Deshpande, S. Gite, and R. Aluvalu, “A review of microscopic analysis of blood cells for disease detection with AI perspective,” *PeerJ Comput Sci*, vol. 7, p. e460, Apr. 2021, doi: 10.7717/peerj-cs.460.

[24] L. Breiman, J. H. Friedman, and R. A. Olshen, *Classification and regression trees*. in Wadsworth and Brooks/Cole statistics/probability series. London: Chapman and Hall, 1993.

[25] E. Sevinç, “An empowered AdaBoost algorithm implementation: A COVID-19 dataset study,” *Comput Ind Eng*, vol. 165, p. 107912, Jan. 2022, doi: 10.1016/j.cie.2021.107912.

[26] Jason Brownlee, “Histogram-Based Gradient Boosting Ensembles in Python,” Dec. 28, 2020. https://machinelearningmastery.com/histogram-based-gradient-boosting-ensembles/#:~:text=Boosting%20With%20LightGBM-,Histogram%20Gradient%20Boosting,models%20to%20an%20ensemble%20sequentially. (accessed May 30, 2023).

[27] G. Ke *et al.*, “LightGBM: A Highly Efficient Gradient Boosting Decision Tree,” in *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

[28] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” *CoRR*, vol. abs/1603.02754, 2016, [Online]. Available: http://arxiv.org/abs/1603.02754

[29] M. Kubat, “Inter-Class Boundaries: Linear and Polynomial Classifiers,” in *An Introduction to Machine Learning*, M. Kubat, Ed., Cham: Springer International Publishing, 2015, pp. 65–90. doi: 10.1007/978-3-319-20010-1_4.

[30] A. J. Smola and B. Schölkopf, “A tutorial on support vector regression,” *Stat Comput*, vol. 14, no. 3, pp. 199–222, 2004, doi: 10.1023/B:STCO.0000035301.49549.88.

[31] “Support Vector Regression (SVR) using linear and non-linear kernels.” https://scikit-learn.org/stable/auto_examples/svm/plot_svm_regression.html#sphx-glr-auto-examples-svm-plot-svm-regression-py (accessed May 29, 2023).

[32] A. V Joshi, “Perceptron and Neural Networks,” in *Machine Learning and Artificial Intelligence*, A. V Joshi, Ed., Cham: Springer International Publishing, 2020, pp. 43–51. doi: 10.1007/978-3-030-26622-6_5.

[33] D. Durstewitz, G. Koppe, and A. Meyer-Lindenberg, “Deep neural networks in psychiatry,” *Mol Psychiatry*, vol. 24, p. 1, Feb. 2019, doi: 10.1038/s41380-019-0365-9.

[34] D. Forsyth, “Simple Neural Networks,” in *Applied Machine Learning*, D. Forsyth, Ed., Cham: Springer International Publishing, 2019, pp. 367–398. doi: 10.1007/978-3-030-18114-7_16.

[35] R. Ng, “Neural Networks (Learning).” https://www.ritchieng.com/neural-networks-learning/ (accessed May 31, 2023).

[36] W. Ngaw, “Neural Networks,” May 06, 2019. https://wngaw.github.io/neural-networks/ (accessed May 31, 2023).

[37] A. Taitler and N. Shimkin, “Learning Control for Air Hockey Striking using Deep Reinforcement Learning,” *CoRR*, vol. abs/1702.08074, 2017, [Online]. Available: http://arxiv.org/abs/1702.08074

[38] Barkhausen Institut, “AI playing AirHockey,” Nov. 05, 2020.

[39] A. Rakhmati, “Deep Learning Air Hockey Robot,” Aug. 14, 2018. https://github.com/arakhmati/deep-learning-air-hockey-robot (accessed May 31, 2023).

[40] C.-L. Chang, S.-T. Chen, C.-Y. Chang, and Y.-C. Jhou, “Application of machine learning in air hockey interactive control system,” *Sensors (Basel)*, vol. 20, no. 24, pp. 1–22, 2020, doi: 10.3390/s20247233.

[41] K. Igeta and A. Namiki, “A decision-making algorithm for an air-hockey robot that decides actions depending on its opponent player’s motions,” in *ROBIO*, IEEE, 2015, pp. 1840–1845. doi: 10.1109/ROBIO.2015.7419040.

[42] J. Il Park, C. B. Partridge, and M. W. Spong, “Neural network-based state prediction for strategy planning of an air hockey robot,” *J Robot Syst*, vol. 18, no. 4, pp. 187–196, 2001, doi: 10.1002/rob.1015.

[43] Beckhoff Automation GmbH & Co. KG, “TwinCAT automation software.”

[44] Beckhoff Automation GmbH & Co. KG, “TwinCAT 3 | Vision,” Dec. 15, 2022. https://download.beckhoff.com/download/document/automation/twincat3/TF7000-TF7300_TC3_Vision_en.pdf (accessed May 31, 2023).

[45] Allied Vision, “Mako_G-030_DataSheet,” Jul. 18, 2022. https://cdn.alliedvision.com/fileadmin/pdf/en/Mako_G-030_DataSheet_en.pdf (accessed May 31, 2023).

[46] MathWorks, “What Is Camera Calibration?” https://www.mathworks.com/help/vision/ug/camera-calibration.html (accessed Jun. 01, 2023).

[47] C. E. Shannon, “Communication In The Presence Of Noise,” *Proceedings of the IEEE*, vol. 86, no. 2, pp. 447–457, 1998, doi: 10.1109/JPROC.1998.659497.

[48] J. Kaur, “Understanding Open Neural Network Exchange Advantages,” Feb. 20, 2022. https://www.xenonstack.com/blog/onnx (accessed Jun. 09, 2023).

[49] Bahr Modultechnik GmbH, “Linear system ELZU 30, 40, 60, 60S, 80, 80S, 100.” https://www.bahr-modultechnik.de/images/pdfs/en/ELZU_EN.pdf (accessed Jun. 09, 2023).