Geautomatiseerde Analyse van Waargenomen Objecten en Kijkduur met behulp van Hoofd Gemonteerde Eyetracking in Zorgsimulaties.

Ilian Bronchart.

Scriptie voorgedragen tot het bekomen van de graad van
Professionele bachelor in de toegepaste informatica

Promotor: Dhr. B. Van Vreckem
Co-promotor: Dhr. J. Campens
Academiejaar: 2024–2025
Eerste examenperiode

Departement IT en Digitale Innovatie .

HO GENT

Woord vooraf

Toen ik op het forum voor potentiële bachelorproefonderwerpen dit project rond computer visie in het Zorglab zag, was mijn interesse onmiddellijk gewekt. Na enkele jaren professioneel actief te zijn in de wereld van computer vision, zag ik hierin een unieke kans om mijn technische vaardigheden in te zetten voor een maatschappelijk relevant doel. Bovendien bood het een gelegenheid om meer te leren over de nuances en valkuilen binnen de zorgverlening – een wereld die tot dan toe relatief onbekend voor me was.

Het zaadje waaruit dit project is gegroeid, werd geplant door dhr. Jorrit Campens, lector en onderzoeker aan HOGENT en een drijvende kracht achter het Zorglab. Zijn domeinexpertise in de zorg vormde een perfecte aanvulling op mijn technische achtergrond. Onze samenwerking voelde als een natuurlijke synergie, waarbij hij me doorheen het hele traject, van gerichte feedback voorzag. Ik herinner me nog goed ons eerste gesprek in het Zorglab, waarbij dhr. Campens zijn droom schetste van een geautomatiseerde analysemethode voor observaties in zorgsimulaties. Hij omschreef het toen als 'naar de maan gaan', een uitdaging die me meteen aansprak. Voor zijn enthousiasme en stimulans, ben ik hem dan ook bijzonder dankbaar.

Toegegeven, het project was ambitieus en uitdagend, zeker in combinatie met mijn deeltijdse werk als softwareontwikkelaar gedurende het semester. Desondanks zag ik het als een perfecte kans om de diverse vaardigheden die ik doorheen de jaren heb opgebouwd, samen te brengen. Mijn IT-reis begon met het ontwikkelen van games in het middelbaar en evolueerde naar het bouwen van webapplicaties en het verkennen van frontend-tecnologieën. Binnen mijn opleiding aan HOGENT koos ik bewust voor de minor Data & AI, een domein waarin ik mijn kennis nog verder in wilde verdiepen. Deze bachelorproef voelde dan ook als een culminatiepunt, waar ik mijn passie voor frontend, backend en data science kon verenigen.

Mijn dank gaat ook uit naar mijn promotor, dhr. Bert Van Vreckem, voor zijn waardevolle inhoudelijke feedback en zijn beschikbaarheid voor overlegmomenten gedurende het traject. Een speciaal woord van dank wil ik richten aan mijn bonusvader, Dirk Coussement. Als auteur voor een lokaal blad heeft hij een scherp oog voor taal en heeft hij mij enorm geholpen bij het nalezen van deze bachelorproef.

Tenslotte wil ik ook de studenten bedanken die de tijd namen om deel te nemen

aan het experiment in het Zorglab. Zonder hun medewerking was het verzamelen van de nodige data niet mogelijk geweest.

Ik hoop dat deze bachelorproef een bruikbare eerste stap vormt naar een meer objectieve en efficiënte manier om observatievaardigheden in zorgopleidingen te evalueren.

Samenvatting

Observatievaardigheden zijn van belang voor zorgverleners, zowel voor accurate diagnoses als voor empathische patiëntondersteuning. De huidige evaluatie van deze vaardigheden in gesimuleerde omgevingen steunt vaak op subjectieve methoden zoals zelfrapportage en directe observatie door docenten. Hoewel de Tobii eyetracking-brillen in het Zorglab van HOGENT objectieve blikdata leveren, ontbreekt er tot op heden geschikte software om automatisch te analyseren welke objecten studenten waarnemen en voor hoe lang. Deze bachelorproef beantwoordt hoe computervisiemodellen geïntegreerd kunnen worden met eyetrackingdata van Tobii Glasses om de observatieprestaties van studenten automatisch te analyseren. De analyses dienen de feedback door docenten in het Zorglab te versterken.

Deze doelstelling werd uitgewerkt door middel van van een proof-of-concept (PoC) softwareapplicatie. Dit proces startte met een literatuurstudie naar eyetrackinganalyse en relevante computervisiemodellen (o.a. YOLO, SAM, DINOv2). Vervolgens werd een prototype applicatie ontworpen en geïmplementeerd (Python, FastAPI, HTMX), inclusief een semi-automatische labeling-tool die gebruik maakt van SAM2 voor objectsegmentatie en -tracking. Om de PoC te valideren, werd een gecontroleerd experiment uitgevoerd in het Zorglab. Hier genereerden studenten aan de hand van Tobii Pro Glasses 3, eyetrackingopnames tijdens gesimuleerde observatietaken. Deze opnames, samen met twee specifieke kalibratieopnames, werden gelabeld met de ontwikkelde tool om een grondwaarheidsdataset te creëren. Een analysepijplijn werd ontworpen en geëvalueerd. In deze analyse werd de trackingfunctionaliteit van FastSAM gecombineerd met blikgestuurde filtering en classificatie van objectsegmenten, middels een getraind YOLOv11-objectdetectiemodel. De prestaties werden geëvalueerd aan de hand van precisie, recall en F1-score, na optimalisatie via een grid search van hyperparameters.

Uit de resultaten bleek dat de combinatie van FastSAM-tracking met een YOLOv11-objectdetector (getraind op 1000 samples per klasse) de beste prestaties opleverde, met een F1-score van 0.80, een precisie van 0.94 en een recall van 0.70. De hoge precisie toont aan dat het systeem met grote zekerheid de correcte objecten identificeert, hoewel een significant deel van de fout-positieven in de werkelijkheid correct gedetecteerde objecten bleken te zijn, die niet in de grondwaarheid waren opgenomen. De lagere recall wijst erop dat niet alle bekeken objecten consistent werden gedetecteerd, voornamelijk door problemen met kleine, transparante objecten en

door inconsistenties tussen de FastSAM-segmentaties en de grondwaarheid. De FastSAM-tracking bleek de meest beperkende factor in de pijplijn.

Deze bachelorproef levert een werkend PoC en een methodologie op die de haalbaarheid van geautomatiseerde analyse van observatievaardigheden aantoont. Het biedt een objectieve, datagestuurde basis om de feedback aan studenten te verbeteren en de effectiviteit van simulatietraining in de zorg te verhogen. Op deze manier legt het een fundament voor verder onderzoek naar robuustere analysemethoden.

Inhoudsopgave

Lijst van figuren

Lijst van codefragmenten

1

Inleiding

Observatievaardigheden van zorgverleners zijn niet enkel belangrijk om nauwkeurige diagnoses te stellen. Het stelt ze tevens in staat de patiënt beter te ondersteunen en te begeleiden. Zo zien we dat studenten bijvoorbeeld in de ouderenzorg vaak moeite hebben met sociale omgang en communicatie met ouderen. In de omgang met kleuters kan de aanwezigheid van gevaarlijke voorwerpen zoals een schaar problemen opleveren. In het 360° Zorglab aan HOGENT worden studenten getraind door middel van simulaties waarbij zorgrelevante taken uitvoeren in een gecontroleerde omgeving. Zo leren ze de nuances van zorgverlening kennen en worden ze geconfonteerd met concrete situaties. Een belangrijk aspect van deze training is dat studenten leren om kritische objecten, zoals een colafles op het nachtkastje van een diabetespatiënt, op te merken. Tot op heden wordt er vooral gewerkt met zelfrapportage en observatie door docenten om de vaardigheden van de studenten te evalueren. Dit is echter een subjectieve methode die niet altijd even betrouwbaar is, waardoor de nood ontstond aan een meer objectieve methode om observatievaardigheden te evalueren. Het Zorglab beschikt over een Tobii eyetracking-bril die de oogbewegingen van de studenten kan registreren en over een camera beschikt die het gezichtsveld van de studenten kan opnemen. Voorgaand onderzoek richtte zich voornamelijk op het visualiseren van het pad waarlangs de blik van de studenten beweegt. Dit betreft echter geen geautomatiseerde analyse maar steunt op het manueel herbekijken van elke opname. Bovendien levert deze methode geen bruikbare metrieken op die de prestaties van de studenten objectief kunnen becijferen. Hierdoor ontstond de nood aan een geautomatiseerde methode om de eyetrackingdata te analyseren en te visualiseren, zodat trainers snel inzicht krijgen in de observatieprestaties van studenten.

1.1. Probleemstelling

Hoewel de Tobii Glasses bruikbare data opleveren, ontbreekt er momenteel geschikte software deze te analyseren. Dit gebrek aan dataverwerking en visualisatie maakt het voor trainers lastig om de observatieprestaties van studenten efficiënt te beoordelen en te verbeteren. Zonder een geautomatiseerde manier van detectie van de specifieke objecten die studenten al dan niet hebben waargenomen, wordt het geven van directe feedback een tijdrovend proces. De huidige feedbackmethoden zijn bovendien te subjectief en kunnen leiden tot inconsistenties in de beoordeling van studenten.

1.2. Onderzoeksvraag

Hoe kunnen computervisie-modellen geïntegreerd worden met eyetrackingdata van Tobii Glasses om observatieprestaties van studenten in het 360° Zorglab automatisch te analyseren? Deze onderzoeksvraag wordt uitgewerkt aan de hand van de volgende deelvragen:

- Welke barrières (cognitief, technisch of didactisch) ervaren trainers en studenten bij de huidige, handmatige observatiemethodes?
- Welke kenmerken moet een geautomatiseerde analysemethode hebben om de huidige beperkingen van handmatige observatie te verhelpen?
- In welke mate kunnen de modellen en de ontwikkelde software:
    1. correct bepalen welke kritische objecten studenten hebben waargenomen?
    2. nauwkeurig meten hoe lang studenten naar deze objecten keken?

1.3. Onderzoeksdoelstelling

Het doel is om trainers in het Zorglab te ondersteunen bij het analyseren en visualiseren van eyetrackingdata, zodat ze snel inzicht krijgen in de observatieprestaties van studenten en kunnen vaststellen of belangrijke objecten tijdens zorgsimulaties zijn waargenomen. Hierbij richten we onze focus op twee items: naar welke objecten de studenten gekeken hebben en voor hoe lang. Hiervoor wordt een proof-of-concept softwareoplossing ontwikkeld die de eyetrackingdata van Tobii Glasses combineert met objectdetectie- en segmentatiemodellen en dit op een gebruiksvriendelijke manier. Ook wordt er onderzocht hoe accuraat het uiteindelijke systeem is in het berekenen van de metrieken.

1.4. Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

- In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie.
- In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.
- In Hoofdstuk 4 worden mogelijke oplossingsstrategieën besproken die kunnen worden toegepast om de eyetrackingdata te analyseren.
- In Hoofdstuk 5 wordt de ontwikkeling van de proof-of-concept applicatie toegelicht.
- In Hoofdstuk 6 wordt het experimenteel onderzoek besproken dat de data opleverde die geanalyseerd dienden te worden.
- In Hoofdstuk 7 wordt het bouwen van de grondwaarheid besproken, die een kwantitatieve basis vormt voor de evaluatie van de analysemethoden.
- In Hoofdstuk 8 worden de experimentele opnames geanalyseerd en de resultaten besproken.
- In Hoofdstuk 9, tenslotte, komen de conclusies aan bod en wordt een antwoord geformuleerd op de onderzoeksvragen. Daarbij worden ook aanbevelingen gegeven voor toekomstig onderzoek binnen dit domein.

2

Stand van zaken

Deze bachelorproef onderzoekt het raakvlak tussen de huidige state-of-the-art Machine Learning-modellen in Computer Vision en de analyse van hoofd-gemonteerde eyetracking data. Daarom kan dit hoofdstuk opgedeeld worden in twee hoofddelen. Het eerste deel geeft een overzicht van hoofd-gemonteerde eyetracking technologie, met in het bijzonder een focus op de Tobii Pro Glasses 3. Het tweede deel biedt een overzicht van de huidige state-of-the-art in Computer Vision, met aandacht voor object detection en segmentation, evenals technieken voor image preprocessing.

2.1. Eyetracking

Historisch gezien steunde medische training op observatie en zelfrapportage om de prestaties van studenten te beoordelen (Pauszek, 2023). Deze methoden hebben echter beperkingen door gebrek aan nauwkeurigheid in de betrouwbare rapportage van visuele aandacht en kijkgedrag van studenten. Recent onderzoek door Clarke e.a. (2017) toont aan dat individuen over het algemeen weinig bewust zijn van hun eigen oogbewegingen, waardoor zelfrapportage een onbetrouwbare maatstaf is om na te gaan of kritische elementen tijdens een simulatie effectief zijn waargenomen. Eyetracking-technologie biedt daarentegen een objectieve methode om visuele aandacht en kijkgedrag in kaart te brengen.

2.1.1. Soorten eyetrackers

Eyetrackers kunnen worden onderverdeeld in twee hoofdtypes: screen-based of mobile devices (Pauszek, 2023). Screen-based eyetrackers worden gebruikt in omgevingen waar de gebruiker stationair blijft en naar een monitor kijkt, zoals een computermonitor. Mobile eyetrackers zijn draagbare toestellen waarmee gebrui-

kers zich vrij kunnen bewegen terwijl hun kijkgedrag nog steeds wordt gevolgd. In de context van medische training en simulatie, waarbij studenten vaak fysieke handelingen uitvoeren en zich door een ruimte verplaatsen, zijn mobile eyetrackers de meest geschikte optie. Hierbij zet de student de bril op en kan er na een initiële kalibratie vrij bewogen worden. Toch zijn er enkele nadelen verbonden aan mobile eyetrackers (Pauszek, 2023). Plotselinge bewegingen kunnen ervoor zorgen dat de bril verschuift waardoor de initiële kalibratie verstoord wordt. Daarnaast is het belangrijk om consistente belichting te hebben om nauwkeurige data te verkrijgen, wat in sommige omgevingen moeilijk kan zijn.

In het Zorglab van de HOGENT wordt specifiek gebruik gemaakt van de Tobii Pro Glasses 3 (Tobii AB, 2025a). Dit apparaat is een mobiele eyetracker die een scene camera heeft die het beeld voor de gebruiker registreert, evenals vier eye cameras die de oogbewegingen van de gebruiker opnemen.

2.1.2. Analyse van eyetracking data

Oogbewegingen kunnen breed worden onderverdeeld in verschillende types, waarbij fixaties en saccades de voornaamste blik-afhankelijke metrieken zijn (Pauszek, 2023). Fixaties zijn periodes waarin de ogen relatief stabiel blijven en zich richten op een specifiek visueel doel of gebied. Deze duren meestal tussen de 100 en 600 milliseconden, hoewel deze duur afhankelijk is van contextuele factoren. Tijdens fixaties wordt visuele informatie actief gecodeerd en verwerkt, wat ze belangrijk maakt voor cognitieve functies zoals objectherkenning, lezen en besluitvorming.

Saccades, daarentegen, zijn snelle oogbewegingen die plaatsvinden tussen fixaties. Deze duren meestal tussen de 20 en 80 milliseconden en dienen om de blik te herpositioneren naar een nieuw aandachtspunt. Tijdens een saccade wordt de verwerking van visuele informatie onderdrukt, een fenomeen dat bekendstaat als saccadic suppression, wat bewegingsonscherpte voorkomt en zorgt voor een vloeiende perceptie van de omgeving. In deze zin komen fixaties overeen met actieve informatieverwerking, terwijl saccades dienen om informatie op te zoeken door de blik naar relevante stimuli te verplaatsen.

De volgorde en het patroon van fixaties en saccades over tijd, scanpaths genoemd, geven inzicht in de dynamiek van visuele aandacht en cognitieve verwerking (Pauszek, 2023).

Aangezien we geïnteresseerd zijn in welke objecten worden bekeken en hoe lang, richten we ons op het detecteren van het Aandachtsgebied (Area of Interest, AOI) tijdens fixaties gedurende de opname. Een AOI is een geselecteerd stimulusgebied, element of regio binnen het visuele veld dat relevant is voor de onderzoeker en dient als een filter waarmee eyetracking-metrieken kunnen worden berekend en gerapporteerd (Pauszek, 2023). Een aantal concrete voorbeelden van AOI's in

de context van het Zorglab zijn: een colafles op de nachtkast van een diabetespatiënt, een infuuspomp in een ziekenhuiskamer of een foto van een familielid op de kamer van een patiënt.

2.1.3. Fovea Centralis

De fovea centralis is een klein gebied in het netvlies van het oog dat verantwoordelijk is voor het scherpste zicht en de meeste visuele waarneming (Remington, 2012). Dit gebied vertegenvoordigt slechts ongeveer 1 graad van het gezichtsveld, maar bevat een hoge concentratie van kegeltjes, de fotoreceptoren die verantwoordelijk zijn voor kleurwaarneming en hoge visuele scherpte. Deze informatie kan belangrijk zijn voor het bepalen van de objecten die de deelnemer van een eyetracking-opname daadwerkelijk heeft gezien, aangezien de blikpunten maar duiden op een enkele pixel in de afbeelding. Zo kunnen we AOI's selecteren die overlappen met de cirkel gedefinieerd door de fovea en eventueel de nauwkeurigheid van de eyetracker. Volgens de specificaties van de Tobii Pro Glasses 3 is de eyetracker nauwkeurig tot op 0.6 graden (Tobii AB, 2025a).

2.1.4. Bestaande oplossingen voor eyetracking data-analyse

Analyse van eyetracking data is een complex proces dat verschillende stappen omvat. Daarom zijn er verschillende bestaande softwarepakketten beschikbaar die deze stappen automatiseren en visualiseren.

Een van de meest gebruikte softwarepakketten is Tobii Pro Lab, die functies biedt voor het importeren, analyseren en visualiseren van eyetracking data. Deze software kan diverse metrieken (Tobii AB, 2025b) berekenen rond fixaties, saccades en AOI's, en biedt een scala aan visualisaties (Tobii AB, 2025c) zoals heatmaps, en scanpaths. Heatmaps tonen de dichtheid van fixaties op een bepaald gebied, terwijl scanpaths de volgorde van fixaties en saccades over tijd visualiseren. Tobii Pro Lab heeft echter geen functionaliteit om automatisch AOI's te detecteren, waardoor men dit handmatig moet doen voor elke opname. Bovendien is het een commercieel product met een jaarlijkse licentie. De limitaties van Tobii Pro Lab zorgen ervoor dat het niet geschikt is voor de noden van het Zorglab.

Een ander softwarepakket dat vaak wordt gebruikt is iMotions, een alles-in-één platform voor het verzamelen, analyseren en visualiseren van biometrische data. Deze software wordt opgedeeld in verschillende modules, waaronder een eyetracking module en een 'Automated AOI'[1] module. Bij iMotions kan de gebruiker klikken op een AOI om deze handmatig te definiëren, waarna de AOI automatisch gevolgd wordt in de rest van de opname. Hoewel dit geavanceerder is dan Tobii Pro Lab, vergt het nog steeds handmatige interventie binnen elke opname. Als men veel opnames heeft, kan dit een tijdrovend proces worden. Daarnaast is iMotions

[1] https://imotions.com/products/imotions-lab/modules/automated-aoi/

ook een commercieel product met een jaarlijkse licentie[2] waarbij elke aparte module beschikbaar is vanaf €3400.

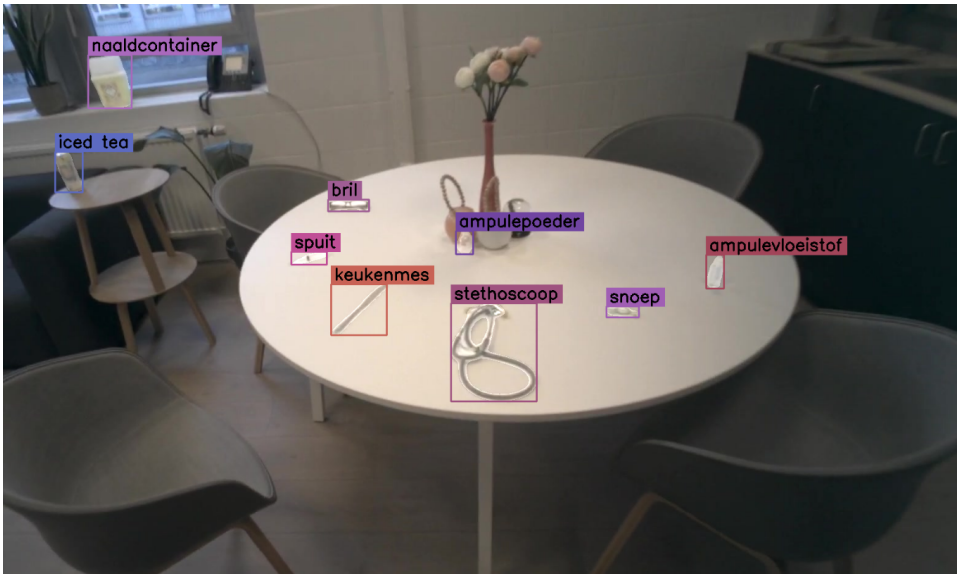
2.2. Computer Vision

Computer Vision is een subveld van Artificiële Intelligentie dat zich bezighoudt met het automatisch extraheren, analyseren en begrijpen van informatie uit digitale beelden of video's. Recente vooruitgangen in Computer Vision maken vandaag de dag nieuwe toepassingen mogelijk waarbij computers visuele informatie autonoom kunnen verwerken en interpreteren zonder menselijke tussenkomst. In deze sectie worden een aantal belangrijke concepten binnen Computer Vision besproken, met een focus op de state-of-the-art technieken voor objectdetectie, segmentatie, en image embedding.

2.2.1. Objectdetectie

Objectdetectie is een taak binnen Computer Vision waarbij een algoritme wordt getraind om automatisch objecten in een afbeelding te lokaliseren en classificeren (zie figuur 2.1). Traditionele methoden maakten gebruik van *features*, zoals de Scale-Invariant Feature Transform (SIFT) en meer recent Oriented FAST and Rotated BRIEF (ORB) om objecten te detecteren (Lindeberg, 2012; Rublee e.a., 2011). SIFT identificeert en beschrijft *keypoints* in een afbeelding op een manier dat deze invariant zijn voor schaal, rotatie en verlichting. ORB is een snellere variant van SIFT die gebruik maakt van BRIEF, een efficiente binaire descriptor die bestaat uit een reeks eenvoudige intensiteitsverschillen tussen pixelparen. Hoewel deze methoden goede resultaten behaalden, waren ze beperkt in hun vermogen om complexe objectvariaties en achtergrondruis aan te pakken. Bovendien vertonen *keypoint*-gebaseerde algoritmen zoals SIFT beperkingen zoals verminderde nauwkeurigheid bij afbeeldingen met een lage resolutie en instabiliteit in *keypoint*-detectie wanneer de beeldschaal verandert of er ruis aanwezig is (Rey-Otero e.a., 2015). De vooruitgang in deep learning heeft geleid tot een aanzienlijke verbetering in prestaties.

[2] https://imotions.com/products/pricing



Figuur 2.1: Voorbeeld van objectdetectie van verschillende objecten in een afbeelding. Hier worden een brede selectie aan objecten gedetecteerd, elk met een label, een bounding box en een bijbehorend masker (segmentatie). De objecten in de afbeelding zijn deel van het experiment in het Zorglab die later in deze bachelorproef verder besproken wordt.

Van R-CNN naar Moderne Objectdetectiemodellen

Een belangrijke doorbraak in objectdetectie kwam met de introductie van Region-based Convolutional Neural Networks (R-CNN) door Girshick e.a. (2014). R-CNN combineerde Convolutional Neural Networks (CNNs) met zogenaamde *region proposals* om objecten nauwkeuriger te detecteren dan eerdere methoden. Hoewel R-CNN goede prestatieverbeteringen liet zien, was het een traag algoritme dat moeilijk te trainen was. Dit kwam door de noodzaak om elke *region proposal* afzonderlijk door een CNN te sturen, wat leidde tot een hoge rekentijd. Om deze nadelen te overwinnen, werden verschillende verbeteringen geintroduceerd, waarvan de meest recente de You Only Look Once (YOLO) en Mask R-CNN modellen zijn.

You Only Look Once (YOLO)

You Only Look Once (YOLO), geïntroduceerd door Redmon e.a. (2016), was een belangrijke stap vooruit binnen objectdetectie doordat het een veel eenvoudiger en sneller proces gebruikt. In plaats van afbeeldingen stap voor stap te analyseren zoals eerdere methoden (bijvoorbeeld Region-based Convolutional Neural Networks, R-CNN), verwerkt YOLO de gehele afbeelding tegelijkertijd in één analyse. Dit zorgt ervoor dat YOLO zeer snel objecten kan herkennen in *real-time*, wat het ideaal maakt voor toepassingen waarbij snelheid belangrijk is. Omdat YOLO de gehele afbeelding in één keer analyseert, wordt contextuele informatie zoals waar objecten zich bevinden tegenover elkaar, beter benut.

Toch heeft YOLO ook enkele nadelen. Het model heeft moeite met het detecteren van kleine en dicht opeengepakte objecten, wat kan leiden tot gemiste of overlap-

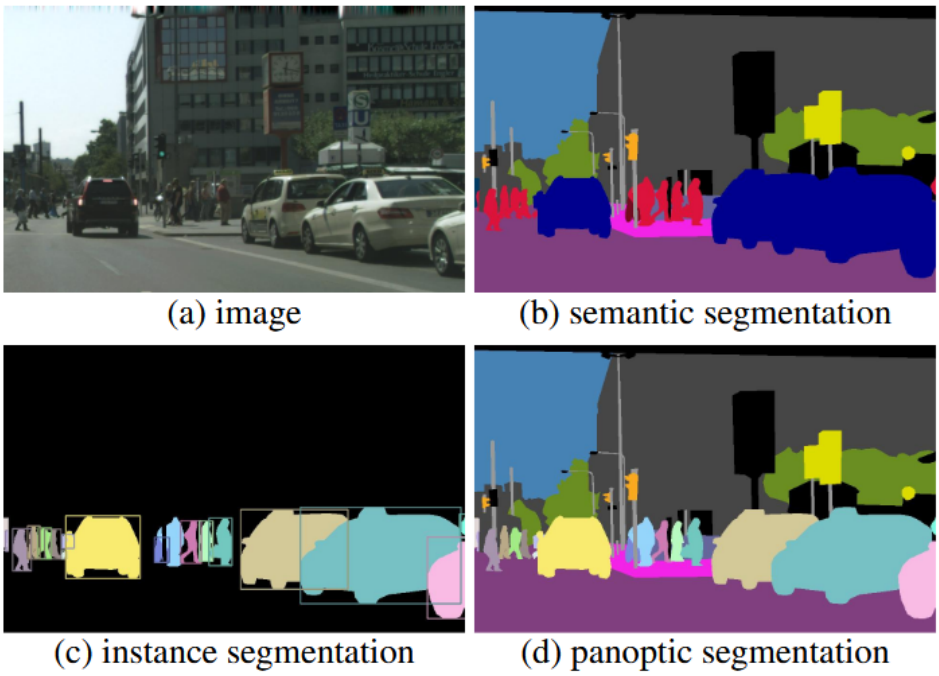
pende detecties. Volgens onderzoek van Huang e.a. (2024) presteert YOLOv8 minder goed bij het detecteren van objecten kleiner dan 8x8 pixels. Dit komt doordat de standaard detectielaag van YOLO onvoldoende gedetailleerd is om kleine objecten nauwkeurig te detecteren. Hoewel optimalisaties zoals het toevoegen van extra detectielagen voor kleinere objecten verbeteringen bieden, blijft YOLO gevoelig voor problemen zoals overlappende en gemiste detecties bij dicht op elkaar geplaatste objecten.

Over de jaren heen zijn er verschillende versies van YOLO uitgebracht, waarvan de meest recente YOLOv11 is (Khanam & Hussain, 2024). Deze modellen zijn beschikbaar via Ultralytics met een betaalde licentie[3] voor commercieel gebruik, maar zijn ook gratis beschikbaar voor open-source projecten en onderzoek.

Mask R-CNN

Mask Region-based Convolutional Neural Network (Mask R-CNN), geïntroduceerd door He e.a. (2018), is een model dat niet alleen objecten detecteert met behulp van rechthoekige kaders (bounding boxes), maar ook precies aangeeft welke pixels bij elk gedetecteerd object horen. Dit gebeurt in twee stappen; eerst worden mogelijke objecten voorgesteld met zogenaamde *region proposals*, en vervolgens worden deze regio's verder geanalyseerd om drie zaken te bepalen. Ten eerste wordt de klasse van het object bepaald, ten tweede wordt de exacte locatie van het object bepaald met behulp van een bounding box, en ten laatste wordt een masker gegenereerd dat aangeeft welke pixels bij het object horen. Het proces waarbij een masker wordt gegenereerd voor elk gedetecteerd object wordt *instance segmentation* genoemd (Hafiz & Bhat, 2020). Wanneer men ook een klasse toekent aan pixels in een afbeelding, wordt dit *semantic segmentation* genoemd. Een laatste vorm van segmentatie is *panoptic segmentation*, geintroduceerd door Kirillov e.a. (2019) waarbij zowel objecten als achtergrond worden geïdentificeerd. Het verschil tussen deze vormen van segmentatie wordt geïllustreerd in figuur 2.2.

[3] https://www.ultralytics.com/license



Figuur 2.2: Verschil tussen instance, semantic en panoptic segmentation (Kirillov e.a., 2019). Zie de tekst hierboven voor meer uitleg over de verschillende vormen van segmentatie.

2.2.2. Transformer-gebaseerde Objectdetectie

Recentelijk zijn transformer-gebaseerde modellen in opkomst binnen Computer Vision als alternatief voor traditionele CNN-gebaseerde modellen, zoals YOLO en Mask R-CNN.

DETR

Een belangrijk voorbeeld hiervan is DETR (DEtection TRansformer), geïntroduceerd door Carion e.a. (2020).

Transformer-gebaseerde modellen zijn interessant vanwege hun vermogen om aandacht (attention) te richten op belangrijke delen van een afbeelding. Dit mechanisme stelt het model in staat om relaties tussen objecten en context beter te begrijpen. Anders dan traditionele modellen gebruikt DETR geen vooraf bepaalde ankerpunten of zogenaamde 'region proposals', om objecten te detecteren. In plaats daarvan behandelt DETR objectdetectie rechtstreeks als een voorspellingstaak, waarbij een afbeelding direct wordt verwerkt om objecten en hun locaties te identificeren. Het DETR-model bestaat uit twee onderdelen: een *encoder* en een *decoder*. De encoder analyseert eerst de hele afbeelding om belangrijke kenmerken te onderscheiden en de context van de afbeelding te begrijpen. Vervolgens gebruikt de decoder specifieke 'object queries' om de exacte locatie en de categorie van objecten te bepalen.

Een voordeel van DETR is dat het geen aparte stap nodig heeft om overlappende

voorspellingen te verwijderen, hoewel dit soms in het beginstadium van voorspellingen wel nuttig kan zijn. Toch heeft DETR ook enkele beperkingen. Zo presteert het model minder goed bij het detecteren van zeer kleine objecten door beperkte resolutie. Desondanks laat het model indrukwekkende resultaten zien in situaties die het tijdens training nog nooit heeft gezien, bijvoorbeeld wanneer er meerdere soortgelijke objecten in één afbeelding staan. Opmerkelijk is dat DETR ook uitgebreid kan worden naar *panoptic segmentation* taken.

DINO

Verder onderzoek door Zhang e.a. (2022) heeft geleid tot de ontwikkeling van DINO (DETR with Improved DeNoising anchOr boxes). DINO bouwt voort op DETR door onder andere verbeterde denoising-technieken tijdens training te gebruiken, wat resulteert in nauwkeurigere detecties. Dit wordt bereikt door het gebruik van zowel positieve als negatieve voorbeelden om verwarring tussen overlappende voorspellingen te verminderen. Daarnaast gebruikt DINO een query-selectiemethode die positie-informatie uit de encoder gebruikt om de initiële voorspellingen (*queries*) van de decoder beter te initialiseren. Ten slotte introduceert DINO een *look forward twice*-techniek, waarbij voorspellingen uit latere decoderlagen worden gebruikt om eerdere voorspellingen verder te verbeteren. Deze innovaties zorgen ervoor dat DINO aanzienlijk beter presteert dan eerdere DETR-gebaseerde modellen.

Grounding DINO

In 2023 introduceerde Liu e.a. (2023) Grounding DINO, een uitbreiding op DINO gericht op *open-set objectdetectie*. Dit houdt in dat het model objecten kan detecteren die tijdens de training niet expliciet zijn aangeleerd. Grounding DINO integreert taalbegrip met visuele detectie, waardoor het mogelijk wordt om objecten op basis van natuurlijke taal te identificeren en te lokaliseren.

Visuele kenmerken uit een afbeelding worden gecombineerd met taalkundige informatie uit teksten, zoals categorieën of beschrijvende zinnen. Dit gebeurt via een *feature enhancer*, die beeld- en tekstinformatie samenbrengt, en een taalgestuurde query-selectiemethode, waarmee relevante beeldgebieden worden geselecteerd op basis van tekstuele input. Vervolgens gebruikt het model een *cross-modality decoder*, waarin informatie uit beide modaliteiten verder gecombineerd wordt voor nauwkeurigere detecties.

Het model wordt getraind op grootschalige datasets, bestaande uit foto's met bijbehorende beschrijvingen en labels. Hierdoor leert Grounding DINO generaliseren naar nieuwe, niet eerder getoonde objecten. Dit stelt het model in staat om, zonder verdere training, objecten te herkennen uit tekstuele beschrijvingen die het nog nooit eerder heeft gezien.

Hoewel de resultaten van Grounding DINO veelbelovend zijn, is het model niet geschikt voor real-time toepassingen, of toepassingen waarbij heel veel afbeeldingen

moeten worden verwerkt (Son & Jung, 2024). Omdat het model gebaseerd is op een transformer-architectuur, en het veel klassen moet kunnen detecteren waardoor het een groot model is, presteert het meer dan 30 keer trager in vergelijking met YOLO modellen.

2.2.3. Segmentatie

In de voorgaande sectie hebben we reeds gesproken over verschillende vormen van segmentatie, zoals instance, semantic en panoptic segmentation. Vaak zijn de taken van objectdetectie en segmentatie nauw met elkaar verbonden, aangezien beide taken objecten in een afbeelding lokaliseren en classificeren. Toch willen we in deze sectie dieper ingaan op een aantal modellen die specifiek gericht zijn op segmentatie, met een oog voor voorgetrainde modellen die geen verdere training vereisen.

Meta Segment Anything Model (SAM)

In 2023 introduceerde Meta het open-source Segment Anything Model (SAM), een model ontworpen om objecten in afbeeldingen te segmenteren op basis van gebruikersinput, ook wel prompts genoemd (Kirillov e.a., 2023). Het doel van SAM is om flexibele en veelzijdige segmentatie mogelijk te maken, waarbij gebruikers eenvoudig kunnen aangeven welke delen van een afbeelding ze willen segmenteren via punten, rechthoeken, maskers of zelfs korte tekstuele beschrijvingen. Dit wordt het 'promptable segmentation'-principe genoemd.

SAM bestaat uit drie belangrijke onderdelen: een beeldencoder, een promptencoder en een masker-decoder. De beeldencoder verwerkt de afbeelding één keer en creëert een zogenoemde image embedding, een compacte representatie van de afbeelding waarin belangrijke kenmerken zijn opgeslagen. Vervolgens wordt de prompt (zoals een punt of rechthoek) verwerkt door de promptencoder en gecombineerd met de representatie uit de beeldencoder. De masker-decoder gebruikt vervolgens deze gecombineerde informatie om het uiteindelijke masker te genereren dat het gewenste object nauwkeurig identificeert.

Een opvallend aspect van SAM is het vermogen om ambiguïteit te herkennen en daarmee om te gaan. Wanneer één prompt meerdere mogelijke interpretaties heeft, bijvoorbeeld een punt dat zowel naar een persoon als een kledingstuk verwijst, kan SAM meerdere segmentatiemaskers genereren die elk een mogelijke interpretatie weergeven. Vervolgens wordt voor elk masker een betrouwbaarheidswaarde berekend, waarmee het model aangeeft welk masker het meest waarschijnlijk overeenkomt met de intentie van de gebruiker.

De kracht van SAM ligt in zijn brede toepasbaarheid zonder dat verdere training vereist is (zero-shot generalisatie). Dit betekent dat SAM in staat is om direct te worden ingezet voor nieuwe beelden en taken die het nog niet eerder heeft ge-

zien, wat erg nuttig is voor interactieve toepassingen en snelle analyses. Hoewel SAM indrukwekkende resultaten levert en een grote stap vooruit betekent voor segmentatiemodellen, heeft het ook enkele beperkingen. De belangrijkste is dat het model, hoewel snel genoeg voor interactieve toepassingen, minder geschikt is voor real-time toepassingen met zeer hoge snelheden zoals videostreaming, vanwege de benodigde rekencapaciteit voor de beeldverwerking.

Het jaar nadien werd een verbeterde versie van het model uitgebracht, SAM 2 (Ravi e.a., 2024). SAM 2 voegt belangrijke verbeteringen toe, zoals een uitbreiding naar videosegmentatie met behulp van een geheugenmodule, waardoor het model eerdere prompts en segmentaties kan onthouden en toepassen op opeenvolgende videoframes. Dit maakt SAM2 heel interessant voor de toepassingen binnen het Zorglab, waarbij het volgen van objecten doorheen de video een belangrijke rol kan spelen. Daarnaast is SAM 2 ongeveer zes keer sneller bij het segmenteren van afbeeldingen dan het oorspronkelijke SAM-model, wat het geschikt maakt voor een bredere reeks real-time toepassingen.

FastSAM

Zou het kunnen dat SAM nog sneller kan? Dat is precies wat Zhao e.a. (2023) zich afvroegen bij het ontwikkelen van FastSAM, een verbeterde versie van SAM die tot 50 keer sneller is dan het originele model, zonder veel in te boeten aan nauwkeurigheid. FastSAM bereikt dit door het segmentatieproces op te splitsen in twee opeenvolgende stappen: het segmenteren van alle objecten in een afbeelding en vervolgens het selecteren van specifieke objecten aan de hand van een gebruikersprompt.

In de eerste fase, genaamd *all-instance segmentation*, maakt FastSAM gebruik van een CNN gebaseerd op het YOLOv8-seg model, een objectdetector met een speciale tak voor *instance segmentation*. De tweede fase, de *prompt-guided selection*, gebruikt vervolgens de gegenereerde segmentatiemaskers uit de eerste fase en selecteert specifiek het gewenste gebied op basis van verschillende prompts. Deze prompts kunnen bestaan uit punten (*point prompts*), rechthoeken (*box prompts*) of zelfs tekstuele beschrijvingen (*text prompts*). Bij een punt-prompt wordt bijvoorbeeld gekeken welk segmentatiemasker het opgegeven punt bevat, terwijl een box-prompt gebruikmaakt van de overlap tussen een opgegeven rechthoek en bestaande segmentatiemaskers. Zo kunnen we bij eye-tracking data bijvoorbeeld een punt-prompt gebruiken op basis van de blikrichting van de gebruiker om objecten te selecteren uit een all-instance segmentatie. Een interessante eigenschap van zowel SAM als FastSAM is de mogelijkheid om een zogenaamde *everything-prompt* te gebruiken, waarbij alle objecten in de afbeelding gesegmenteerd worden zonder dat de gebruiker specifieke prompts hoeft op te geven. Zo is het mogelijk om een everything-prompt te gebruiken, en vervolgens de segmentatiemaskers te filteren op basis van de blikpunten van de gebruiker, om zo de re-

levante objecten te selecteren. Merk op dat hier geen klassen worden toegewezen aan objecten (behalve bij text-prompting), maar enkel segmentatiemaskers worden gegenereerd. Daarom zullen segmentatiemodellen zoals FastSAM en SAM moeten worden gecombineerd met objectdetectiemodellen om na te gaan welke objecten er in de afbeelding aanwezig zijn.

FastSAM is beschikbaar in de Model Hub van Ultralytics[4] en kan gratis worden gebruikt voor open-source projecten en onderzoek.

2.2.4. Image Embedding

Naast objectdetectie en segmentatie speelt ook image embedding een steeds belangrijkere rol binnen Computer Vision. Image embedding is een techniek waarbij afbeeldingen worden omgezet naar numerieke vectoren, ook wel feature vectors genoemd. Deze vectoren zijn eigenlijk een reeks getallen die belangrijke visuele kenmerken en eigenschappen van een afbeelding beschrijven. Ze kunnen worden beschouwd als een compacte, numerieke 'vingerafdruk' van een afbeelding, waarbij visueel vergelijkbare afbeeldingen corresponderen met vectoren die dicht bij elkaar liggen in de hoog-dimensionale ruimte. Enkele voorbeelden van de mogelijkheden van image embeddings zijn te zien in figuur 2.3.

Deze techniek is vooral interessant voor toepassingen waarbij men op een efficiënte manier wil bepalen om welk object het gaat, zonder hiervoor telkens nieuwe detectiemodellen te moeten trainen. Wanneer een selectie van objecten via segmentatie (zoals met SAM) gedetecteerd wordt, kan een image embedding gebruikt worden om automatisch te bepalen om welk specifiek object het gaat. Dit gebeurt door de gegenereerde vector van het gedetecteerde object te vergelijken met een vooraf opgebouwde database van bekende objecten. Het gebruik van image embeddings biedt een aantal voordelen: het is robuust tegen variaties in belichting, perspectief, en andere visuele veranderingen, waardoor het model goed blijft presteren, zelfs wanneer objecten er anders uitzien dan tijdens de training.

CLIP

Een van de eerdere modellen die image embeddings populair maakten is CLIP (Contrastive Language-Image Pre-training), geïntroduceerd door OpenAI in 2021 (Radford e.a., 2021). CLIP koppelt tekstuele beschrijvingen aan afbeeldingen tijdens de training, waardoor het model leert om afbeeldingen en teksten in dezelfde vectorruimte te plaatsen. Dit betekent dat afbeeldingen niet alleen herkend kunnen worden op basis van visuele kenmerken, maar ook gekoppeld kunnen worden aan tekstuele beschrijvingen zonder dat hiervoor expliciete labels nodig zijn.

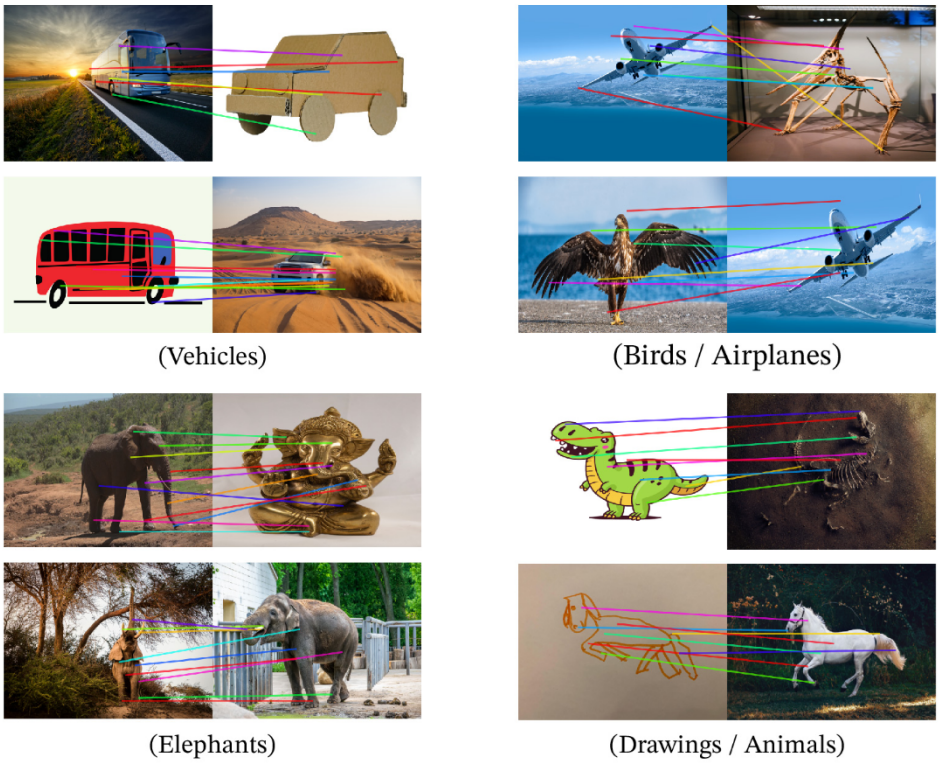
[4] https://docs.ultralytics.com/models/fast-sam/

DINOv2

DINOv2, geïntroduceerd in 2024 door Oquab e.a. (2024), bouwt voort op eerdere modellen zoals DETR en DINO, met verbeteringen in nauwkeurigheid en toepasbaarheid zonder extra finetuning. Net zoals eerdere embedding-modellen genereert DINOv2 numerieke representaties van afbeeldingen, maar onderscheidt zich door een sterk verbeterde generalisatiekracht. Dit betekent dat het model beter in staat is om nieuwe objecten en situaties te herkennen zonder verdere training.

DINOv2 is gebaseerd op transformer-architecturen en maakt gebruik van self-supervised learning (SSL). Hierbij leert het model zelfstandig relevante kenmerken van afbeeldingen zonder gebruik te maken van expliciete labels. In tegenstelling tot veel eerdere modellen presteert DINOv2 zowel goed op algemene taken, zoals het herkennen van categorieën van objecten (auto's of vliegtuigen), als op taken waarbij het exacte exemplaar (een specifiek gebouw of schilderij) geïdentificeerd moet worden.

Daarnaast biedt DINOv2 ook goede prestaties bij segmentatietaken. Door een eenvoudige lineaire laag aan het model toe te voegen, kunnen zeer goede segmentatiemaps worden gegenereerd zonder verdere optimalisatie of training van het volledige model. Met name bij gebruik van zogenaamde multiscale augmentaties (waarbij het model de afbeelding op meerdere resoluties analyseert), behaalt DINOv2 segmentatieprestaties die vergelijkbaar zijn met gespecialiseerde segmentatiemodellen zoals SAM en FastSAM.



Figuur 2.3: Enkele voorbeelden van de mogelijkheden van DINOv2 image embeddings (afbeelding afkomstig van Oquab e.a. (2024)).

2.3. Eyetracking en Computer Vision

Ten slotte is het interessant om stil te staan bij recente implementaties van Computer Vision in combinatie met eyetracking-technologie.

In 2023 onderzochten Cederin en Bremberg (2023) de mogelijkheden van bestaande computer vision modellen om automatisch te detecteren welke objecten een gebruiker bekijkt tijdens een eyetrackingsessie. Net zoals deze bachelorproef gebruikten ze de Tobii Pro Glasses 3, wat hun onderzoek interessant maakt als referentie voor het omgaan met de eyetracking data van deze specifieke bril. Een probleem bij hoofd-gemonteerde eyetrackers is dat er vaak bewegingsruis aanwezig is, zeker als het hoofd van de gebruiker veel beweegt. Dit maakt het toepassen van computer vision modellen moeilijker. In hun onderzoek vergeleken ze twee ruisreductiemodellen, namelijk Restormer en DeblurGAN-v2, om de kwaliteit van de eyetracking data te verbeteren. Ze concludeerden dat DeblurGAN-v2 betere resultaten opleverde dan Restormer. Ook onderzochten ze de prestatieverschillen tussen YOLOv8 en DINO modellen op de eyetracking data. Uit hun resultaten blijkt dat DINO beter presteert dan YOLOv8 in elk testscenario. Ze concludeerden echter ook dat DINO modellen een langere rekentijd nodig hebben dan de YOLO modellen.

SAM werd ook onderzocht in combinatie met eyetracking data door

Wang e.a. (2023) in hun applicatie genaamd 'GazeSAM'. Dit onderzoek richtte zich op het segmenteren van de objecten waarnaar de gebruiker kijkt tijdens een eyetrackingsessie. Een belangrijk onderschijd tussen de context van dit onderzoek en deze bachelorproef is dat er gebruik gemaakt werd van screen-based eyetracking, waarbij de gebruiker naar een computermonitor keek in plaats van een fysieke omgeving. De blikrichting van de gebruiker werd gebruikt als basis voor een punt-prompt, waarmee SAM de objecten in de afbeelding kon segmenteren. Het bleek dat SAM soms moeilijkheden had met het segmenteren van medische afbeeldingen, omdat SAM voornamelijk getraind is op alledaagse objecten. Dit zal echter geen probleem vormen voor de toepassingen binnen het Zorglab, waar de objecten voornamelijk alledaagse objecten zijn. Een ander nadeel van SAM was dat het model moeilijkheden had met het correct segmenteren van objecten op basis van een enkele punt-prompt, waardoor verdere manuele interventie nodig was.

3

Methodologie

Het onderzoek werd uitgevoerd in een iteratieve cyclus, waarbij de verschillende fasen van het project elkaar opvolgden en elkaar beïnvloedden. Deze sectie beschrijft in grote lijnen de verschillende fasen van het onderzoek, de doelstellingen en de gebruikte methodologieën. Figuur 3.1 toont een flowchart van de methodologie, waarin de verschillende fasen en hun onderlinge relaties worden weergegeven. De labels op de pijlen geven de belangrijkste deliverables van elke fase aan, die als input dienen voor de volgende fase(s).

Figuur 3.1: Flowchart van de methodologie van het onderzoek.

3.1. Literatuurstudie

De literatuurstudie vormde een iteratief proces gedurende het gehele onderzoek, beginnend met een brede initiële verkenning en later verfijnd en aangevuld naarmate specifieke vragen tijdens het project opdoken. Het overkoepelende doel was een grondig en actueel inzicht te verwerven in de relevante domeinen. Hierbij lag de focus op de volgende gebieden:

1. Onderzoeken hoe de evaluatie van observatieprestaties momenteel wordt uitgevoerd, met bijzondere aandacht aan de nood aan een geautomatiseerde aanpak.
2. De huidige stand van zaken in eyetracking-technologie, met aandacht voor hoofd-gemonteerde systemen zoals de Tobii Pro Glasses 3 en de analyse van de hieruit voortkomende data.
3. State-of-the-art technieken binnen Computer Vision, waaronder objectdetectie (bv. YOLO, DETR, Grounding DINO), segmentatie (bv. Mask R-CNN, SAM, FastSAM) en image embedding (bv. CLIP, DINOv2).
4. Bestaande integraties van eyetracking en Computer Vision.

Wetenschappelijke publicaties, technische documentatie en relevante open-source projecten werden hiervoor systematisch geraadpleegd en kritisch geanalyseerd. Hoofdstuk 2 beschrijft uitvoerig de bevindingen van deze literatuurstudie.

3.2. Oplossingsverkenning

Voortbouwend op de inzichten uit de literatuurstudie, richtte de tweede fase zich op het verkennen en evalueren van verschillende conceptuele oplossingsstrategieën om de centrale onderzoeksvraag te beantwoorden. Het doel was om een reeks potentiële benaderingen te formuleren voor het automatisch analyseren van eyetrackingdata in combinatie met computervisiemodellen, en hieruit de meest veelbelovende te selecteren voor de ontwikkeling van de Proof-of-Concept (PoC). Dit proces omvatte het conceptueel ontwerpen van diverse pipelines, waarbij de inputs (eyetracking-opnames, objectdefinities) en de gewenste outputs (geobserveerde objecten, observatieduur) als leidraad dienden. De strategieën varieerden op vlak van algemene aanpak, de typen computervisiemodellen en de rol van de blikdata in het proces. Elke strategie werd kwalitatief beoordeeld op criteria zoals verwachte accuraatheid, complexiteit van implementatie, computationele vereisten en flexibiliteit. Deze analyse resulteerde in een beargumenteerde keuze voor de strategie die als basis diende voor het onderzoek, zoals gedetailleerd in Hoofdstuk 4.

3.3. Huisgemaakte Eyetrackingdata

Voor het onderzoek en het uitwerken van de PoC software-applicatie, was het belangrijk om zicht te krijgen op de werking van de Eyetracker en de bijhorende software. Daarom werd deze ontleend uit het Zorglab van HoGent voor een periode van 3 weken. Tijdens deze periode werden thuis verschillende opnames gemaakt in een woonkamer met diverse objecten, met de volgende specifieke doelstellingen:

- Het leren werken met de Tobii Pro Glasses 3 en de bijhorende software.
- Nagaan hoe men de eyetrackingdata kan exporteren naar een bruikbaar formaat via de WiFi-verbinding van de eyetracker.
- Een dataset creëren die kan dienen als basis voor het uitwerken van de PoC.

Deze opnames werden niet gebruikt voor de uiteindelijke evaluatie van de PoC, aangezien de data afkomstig zijn van tests waarbij de onderzoeker zelf de eyetracker droeg. Hierdoor is er sprake van mogelijke bias, wat de objectiviteit van de data in het gedrang brengt.

3.4. Proof of Concept Software Applicatie

Een kernonderdeel van deze bachelorproef was de ontwikkeling van een Proof-of-Concept (PoC) softwareapplicatie. Het hoofddoel van deze fase was het ontwerpen en implementeren van een werkend prototype dat de beoogde workflow ondersteunt: van het importeren van ruwe eyetracking-opnames tot het genereren van gelabelde data die als basis kan dienen voor analyse. De methodologie omvatte de selectie van een passende, moderne technologie-stack (o.a. Python, FastAPI, HTMX, SQLite) gericht op modulariteit en toekomstige uitbreidbaarheid. Een significant onderdeel was het ontwerp en de implementatie van een semi-automatische labeling-tool, bedoeld om de efficiëntie van het labelingsproces te verhogen. Er werd ingezet op duurzame software-ontwikkelpraktijken zoals type hinting en containerisatie (Docker) om de mogelijkheid tot onderhoud en reproduceerbaarheid te waarborgen. De resulterende applicatie, inclusief de architectuur, componenten en technische keuzes, wordt uitvoerig beschreven in Hoofdstuk 5.

3.5. Experimenteel Onderzoek

Zoals eerder vermeld, werd de thuisopgenomen dataset niet gebruikt voor de evaluatie van de PoC. Om een onbevooroordeelde dataset te verkrijgen en om de modaliteiten van de uitwerking te valideren (bijvoorbeeld: invloed van de afstand tussen de Eyetracker en het object, en de aard van de objecten), werd er een gecontroleerd experiment opgezet in het Zorglab van HoGent. Het doel van dit experiment

was om een dataset te creëren die als grond-waarheid diende voor de metrieken die de PoC berekent (bekeken objecten en tijdsduur). Op de campus werden studenten van diverse opleidingen gevraagd om deel te nemen aan het experiment. De 14 resulterende opnames werden daarna gelabeld via de labeling-tool van de PoC, met een manuele kwaliteitscontrole om de betrouwbaarheid van de data te waarborgen. Voor een uitgebreide beschrijving van het opzet en de uitvoering van het experiment, inclusief de gebruikte methodologieën, wordt verwezen naar Hoofdstuk 6.

3.6. Creatie van een Grondwaarheidsdataset

Om de prestaties van de geautomatiseerde analysemethoden objectief te kunnen evalueren, was de creatie van een nauwkeurige grondwaarheidsdataset noodzakelijk. Het doel van deze fase was om, voor elke evaluatieopname uit het experiment, frame-per-frame vast te stellen welke van de 15 gedefinieerde objecten daadwerkelijk door de deelnemer werden bekeken. De methodologie startte met het voorbereiden van de ruwe opnamedata (video en blikdata). Vervolgens werden de evaluatieopnames gelabeld met behulp van de ontwikkelde labeling-tool (zie Hoofdstuk 5). Hierbij segmenteerde de onderzoeker objecten waar de blik van de deelnemer op rustte, met behulp van het SAM2-model en de trackingfunctionaliteit van de labeling-tool. Deze initiële segmentaties werden daarna gefilterd op basis van de geregistreerde blikpunten, waarbij een cirkelvormig kijkgebied moest overlappen met het segmentatiemasker. Dit resulteerde in een dataset die per frame de bekeken objecten, hun bounding boxes en maskeroppervlakte specificeert. De correctheid werd manueel gevalideerd. De gedetailleerde stappen van dit proces zijn beschreven in Hoofdstuk 7.

3.7. Analyse van Observatieprestaties

De laatste fase van het onderzoek focuste op het implementeren en evalueren van een geautomatiseerde analysepipeline om observatieprestaties te meten, conform de gekozen oplossingsstrategie (Strategie 4 uit Hoofdstuk 4). Het doel was om automatisch te bepalen welke kritische objecten werden waargenomen en hoe lang, en dit te vergelijken met de grondwaarheidsdataset. De methodologie omvatte:

1. Het toepassen van “everything-segmentation” en tracking met FastSAM op de evaluatieopnames.
2. Het filteren van de resulterende segmenten op basis van objectgrootte en de geregistreerde blikdata (overlap met het blikpunt), wat resulteerde in de te classificeren object ROIs (Regions of Interest).
3. Het classificeren van deze ROIs met drie verschillende methoden:

- Een DINOv2 image embedding model, dat de ROIs omzet naar vectorrepresentaties en deze vergelijkt met een Faiss vector-index van de kalibratieopnames.
- Een YOLOv11-classificatiemodel, dat specifiek getraind is op de objecten uit de kalibratieopnames.
- Het toepassen van een YOLOv11-objectdetectiemodel, dat de ROIs detecteert en de voorspellingen combineert met de eerdere FastSAM-segmentaties.

De prestaties van deze geautomatiseerde analysepipeline werden kwantitatief beoordeeld aan de hand van precisie, recall en F1-score ten opzichte van de grondwaarheidsdataset. De evaluatie ging echter verder dan deze kernmetrieken. Er werd een systematische hyperparameteroptimalisatie via een grid search uitgevoerd en modellen getraind met verschillende hoeveelheden data werden vergeleken. Bovendien vond een diepgaande analyse plaats van de aard van vals-positieve en vals-negatieve detecties, waarbij ook de invloed van factoren zoals objectgrootte werd onderzocht. Dit proces, inclusief de gedetailleerde methodologie en resultaten, wordt uitvoerig beschreven in Hoofdstuk 8.

4

Mogelijke Oplossingsstrategieën

Bij het ontwerpen van een oplossing voor de gestelde problematiek, is het belangrijk om de kernfunctionaliteit van het systeem te beschouwen in termen van zijn inputs en outputs. Het systeem dient de eyetracking-opnames als input te verwerken en als output de relevante metrieken op te leveren. Hoewel verschillende strategieën zullen leiden tot een proof-of-concept systeem met verschillende eisen, zullen ze allemaal grotendeels voldoen aan dezelfde input-output specificatie.

4.1. Inputs van het Systeem

Opnames

De tobii eyetracking-opnamen bevatten verschillende gegevens, waarvan de volgende nuttig zijn voor de doeleinden van het probleem (Tobii AB, 2023):

- **Video-opname**: De video-opname van de eyetracking-bril, met een resolutie van 1920x1080 pixels en een framerate van 25 fps (frames per seconde).
- **Blikdata**: De blikdata van de eyetracking-bril, die de coördinaten van het blikpunt (het punt gedefinieerd door de samenkomst van de twee ooglijnen) bevat in de vorm van een 2D-coördinaatssysteem. Deze worden opgenomen met een frequentie van 50 Hz, wat betekent dat er 50 blikpunten per seconde worden geregistreerd.
- **Metadata**: Metadata van de opname, zoals ID van de opname, naam van de deelnemer, tijdstempel, en andere relevante informatie.

Andere gegevens omvatten onder andere data afkomstig van de IMU (Inertial Measurement Unit) van de eyetracker, zoals de oriëntatie van de bril, de acceleratie, en metingen van het magnetisch veld rond de bril. Deze kunnen eventueel gebruikt worden als secundaire gegevens voor het verbeteren van de resultaten, maar zijn

niet noodzakelijk voor de kernfunctionaliteit van het systeem.

Objecten

Naast de eyetracking-opnames zelf, vereist het systeem ook een vooraf gedefinieerde lijst van de specifieke objecten waarvan de observatiestatus moet worden bepaald. Afhankelijk van de gekozen oplossingsstrategie, kan het gaan om een enkele foto van elk object, of een dataset met meerdere beelden (samples) van elk object vanuit verschillende hoeken en onder verschillende belichtingsomstandigheden.

4.2. Outputs van het Systeem

De uiteindelijke doelmetrieken werden eerder gedefinieerd als de specifieke objecten die studenten hebben bekeken en de totale tijdsduur van deze observaties per object binnen een opname. Deze metrieken zijn echter afgeleide, geaggregeerde waarden die niet rechtstreeks uit de ruwe eyetracking-data kunnen worden geëxtraheerd.

Om deze doelmetrieken te kunnen berekenen, moet het systeem eerst een meer fundamentele, primaire output genereren. Deze primaire output bestaat, voor elke individuele frame van de video-opname, uit een identificatie van het (de) object(en) waarop de blik van de deelnemer op dat specifieke moment gericht was. Met andere woorden, gegeven alle frames uit een opname, is het de taak van het systeem om per frame te bepalen welk(e) relevant(e) object(en) zich in het blikveld bevonden en daadwerkelijk werden bekeken.

Vanuit deze frame-per-frame objectidentificatie kunnen vervolgens de twee beoogde hoofdmetrieken worden afgeleid:

- **Geobserveerde objecten:** Een lijst van alle unieke objecten die gedurende de opname minstens één keer zijn bekeken.
- **Observatieduur per object:** Voor elk geobserveerd object, de cumulatieve tijd (of het aantal frames) waarop de blik van de deelnemer gericht was.

4.3. Oplossingsstrategieën

De uitdaging is om een systeem te ontwerpen dat als brug fungeert tussen de inputgegevens en de outputmetrieken. Zoals eerder aangegeven in Hoofdstuk 2, zijn er binnen computervisie verschillende technieken beschikbaar. Op basis van deze technieken werden verschillende oplossingsstrategieën geformuleerd, elk met hun eigen voor- en nadelen. Bij elk van deze strategieën is de laatste stap de koppeling van de blikdata aan de objectdetectie-output, waarbij enkel de objecten die ook daadwerkelijk bekeken zijn, worden behouden.

4.3.1. Strategie 1: Objectdetectie met Vooraf Getrainde Specifieke Modellen

Deze strategie is gebaseerd op het trainen van een objectdetectiemodel, zoals een variant van YOLO (zie 2.2.1), op de voorgedefiniëerde objecten.

Conceptuele Werking:

1. **Dataverzameling & Labeling**: Er wordt een uitgebreide dataset gecreëerd met beelden van elk te detecteren object. Deze beelden dienen de objecten vanuit verschillende hoeken, onder variërende belichtingscondities en tegen diverse achtergronden te tonen. Elk object in deze trainingsbeelden wordt vervolgens gelabeld met bounding boxes.
2. **Modeltraining**: Een objectdetectiemodel wordt getraind op deze dataset.
3. **Analyse van Evaluatieopnames:** Tijdens de analyse van een eyetracking-opname wordt het getrainde model frame-per-frame toegepast op de videobeelden.
4. **Koppeling met Blikdata**

Voordelen:

- **Snelheid tijdens Analyse:** Zoals vermeld in de stand van zaken, zijn YOLO-modellen geoptimaliseerd voor snelheid.
- **Eenvoudige Analyse:** Deze methode maakt gebruik van een enkel model, tegenover andere strategieën die meerdere modellen combineren.

Nadelen:

- **Dataverzameling:** Het creëren van een dataset met voldoende variatie in objecten kan tijdrovend zijn.
- **Modeltraining:** Het trainen van een model vereist aanzienlijke rekenkracht en tijd.

4.3.2. Strategie 2: Zero-Shot Objectdetectie en Tracking

Deze aanpak maakt gebruik van recente ontwikkelingen in zero-shot objectdetectie, zoals Grounding DINO, gecombineerd met segmentatie- en trackingmodellen zoals SAM 2. Met zero-shot wordt bedoeld dat het model geen specifieke training vereist voor de objecten die het moet detecteren.

Conceptuele Werking:

1. **Zero-Shot Detectie:** Een model zoals Grounding DINO wordt gebruikt om objecten in de videoframes te detecteren op basis van tekstuele prompts (bv. de namen van objecten).

2. **Segmentatie en Initialisatie van Tracking:** De output van Grounding DINO (bounding boxes en labels) kan worden gebruikt om initiële segmentatiemaskers te verkrijgen, eventueel verfijnd met SAM. Deze maskers dienen als input voor een trackingalgoritme.
3. **Object Tracking:** Een model zoals SAM 2, dat videosegmentatie ondersteunt, wordt gebruikt om de gedetecteerde en gesegmenteerde objecten doorheen de videosequentie te volgen.
4. **Koppeling met Blikdata**

Voordelen:

- **Geen Specifieke Training Nodig:** Dit elimineert de noodzaak voor het verzamelen van een uitgebreide, gelabelde dataset en het trainen van een specifiek model voor de Zorglab-objecten.
- **Potentieel Hogere Generaliseerbaarheid:** Foundation models zoals Grounding DINO zijn getraind op zeer grote en diverse datasets, waardoor ze potentieel beter omgaan met variaties in omgeving en objectuiterlijk, en makkelijker generaliseren naar nieuwe objecten.

Nadelen:

- **Filteren van Fout-Positieven:** Zero-shot modellen kunnen soms objecten detecteren die niet relevant zijn (fout-positieven) of incorrecte labels toekennen, wat filtering achteraf noodzakelijk maakt.
- **Computationele Kosten:** Foundation models zijn vaak groter en computationeel intensiever dan gespecialiseerde modellen zoals YOLO.
- **Complexiteit Pipeline:** Het combineren van meerdere modellen (detectie, segmentatie, tracking) kan leiden tot een complexere implementatie en potentiële accumulatie van fouten tussen de modules.

4.3.3. Strategie 3: Handmatige Initialisatie Gevolgd door Tracking

Deze strategie minimaliseert de afhankelijkheid van automatische objectdetectie, door een menselijke operator de initiële objecten te laten aanwijzen, waarna een trackingalgoritme het overneemt.

Conceptuele Werking:

1. **Handmatige Initialisatie:** In één of meerdere keyframes van de video klikt een operator handmatig op de te volgen objecten.
2. **Object Tracking:** Vanaf deze initiële handmatige definitie wordt een trackingalgoritme ingezet om de objecten doorheen de rest van de video te volgen.

3. **Koppeling met Blikdata**

Voordelen:

- **Hoge Accuraatheid bij Initialisatie:** Hoge accuraatheid bij handmatige initialisatie vermindert de kans op fout-positieven.
- **Geen Model Training Nodig:** Elimineert de noodzaak voor het trainen van een gespecialiseerd detectiemodel.

Nadelen:

Het handmatig initialiseren van objecten in elke video is extreem tijdrovend en schaalt slecht bij een groot aantal opnames.

4.3.4. Strategie 4: Blikgestuurde Segmentatie Gevolgd door Classificatie

Deze strategie stelt de eyetracking-data centraal in het segmentatieproces, waarna diverse modellen kunnen worden toegepast voor classificatie. Het maakt gebruik van de 'segment everything' capaciteit van moderne segmentatiemodellen, waarbij de blikdata vervolgens wordt ingezet om de voor classificatie relevante segmenten te identificeren.

Conceptuele Werking:

1. **Segment Everything en Object Tracking:** Een model zoals SAM 2 of FastSAM wordt toegepast om in elk frame van de video alle potentiële objecten te segmenteren. Deze gesegmenteerde objecten krijgen een unieke identifier en worden over de frames heen getrackt. Dit resulteert in een verzameling van segmentatiemaskers met bijbehorende object-ID's per frame.
2. **Koppeling met Blikdata:** Per frame wordt het blikpunt gebruikt om te bepalen welk(e) van de gesegmenteerde en getrackte objecten daadwerkelijk door de deelnemer worden bekeken. Dit kan bijvoorbeeld door te controleren of het blikpunt binnen een bepaald segmentatiemasker valt, rekening houdend met de inherente onnauwkeurigheid van de eyetracker.
3. **Segment Uitsnijden en Classificatie:** Enkel de segmenten die als 'bekeken' zijn geïdentificeerd, worden uit het frame geknipt. Deze uitsnedes kunnen vervolgens geclassificeerd worden met verschillende mogelijke modellen.

Hier is het mogelijk om verschillende benaderingen te gebruiken voor de classificatie van de segmenten:

- **Zero-Shot Classificatie:** Zero-shot classificatiemodellen zoals Grounding DINO kunnen worden toegepast op de gesegmenteerde objecten, waardoor geen specifieke training vereist is.

- **Getraind Classificatiemodel:** Een specifiek getraind model kan worden ingezet om de gesegmenteerde objecten te classificeren.
- **Keypoint-gebaseerde Classificatie:** Een keypoint-gebaseerd model, zoals SIFT of ORB, kan worden gebruikt om de segmenten te classificeren op basis van hun visuele kenmerken.
- **Feature-gebaseerde Classificatie:** Voorgetrainde modellen zoals DINOv2 kunnen in combinatie met een vector-database worden gebruikt om de segmenten te classificeren op basis van hun visuele kenmerken. Deze modellen genereren een vingerafdruk (feature vector) van elk segment, die vervolgens kan worden vergeleken met een database van gekende objecten.
- **Classificatie met Vision-API**: Voor classificatie kan ook eventueel gebruik gemaakt worden van bestaande foundation-model gebaseerde computer-vision diensten (Google, OpenAI, … APIs).

Voordelen:

- **Potentieel Robuustere Segmentatie en Tracking:** Door alle objecten te segmenteren en te tracken, kan het systeem stabieler zijn tegen kortstondige occlusies of snelle blikverschuivingen. Een object dat even niet wordt bekeken, blijft toch getrackt en kan later opnieuw als 'bekeken' worden geïdentificeerd zonder nieuwe segmentatie-initiatie.

Nadelen:

- **Beheer van Groot Aantal Segmenten:** Het tracken en beheren van ID's voor potentieel veel segmenten per frame kan complex zijn, vooral in drukke omgevingen.
- **Complexe Pipeline:** Het combineren van segmentatie, tracking en classificatie kan leiden tot een complexe implementatie.
- **Trager dan Directe Objectdetectie:** Het toepassen van meerdere modellen (segmentatie/tracking, classificatie) kan de snelheid van de analyse verminderen in vergelijking met een directe objectdetectie-aanpak.

4.4. Keuze van de Oplossingsstrategie

Na een zorgvuldige afweging van de hierboven beschreven strategieën, elk met hun inherente voor- en nadelen, is voor de ontwikkeling van de PoC applicatie gekozen voor **Strategie 4: 'Segment Everything' Gevolgd door Blikselectie en Diverse Classificatiemethoden**.

Deze keuze is primair ingegeven door de veelbelovende resultaten uit initiële, exploratieve tests. Waar zero-shot objectdetectie (Grounding DINO), in eerste experi-

menten nog geen consistent robuuste resultaten opleverde (met name wat betreft fout-positieven en generalisatie naar sommige van de specifieke Zorglab-objecten), toonde de 'segment everything' en tracking-capaciteiten van FastSAM direct indrukwekkende prestaties. Veel van de door de eyetracker geregistreerde objecten konden effectief worden gesegmenteerd en gevolgd. De uitdaging verschoof daarmee naar het vinden van een betrouwbare methode om deze dynamisch geïdentificeerde segmenten correct te classificeren.

- Strategie 1 (Trainen van een Objectdetector) kon op dit punt in het onderzoekproces niet worden overwogen, omdat er nog geen dataset beschikbaar was.
- Strategie 2 (Zero-Shot Objectdetectie) werd wel overwogen, maar de initiële resultaten waren niet robuust genoeg voor de beoogde toepassing.
- Strategie 3 (Handmatige Initialisatie) werd als te arbeidsintensief en niet schaalbaar beschouwd.

De gekozen Strategie 4 biedt de flexibiliteit om verschillende classificatietechnieken te exploreren voor de bekeken segmenten. Dit maakt een iteratieve ontwikkeling en optimalisatie van de PoC mogelijk zonder de onderliggende pipeline aanzienlijk te hoeven aanpassen. Hoewel het efficiënt beheren van vele segmenten en de complexiteit van de pipeline potentiële uitdagingen kent, wogen de initiële positieve resultaten van de segmentatie- en trackingcomponenten en de modulaire opzet voor de classificatiestap zwaarder door in de besluitvorming. De PoC applicatie werd uitgewerkt in functie van deze gekozen oplossingsstrategie, en zal in het volgende hoofdstuk verder worden toegelicht.

5

Proof-of-Concept Applicatie

5.1. Inleiding

Dit hoofdstuk dient als een uitgebreide toelichting op de softwareoplossing die binnen deze bachelorproef is ontwikkeld. Eerst zullen de verschillende componenten van de applicatie worden besproken, gevolgd door een gedetailleerde uitleg hoe de applicatie kan worden gebruikt. Aangezien er rekening gehouden werd met het effectief inzetten van de applicatie in de praktijk, zullen ook de gebruikte technologieën en technische keuzes worden toegelicht. Zo kan de software bij opeenvolgende bachelorproeven iteratief verder ontwikkeld worden voor nieuwe toepassingen en verbeteringen.

5.2. Applicatiecomponenten en Gebruikersinteractie

De workflow die bedacht werd voor de proof-of-concept applicatie, om van ruwe data naar bruikbare metrieken te gaan, kan als volgt worden samengevat:

1. Er worden één of meerdere 'kalibratie-opnames' gemaakt met de eyetrackingbril, waarbij elk object dat onderdeel is van het onderzoek, vanuit verschillende hoeken wordt gefilmd.
2. Ook worden er evaluatieopnames gemaakt van studenten die de simulatieomgeving gebruiken, waarbij de eyetracker op het hoofd van de student wordt geplaatst.
3. De opnames worden geïmporteerd in de applicatie via de WiFi-verbinding van de eyetrackingbril.
4. Men geeft een naam aan de simulatieomgeving (bijvoorbeeld 'Zorglab Zorgkamer') binnen de applicatie en definieert de objecten.

5. Daarna kan men beginnen met het labelen van de objecten in de kalibratie-opnames via de ingebouwde labeling-tool. De labeling-data dienen als basis voor het trainen of initialiseren van de analysemodellen.
6. De applicatie is nu in staat de eyetracking-opnames van de studenten te analyseren en de metrieken te visualiseren. Het visualisatiegedeelte viel buiten scope voor deze bachelorproef.

De applicatie is dus opgebouwd uit verschillende componenten die samen een stapsgewijs proces vormen. Deze zullen hieronder verder worden toegelicht.

5.2.1. Opnames Maken

Alvorens de applicatie ontwikkeld kon worden, waren er eerst een aantal video-opnames nodig. In deze fase werd de Tobii eyetracking-bril van het Zorglab ontleend om vertrouwd te raken met het maken van opnames. Deze opnames werden thuis uitgevoerd en dienden zowel als basis voor de applicatie zelf als voor initiële experimenten met de computer vision-modellen die later zouden worden ingezet in de labeling-tool en de analyses. Om een opname te maken hebben we twee zaken nodig: de eyetracker zelf met bijbehorende accessoires, en een laptop of computer waarop de 'Tobii Pro Glasses 3 Controller'[1] app is geïnstalleerd.

De eyetracker zelf bestaat uit een hub die stroom levert aan de bril en de opnames opslaat op een SD-kaart. Deze hub is verbonden met de bril zelf via een ingebouwde HDMI-kabel. De hub heeft een batterij die opgeladen kan worden met de bijgeleverde oplader. Eenmaal de hub ingeschakeld is, verschijnt er een WiFi-verbinding op de laptop of computer. Deze verbinding start met 'TG', gevolgd door het serienummer van de bril. Het wachtwoord voor de verbinding is 'TobiiGlasses'. Open de Tobii Pro Glasses 3 Controller-app en observeer of het camerabeeld van de bril zichtbaar is in de app.

Na het invoegen van de naam van de opname (of participant), drukt men op de knop 'Record' om de opname te starten. Voordat de opname kan starten, zal de app vragen om de bril te kalibreren. Dit kan gedaan worden met de bijgeleverde kalibratiekaart, die op armlengte afstand van de bril dient gehouden te worden. Aan de participant wordt gevraagd om naar de centrale stip op de kaart te kijken, en vervolgens wordt er op de knop 'Calibrate' gedrukt. De app begint nu met het opnemen van de video, en de participant kan nu vrij rondlopen in de simulatieomgeving.

Opmerking over Belichting

Bij het maken van opnames werd een probleem vastgesteld wanneer een object zich op een locatie bevond met een hoge lichtinval. In deze situaties gingen de objecten vaak verloren in de achtergrond, waardoor het moeilijk werd om deze te

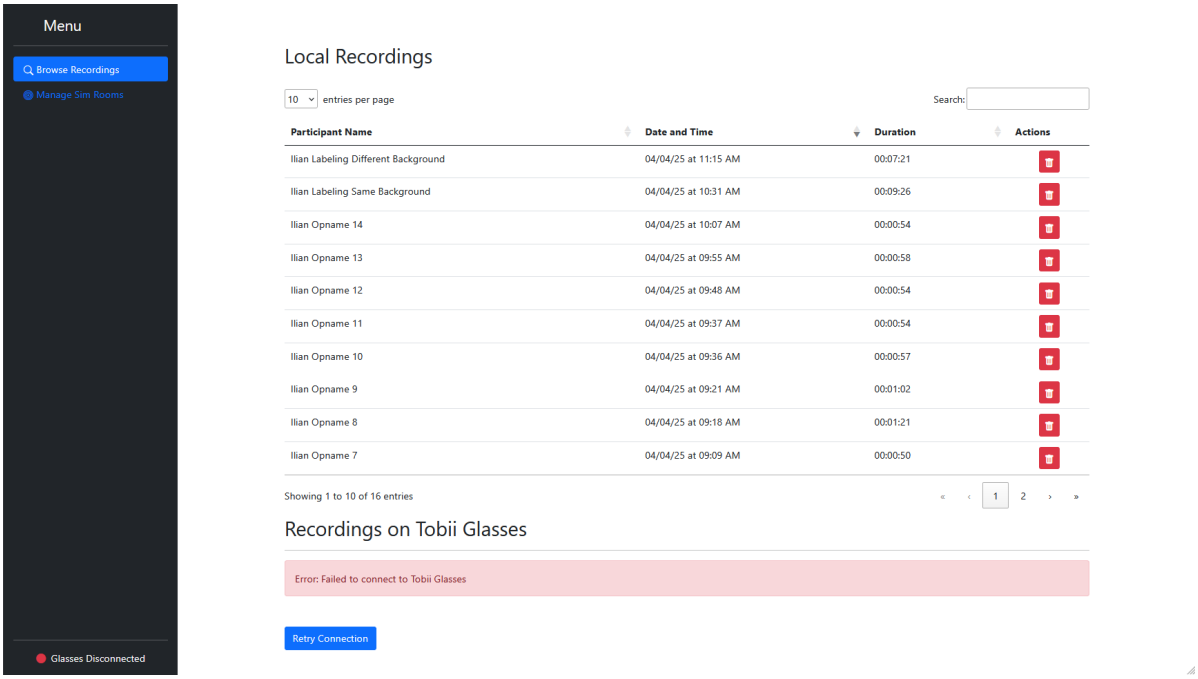
[1] https://www.tobii.com/products/eye-trackers/wearables/tobii-pro-glasses-3/controller-app

detecteren. Daarom is het belangrijk om bijzondere aandacht te besteden aan de belichting van de omgeving waarin de opnames worden gemaakt.

5.2.2. Importeren van Eyetracking-Opnames

Zoals eerder vermeld heeft de Tobii eyetracker twee componenten: de bril zelf en een 'hub' die stroom levert aan de bril en de opnames opslaat op een SD-kaart. Het zou dus mogelijk zijn om de opnames via de SD-kaart te importeren, maar dit is niet praktisch aangezien de SD-kaart steeds uit de hub gehaald dient te worden. Daarom biedt de hub ook een WiFi-verbinding aan, zodat de opnames via een netwerkverbinding kunnen worden geïmporteerd.

Wanneer men de applicatie opent op de pagina 'Browse Recordings' via de navigatiekolom, worden er twee tabellen getoond (zie figuur 5.1)

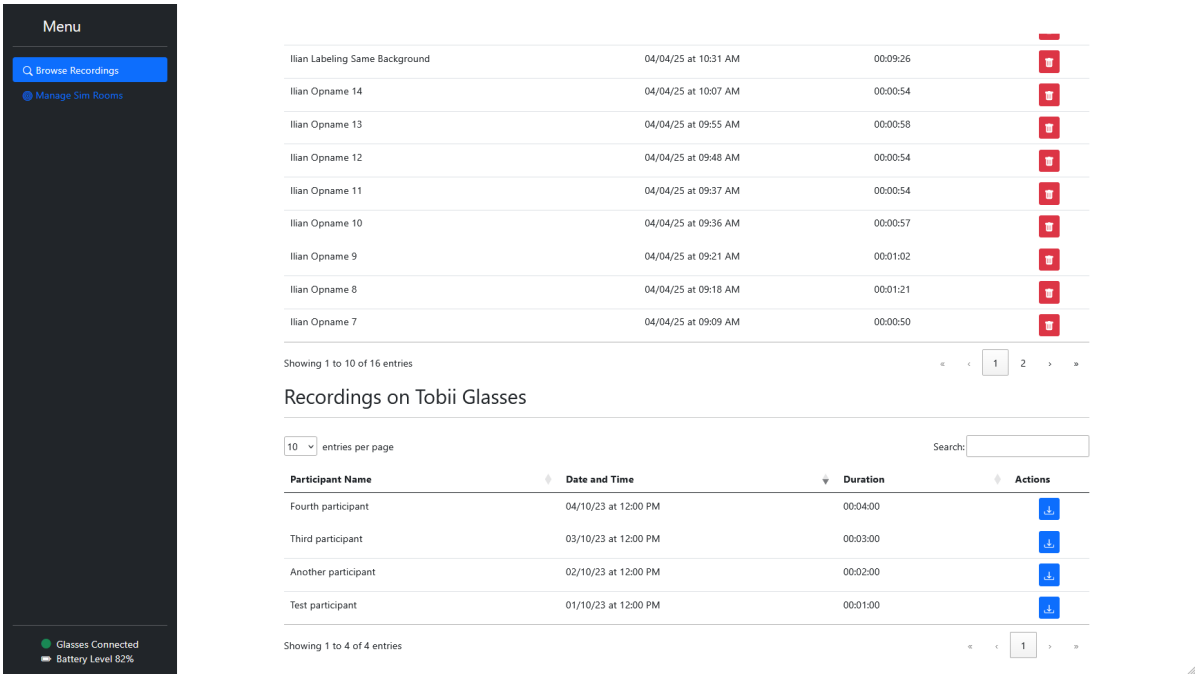


Figuur 5.1: Screenshot van de 'Browse Recordings'-pagina waar opnames kunnen worden geïmporteerd. Hier is de bril niet verbonden met de computer.

De eerste tabel 'Local Recordings' toont de opnames die lokaal zijn opgeslagen op de computer, en de onderste tabel 'Recordings on Tobii Glasses' toont de opnames die zijn opgeslagen op de hub. Indien een tabel geen opnames bevat, wordt er een passend bericht getoond. De opnames worden getoond in tabellen met de naam van elke opname, de datum en tijd waarop deze werd gemaakt, evenals de duur van de opname. Het is mogelijk om opnames te verwijderen uit de 'Local Recordings'-tabel door op de rode knop met het prullenbakje te klikken naast een opname. Dit heeft geen invloed op de opnames die zijn opgeslagen op de hub. Ook kan men zoeken naar opnames via de zoekbalk bovenaan de tabellen, en kan

men deze sorteren op naam, datum of duur door op de bijbehorende kolomkop te klikken.

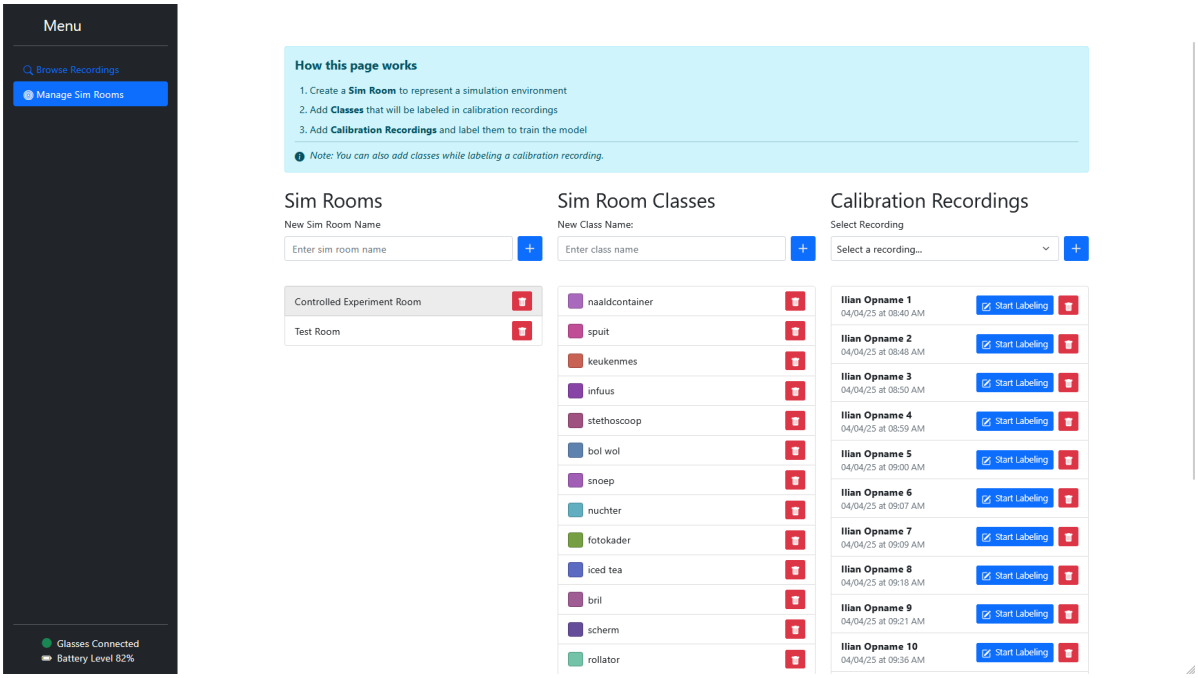
Wanneer de bril niet verbonden is met de computer, wordt er een aangepaste melding getoond, met een knop om opnieuw te verbinden. Figuur 5.1 toont een voorbeeld van de interface van de applicatie, waar de bril niet verbonden is met de computer. Linksonder in de interface toont de applicatie de huidige status van de verbinding met de bril, inclusief de batterijstatus. Bij figuur 5.2 is de bril wel verbonden met de computer. Elke rij in de onderste tabel bevat een blauwe knop met een pijl naar beneden, waarmee een opname kan worden geïmporteerd vanuit de bril naar de computer. Dit kan enige tijd duren, afhankelijk van de grootte van de opname.



Figuur 5.2: Bij dit voorbeeld is de bril wel verbonden met de computer. Men ziet linksonder de batterijstatus van de bril. In de onderste tabel is het mogelijk om opnames te importeren vanuit de bril naar de computer.

5.2.3. Simulatieruimten en Objecten

Eens de opnames zijn geïmporteerd, kunnen we overgaan tot het definiëren van de objecten in de kalibratie-opnames. Één van de design-keuzes was om met zogenaamde 'simulatieruimten' te werken die verschillende omgevingen voorstellen, elk met hun eigen objecten. Men kan zich dus voorstellen dat men niet enkel in het Zorglab werkt, maar bijvoorbeeld ook in een echt ziekenhuis of een woonzorgcentrum. Het doel was dus om de applicatie zo gebruiksvriendelijk mogelijk te maken, om het werk van de trainers te vereenvoudigen.



Figuur 5.3: Voorbeeld van de 'Manage Sim Rooms'-pagina waar de simulatieomgevingen kunnen worden gedefinieerd. Hier is de 'Controlled Experiment Room' geselecteerd, met een aantal objecten en kalibratie-opnames.

In figuur 5.3 is een voorbeeld te zien van de pagina waar simulatieomgevingen kunnen worden gedefinieerd. Deze pagina heeft drie kolommen:

- De eerste kolom toont de reeds gedefinieërde simulatieomgevingen, met hun naam, een knop om deze te verwijderen, en een invoerveld om meer simulatieomgevingen toe te voegen.
- Wanneer een simulatieomgeving is geselecteerd, worden de bijbehorende objecten getoond in de tweede kolom. Elk object krijgt automatisch een kleur toegewezen die ook gebruikt zal worden in de labeling-tool en eventueel de visualisatie van de metrieken.
- In de derde kolom kan men geïmporteerde opnames selecteren om deze te gebruiken als kalibratie-opnames. Elke kalibratie-opname heeft een knop 'Start Labeling'. Wanneer men hierop klikt, wordt de labeling-tool geopend met de geselecteerde opname.

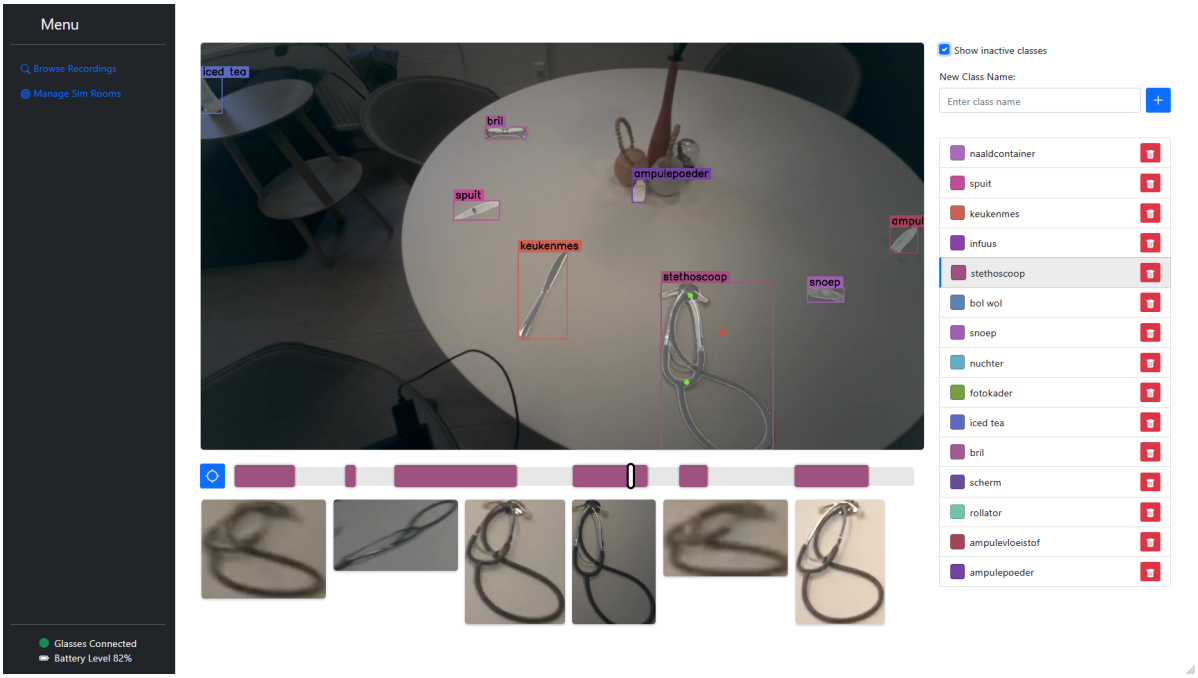
Merk op dat het mogelijk is om om het even welke opname te selecteren als kalibratie-opname, waardoor ook evaluatieopnames kunnen worden gebruikt! Zo kunnen we reeds gemaakte opnames van studenten ook gebruiken om de analysemodellen te trainen. Dit maakt het mogelijk om de opnames te analyseren op basis van manuele labels door de trainers. Deze aanpak zal meer tijd vergen, maar is veel nauwkeuriger dan een automatische analyse. Aangezien het in deze bachelorproef gaat om geautomatiseerde analyse, werd deze optie niet verder uit-

gewerkt.

Wanneer men een kalibratie-opname verwijdert, heeft dit geen invloed op de opname zelf, maar enkel op de associatie met de simulatieomgeving. Indien er in de opname werd gelabeld binnen deze simulatieomgeving, worden deze labels wel verwijderd. Hetzelfde geldt voor de objecten: indien een object wordt verwijderd, worden ook de labels die gemaakt werden met dit object in alle kalibratie-opnames verwijderd.

5.2.4. Labeling Tool

We hebben het al eerder gehad over de labeling-tool, maar hoe werkt deze nu precies? Dit is één van de belangrijkste onderdelen van de applicatie, en ook het meest complexe. De labeler kunnen we openen door op de knop 'Start Labeling' te klikken van een kalibratie-opname binnen de 'Manage Sim Rooms'-pagina. Bij het laden van de labeler worden alle individuele frames uit de opname gehaald en in een map opgeslagen, waardoor het enige tijd kan duren voor de tool opent. De specifieke implementatieredenen hiervoor komen later aan bod. Het doel van de labeler is om een masker en een 'bounding box' te verkrijgen binnen elke frame van de opname waarin een object zich bevindt. Natuurlijk zou dit een grote tijdsinvestering vragen bij een volledig manuele tool: aangezien de framerate van de eyetracker 25fps (Frames per Second) is, zou dit betekenen dat we 1500 frames moeten labelen voor een opname van 1 minuut, en dit voor elk object dat we willen labelen. Het probleem werd opgelost met een semi-automatische labeling-tool. Daarbij hoeft men enkel objecten een aantal keer aan te klikken. De tool volgt vervolgens elk object automatisch doorheen de hele opname. De functies van de labeling-tool worden hieronder verder toegelicht.



Figuur 5.4: Een screenshot van de labeling-tool met vier onderdelen: de labeling-canvas, de tijdlijn, de objectenlijst rechts, en een lijst van gemaakte annotaties onderaan. In dit voorbeeld is het object 'stethoscoop' geselecteerd.

Eenmaal de labeling-tool geopend is, zien we een pagina met vier onderdelen (zie figuur 5.4). Linksboven zien we het 'labeling-canvas', waar de huidige geselecteerde frame van de opname wordt getoond. Onder het canvas is er een tijdlijn die aangeeft waar we ons bevinden in de opname. De tijdlijn markeert ook welke frames het gevolgde object bevatten. Binnen deze canvas kunnen we objecten labelen door eerst een object in de lijst rechts te selecteren, en vervolgens te klikken op het canvas. Hierbij kunnen we een linkermuisklik gebruiken om aan te geven welke delen van de afbeelding deel uitmaken van het object dat we willen labelen (zie de stethoscoop in figuur 5.4). Met de rechtermuisklik geven we aan welke regio's buiten het object vallen. Deze zijn respectievelijk gemarkeerd met een groene en een rode stip. Als we met de muis over een stip gaan, wordt deze gemarkeerd. Vervolgens kunnen we erop klikken om de stip te verwijderen. Na elke muisklik worden het masker en de bijbehorende *bounding box* automatisch geüpdatet om feedback te geven aan de gebruiker.

Onder de tijdlijn zien we een reeks annotaties die we hebben gemaakt, van links naar rechts gesorteerd op tijd. Men kan op deze annotaties klikken om naar de bijbehorende frame te gaan, en ze ook verwijderen door met de muis erover te gaan en op de rode knop te klikken. Indien men alle stippen verwijdert, wordt de annotatie ook verwijderd. Eens we tevreden zijn met de annotaties, kunnen we op de blauwe 'tracking'-knop klikken aan de linkerkant van de tijdlijn. Dit start de automatische tracking van het huidige geselecteerde object doorheen de opname,

zoals getoond in figuur 5.5.

Figuur 5.5: In deze screenshot zien we dat er een tracking-opdracht gaande is. Hier is ook de optie 'Show inactive classes' uitgevinkt, waardoor enkel het actieve object wordt getoond.

Uit praktische overweging worden niet alle frames bekeken binnen een tracking-opdracht. Een annotatie wordt als basis gebruikt voor tracking, en wanneer het object voor een bepaalde tijd uit het beeld verdwijnt, wordt de trackingopdracht stopgezet tenzij er nog annotaties overblijven. Over het algemeen hebben we slechts één annotatie nodig voor elk gedeelte van de video waarin het object voortdurend in beeld is. Soms zijn echter meerdere annotaties per gedeelte nodig om het trackingresultaat te optimaliseren. Het specifieke algoritme dat bedacht werd voor de tracking komt aan bod in de technische sectie van dit onderdeel. Tenslotte bestaat de mogelijkheid dat er veel objecten in beeld zijn waardoor maskers, bounding boxes en labels elkaar overlappen. Om deze reden kunnen we rechtsboven een optie 'Show inactive classes', uitvinken om enkel het actieve object te tonen in het canvas.

5.2.5. Analyse van Eyetracking-Opnames

Hoewel het aanvankelijk de bedoeling was, werd in de context van deze bachelorproef, geen werkende analyse-tool ontwikkeld maar enkel een proof-of-concept. De labeling-tool was in dit opzicht de belangrijkste component, omdat deze de labels genereert die nodig zijn voor de analyse. Aangezien het hier enkel een proof-of-concept betrof, werd hier ook geen pagina voor ontwikkeld. Indien men in de toekomst een betrouwbaar analysemodel kan ontwikkelen, bestaat de mogelijkheid om een nieuwe pagina te maken die de resultaten van de analyses toont. Naast

de pagina zelf dient dan ook een bijbehorende optie in de navigatiekolom aangemaakt te worden. De analyses die uitgevoerd werden in deze bachelorproef komen aan bod in Hoofdstuk 6.

5.3. Technische Specificaties

We hebben het gehad over de features van de applicatie vanuit het perspectief van een gebruiker. In deze sectie gaan we dieper in op de technische aspecten van de applicatie, met als doel de keuzes te verduidelijken en eventuele nieuwe developers wegwijs te maken in de codebase.

5.3.1. Softwarearchitectuur

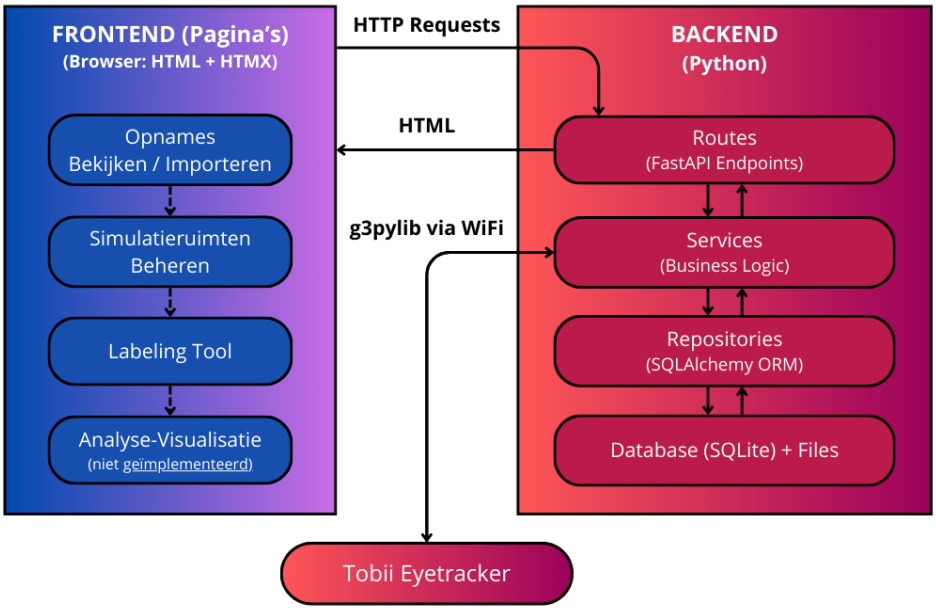
Voor de ontwikkeling van de applicatie werd een lichtgewicht, modulaire softwarestack gekozen, afgestemd op de specifieke noden van het project. Er is geopteerd voor een web-applicatie die via een browser toegankelijk is en lokaal kan draaien op een laptop (met een degelijke grafische kaart) of een desktop computer.

- **Backend Framework:** Python met `FastAPI` werd gekozen omwille van zijn gebruiksvriendelijkheid, asynchrone mogelijkheden (belangrijk voor I/O-intensieve taken zoals data-import en analyse), automatische documentatie en type-hinting-integratie.
- **Frontend Aanpak:** In plaats van een zwaar JavaScript-framework, werd geopteerd voor `HTMX`. Dit laat toe om dynamische interacties te bouwen door HTML direct uit te wisselen met de server, wat de complexiteit van de frontend aanzienlijk reduceert. `Jinja2` dient als templating engine om deze HTML serverside te genereren. Tenslotte werd `Bootstrap` gebruikt voor de opmaak van de applicatie.
- **Database:** Een `SQLite` database werd geselecteerd vanwege de eenvoud en het feit dat de applicatie bedoeld is voor lokaal gebruik, waardoor een complexe database server overbodig is. `SQLAlchemy` werd ingezet als ORM (Object-Relational Mapping) voor een gestructureerde interactie met de database vanuit Python.
- **Hardware Communicatie:** Voor de interactie met de Tobii Pro Glasses 3 werd de officiële `g3pylib`[2] SDK gebruikt, die toegang biedt tot opnames en statusinformatie via de WiFi-verbinding.
- **Development & Kwaliteitsborging:** `Poetry` werd gebruikt voor dependency management. Strikte type hinting met `Mypy` en code linting/formatting met `Ruff` dragen bij aan de robuustheid en onderhoudbaarheid van de code. `Doc-`

[2] https://github.com/tobiipro/g3pylib(laatstgeraadpleegdop2025-05-30)

`ker` werd ingezet om de applicatie te containeriseren, wat zorgt voor een consistente uitvoeringsomgeving en eenvoudige deployment.

Deze keuzes resulteerden in een applicatie die relatief eenvoudig te onderhouden en verder te ontwikkelen is, conform de doelstelling om iteratieve verbeteringen in volgende bachelorproeven mogelijk te maken. Er werd gewerkt met een gelaagde architectuur om een duidelijke scheiding van verantwoordelijkheden (Separation of Concerns) te waarborgen en onderhoud en testing te vereenvoudigen. De data en requests vloeien doorgaans van de `frontend` via de `routes` naar de `services`, die vervolgens de `repositories` aanroepen voor databasetoegang (zie figuur 5.6).



Figuur 5.6: Software-architectuur van de applicatie. De applicatie is opgebouwd uit verschillende lagen die elk hun eigen verantwoordelijkheden hebben.

Frontend (HTML, HTMX, en Jinja2)

De gebruikersinterface (UI) wordt weergegeven in de browser en is opgebouwd uit HTML-pagina's die server-side worden gegenereerd met behulp van de `Jinja2` templating engine. Deze engine maakt het mogelijk om inhoud (data vanuit de Backend) te integreren in de HTML-pagina's door gebruik te maken van variabelen en controle-structuren. Voor de dynamische aspecten van de applicatie, waaronder het bijwerken van de UI zonder de volledige pagina te herladen, wordt `HTMX` ingezet.

`HTMX` maakt het mogelijk om via HTML-attributen oproepen te doen naar de backend (vb: als een knop wordt ingedrukt). De backend genereert vervolgens het gevraagde HTML-fragment—zoals een tabel met opnames of een lijst van objecten—die HTMX vervolgens injecteert in de huidige pagina. Op deze manier kunnen

eenvoudige interacties worden gerealiseerd zonder de noodzaak van een volledig JavaScript-framework zoals React of Vue.js.

Binnen de 'Browse Recordings'-en 'Manage Sim Rooms'-pagina's is HTMX grotendeels voldoende, maar toch valt JavaScript niet volledig te vermijden. De labeling-tool vergt immers complexe interacties die niet eenvoudig gerealiseerd kunnen worden met enkel HTML en HTMX. Zo werd de labeling-tool opgesplitst in vijf componenten die elk hun eigen verantwoordelijkheden hebben:

- De **labeling-canvas** die de huidige frame toont en het mogelijk maakt om objecten te labelen.
- De **tijdlijn** die de voortgang van de opname toont en het mogelijk maakt om naar een specifieke frame te navigeren.
- De **objectenlijst** die de beschikbare objecten toont en het mogelijk maakt om objecten te beheren en te selecteren.
- De **annotatielijst** die de gemaakte annotaties van het momenteel geselecteerde object toont.
- De **labeling settings** die instellingen van de labeling-tool toont ('Show inactive classes').

Elk van deze componenten communiceert apart via HTMX-attributen met de backend, en kunnen onafhankelijk van elkaar worden geladen. Wanneer toch communicatie nodig is tussen de componenten, gebeurt dit via `Events` die door de componenten worden uitgezonden en opgevangen. Een voorbeeld hiervan is wanneer de gebruiker klikt op de tijdlijn om naar een specifieke frame te navigeren. De tijdlijn stuurt een event naar de labeling-canvas, die vervolgens een oproep doet naar de backend om de juiste frame op te halen.

Aangezien het gaat om een single-user applicatie, is het toch mogelijk om een groot deel van de logica over te dragen aan de backend. De frontend is dus voornamelijk verantwoordelijk voor de presentatie van de data en de interactie met de gebruiker. De backend is verantwoordelijk voor de verwerking van de data, en het bijhouden van de status van de applicatie (de huidige frame, de geselecteerde objecten, de machine-learning modellen, etc.).

Backend (Python met FastAPI)

De backend, geschreven in Python met het `FastAPI` framework, is verantwoordelijk voor het verwerken van requests, het uitvoeren van de businesslogica, en de interactie met de datalaag en externe hardware.

Routes (FastAPI Endpoints)
De `routes` definiëren de API-endpoints waarnaar de frontend requests stuurt. Deze laag ontvangt HTTP-requests, valideert inkomende data (via Pydantic modellen die FastAPI automatisch uit request bodies of query parameters haalt), roept de juiste methoden aan in de `services`-laag, en formatteert de respons. Voor HTMX-requests is dit typisch een HTML-fragment gerenderd met Jinja2; voor andere endpoints kan dit JSON zijn (bv. het ophalen van annotatiepunten voor de labeler).

Services (Business Logic)
De `services`-laag bevat de kernlogica van de applicatie. Services coördineren operaties en implementeren de business rules. Ze zijn ontkoppeld van de HTTP-laag en de directe database-implementatie. In plaats daarvan gebruiken ze methoden uit de `repositories`-laag voor datatoegang- en persistentie, en kunnen ze andere utilities of externe bibliotheken aanroepen (bv. `g3pylib` voor communicatie met de Tobii-bril, of `SAM2ImagePredictor` voor segmentatie).

Repositories (Data Access)
De `repositories`-laag vormt de abstractielaag boven de database en het filesysteem. Het bevat alle logica voor het ophalen, opslaan, en beheren van data in de `SQLite` database via `SQLAlchemy` ORM. Ook bevat het logica voor het beheren van bestanden op de schijf, zoals resultaten van de labeling-tool of de video-opnames. Dit zorgt ervoor dat de `services`-laag onafhankelijk is van de specifieke database-implementatie.

5.3.2. Backend van de Labeling Tool en Tracking-logica

In deze sectie gaan we dieper in op wat er in de achtergrond gebeurt wanneer de gebruiker objecten labelt in de labeling-tool.

Decoderen van Video-opnames

Alvorens de gebruiker kan beginnen met labelen, moeten eerst de frames van de opname worden geëxtraheerd. Op deze manier moeten niet alle frames in het geheugen geladen worden, wat onmogelijk zou zijn voor lange opnames. Ook neemt het enige tijd om een enkele frame te extraheren uit een video-opname, wat de gebruikerservaring zou beïnvloeden. Voor het decoderen van een opname wordt gebruik gemaakt van `ffmpeg`, een populaire open-source tool die verschillende functies biedt voor het verwerken van video- en audiobestanden. Zo worden alle afzonderlijke frames opgeslagen in een map, met als naam het indexnummer van het frame.

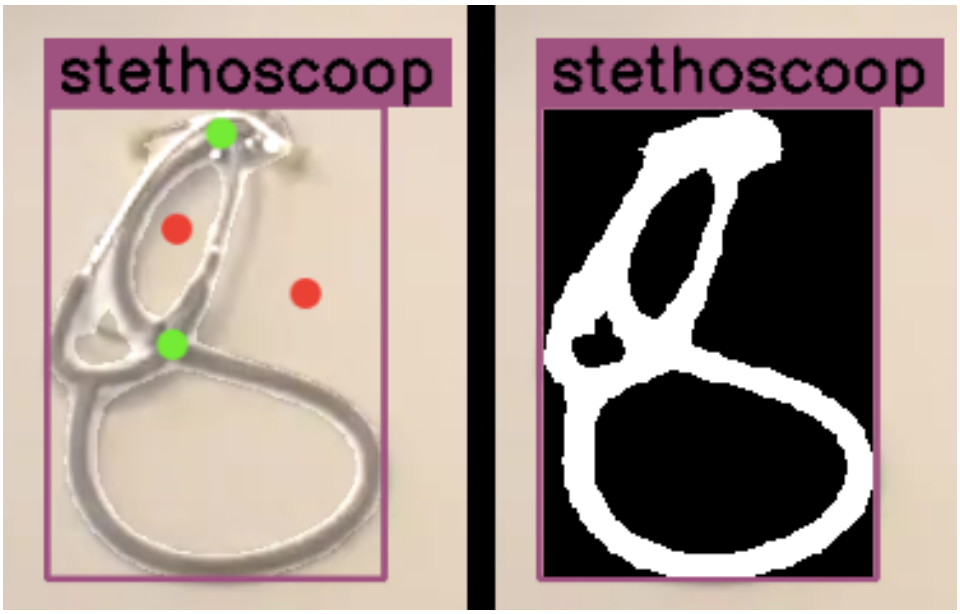
Doorheen zowel de applicatie als de analyses is het heel belangrijk om dezelfde video-decoder te gebruiken. Verschillende decoders (zoals `ffmpeg` of `OpenCV`) kun-

nen andere frame- en tijdsindexering gebruiken. Als men bijvoorbeeld een frame-index van 100 opgeeft, kan dit in de ene decoder overeenkomen met de 100ste frame, terwijl het in een andere decoder de 101ste frame is. Dit kan leiden tot inconsistenties in de resultaten van de analyses, en dus werd er gekozen om steeds `ffmpeg` te gebruiken.

Annotaties en Tracking

Voor een beter begrip van het waarom van manuele annotaties, en hoe de geautomatiseerde tracking werkt, moeten we eerst begrijpen hoe het achterliggende machine-learning model werkt. Er werd gekozen om het Segment Anything Model 2 (SAM2) van Meta te gebruiken, dat in staat is om objecten te segmenteren (een masker genereren) in zowel een afbeelding als een video.

Via het 'promptable-segmentation'-mechanisme van SAM2 kunnen we een object segmenteren door een aantal punten te geven die het object beschrijven. Men kan een positief punt geven dat 'binnen' het object ligt (voorgesteld door een 1), en een negatief punt dat 'buiten' het object ligt (voorgesteld door een 0). Zo krijgen we een annotatie die uit meerdere punten bestaat (zie figuur 5.7).



Figuur 5.7: Links een annotatie met een aantal positieve (groene) en negatieve (rode) punten zoals zichtbaar in de labeling-tool. Rechts het overeenkomstig masker dat werd gegenereerd door SAM2.

De manuele annotaties vormen de basis voor de automatische tracking van het object doorheen de opname. Zoals eerder vermeld, worden niet alle frames bekeken binnen deze tracking-opdracht. Het frame-per-frame verwerken van een volledige video-opname is immers onnodig (wanneer een object voor een lange tijd niet in beeld is) en tijdrovend.

Om de tracking efficiënt uit te voeren, fungeert elke manuele annotatie als een

startpunt. De `TrackingJob` klasse initieert vanuit deze 'ankerpunten' het tracking-proces, zoals geïmplementeerd in de run methode:

```
    def run(self) → None:
        self.initialize()

        annotations_frame_idx = [annotation.frame_idx for annotation
            in self.annotations]
        last_tracked_annotation = -1

        while last_tracked_annotation ≠ len(self.annotations) - 1:
            start_frame_idx =
                annotations_frame_idx[last_tracked_annotation + 1]
            self.progress = start_frame_idx / self.frame_count

            # Voorwaarts tracken tot tracking verlies:
            list(self.track_until_loss(start_frame_idx, reverse=True))

            # Achterwaarts tracken tot tracking verlies:
            for frame_idx in self.track_until_loss(start_frame_idx):
                self.progress = frame_idx / self.frame_count
                if frame_idx in annotations_frame_idx:
                    last_tracked_annotation =
                        annotations_frame_idx.index(frame_idx)
```

Codefragment 5.1: De `run` methode van `TrackingJob` itereert over de gesorteerde manuele annotaties en start vanuit elk ankerpunt een gelokaliseerd tracking-proces om de rekentijd te beperken.

De strategie is als volgt:

1. De `TrackingJob` ontvangt een lijst van manuele annotaties, die eerst gesorteerd worden op frame-index.
2. Vervolgens wordt er over deze annotaties geïtereerd. Voor elke nog niet verwerkte annotatie (`start_frame_idx`) wordt de functie `track_until_loss` aangeroepen, eerst in achterwaartse richting (`reverse=True`) en daarna in voorwaartse richting.
3. Dit zorgt ervoor dat de tracking zich vanuit elk betrouwbaar, manueel gelabeld punt propageert over videosegmenten waar het object vermoedelijk aanwezig is. Dit reduceert het totaal aantal te verwerken frames aanzienlijk.

De kern van het daadwerkelijke volgen binnen deze segmenten zit in de `track_until_loss` methode. Deze methode maakt gebruik van de

`propagate_in_video` functionaliteit van de `SAM2VideoPredictor`.

```python
    def track_until_loss(
        self, start_frame_idx: int, reverse: bool = False
    ) -> Generator[int, None, None]:
        tracking_loss = 0 # Teller voor het aantal frames zonder
            tracking

        with torch.amp.autocast("cuda"):
            for (
              out_frame_idx, _, out_mask_logits
            ) in self.video_predictor.propagate_in_video(
                inference_state=self.inference_state,
                start_frame_idx=start_frame_idx,
                reverse=reverse,
            ):
                yield out_frame_idx # Rapporteer de verwerkte frame
                    index voor de progress bar

                mask_torch = out_mask_logits[0] > 0.5
                if mask_torch.any(): # Is er een masker getedecteerd?
                    tracking_loss = 0 # Reset teller bij succesvolle
                        tracking

                    # ... (opslaan van resultaten: masker, box, roi)
                        ...

                    file_path = self.results_path /
                        f"{out_frame_idx}.npz"
                    np.savez_compressed(
                        file_path,
                        # ... (data)
                    )
                else:
                    tracking_loss += 1 # Verhoog teller bij mislukte
                        tracking

                if tracking_loss >= self.GRACE_PERIOD: # Tolerantie
                    overschreden?
                    break # Stop tracking in deze richting
```

Codefragment 5.2: De `track_until_loss` methode propageert het masker frame-per-frame binnen een beperkt segment. De `GRACE_PERIOD` helpt om de tracking robuust te houden tegen kortstondige detectieproblemen binnen dit segment.

Waarom geen FastSAM?

Men zou zich kunnen afvragen waarom het FastSAM-model niet werd gebruik, aangezien het ook in staat is om objecten te segmenteren in video-opnames en bovendien veel sneller is. Het probleem is dat de gegenereerde maskers van FastSAM minder nauwkeurig zijn dan die van SAM2 (Zhao e.a., 2023). Aangezien het om labeling gaat, is het belangrijk dat de maskers zo nauwkeurig mogelijk zijn. Het is mogelijk dat dit verschil in de context van deze bachelorproef niet zo belangrijk is, maar het zou onnauwkeurig zijn om dit te stellen zonder de verschillen in detail te analyseren. Hoewel het testen van FastSAM binnen de labeling-tool niet verder werd uitgewerkt, werd dit model wel gebruikt voor de analyses in Hoofdstuk 6. Zo zijn er voorbeelden van het gebruik van beide modellen binnen de codebase terug te vinden, en kan de labeling-tool overschakelen naar FastSAM indien gewenst.

Prestaties en Geheugenverbruik

Belangrijke overwegingen bij het implementeren van de applicatie in het Zorglab zijn de hardwarevereisten. Het Zorglab is uitgerust met een computer met een krachtige CPU (Central Processing Unit) en 32GB RAM, maar heeft momenteel geen GPU (Graphics Processing Unit). Daarom is het interessant om te kijken wat de vereisten zijn voor een vlotte werking van de applicatie.

De ontwikkeling van de applicatie, inclusief alle analyses, werd uitgevoerd op een computer met de volgende specificaties:

- **Processor:** Ryzen 7 7700X @ 4.50 GHz (8 cores)
- **Geheugen:** 32GB DDR5
- **Grafische Kaart:** NVIDIA GeForce RTX 4090 (24GB VRAM)
- **Opslag:** 1TB M.2 SSD

De machine in het Zorglab is een Dell Precision 7550 met de volgende specificaties:

- **Processor:** Intel Core i9-10885H CPU @ 2.40 GHz (16 cores)
- **Geheugen:** 32GB
- **Grafische Kaart:** Geen

Verschil tussen CPU en GPU

Voor taken binnen machine-learning, zoals de hier toegepaste beeldsegmentatie, is een GPU significant sneller dan een CPU. Een CPU specialiseert zich in het sequentieel uitvoeren van een beperkt aantal complexe taken. Een GPU daarentegen is ontworpen met duizenden kleinere cores (de RTX 4090 heeft er bijvoorbeeld 16,384) om een grote hoeveelheid relatief eenvoudigere taken parallel uit te voeren.

Dit parallellisme is ideaal voor de matrixoperaties die de ruggengraat vormen van neurale netwerken zoals SAM2.

Niet enkel het aantal cores is bepalend, maar ook de hoeveelheid en snelheid van het geheugen op de GPU (VRAM). Zowel het model zelf (de parameters of 'gewichten') als de te verwerken data (in dit geval de videoframes) moeten tegelijkertijd in het VRAM geladen kunnen worden voor efficiënte verwerking. Pogingen om dergelijke modellen uitsluitend op een CPU uit te voeren, resulteren in onpraktisch lange verwerkingstijden, waardoor de interactiviteit van de labeling tool verloren zou gaan.

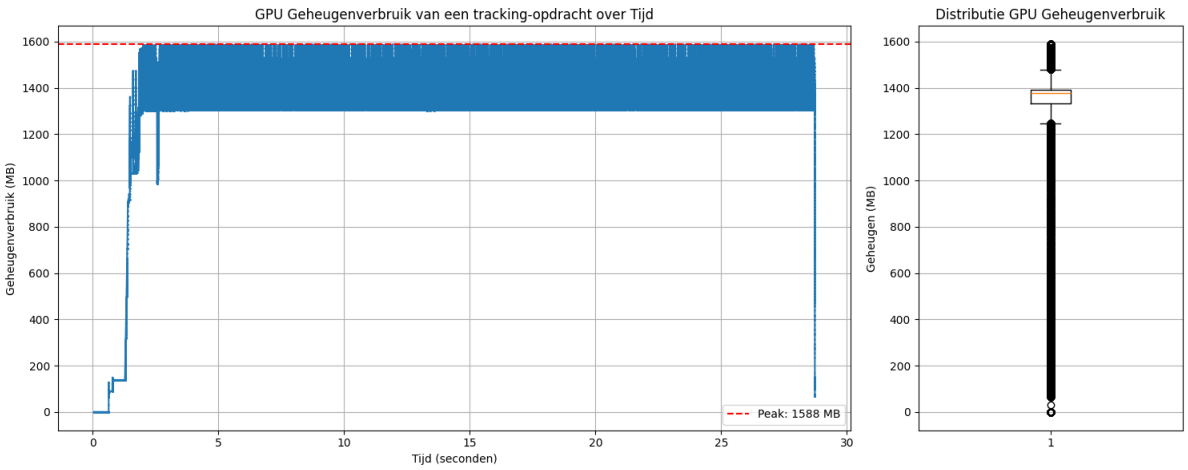
Analyse van de prestaties

Het SAM2-model komt in vier verschillende varianten, elk met een andere hoeveelheid parameters en dus ook andere geheugenvereisten. Via de `torch.profiler` module werd het geheugengebruik van elk model gemeten tijdens het laden van de parameters. De resultaten hiervan zijn als volgt:

- `sam2.1_hiera_large.pt`: 1000 MB
- `sam2.1_hiera_base_plus.pt`: 450 MB
- `sam2.1_hiera_small.pt`: 315 MB
- `sam2.1_hiera_tiny.pt`: 288 MB

Aangezien geheugenvereisten relatief laag zijn, werd besloten om het grootste model te gebruiken, dat ook de beste resultaten oplevert.

Naast de modelparameters is het ook belangrijk het geheugengebruik te meten bij de tracking-opdracht. Zo werd doorheen een tracking-opdracht gemeten hoeveel videogeheugen er werd gebruikt door het SAM2 model. Het ging om een totaal van 360 getrackede frames aan een snelheid van 10.5fps, met een piek geheugenverbruik van 1599 MB (zie figuur 5.8).



Figuur 5.8: Het geheugengebruik van SAM2 tijdens een tracking-opdracht. Links het cumulatieve geheugengebruik doorheen de tijd met een piek van 1599MB, rechts een boxplot van het geheugengebruik met een gemiddelde van 1374 MB.

Deze analyses zijn terug te vinden in de file `experiment/12_performance.ipynb` in de Git repository.

Optimalisatie van Geheugenbeheer in de SAM2-bibliotheek

Tijdens de ontwikkeling van de tracking-functionaliteit werd een probleem met toenemend GPU-geheugenverbruik in de originele SAM2-bibliotheek van Meta geïdentificeerd (Hu e.a., 2024). Bij langdurige tracking-opdrachten, bleek dat de bibliotheek intermediaire resultaten of toestanden van reeds verwerkte frames in het GPU-geheugen vasthield. Dit leidde tot een lineaire toename van het VRAM-gebruik naarmate de tracking vorderde. Uiteindelijk kon dit de beschikbare GPU-geheugenlimiet overschrijden, en zo doende resulteren in een crash van de applicatie.

Om dit probleem aan te pakken en het GPU-geheugenverbruik te stabiliseren, werd er een fork gemaakt van de originele SAM2-bibliotheek waarin een aanpassing is doorgevoerd (Hu e.a., 2025). De oplossing is geïnspireerd door een Github-issues comment (heyoeyo, 2024).

De volgende code-snippet bevat de logica die werd toegevoegd binnen de propagatielus van de video-predictor:

```
# Fragment uit de aangepaste SAM2VideoPredictor binnen de video propagatie lus
# Bepaal de dictionary waar frame outputs worden opgeslagen
obj_output_dict = inference_state["obj_wise_inference_states"][obj_id]
storage_key = "non_cond_frame_outputs"
obj_specific_storage_key = "output_dict_per_obj"

# Identificeer oude frames die verwijderd moeten worden
if reverse: # Bij achterwaarts tracken
    # Verwijder frames die te ver vooruit liggen
    newest_allowed_idx = frame_idx + 16 # Behoud een buffer van 16 frames
    all_frame_idxs = obj_output_dict[storage_key].keys()
    old_frame_idxs = [idx for idx in all_frame_idxs if idx > newest_allowed_idx]
else: # Bij voorwaarts tracken
    # Verwijder frames die te ver achterliggen
    oldest_allowed_idx = frame_idx - 16 # Behoud een buffer van 16 frames
    all_frame_idxs = obj_output_dict[storage_key].keys()
    old_frame_idxs = [idx for idx in all_frame_idxs if idx < oldest_allowed_idx]

# Verwijder de geïdentificeerde oude frames uit de cache
for old_idx in old_frame_idxs:
    obj_output_dict[storage_key].pop(old_idx, None) # Verwijder uit algemene cache
    # Verwijder ook uit object-specifieke caches indien aanwezig
    for objid_iter in inference_state[obj_specific_storage_key].keys():
        if old_idx in inference_state[
          obj_specific_storage_key
        ][objid_iter][storage_key]:
            inference_state[
              obj_specific_storage_key
            ][objid_iter][storage_key].pop(old_idx, None)
```

Codefragment 5.3: Toegevoegde logica om gecachte frame-specifieke outputs selectief te verwijderen, waardoor het GPU-geheugengebruik tijdens lange tracking-sessies constant blijft.

De oplossing werkt als volgt:

1. De code richt zich op specifieke dictionaries binnen de `inference_state` waar frame-gerelateerde data worden opgeslagen.
2. Er wordt een buffer rond de huidige `frame_idx` gedefinieerd. Data van frames die buiten dit venster vallen, worden beschouwd als 'oud' en komen in aanmerking voor verwijdering.
3. De geïdentificeerde `old_frame_idxs` worden vervolgens uit de relevante cache-dictionaries verwijderd. Dit gebeurt zowel voor de algemene frame outputs als voor de object-specifieke outputs.

Merk op dat de buffer van 16 frames hier hardgecodeerd is. In de toekomst kan het nuttig zijn om deze waarde aanpasbaar te maken door middel van een configura-

tieparameter binnen de `SAM2VideoPredictor` klasse.

5.3.3. Omgaan met Blikdata

Om de analyses in Hoofdstuk 8 te vergemakkelijken, en voor het bouwen van de grondwaarheid in Hoofdstuk 7, biedt de applicatie in de service-laag (geïmplementeerd in `src.api.services.gaze_service`) ook functionaliteit om de blikdata van de eyetracker te verwerken. Deze functionaliteit omvat het laden, verwerken en synchroniseren van de blikdata met de videoframes. Hier worden de kernstappen van deze functionaliteit besproken.

Parseren van Blikdatabestanden

De blikdata van de eyetracker worden opgeslagen in een `.tsv`-bestand. Hoewel men er kan van uitgaan dat het hier om tab-separated values gaat (tsv), blijkt dit niet het geval te zijn. In de werkelijkheid bestaan deze bestanden uit een reeks van `json`-objecten, waarbij elke regel metadata bevat over een bepaald blikpunt. De `parse_gazedata_file` functie is verantwoordelijk voor het inlezen van de bestanden, en levert een lijst aan `GazeData` objecten op.

Het volgende is een voorbeeld van een regel uit een blikdatabestand:

```
{
    "type": "gaze",
    "timestamp": 0.014574,
    "data": {
        "gaze2d": [
            0.42802076888650348,
            0.59985904570034698
        ],
        "gaze3d": [
            5524.405064657687,
            -4220.9188210373977,
            38060.311822743075
        ],
        "eyeleft": {
            "gazeorigin": [
                32.591088912983722,
                -9.3800673986482224,
                -28.144596430671765
            ],
            "gazedirection": [
                0.14673772446588171,
                -0.10725782137152362,
                0.98334317507836977
            ],
            "pupildiameter": 3.078226794385742
        },
        "eyeright": {
            "gazeorigin": [
                -28.884017016163959,
                -9.1355755568510908,
                -27.103885607429742
            ],
            "gazedirection": [
                0.13854486333094154,
                -0.11030711082172073,
                0.98419391491045893
            ],
            "pupildiameter": 3.1712040692965147
        }
    }
}
```

Codefragment 5.4: Een voorbeeld van een regel in een blikdatabestand. Het bevat metadata over de blikdata, zoals de tijdstempel, de 2D- en 3D-coördinaten van de blik, en de pupilgrootte.

Zoals men kan zien bevat elke regel van het bestand de volgende metadata, waarvan de uitleg afkomstig is uit de Tobii Glasses 3 Developer Guide (Tobii AB, 2023):

- `type`: het type van de data (in dit geval 'gaze').
- `timestamp`: de tijdstempel van de data in seconden.
- `data`: een dictionary met de blikdata. Het is mogelijk dat deze leeg is wanneer er geen blikdata beschikbaar is. Het bevat de volgende velden:
  - `gaze2d`: De 2D-coördinaten van de blik. Merk op dat dit genormaliseerde coördinaten zijn ten opzichte van de resolutie van de video-opname, waarbij (0, 0) de linkerbovenhoek van de video-opname is en (1, 1) de rechteronderhoek.
  - `gaze3d`: Dit is een 3D-coördinaat in millimeters, dat de positie van de blik in de ruimte weergeeft tegenover de positie van de camera.
  - `eyeleft` en `eyeright`: metadata over het linker en rechter oog, waaronder de oorsprong van de blik, de richting van de blik, en de pupilgrootte. Het is mogelijk dat één van deze velden leeg is wanneer er geen data beschikbaar is voor dat oog. Indien beide velden leeg zijn, betekent dit dat er geen blikdata beschikbaar zijn (het 'data' veld is dan ook volledig leeg).

Converteren naar Bruikbare Blikpunten

Nadat de ruwe blikdata zijn geparsed, worden deze verder verwerkt door de `get_gaze_points` functie. Het resultaat van deze functie is een lijst van `GazePoint`-objecten, die elk een x- en y-coördinaat, de geschatte diepte, en de oorspronkelijke tijdstempel bevatten. Hoewel de diepte binnen de analyses in Hoofdstuk 8 niet verder wordt gebruikt, kan deze in de toekomst nuttig zijn voor andere analyses.

```python
    def get_gaze_points(
        gaze_data: list[GazeData], resolution: tuple[int, int]
    ) -> list[GazePoint]:
        gaze_points = []

        for data in gaze_data:
            if data.gaze2d is None:
                # Geen 2D-coördinaten beschikbaar, dus deze data is niet bruikbaar
                continue

            gaze_depth = None
            if data.gaze3d and data.eye_data_left and data.eye_data_right:
                eyeleft = data.eye_data_left
                eyeright = data.eye_data_right

                # Bereken de oorsprong van de blik als het
                # gemiddelde van de oorsprong van beide ogen
                gaze_origin = (
                    np.add(eyeleft.origin, eyeright.origin) / 2
                    if (eyeleft and eyeright)
                    else eyeleft.origin
                    if eyeleft
                    else eyeright.origin
                )

                # Bereken de diepte van de blik als de
                # euclidische afstand tussen de oorsprong en het 3D-punt
                gaze_depth = np.sqrt(np.sum((data.gaze3d - gaze_origin) ** 2, axis=0))

            # Denormaliseer de 2D-coördinaten naar pixelcoördinaten
            x = int(clamp(data.gaze2d[0], 0, 1) * resolution[1])
            y = int(clamp(data.gaze2d[1], 0, 1) * resolution[0])

            gaze_points.append(GazePoint(x, y, gaze_depth, data.timestamp))

        return gaze_points
```

Codefragment 5.5: De `get_gaze_points` functie verwerkt de ruwe blikdata en zet deze om naar bruikbare blikpunten. Het resultaat is een lijst van `GazePoint` objecten met de x- en y-coördinaten, de geschatte diepte, en de oorspronkelijke tijdstempel.

De functie heeft twee hoofddoelen:

1. **Filteren van Ongeldige Data:** Blikregistraties waarbij geen `gaze2d`-coördinaten beschikbaar zijn, worden genegeerd, omdat deze niet bruikbaar zijn voor 2D-analyse in het videobeeld.
2. **Denormalisatie en Diepteberekening:**
   - Een schatting van de diepte van het blikpunt (`gaze_depth`) wordt berekend. Indien data voor beide ogen beschikbaar zijn, wordt de 3D-oorsprong van de blik benaderd als het gemiddelde van de `gazeorigin` van het linker- en rechteroog. De diepte is dan de Euclidische afstand tussen dit gemiddelde oorsprongspunt en het geregistreerde `gaze3d`-punt.
   - De genormaliseerde `gaze2d`-coördinaten (x, y) worden omgezet naar absolute pixelcoördinaten ten opzichte van de resolutie van de video-opname. Hierbij wordt gebruik gemaakt van de `clamp`-functie om te verzekeren dat de coördinaten binnen de grenzen van de videoframe blijven alvorens te vermenigvuldigen met de respectievelijke breedte en hoogte.

Synchronisatie van Blikpunten met Videoframes

Een belangrijke stap voor analyse is het correct toewijzen van de continue stroom van blikpunten aan de verschillende videoframes. De Tobii Pro Glasses 3 registreert blikdata met hogere frequentie (50Hz) dan de videoframerate (25fps). Dit betekent dat er meerdere blikpunten kunnen corresponderen met een enkele videoframe.

De functie `match_frames_to_gaze` implementeert de logica voor deze synchronisatie:

```python
    def match_frames_to_gaze(
        frame_count: int, gaze_points: list[GazePoint], fps: float
    ) → list[list[GazePoint]]:
        # Lijst van lijsten om de blikpunten per frame op te slaan
        frame_gaze_mapping = []

        # Bijhouden waar we zijn in de lijst van blikpunten
        gaze_index = 0

        for frame_num in range(frame_count):
            # Bepaal de tijdstempel van de volgende frame
            next_frame_timestamp = (frame_num + 1) / fps

            # Maak een lijst voor de blikpunten van deze frame
            frame_gazes = []

            while (
                gaze_index < len(gaze_points)
                and gaze_points[gaze_index].timestamp <
                    next_frame_timestamp
            ):
                # Als de tijdstempel van het blikpunt kleiner is
                # dan de tijdstempel van de volgende frame,
                # voeg het blikpunt toe aan de lijst van deze frame
                gaze_point = gaze_points[gaze_index]
                frame_gazes.append(gaze_point)
                gaze_index += 1

            frame_gaze_mapping.append(frame_gazes)

        # Controleer of er onverwacht veel blikpunten per frame zijn
        for points in frame_gaze_mapping:
            if len(points) ⩾ 3:
                print(
                    f"Warning: Detected {len(points)} gaze points for
                        a frame in the video. This is unexpected."
                )

        return frame_gaze_mapping
```

Codefragment 5.6: De `match_frames_to_gaze` functie synchroniseert de blikpunten met de videoframes. Het resultaat is een lijst van lijsten, waarbij elke sublijst de blikpunten bevat die overeenkomen met de respectievelijke frame-index.

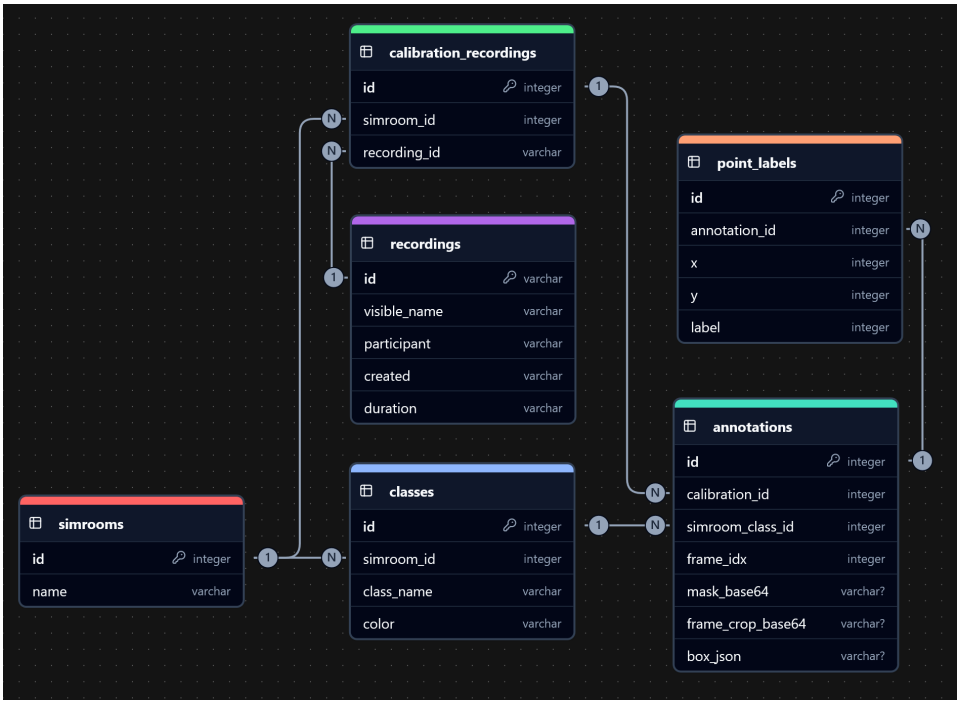
Gegeven het totaal aantal frames in de video, de lijst van `GazePoint`-objecten (gesorteerd op tijdstempel), en de framerate (fps) van de video, doorloopt deze functie elke frame-index:

1. Voor elke frame wordt de eindtijd berekend.
2. Vervolgens worden alle `GazePoint`-objecten verzameld waarvan de tijdstempel valt vóór het einde van de huidige, maar na het einde van de vorige frame (impliciet, doordat de `gaze_index` behouden blijft).
3. Dit resulteert in een mapping waarbij elke frame-index wordt geassocieerd met een lijst van één of meerdere `GazePoint`-objecten die tijdens de duur van dat frame zijn geregistreerd. Typisch zullen dit, afhankelijk van de sampling rates, nul, één of twee blikpunten per frame zijn. Een waarschuwing wordt gegenereerd indien er onverwacht drie of meer blikpunten aan een frame worden toegewezen.

Bij sommige opnames is het heel zelden mogelijk dat er meer dan twee blikpunten worden geregistreerd binnen een frame. Hoewel dit onverwacht is, zullen hier geen problemen door ontstaan, aangezien de applicatie het vroegste blikpunt dat overeenkomt met de frame-index zal gebruiken.

5.3.4. Database

We hebben het al eerder gehad over opnames, simulatieomgevingen, objecten en annotaties, maar hoe worden deze nu opgeslagen in de database? Om hier een beter zicht op te krijgen, werd er een Entity-Relationship Diagram (ERD) gemaakt dat de verschillende entiteiten en hun onderlinge relaties toont (zie figuur 5.9).



Figuur 5.9: Entity-Relationship Diagram (ERD) van de applicatie. De verschillende entiteiten en hun onderlinge relaties worden weergegeven: simulatieruimten, objecten (classes), opnames, kalibratie-opnames, annotaties, en point-labels.

- **simrooms** zijn de verschillende omgevingen waarin de opnames worden gemaakt. Elke simulatieomgeving heeft een naam en kan meerdere objecten bevatten.
- **classes** zijn de verschillende objecten die in de simulatieomgeving aanwezig zijn. Elk object heeft een naam, een kleur, en kan meerdere annotaties bevatten.
- **recordings** zijn de video-opnames die gemaakt worden met de eyetracker. Elke opname heeft een naam, een datum, en een duur. Een opname kan ook dienen als kalibratie-opname voor meerdere simulatieruimten. Opnames hebben ook een pad naar hun video- en blikdata bestand. Deze paden worden niet opgeslagen in de database, maar worden gegenereerd op basis van het `id` van de opname.
- **calibration_recordings** zijn louter een referentie naar de opname die als kalibratie-opname wordt gebruikt voor de simulatieomgeving. Ze kunnen meerdere annotaties bevatten.
- **annotations** worden gemaakt in de labeling-tool en hebben een `class`, een `frame index`, een base64-gecodeerde afbeelding van het masker evenals van de regio in de originele frame. Het bevat eveneens een bounding box. Zowel het base64-gecodeerde masker en de bounding-box worden getoond in de labeler bovenop de originele frame. De bounding-box is de rechthoek die de

regio van de annotatie omsluit, met het json formaat: `[x1, y1, x2, y2]`. Hier zijn `x1` en `y1` de coördinaten van de linkerbovenhoek van de rechthoek, en `x2` en `y2` de coördinaten van de rechteronderhoek. Tenslotte wordt ook een 'crop' van de originele frame opgeslagen, dat enkel de regio van de annotatie bevat. Dit is een afbeelding van dezelfde grootte als het masker, en wordt gebruikt om de annotatie te tonen in de annotatielijst onderaan de labeler.

- **point_labels** zijn de individuele punten die aan een annotatie worden toegekend. Elk point-label heeft een x- en y-coördinaat, en een label dat aangeeft of het punt binnen (1) of buiten (0) de annotatie ligt.

Merk op dat de geautomatiseerde tracking-resultaten niet in de database worden opgeslagen, maar enkel de manuele annotaties. Dit is omdat tracking grote hoeveelheden data genereereert die op een optimale manier moeten worden opgeslagen. De tracking-resultaten worden opgeslagen in een aparte map, onderverdeeld per kalibratie-opname en per klasse. Elke file bevat de tracking-resultaten van één object in één kalibratie-opname, met als naam het frame index. Ze worden opgeslagen in `.npz`-bestanden, die een gecomprimeerd formaat zijn voor numpy arrays. Ze bevatten de volgende data:

- `mask`: de gecomprimeerde versie van het masker dat werd gegenereerd door het segmentatie-algoritme.
- `bbox [x1, y1, x2, y2]`: de coördinaten van de bounding-box die het object omsluit.
- `roi` (Region of Interest): de regio van de originele frame die overeenkomt met het masker.
- `class_id`: de id van de klasse waartoe het object behoort. Zo kan andere informatie over het object—zoals de naam en de kleur—worden opgehaald uit de database.
- `frame_index`: de index van het frame waarin het object werd getracked.

5.4. Installatie en Configuratie

De instructies voor het installeren en configureren van de applicatie zijn uitvoerig beschreven in de README[3] file van de Git repository.

5.5. Gekende Problemen en Beperkingen

Hoewel de kernfunctionaliteit van de applicatie robuust is, zullen we het in deze sectie hebben over enkele gekende problemen en beperkingen die de gebruikerservaring kunnen beïnvloeden.

[3] https://github.com/ilianbronchart/bachelorproef/blob/master/README.md

Importeren van Opnames

De applicatie zou eventueel baat hebben bij de mogelijkheid om opnames te importeren vanuit een andere bron, zoals een bestand op de harde schijf. Zo kunnen opnames die niet op de hub van de eyetracker staan, maar wel lokaal beschikbaar zijn, worden toegevoegd aan de applicatie.

Labeling Tool

De grootste beperking van de labeling tool is dat men slechts één annotatie per klasse per frame kan maken. Dit betekent dat, indien er meerdere objecten van dezelfde klasse in een frame aanwezig zijn, men slechts één van deze objecten kan annoteren. Daarnaast biedt de labeling tool momenteel geen manier om een tracking-opdracht te annuleren. Ook is er geen automatisch beheer van de geladen computer-visie modellen. Wanneer men de labeling tool afsluit (door bijvoorbeeld naar een andere pagina te navigeren), worden de modellen niet automatisch uit het geheugen verwijderd. Tenslotte zou het handig zijn als er een mogelijkheid bestond om de gemaakte annotaties te exporteren. Op deze manier kan er gemakkelijker geëxperimenteerd worden met de annotaties, zoals bij het ontwikkelen van nieuwe analysemethoden.

6

Experimenteel Onderzoek

In het voorgaande hoofdstuk werd de ontwikkeling van de proof-of-concept softwareapplicatie uitvoerig besproken. Voor ontwikkelingsdoeleinden werd gebruik gemaakt van een dataset die thuis werd opgenomen met de Tobii Pro Glasses 3. Deze dataset was echter niet geschikt voor evaluatie van de uiteindelijke analysemethoden, aangezien ze door de onderzoeker zelf werden opgenomen en geen objecten bevatten die relevant zijn voor de zorgcontext. Om een dataset te verkrijgen die wel aan deze kwaliteitseisen voldoet, werd er een experiment opgezet in het Zorglab van HoGent waarin studenten werden gevraagd om de eyetracker te dragen en te kijken naar verschillende objecten. Het opzet, uitvoering en de resulterende data van dit experiment worden in dit hoofdstuk nader toegelicht.

6.1. Doelstellingen van het Experiment

Het hoofddoel van het experiment was om een dataset te creëren die als grondwaarheid kon dienen voor het valideren van de twee kernmetrieken van de PoC applicatie:

- De objecten die de studenten hebben bekeken.
- De tijdsduur dat de studenten naar deze objecten hebben gekeken.

Om deze validatie mogelijk te maken, werden twee soorten opnames gegenereerd. Ten eerste werden er zogenaamde *kalibratieopnames* gemaakt door de onderzoeker zelf. Deze dienden primair om de computervisiemodellen binnen de analyses te initialiseren met de visuele kenmerken van de te detecteren objecten. Ten tweede werden er *evaluatieopnames* verzameld waarbij studenten de eyetracker droegen en de observatietaak uitvoerden. Deze laatste categorie de te analyseren data waarvoor de ground-truth werd vastgesteld.

Naast het genereren van deze ground-truth, had het experiment ook als doel om de robuustheid van de ontwikkelde analysemethoden te onderzoeken. Dit omvatte het evalueren van de prestaties van het systeem onder invloed van verschillende factoren, waaronder:

- De invloed van **variërende kijkafstanden tot objecten** op de detectieaccuraatheid, resulterend in variaties in objectgrootte en detail in het camerabeeld.
- De prestaties van de modellen bij **diverse objectkarakteristieken**, zoals verschillen in vorm, grootte, textuur en mate van transparantie.
- De impact van **wisselende achtergronden** op de betrouwbaarheid van de objectherkenning.

6.2. Opzet van het Experiment

De concrete uitwerking van het experiment omvatte de inrichting van de experimentele omgeving, de keuze van kritische objecten, het opstellen van een gestandaardiseerde procedure, en de selectie van deelnemers.

6.2.1. Experimentele Omgeving en Materialen

De experimentele opnames vonden plaats in het Zorglab van HOGENT, dat beschikt over verschillende gesimuleerde zorgomgevingen. Voor deze bachelorproef werd specifiek gebruik gemaakt van de ruimte die is ingericht als een **woonkameromgeving**. Deze setting omvat typische elementen zoals een zithoek met een zetel, een keukengedeelte en een eettafel. Hoewel het Zorglab ook een gesimuleerde ziekenhuiskamer met twee bedden ter beschikking stelt, werd omwille van praktische overwegingen, zoals de positionering van objecten en de bewegingsvrijheid van de deelnemers, gekozen voor de woonkameropstelling.

Figuur 6.1: De woonkameromgeving in het Zorglab van HoGent.

Het Zorglab beschikt over een uitgebreid assortiment aan objecten die ingezet kunnen worden tijdens simulatietrainingen. Uit deze collectie werden 15 specifieke objecten geselecteerd (zie Figuur 6.2). Dit aantal werd gekozen als een pragmatisch compromis tussen het verkrijgen van voldoende variatie in objecttypes, die relevant zijn voor de zorgcontext en de technische uitdagingen. Het behouden van een haalbare werklast voor de latere annotatie van de grondwaarheid werd ook in overweging genomen. Deze objecten werden vervolgens in de woonkameromgeving geplaatst.

Figuur 6.2: De geselecteerde objecten voor het experiment. Van links naar rechts en van boven naar beneden: een bol wol, een ampule met vloeistof, een monitor, een snoepje, een infuus, een indicator 'Nuchter voor operatie', een stethoscoop, een transparante ampule, een naaldcontainer, een bril, een blikje iced tea, een keukenmes, een rollator, een spuit, en een fotokader.

Elk van deze objecten werd geselecteerd om redenen die zowel te maken hebben met de zorgcontext als met de technische haalbaarheid van de detectie:

- **Bol wol**: Indicatief voor een hobby activiteit van de patiënt. Dit kan interessant zijn als gespreksonderwerp.
- **Ampule met vloeistof**: Een veelvoorkomend medisch object, hier specifiek gekozen voor de kleine afmetingen en de reflecterende eigenschappen van het glas.
- **Monitor**: Monitor waarop parameters (bv. bloeddruk/pols/ECG/...) aan de zorgverlener en/of patiënt worden getoond.
- **Snoepje**: Voornamelijk gekozen omwille van de kleine afmetingen met een hoog kleurcontrast.
- **Infuus**: Het is belangrijk dat zorgverleners nakijken of een infuus voldoende is gevuld, en of het druppelmechanisme geactiveerd is.
- **Nuchter voor operatie**: Indicatoren zoals deze worden gebruikt in het zorglab om binnen een simulatie contextuele informatie te geven aan de studenten.
- **Stethoscoop**: Voornamelijk gekozen omwille van de complexe en potentieel variërende vorm van het object.
- **Transparante ampule**: Dit object is gekozen omwille van de reflecterende eigenschappen van het glas. Het werd naast andere objecten geplaatst om de detectie potentieel te bemoeilijken.

- **Naaldcontainer**: Een naaldcontainer mag nooit binnen handbereik van een patiënt komen en is dus een object dat zorgverleners altijd in het oog moeten houden.
- **Bril**: Een zorgverlener zou zich kunnen afvragen of een patiënt een bril nodig heeft, of hij hem inderdaad draagt.
- **Blikje Iced Tea**: Bij een diabetespatiënt is het belangrijk dat de zorgverlener controleert of de patiënt geen suikerhoudende dranken consumeert.
- **Keukenmes**: Suïcidepreventie is een belangrijk aandachtspunt in de zorg, en dit object kan potentieel gevaarlijk zijn voor patiënten met suïcidale gedachten. Hier werd een keukenmes gekozen, omdat het Zorglab geen gevaarlijke objecten ter beschikking had.
- **Rollator**: Dit object werd gekozen omwille van de grote afmetingen, en omdat een zorgverlener moet controleren of de rollator binnen bereik is van de patiënt.
- **Spuit**: Een veelvoorkomend medisch object, dat ook doorschijnend kan zijn maar toch een aantal specifieke visuele kenmerken heeft. Het kan tevens gevaarlijk zijn als het binnen handbereik van een patiënt is.
- **Fotokader**: Een fotokader kan eventueel familieleden of vrienden van de patiënt tonen, wat een gespreksonderwerp kan zijn.

6.2.2. Deelnemers

Voor het verzamelen van de evaluatieopnames werd een beroep gedaan op vrijwilligers. Deze werden gerekruteerd onder de studentenpopulatie op de campus van HoGent. Potentiële deelnemers werden voorafgaand aan hun deelname geïnformeerd over de doelstellingen en de aard van het onderzoek via een informed-consent formulier. Dit document bevatte tevens de voorwaarden voor deelname, waarbij benadrukt werd dat deelname volledig vrijwillig was en dat zij zich op elk gewenst moment zonder opgaaf van reden konden terugtrekken uit het experiment. Enkel na ondertekening van dit formulier werd de deelname bevestigd.

6.2.3. Procedure

De opnameprocedure voor zowel de evaluatie- als de kalibratieopnames werd gestandaardiseerd om consistente en reproduceerbare data te verzamelen.

Procedure bij Evaluatieopnames

Om gestandaardiseerde data te verzamelen, werd een vaste procedure gevolgd voor elke opnamesessie met een deelnemer.

1. **Voorbereiding object selectie:** Voorafgaand aan het experiment werden 15 unieke sets samengesteld, elk bestaande uit vijf van de 15 geselecteerde objec-

ten. De keuze voor een beperkt aantal van vijf objecten per set werd gemaakt om meerdere redenen: enerzijds om de benodigde tijd voor de latere annotatie van de ground-truth data per opname te beperken, en anderzijds om de duur van elke individuele opnamesessie en de belasting voor de deelnemer te beperken. Elke set werd gevisualiseerd in een afzonderlijk PDF-document, met daarin duidelijke afbeeldingen en de namen van de betreffende vijf objecten. Bij het samenstellen van de sets werd gezorgd voor een evenwichtige spreiding, zodat elk van de 15 objecten over het totaal van de sets ongeveer even vaak voorkwam.

2. **Initiële briefing:** Bij aanvang van een opnamesessie werd aan de deelnemer eerst de specifieke set van vijf objecten voor die sessie getoond aan de hand van het corresponderende PDF-document. Tegelijkertijd wees de onderzoeker de fysieke locatie van elk van deze vijf objecten in de experimentele ruimte aan.

3. **Uitleg procedure:** Na de identificatie van de objecten en hun locaties, werd het volledige protocol van de observatietaak mondeling aan de deelnemer uitgelegd. Dit omvatte de instructie om op een specifiek startpunt plaats te nemen, te wachten tot de onderzoeker een objectnaam noemt, de blik op dat object te richten, er langzaam naartoe te lopen terwijl men blijft kijken. Op aangeven van de onderzoeker diende de deelnemer terug te keren naar het startpunt, en dit proces te herhalen voor elk van de vijf objecten. De deelnemer kreeg hierbij de gelegenheid om vragen te stellen, zodat de taakvereisten volledig duidelijk waren alvorens verder te gaan.

4. **Setup eyetracker:** Vervolgens werd de Tobii Pro Glasses 3 eyetracker bij de deelnemer opgezet. Hierna volgde een standaard kalibratieprocedure via de bijbehorende software. De opname werd gestart na succesvolle kalibratie.

5. **Start observatietaak:** De deelnemer werd verzocht plaats te nemen op het vooraf bepaalde startpunt in de ruimte. De onderzoeker noemde vervolgens de naam van het eerste object uit de geselecteerde set.

6. **Uitvoering observatietaak:** Bij het horen van de naam van een object, voerde de deelnemer de eerder uitgelegde instructie uit: de blik op het specifieke object richten en er vervolgens langzaam naartoe lopen, terwijl de blik continu op het object gericht bleef. Deze beweging naar het object toe was cruciaal om variatie in de kijkafstand te introduceren. Dit resulteerde in opnames waarbij de objecten in verschillende schijnbare groottes in het gezichtsveld van de camera verschijnen. Op deze manier konden de effecten van kijkafstand op de detectieprestaties worden geëvalueerd.

7. **Terugkeer en herhaling:** Na een signaal van de onderzoeker, keerde de deelnemer terug naar het oorspronkelijke startpunt. Vervolgens noemde de on-

derzoeker het volgende object in de set, waarna stappen 6 en 7 werden herhaald.

8. **Afronding sessie:** Nadat het vijfde object was bekeken en de deelnemer was teruggekeerd naar het startpunt, werd de opname gestopt. De eyetracker werd afgenomen en de deelnemers werd bedankt voor de medewerking.

Het initiële streven was om ongeveer 15 evaluatieopnames te verzamelen, corresponderend met de 15 unieke objectsets. Dit aantal werd, net als het aantal geselecteerde objecten, als realistisch ingeschat met het oog op de benodigde tijd voor de latere handmatige annotatie van de ground-truth data. Uiteindelijk werd de beschreven procedure in totaal 14 maal herhaald met verschillende deelnemers, waarbij sommigen tweemaal deelnamen aan het experiment met een verschillende selectie van objecten. Door een administratieve fout tijdens de uitvoering werd echter één van de objectselecties dubbel opgenomen en een andere niet, waardoor de finale dataset evaluatieopnames bevat van 13 unieke objectselecties. Elke opname had een gemiddelde duur van ongeveer één minuut.

Proces bij Kalibratieopnames

Naast de evaluatieopnames werden, zoals eerder toegelicht, twee specifieke kalibratieopnames verricht door de onderzoeker. Bij beide opnames werd elk van de 15 objecten gedurende 30 seconden (gemeten met een timer) in beeld gebracht vanuit verschillende invalshoeken en afstanden. De eerste kalibratieopname vond plaats met de geselecteerde objecten in hun oorspronkelijke posities binnen de woonkameromgeving. Dit was identiek aan de setting voor de evaluatieopnames. Voor de tweede kalibratieopname werden dezelfde objecten vervolgens verplaatst naar een locatie met een significant afwijkende achtergrond. Het doel hiervan was specifiek om de invloed van de achtergrondcontext op de detectieprestaties van de modellen te kunnen beoordelen. Aangezien beide opnames binnen een korte tijdspanne werden gemaakt, bleven de lichtomstandigheden consistent tussen de twee kalibratieopnames.

7

Creatie van de Grondwaarheidsdataset

7.1. Inleiding

De evaluatie van elk geautomatiseerd systeem, staat of valt met de beschikbaarheid van een betrouwbare referentiestandaard, ook wel de grondwaarheid genoemd. In de context van deze bachelorproef, is een dergelijke grondwaarheid essentieel om de validiteit van de ontwikkelde methoden (zie Hoofdstuk 4, Strategie 4) te kunnen beoordelen. De kern van deze grondwaarheid ligt in het van frame-per-frame nauwkeurig vaststellen welke kritische objecten door een deelnemer werden bekeken. Dit hoofdstuk beschrijft het proces dat gevolgd werd om deze grondwaarheidsdataset te creëren. De code en data voor zowel dit hoofdstuk als Hoofdstuk 8 is beschikbaar in de git-repository, in de map `experiment`.

7.2. Voorbereiding van de Data

Alvorens de annotatie kon plaatsvinden, was een voorbereiding van de verzamelde ruwe data nodig. Dit voorbereidingsproces werd geautomatiseerd in de python-notebook `02_preprocess_data.ipynb`. Hier worden zowel de evaluatie- als kalibratieopnames voorbereid voor annotatie. We zullen ons hier echter alleen richten op de evaluatieopnames. De kalibratieopnames dienden enkel voor het creëren van visuele voorbeelden van de objecten, die op hun beurt als trainingsdata dienden voor de modellen in Hoofdstuk 8.

De notebook is opgedeeld in verschillende secties, die elk een specifiek aspect van de voorbereiding behandelen:

1. De ruwe opnamemappen werden geïnventariseerd. Er werd gecontroleerd of

het verwachte aantal van 14 evaluatieopnames en 2 kalibratieopnames aanwezig was.

2. Alle relevante opnamedata (afkomstig van de SD-kaart van de Tobii-glasses) werden gekopieerd naar een centrale mapstructuur (`data/recordings/`) zoals benodigd door de PoC applicatie. Hier werden de gazedata (uitgepakt met gzip) en de video-opnames respectievelijk opgeslagen onder `<recording_id>.tsv` en `<recording_id>.mp4`.
3. Voor elke video-opname werden alle individuele frames geëxtraheerd en opgeslagen in een aparte submap (`data/recording_frames/<recording_id>/xxxxx.png`).
4. Tenslotte werd de database van de applicatie geïnitialiseerd met alle nodige data. De metadata van de opnames werden ingelezen en opgeslagen in de database. Ook werd er een simulatieruimte aangemaakt met de naam 'Controlled Experiment Room' met de 15 objecten. Ook werden de kalibratieopnames toegewezen aan deze simulatieruimte. Merk op dat de evaluatieopnames hier ook als kalibratieopnames werden beschouwd binnen de applicatie, omdat deze ook geannoteerd dienden te worden voor het maken van de grondwaarheid.

7.3. Annotatie van Evaluatieopnames

Een accurate en consistente annotatie van de experimentele data vormt de spil voor een accurate grondwaarheid. Dit proces werd uitgevoerd met behulp van de in Hoofdstuk 5 beschreven labeling-tool binnen de ontwikkelde PoC-applicatie.

Voor de 14 evaluatieopnames was het hoofddoel het creëren van een nauwkeurige grondwaarheid van het kijkgedrag van de deelnemers. Hoewel elke deelnemer instructies kreeg om naar een specifieke set van vijf objecten te kijken, kon niet worden uitgesloten dat hun blik, al dan niet bewust, ook op andere objecten in de omgeving zou vallen. Een initiële overweging was om enkel de vijf doelobjecten per opname te annoteren. Om te voorkomen dat het geautomatiseerde analysesysteem een correct gedetecteerd, maar niet-geïnstrueerd, object als fout-positief zou classificeren (omdat dit niet in een beperkte grondwaarheid zou voorkomen), werd voor een meeromvattende aanpak gekozen.

De onderzoeker bekeek daartoe elke evaluatieopname integraal, waarbij de videofeed werd gecombineerd met een overlay van de geregistreerde blikpunten. Deze visuele combinatie maakte het mogelijk om dynamisch te beoordelen welke van de 15 potentiële objecten op enig moment relevant waren voor annotatie. Dit waren die objecten waar de blik van de deelnemer daadwerkelijk op rustte, ongeacht of dit een geïnstrueerd doelobject was of een object dat 'toevallig' werd aangekeken. Zodra een fixatie op een van de 15 objecten werd vastgesteld, werd dit speci-

fieke object in de labeling-tool geselecteerd. Vervolgens werden met behulp van positieve (en eventueel negatieve) interactiepunten de SAM2-segmentatie en de semi-automatische tracking ingezet om het object te volgen. Waar nodig werden manuele correcties of herinitialisaties van de tracking uitgevoerd.

7.4. Genereren van de Grondwaarheid

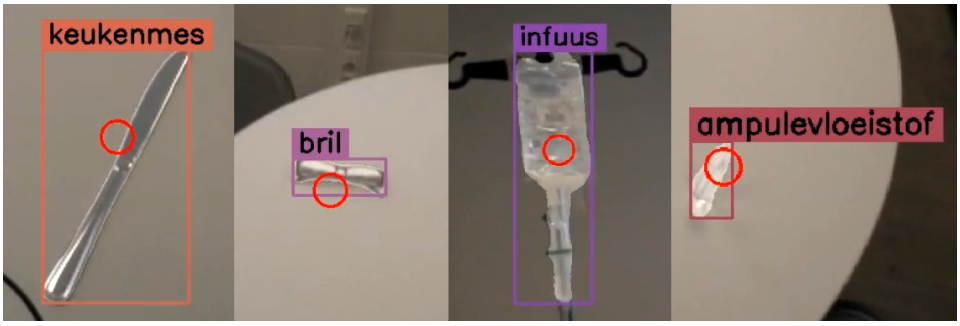
De resultaten van het annotatieproces werden door de applicatie opgeslagen in de map `data/labeling_results`, en werden verder verwerkt in het `03_create_ground_truth_dataset.ipynb`-notebook tot een grondwaarheidsdataset. Merk op dat de output van het annotatieproces van de evaluatieopnames niet direct gelijkgesteld kan worden aan de uiteindelijke grondwaarheid. De trackingfunctionaliteit van de labeling-tool volgt immers objecten zodra ze gemarkeerd zijn, onafhankelijk van de continue blikrichting van de deelnemer. Dit resulteert in een dataset die ook segmentaties bevat van objecten waar de deelnemer op dat specifieke moment niet (meer) naar keek.

7.4.1. Filtering van de Annotaties

Een aanvullende filterstap, gebaseerd op de geregistreerde blikdata, is noodzakelijk om enkel die objectsegmentaties te behouden die daadwerkelijk samenvallen met de fixaties van de deelnemer.

De basis hiervoor werd gelegd in Sectie 5.3.3 (in Hoofdstuk 5), waar de methoden voor het parsen, verwerken en synchroniseren van blikdata met videoframes werden toegelicht. De functie `match_frames_to_gaze` leverde per videoframe een lijst op van de blikpunten die binnen de tijdsduur van dat frame werden geregistreerd. Gezien de eyetracker (50Hz) een hogere samplingfrequentie heeft dan de videoframerate (25fps), kan een frame nul, één, of typisch twee blikpunten bevatten. Voor de constructie van de grondwaarheid werd per frame, indien beschikbaar, het eerste blikpunt uit deze lijst geselecteerd als representatief voor de blikrichting gedurende dat frame. Deze keuze is gebaseerd op de aanname dat het eerste geregistreerde blikpunt binnen een frame-interval het dichtst aansluit bij de visuele informatie aan het begin van dat frame.

Met een representatief blikpunt per frame kon vervolgens de filtering worden uitgevoerd. De uitdaging hierbij is dat een blikpunt, zoals geregistreerd door de eyetracker, één enkel pixelcoördinaat representeert, terwijl visuele waarneming in de werkelijkheid plaatsvindt binnen een bepaald gebied van het gezichtsveld. Dit gebied wordt in de literatuur vaak aangeduid als de ‘foveale’ kijkzone, of kortweg ‘fovea’. Om te bepalen of een segmentatiemasker daadwerkelijk ‘bekeken’ werd, dient dus berekend te worden of er een overlap bestaat tussen het blikveld en het segmentatiemasker. Figuur 7.1 illustreert dit principe.



Figuur 7.1: Voorbeelden van de overlap tussen een blikpunt en een segmentatiemasker. Indien het blikpunt geheel buiten het masker valt, wordt dit beschouwd als een 'niet bekeken' object. De rode cirkel stelt het blikpunt voor, met een straal die de fovea en de nauwkeurigheid van de eyetracker modelleert.

Dit cirkelvormige kijkgebied wordt gedefinieerd op basis van twee principes die in Sectie 2.1.3 (Stand van Zaken) werden besproken:

1. De fovea centralis, het gebied in het netvlies verantwoordelijk voor het scherpste zicht, beslaat ongeveer 1 graad van het menselijk gezichtsveld.
2. De nauwkeurigheid van de Tobii Pro Glasses 3 eyetracker, die volgens specificaties ongeveer 0.6 graden bedraagt (Tobii AB, 2025a).

Om rekening te houden met beide factoren, wordt de effectieve Field of View (FOV) voor 'aandachtig kijken' benaderd als de som van de foveale FOV en de eyetracker-nauwkeurigheid, dus $1° + 0.6° = 1.6°$. Deze gecombineerde hoek wordt vervolgens omgezet naar een pixelradius op het camerabeeld. Gegeven de horizontale FOV van de Tobii Pro Glasses 3 camera (95°, volgens Tobii AB (2025a)) en de horizontale resolutie van de video (1920 pixels), wordt de straal in pixels van het cirkelvormige kijkgebied berekend als:

$$\text{straal} = \left(\frac{1.6°}{95°} \times 1920\right) / 2$$

Deze waarde maakt de `mask_was_viewed` functie mogelijk, die controleert of het blikpunt van de deelnemer overlapt met een segmentatiemasker.

```python
    def mask_was_viewed(
        mask: torch.Tensor,
        gaze_position: tuple[float, float],
        viewed_radius: float = VIEWED_RADIUS,
    ) -> bool:
        # De functie gaat ervan uit dat het masker dezelfde
        # afmetingen heeft als de videoframes (hoogte, breedte).
        # Het blikpunt is een tuple van (x, y) coördinaten.
        # viewed_radius is de straal van de cirkel rond het blikpunt
        height, width = mask.shape
        device = mask.device

        # Creëer een coördinatenrooster voor het masker.
        y_coords = torch.arange(0, height, device=device).view(-1,
            1).repeat(1, width)
        x_coords = torch.arange(0, width, device=device).view(1,
            -1).repeat(height, 1)

        # Bereken het kwadraat van de afstand van elk pixel tot het
            blikpunt.
        dist_sq = (x_coords - gaze_position[0]) ** 2 + (y_coords -
            gaze_position[1]) ** 2

        # Creëer een cirkelvormig masker gebaseerd op viewed_radius.
        # Pixels binnen de straal krijgen waarde 1.0, daarbuiten 0.0.
        circular_mask = (dist_sq ≤ viewed_radius**2).float()

        # Pas het cirkelvormige blikmasker toe op het input
            (segmentatie)masker.
        # Dit gebeurt door een element-wise vermenigvuldiging
        overlapped_mask = mask * circular_mask

        # Indien de som van de resulterende maskerwaarden groter is
            dan 0,
        # betekent dit dat er overlap was.
        return bool(overlapped_mask.sum() > 0)
```

Codefragment 7.1: Controleert of het blikpunt van de deelnemer overlapt met een segmentatie-masker.

De werking van deze functie kan als volgt worden samengevat:

1. Eerst wordt een binair masker gegenereerd dat het cirkelvormige kijkgebied rond de `gaze_position` representeert, gebruikmakend van de berekende `viewed_radius`. Pixels binnen deze cirkel krijgen de waarde 1, pixels daarbuiten de waarde 0.
2. Vervolgens wordt een element-wise vermenigvuldiging uitgevoerd tussen dit binaire kijkgebied-masker en het input segmentatiemasker (dat eveneens binair is, waarbij 1 objectpixels en 0 achtergrondpixels aanduidt).
3. Het resultaat is een nieuw masker (`overlapped_mask`) dat enkel pixels met waarde 1 bevat waar zowel het oorspronkelijke segmentatiemasker als het cirkelvormige kijkgebied een pixel hadden. Indien de som van alle pixelwaarden in dit `overlapped_mask` groter is dan nul, betekent dit dat er ten minste één pixel overlap is, en wordt het object als 'bekeken' beschouwd.

7.4.2. Bouwen van de Grondwaarheid

De filtering van de annotaties resulteert in een lijst van bekeken segmentatiemaskers per frame, wat een directe input vormt voor het construeren van de finale grondwaarheidsdataset. Voor elke evaluatieopname werden de opgeslagen trackingresultaten (de `.npz`-bestanden, zie Sectie 5.3.2) verder verwerkt. Uit elk relevant `.npz`-bestand werden de volgende kerneigenschappen geëxtraheerd:

- De unieke identificatie van de opname (`recording_id`).
- De index van het videoframe (`frame_idx`).
- De numerieke ID van het bekeken object (`class_id`).
- De coördinaten van de bounding box rond het object (`x1, y1, x2, y2`). Zoals eerder aangegeven, duidt het punt `(x1, y1)` de linkerbovenhoek aan en `(x2, y2)` de rechteronderhoek van de bounding box.
- De oppervlakte van het segmentatiemasker in pixels (`mask_area`), als indicatie van de objectgrootte.

Deze gegevens werden samengevoegd tot één tabel, waarbij elke rij een uniek, door een deelnemer bekeken object, in een specifiek frame van een evaluatieopname representeert. De tabel, opgeslagen als `data/ground_truth.csv`, constitueert de finale grondwaarheid.

7.4.3. Valideren van de Grondwaarheid

Om de correctheid van de gegenereerde grondwaarheid te waarborgen, werd een iteratieve, manuele validatiestap uitgevoerd. Voor elke evaluatieopname werd op elke frame de volgende informatie gevisualiseerd (indien beschikbaar):

1. **Bekeken Objecten:** Voor de objecten die volgens de grondwaarheid in dat

specifieke frame als 'bekeken' waren geïdentificeerd, werd het bijbehorende segmentatiemasker en een gelabelde bounding box (met objectnaam en -kleur) op het frame getekend.

2. **Blikpunt:** Het representatieve blikpunt voor dat frame, werd als een rode cirkel gevisualiseerd.

Deze geannoteerde frames werden vervolgens samengevoegd tot nieuwe video's, opgeslagen in de map `data/labeling_validation_videos`.

Indien een segmentatiemasker niet correct was (te groot of te klein) of ontbrak (blikpunt op het object, maar geen segmentatie zichtbaar), ging de onderzoeker terug naar de labeling-tool om dit te corrigeren.

8

Analyse van Observatieprestaties

8.1. Inleiding

In de voorgaande hoofdstukken werden de ontwikkeling van de PoC applicatie, de methodologie voor het verzamelen van experimentele data, en het creëren van een grondwaarheidsdataset uitvoerig besproken. Dit hoofdstuk richt zich op de kern van het onderzoek: de geautomatiseerde analyse van de observatieprestaties van studenten aan de hand van de verzamelde eyetracking-opnames.

Het hoofddoel van de hier beschreven analyse is om op basis van de videofeed en blikdata van de Tobii Pro Glasses 3, automatisch te bepalen (1) welke van de vooraf gedefinieerde kritische objecten door een student zijn waargenomen en (2) hoe lang de aandacht op elk van deze objecten gericht was. Om dit te realiseren, werd een analysepipeline ontworpen en geïmplementeerd, die de output van verschillende computervisiemodellen combineert.

De ontwikkelde analysepipeline, zoals conceptueel voorgesteld in Strategie 4 van Hoofdstuk 4, transformeert de ruwe video- en blikdata, frame-per-frame tot een identificatie van bekeken, kritische objecten. Dit proces omvat drie hoofdfasen: (1) het segmenteren en tracken van alle potentiële objecten in beeld, (2) het filteren van deze segmenten op basis van objectgrootte en daadwerkelijke observatie door de student, en (3) het classificeren van de overgebleven objectsegmenten. Er werden drie verschillende benaderingen voor de classificatiestap geëvalueerd:

1. **Vector-Index Classificatie:** Hierbij worden eerst image embeddings van de bekeken segmenten gegenereerd met DINOv2. Hierna worden deze embeddings vergeleken met voorbeelden van de kritische objecten binnen een

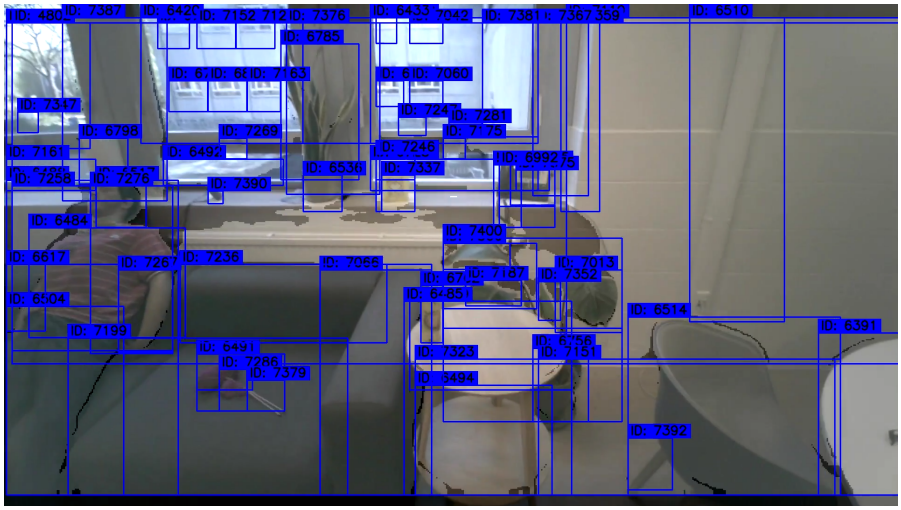
`Faiss (Facebook AI Similarity Search)` vector-index.

2. **YOLOv11 Classificatie:** In deze benadering wordt een YOLOv11-model getraind om de segmenten te classificeren.
3. **YOLOv11 Object Detectie:** Deze aanpak verschilt van de vorige doordat het model niet enkel classificeert, maar ook de locatie van de objecten in het frame bepaalt. De objectdetector genereert bounding boxes, die vervolgens worden gecombineerd met de trackingresultaten van FastSAM om tot een definitieve classificatie te komen van elk bekeken object.

Merk op dat het bij de eerste fase niet enkel over frame-per-frame segmentatie gaat, maar ook over het tracken van deze segmenten doorheen de video. Deze aanpak maakt het mogelijk om na classificatie van de individuele segmenten, de resultaten te aggregeren over de volledige tracking-sessie van elk object.

De ontwikkelde methoden werden beoordeeld aan de hand van de in Hoofdstuk 7 gecreëerde grondwaarheid.

Figuur 8.1 illustreert de fasen van dit proces aan de hand van illustratieve beelden.



(a) Everything-Segmentatie (FastSAM)

(b) Filtering op basis van blikpunt en objectgrootte

(c) Classificatiestap

Figuur 8.1: Visualisatie van de stappen in de analysepipeline. (a) FastSAM segmenteert alle objecten in het beeld en volgt deze doorheen de video. (b) De segmenten worden gefilterd; enkel de segmenten die daadwerkelijk met de blik van de gebruiker overlappen worden behouden. (c) De overgebleven segmenten worden uit de originele frame geknipt en dienen als input voor een classificatiemodel.

8.2. Tracking en Segmentatie van Objecten

De eerste fase van de analysepipeline is erop gericht om uit de continue videostroom alle potentieel relevante objectregio's te isoleren die daadwerkelijk door de student zijn bekeken. De implementatie van dit proces werd vastgelegd in de python-notebook `04_gaze_segmentation.ipynb`, en wordt hieronder toegelicht. De besproken code binnen deze fase is terug te vinden in de `GazeSegmentationJob` klasse in de notebook.

Voor deze fase werd gekozen om gebruik te maken van het FastSAM-model, vanwege zijn hoge snelheid. Dit maakte het mogelijk om snel de aanpak te optimaliseren en de pipeline te testen, zonder dat de tijdsduur van de analyse een beperkende factor werd. Het model komt in twee varianten: een 'large' (`FastSAM-x`) en een 'small' (`FastSAM-s`) versie. Er werd gekozen om de 'large' versie te gebruiken, vanwege de betere segmentatiekwaliteit. FastSAM is beschikbaar via de `ultralytics`[1] python-bibliotheek, die een `track` functie bevat die het mogelijk maakt om alle objecten in een video zowel te segmenteren als te tracken.

8.2.1. Tracking en Segmentatie van Objecten

In een eerste stap worden alle frames van de evaluatieopname geëxtraheerd naar een tijdelijke map met behulp van `ffmpeg`. Daarna is het mogelijk om de `track` functie toe te passen op deze frames:

```
1     frame_paths = list(self.frames_path.iterdir())
2     # Frames sorteren op basis van hun naam (index)
3     frame_paths.sort(key=lambda x: int(x.stem))
4
5     for frame_path in frame_paths:
6         frame_idx = int(frame_path.stem)
7         results = self.model.track(
8             source=str(frame_path), imgsz=1024, verbose=False,
                  persist=True
9         )[0]
```

Codefragment 8.1

Hier zijn de volgende zaken belangrijk om op te merken:

- De frames dienen gesorteerd te worden op basis van hun volgorde in de video.
- De `track` functie neemt een parameter `imgsz` aan, die de grootte van de inputafbeeldingen bepaalt. Indien de afbeeldingen te groot of te klein zijn, worden ze automatisch geschaald. Deze parameter heeft zowel invloed op de

[1] https://docs.ultralytics.com/models/fast-sam/ (laatst geraadpleegd op 2025-05-21)

snelheid van de segmentatie als op de kwaliteit ervan.

- Het is belangrijk om de `persist` parameter op `True` te zetten, zodat het model de tracking-informatie kan behouden tussen opeenvolgende frames.
- De `track` functie levert een lijst op van `Results`-objecten, maar bevat hier slechts één element, aangezien de functie telkens één enkele frame behandelt. Dit `Results`-object bevat de segmentatiemaskers, bounding boxes, en tracking-informatie voor elk object in het frame.

Het is ook mogelijk om een `iou` (Intersection over Union) parameter in te stellen, evenals een `conf` (vertrouwensscore) parameter. De `iou` parameter bepaalt hoeveel overlap nodig is tussen objecten om ze als hetzelfde object te beschouwen tijdens tracking. Een hoge waarde zal meer objecten als verschillend beschouwen, terwijl een lage waarde meer objecten zal samenvoegen. Echter werden deze parameters binnen dit onderzoek niet verder geoptimaliseerd.

8.2.2. Filteren van Tracking-Resultaten

Na het uitvoeren van de tracking en segmentatie, worden de resultaten gefilterd op basis van twee criteria:

- **Objectgrootte:** Segmenten die een onrealistisch groot deel van het beeld beslaan (bijvoorbeeld muren, vloeren, of de gehele achtergrond) worden weggelaten.
- **Blikdata:** Met behulp van de `mask_was_viewed` functie (zie Sectie 7.4.1) wordt voor elk overgebleven segment gecontroleerd of het blikveld van de student daadwerkelijk overlapt met het segmentatiemasker in dat specifieke frame. Enkel de 'bekeken' segmenten worden behouden voor verdere analyse.

Hier zullen we enkel de `mask_too_large` functie toelichten.
Codefragment 8.2 toont de implementatie hiervan.

```
1     def mask_too_large(self, mask: torch.Tensor) → bool:
2         MAX_MASK_AREA = 0.1
3         height, width = mask.shape
4         frame_area = height * width
5         max_mask_area = MAX_MASK_AREA * frame_area
6
7         mask_area = mask.sum()
8         return mask_area ⩾ max_mask_area
```

Codefragment 8.2: Deze functie controleert of een segment te groot is op basis van de oppervlakte van het segmentatiemasker.

Hier wordt een som berekend van alle pixels in het segmentatiemasker (aangezien het masker binair is). Indien deze som groter is dan de maximaal toegestane oppervlakte, wordt het masker als 'te groot' beschouwd. De maximale oppervlakte van een segment wordt gedefinieerd als 10% van de totale oppervlakte van het frame. Dit is momenteel een arbitraire waarde, maar kan in de toekomst verder geoptimaliseerd worden, of zelfs dynamisch worden ingesteld op basis van objecten binnen kalibratieopnames in de applicatie.

8.2.3. Opslaan van de Resultaten

Na het filteren van de segmenten, worden de resultaten van elke frame opgeslagen in gecomprimeerde numpy-bestanden (`.npz`) onder `data/gaze_segmentation_results`.

```
    executor.submit(
        np.savez_compressed,
        self.results_path / f"{frame_idx}.npz",
        boxes=boxes,
        rois=rois_array,
        masks=masks_array,
        object_ids=object_ids,
        frame_idx=frame_idx,
        gaze_position=gaze_position,
        confidences=confidences,
    )
```

Codefragment 8.3: Deze code slaat de resultaten van de segmentatie en tracking op in een gecomprimeerd numpy-bestand. De resultaten worden opgeslagen per frame, met de relevante metadata.

De volgende gegevens worden opgeslagen:

- **boxes:** De bounding boxes van de segmenten, handig voor het visualiseren van de segmenten in de video.
- **rois:** De ROI's (Region of Interest) van de segmenten. Deze worden later gebruikt bij de classificatiestap om de objecten te identificeren.
- **masks:** De segmentatiemaskers van de objecten, die ook worden gebruikt voor visualisatie.
- **object_ids:** De unieke ID's van de objecten, die worden toegewezen door het FastSAM-model. Deze ID's blijven consistent voor elk specifiek object over meerdere frames, tenzij de tracking verloren gaat (bijvoorbeeld wanneer het object tijdelijk uit beeld is). Wanneer dit gebeurt, wordt een nieuwe ID toegewezen aan het object als het opnieuw in beeld komt. Deze ID maakt het

mogelijk om de resultaten van de classificatiestap te aggregeren over meerdere frames, om zo een beter resultaat te krijgen.

- **frame_idx:** De index van het frame, die wordt gebruikt om de resultaten te koppelen aan het juiste frame in de video.
- **gaze_position:** De blikpositie van de student in dat specifieke frame (indien beschikbaar).
- **confidences:** De vertrouwensscore van het model voor elk segment, die aangeeft hoe zeker het model is dat het segment correct is. Dit kan nuttig zijn voor het filteren van segmenten die een lage vertrouwensscore hebben, of het vinden van correlaties tussen de vertrouwensscore en de uiteindelijke classificatie.

Aangezien het opslaan van de resultaten IO-intensief is, wordt dit proces uitgevoerd met behulp van multithreading (`executor.submit`).

8.2.4. Voorbereiding van de Tracking Resultaten voor Classificatie

De output van de vorige stap bestaat uit een reeks `.npz`-bestanden, één per frame, die de gefilterde segmenten, ROIs, en bijbehorende metadata bevatten. Om deze data efficiënt te kunnen gebruiken als input voor de classificatiestrategieën, werd een aanvullende voorbereidingsstap uitgevoerd. Deze stap werd geïmplementeerd in de notebook `05_create_object_datasets.ipynb` en consolideert de frame-per-frame resultaten tot een gestructureerde dataset per evaluatieopname. Deze dataset, hierna 'object-dataset' genoemd, aggregeert alle metadata van de bekeken segmenten binnen een enkele tabel. Hoewel veel van deze metadata ook in de individuele `.npz`-bestanden te vinden zijn, resulteert de consolidatie naar één `.csv`-bestand per opname in een efficiëntere dataverwerking tijdens de classificatiefase. Het vermijdt het herhaaldelijk openen en parsen van potentieel honderden of duizenden afzonderlijke `.npz`-bestanden.

Het creëren van de object-dataset omvat het itereren over de `.npz`-bestanden van elke opname. Voor elk gedetecteerd en gefilterd object (ROI) binnen een frame worden de volgende kenmerken geëxtraheerd en samengevoegd tot een rij in een `Pandas DataFrame`:

- `frame_idx`: De index van het frame waarin het object oorspronkelijk werd gedetecteerd. Dit koppelt het object aan een specifiek tijdstip in de video.
- `object_id`: De unieke, tijdelijke ID die door het FastSAM-model aan het getrackte object is toegewezen binnen de scope van die specifieke trackingsessie.
- `confidence`: De vertrouwensscore van het FastSAM-model voor de detectie van dit specifieke segment. Deze score geeft een indicatie van hoe zeker het

model was van zijn segmentatie.

- `embedding`: Een dense vectorrepresentatie (embedding) van de visuele kenmerken van de ROI, gegenereerd met het DINOv2-model. Deze embedding dient als input voor de op similariteit gebaseerde classificatie met een vectorindex. Meer hierover in de volgende sectie.
- `mask_area`: De totale oppervlakte van het segmentatiemasker van het object in pixels. Dit geeft een maat voor de (schijnbare) grootte van het object in het frame.
- `x1, y1, x2, y2`: De coördinaten die de bounding box rondom het gedetecteerde object definiëren.

De resulterende `DataFrame` wordt vervolgens opgeslagen als een `.csv`-bestand onder `data/object_datasets/<recording_id>`. Het resultaat is een set van tabelvormige datasets die klaar zijn voor de daadwerkelijke classificatietaken.

8.3. Classificatie van Objecten

De vorige fase leverde een dataset op met bekeken objectsegmenten (ROIs) per evaluatieopname. Deze kregen echter nog geen label toegewezen, dat aangeeft tot welk van de kritische objecten ze behoren. Om dit te realiseren, werd een classificatiestap geïmplementeerd die de ROIs labelt op basis van hun visuele kenmerken.

8.3.1. Data Labeling

Voor het initialiseren en trainen van de classificatiestrategieën was het noodzakelijk om een dataset te hebben met gelabelde objecten. Zoals eerder beschreven in Hoofdstuk 6 (Sectie 6.2.3), werden hiertoe twee specifieke kalibratieopnames gemaakt door de onderzoeker. De eerste opname bevatte de objecten in hun oorspronkelijke posities en achtergrond, identiek aan de evaluatieopnames. De tweede opname toonde dezelfde objecten tegen een significant afwijkende achtergrond, primair bedoeld om de invloed van contextvariatie te onderzoeken.

Bij de analyses die in dit hoofdstuk worden gepresenteerd, is uitsluitend gebruik gemaakt van de data uit de kalibratieopname met de originele achtergrond. De beslissing om de tweede kalibratieopname buiten beschouwing te laten, werd genomen om de scope van deze bachelorproef beheersbaar te houden. Een discussie over het potentieel van deze tweede dataset voor verder onderzoek is terug te vinden in Hoofdstuk 9.

Labeling voor Classificatie

Voor de initiële, op ROI-gebaseerde classificatiepogingen, was de labelingstrategie relatief eenvoudig. Het volstond om voor elk van de objecten een representatief

aantal ROIs te verzamelen en te labelen gedurende de tijdsvensters van 30 seconden waarin elk object centraal werd bekeken. Dit leverde voldoende voorbeelden van elk object op voor het trainen van de classificatiemodellen.

Labeling voor Objectdetectie

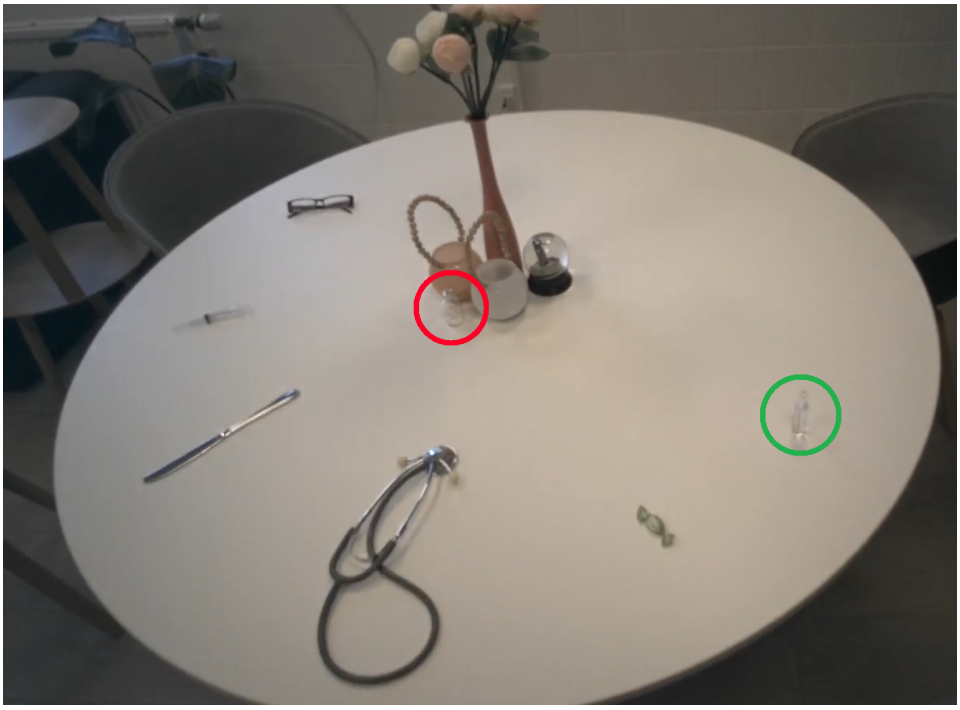
Voor het trainen van het objectdetectiemodel was echter een meer omvattende aanpak vereist. In tegenstelling tot ROI-classificatie die op geïsoleerde objectsegmenten werkt, wordt een objectdetector getraind om objecten te lokaliseren binnen een breder beeld. Tijdens de training analyseert het model 'crops' (vierkante regio's van het beeld) en leert het model om objecten te detecteren binnen zulke regio's. Hierbij is het belangrijk dat binnen de labeling alle objecten in het beeld worden gemarkeerd, die potentieel binnen een crop kunnen vallen (zie Figuur 8.2 voor een conceptuele visualisatie van een crop). De manier waarop deze crops worden gedefinieerd binnen de trainingsdataset, komt later in dit hoofdstuk aan bod.

Figuur 8.2: Voorbeeld van een frame uit de labeling tool. Hier wordt een voorbeeld van een crop getoond die gebruikt kan worden voor het trainen van een objectdetectiemodel. Het is dus belangrijk dat alle objecten die binnen deze crop kunnen vallen, worden gelabeld, zelfs als ze niet volledig zichtbaar zijn. Merk op dat dit niet de enige mogelijke crop is binnen deze afbeelding, men kan ook andere regio's selecteren met andere objecten.

Opmerking: Ampule Poeder Niet Gelabeld

Het object 'ampule poeder' werd in de labeling niet opgenomen. Origineel werd het object gekozen omwille van zijn doorschijnende karakter (glazen ampule). Het werd ook naast een andere groep objecten geplaatst om de detectie alsnog te bemoeilijken (zie Figuur 8.3). Echter, tijdens de labeling bleek het object te moeilijk

voor het SAM2 model om te segmenteren en te tracken. Dit resulteerde in een lage labelingkwaliteit, waarbij de segmenten inaccuraat waren. Soms werden zelfs ook foutief, aangrenzende objecten meegenomen in de segmentaties. Om deze reden werd besloten om het object niet mee te nemen in de labeling en de uiteindelijke analyse. Een ander object, de 'ampule vloeistof', werd wel opgenomen ondanks zijn sterk doorschijnende karakter. Dit object bleek veel gemakkelijker te volgen en te segmenteren door het FastSAM-model, vermoedelijk omdat het afgezonderd was van andere, potentieel verwarrende objecten.



Figuur 8.3: Locatie van de ampule poeder in de kalibratieopname, aangeduid met een rode cirkel. De ampule vloeistof werd wel opgenomen in de labeling, en is hier aangeduid met een groene cirkel.

8.3.2. Initiële Classificatiepogingen en Uitdagingen

Nadat de object-datasets waren aangemaakt en de referentiedata uit de kalibratieopname was gelabeld, was de volgende stap het toewijzen van een label aan elk gedetecteerd en door de student bekeken objectsegment (ROI). Er werden initieel twee strategieën onderzocht voor het classificeren van deze ROIs, waarvan de relevante data beschikbaar waren in de object-datasets:

1. **Vector-Index Classificatie:** Hierbij werden de DINOv2-embeddings van de ROIs vergeleken met referentie-embeddings van de kritische objecten afkomstig uit de kalibratieopname. Er werd een Faiss (Facebook AI Similarity Search) index gebruikt om de meest vergelijkbare referentie-embeddings te vinden. Faiss is een bibliotheek die het mogelijk maakt om snel zoekopdrachten uit te voeren op grote datasets van vectoren.

2. **YOLOv11 Classificatie:** Een YOLO-model werd getraind, niet voor detectie in het volledige frame, maar specifiek voor het classificeren van de reeds geïsoleerde ROIs uit de evaluatieopnames.

Deze twee benaderingen zullen hier echter niet diepgaand worden behandeld, aangezien beide al in een vroeg stadium van evaluatie een fundamenteel gebrek vertoonden, namelijk een onacceptabel hoge mate van vals-positieven. Het probleem lag niet zozeer in de specifieke modelkeuzes, maar in een verkeerde definitie van het classificatieprobleem. De gekozen classificatietechnieken zijn inherent ontworpen voor *gesloten-set* scenario's. In dergelijke scenario's wordt aangenomen dat elke te classificeren input (elke bekeken ROI) daadwerkelijk tot één van de vooraf gedefinieerde klassen behoort.

De realiteit van de observaties is echter complexer en sluit veel beter aan bij het concept van *Open Set Recognition (OSR)*. Zoals Geng e.a. (2018) beschrijven in hun overzichtsartikel, is het "doorgaans moeilijk om, wanneer men een herkenner of classificator traint, trainingsvoorbeelden te verzamelen die alle (mogelijke) klassen omvatten". OSR beschrijft een scenario waarin "er ten tijde van de training onvolledige kennis van de wereld bestaat, en onbekende klassen tijdens het testen aan een algoritme kunnen worden voorgelegd". Dit vereist dat een classificatiemodel "niet alleen de geziene klassen accuraat classificeert, maar ook effectief omgaat met de ongeziene" (citaten uit Geng e.a. (2018), eigen vertaling).

In de context van dit onderzoek betekende dit dat de studenten talloze objecten en achtergrondelementen bekeken die *niet* tot de kritische objecten behoorden. De geïmplementeerde ROI-classificatiemodellen waren niet in staat om deze 'onbekende' inputs effectief te herkennen en te verwerpen. In plaats daarvan waren ze geneigd elke ROI te classificeren als één van de objecten, wat resulteerde in de waargenomen overvloed aan vals-positieven.

Gezien deze mismatch tussen de aard van het probleem en de gekozen aanpak, werd besloten deze classificatiestrategieën niet verder te optimaliseren. De focus verschoof naar een methode die beter is uitgerust om specifieke, bekende objecten te identificeren: objectdetectie.

8.4. Objectdetectie met YOLOv11

Bij de voorgaande classificatiestrategieën lag de focus op het classificeren van geïsoleerde objectsegmenten afkomstig uit de FastSAM everything-segmentatie. Echter, de eerder beschreven problemen met vals-positieven maakten duidelijk dat een andere aanpak nodig was. Objectdetectie biedt een krachtigere oplossing, omdat het niet alleen in staat is om objecten te classificeren, maar ook om hun locatie in het beeld te bepalen met behulp van bounding boxes. De toegevoegde beeldcontext rond een object zorgt voor lagere gevoeligheid voor de problemen

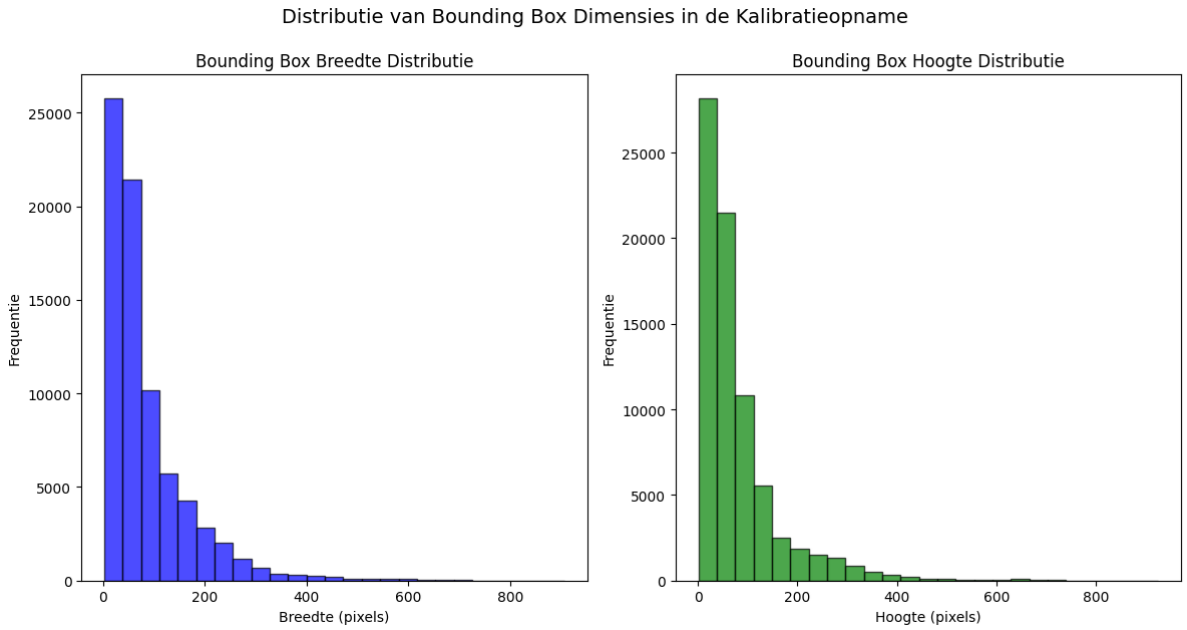
die zich voordeden bij de eerdere classificatiestrategieën.

8.4.1. Creatie van de Trainingsdataset voor Objectdetectie

Gebaseerd op de gelabelde kalibratieopname die eerder werd besproken, werd een specifieke trainingsdataset voor objectdetectie gecreëerd. Zoals eerder vermeld, werd het object 'ampule poeder' niet opgenomen in de labeling. Dit resulteerde in een dataset met 14 objecten. Deze diende te bestaan uit beelduitsnedes (hierna 'crops' genoemd) waarin de kritische objecten gelabeld zijn met bounding boxes. De creatie van deze dataset werd geïmplementeerd in de python notebook `12_prepare_object_detection_datasets.ipynb`. Omwille van de beknoptheid zullen in deze sectie enkel de belangrijkste codefragmenten worden getoond; de geïnteresseerde lezer wordt verwezen naar de zonet vernoemde notebook voor de volledige implementatiedetails van de hieronder beschreven stappen.

Formaat van de Trainingsdataset

Alvorens de trainingsdataset te creëren, was het belangrijk om het beoogde formaat van de dataset te bepalen. Een eerste overweging was de resolutie van de crops. Om tot een geïnformeerde keuze te komen, werd een analyse uitgevoerd op de afmetingen van de bounding boxes van de gelabelde objecten in de kalibratieopname. Figuur 8.4 toont histogrammen van zowel de breedte (links) als de hoogte (rechts) in pixels van deze bounding boxes. Uit de histogrammen blijkt dat de meerderheid van de objecten een breedte en hoogte hebben van minder dan 400 pixels. Enkele objecten die van zeer dichtbij waren opgenomen, vertoonden grotere afmetingen. Op basis van deze observatie, en rekening houdend met het feit dat objecten in de praktijk niet altijd perfect in het midden van een crop zullen liggen, werd gekozen voor een vierkante crop-grootte van 640x640 pixels. Deze afmeting biedt een ruime marge om de meeste objecten volledig te omvatten, zelfs als ze enigszins verschoven zijn ten opzichte van het centrum van de crop. Deze grootte is tevens een gangbare inputresolutie voor YOLO-modellen.



Figuur 8.4: Histogrammen van de breedte (links) en hoogte (rechts) in pixels van de bounding boxes in de kalibratieopname. De meeste objecten waren minder dan 400 pixels in zowel breedte als hoogte, met een aantal uitzonderingen.

De volgende overweging betrof de wijze waarop deze crops gegenereerd zouden worden uit de kalibratieopname. Aangezien het uiteindelijke doel was om de ROIs die gegenereerd werden door FastSAM, te classificeren, dient de trainingsdataset zo goed mogelijk de karakteristieken van deze te classificeren ROIs te weerspiegelen. In de praktijk zullen deze ROIs vaak (maar niet altijd perfect) gecentreerd zijn rond het blikpunt van de student, omdat de FastSAM-segmentatie en de blikpuntfiltering hierop aansturen. Daarom werd voor de creatie van de trainingsdataset besloten om de crops te genereren door ze te centreren op het middelpunt van de bounding box van een *geselecteerd doelobject* uit de kalibratieopname. Dit simuleert de situatie waarbij het model een ROI aangeboden krijgt waarin het doelobject centraal staat.

Tenslotte vereiste de trainingsdataset voor objectdetectie ook een specifieke structuur, zijnde het *Ultralytics YOLO formaat*[2] (zie Codefragment 8.4).

[2] https://docs.ultralytics.com/datasets/detect/#ultralytics-yolo-format (laatst geraadpleegd op 2025-05-21).

```
data/
├── images/
│   ├── train/
│   │   ├── 0000000001.jpg
│   │   ├── 0000000002.jpg
│   │   ├── 0000000003.jpg
│   │   └── ...
│   └── val/
│       ├── 0000000001.jpg
│       ├── 0000000002.jpg
│       ├── 0000000003.jpg
│       └── ...
├── labels/
│   ├── train/
│   │   ├── 0000000001.txt
│   │   ├── 0000000002.txt
│   │   ├── 0000000003.txt
│   │   └── ...
│   └── val/
│       ├── 0000000001.txt
│       ├── 0000000002.txt
│       ├── 0000000003.txt
│       └── ...
└── data.yml
```

Codefragment 8.4: Voorbeeld van de structuur van de trainingsdataset voor objectdetectie in het Ultralytics YOLO formaat. De dataset bestaat uit een map met afbeeldingen (in dit geval crops) en een map met labels. Elke afbeelding heeft een bijbehorend labelbestand met dezelfde naam, waarin de bounding boxes en klassenamen van elk object in de afbeelding zijn gedefinieerd. Het `data.yml` bestand bevat de metadata van de dataset. Dit komt later aan bod.

Genereren van de Trainingsdataset

Het eigenlijke proces voor het creëren van de trainings- en validatiedatasets werd gecoördineerd door de functie `create_dataset` (zie Codefragment 8.5). Deze functie neemt de metadata van de gelabelde objecten, de geëxtraheerde frames van de kalibratieopname en de configuratieparameters zoals de crop-grootte en het gewenste aantal samples per klasse, als input.

```python
def create_dataset(
    per_class_metadata: dict,
    frames: list[Path],
    datasets_path: Path,
    crop_size: int,
    dataset_type: str,
    num_samples_per_class: int,
):
    # Definieren van de paden voor de trainings- en validatiedatasets
    # ... Code weggelaten voor beknoptheid ...

    # Samples selecteren per klasse en train/val splitsen
    selected_samples_per_class = select_samples_per_class(
        per_class_metadata, num_samples_per_class
    )
    all_samples_per_frame = get_samples_per_frame(per_class_metadata)
    train_samples_per_class, val_samples_per_class = get_train_val_split(
        selected_samples_per_class, train_ratio=0.8
    )

    # Datasetfiles aanmaken
    class_label_to_model_id = create_metadata_yaml(dataset_path,
        per_class_metadata)
    create_train_or_val_dataset(
        per_class_metadata,
        class_label_to_model_id,
        train_samples_per_class,
        all_samples_per_frame,
        frames,
        train_images_path,
        train_labels_path,
        crop_size=crop_size,
    )
    create_train_or_val_dataset(
        per_class_metadata,
        class_label_to_model_id,
        val_samples_per_class,
        all_samples_per_frame,
        frames,
        val_images_path,
        val_labels_path,
        crop_size=crop_size,
        is_validation=True,
    )

    return dataset_path
```

Codefragment 8.5: De `create_dataset` functie coördineert het creëren van de trainings- en validatiedatasets voor objectdetectie. Deze functie definieert de paden voor de datasets, maakt de benodigde mappen aan, selecteert de samples per klasse, splitst deze in train- en validatiesets en roept de `create_train_or_val_dataset` functie aan om de crops en labels te genereren.

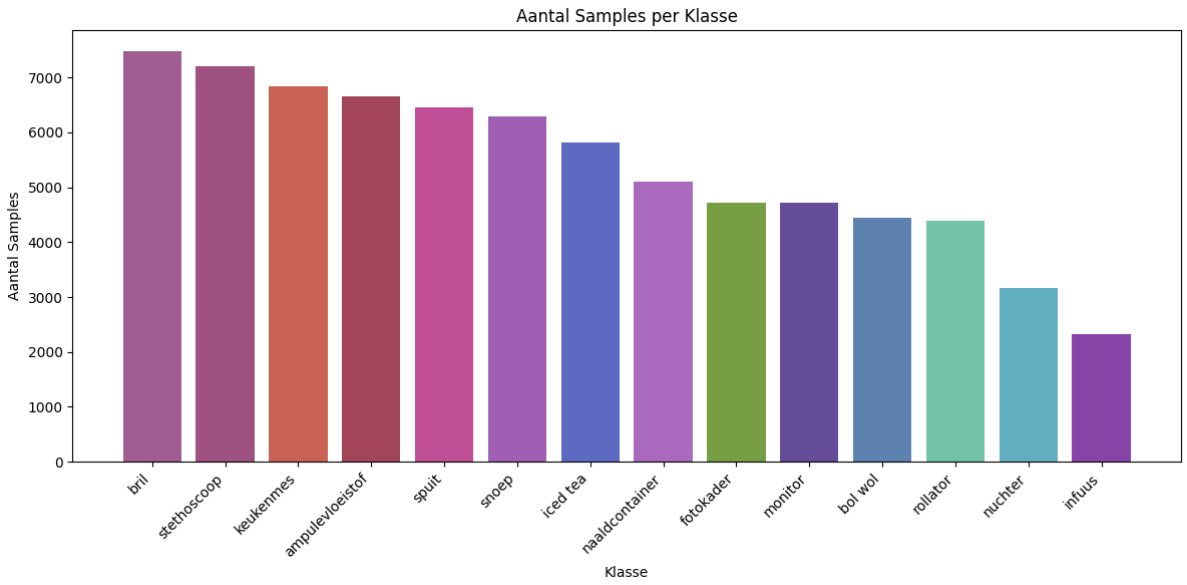
De benodigde stappen voor het creëren van de trainingsdataset kunnen worden samengevat in drie hoofdfasen, zoals hieronder beschreven.

1. Verzamelen van Metadata per Klasse
De notebook laadt in een eerste stap de bestandspaden van de trackingresultaten per klasse voor de kalibratieopname via de functie `get_tracking_results_per_class`.
Vervolgens worden op basis van de trackingresultaten, metadata per klasse verzameld door de functie `get_metadata_per_class` aan te roepen. Dit resulteert in een dictionary waarin elke klasse ID is gekoppeld aan een dictionary met de volgende informatie:

- `class_name`: De naam van de klasse, bijvoorbeeld 'monitor'.
- `color`: De kleur van de klasse in hexadecimale notatie.
- `frame_indexes`: Een lijst van frame-indexen waarin de klasse voorkomt in de trackingresultaten.
- `mask_areas`: Een lijst van de oppervlakte van het segmentatiemasker voor elk frame waarin de klasse voorkomt.
- `bboxes`: Een lijst van bounding boxes voor elk frame waarin de klasse voorkomt.

Elke combinatie van een frame-index en bijbehorende bounding box uit deze lijsten representeert een uniek gelabeld voorbeeld (een 'sample') van een object in de kalibratieopname.

Figuur 8.5 toont het aantal verzamelde samples per klasse in de kalibratieopname. Aangezien sommige objecten dichter bij elkaar stonden, zijn er voor die objecten meer samples verzameld dan voor andere. Zo zien we dat het infuus relatief weinig samples heeft, omdat het in een afgezonderde hoek van het beeld stond. Hierdoor zijn er enkel samples verzameld binnen de 30 seconden waarin het infuus werd bekeken, of wanneer het zichtbaar was in de achtergrond.



Figuur 8.5: Aantal samples per klasse in de kalibratieopname. De getoonde aantallen zijn het resultaat van de tracking binnen de labeling tool.

De `per_class_metadata` dictionary werd vervolgens doorgegeven aan de `create_dataset` functie. Elke dataset werd opgeslagen in de map `data/training_datasets/object_detection`.

2. Selectie van Voorbeeldsamples per Klasse

Om een gebalanceerde dataset te creëren voor training, en om de impact van de datasetgrootte te kunnen onderzoeken, werd per objectklasse een specifiek aantal voorbeeld-samples geselecteerd. De functie `select_samples_per_class` implementeert deze selectie. Er werd geëxperimenteerd met verschillende aantallen samples per klasse: 500, 1000, 2000 en 3000 via de parameter `num_samples_per_class`. Indien een klasse onvoldoende unieke samples bevatte ten opzichte van het opgegeven aantal, werd oversampling toegepast. Alle beschikbare unieke samples werden eerst geselecteerd, waarna de resterende samples willekeurig met vervanging uit de beschikbare pool werden getrokken.

Met deze mapping werden de train- en validatiesets gedefinieerd. De functie `get_train_val_split` verdeelt de geselecteerde samples in een train- en validatieset, met een splitsing van 80% voor training en 20% voor validatie. Deze stap resulteerde in twee mappings: `train_samples_per_class` en `val_samples_per_class`.

Tenslotte werd ook per frame een lijst van *alle* (dus niet enkel de geselecteerde) bounding boxes verzameld. Dit is van belang omdat bij het maken van een crop rond een *geselecteerd doelobject*, ook alle *andere* objecten die toevallig in die crop vallen, correct voorspeld moeten worden door de objectdetector. De functie `get_samples_per_frame` creëerde hiertoe een dictionary

`all_samples_per_frame`, waarbij elke frame-index gemapt werd naar een lijst van alle objecten (klasse-ID en bounding box) die in dat frame voorkomen.

3. Genereren van Crops en YOLO-Labels voor Trainings- en Validatiesets
Met de voorbereide lijsten van geselecteerde trainings- en validatiesamples per klasse (`train_samples_per_class` en `val_samples_per_class`) en de complete lijst van alle gelabelde objecten per frame (`all_samples_per_frame`), kon de daadwerkelijke generatie van de dataset beginnen. Dit proces werd uitgevoerd door de functie `create_train_or_val_dataset`, die afzonderlijk werd aangeroepen voor het creëren van de trainingsdata en de validatiedata. De functie `create_dataset` fungeerde als een overkoepelende wrapper die de mappenstructuur aanmaakte en de `create_train_or_val_dataset` functie aanriep.

```
    def create_train_or_val_dataset(
        per_class_metadata,
        class_label_to_model_id,
        selected_samples_per_class,
        all_samples_per_frame,
        frames,
        images_path,
        labels_path,
        crop_size: int,
        is_validation=False,
    ):
        selected_samples_per_frame = get_selected_samples_per_frame(
            selected_samples_per_class
        )

        current_sample_idx = 0
        for frame_idx, frame in enumerate(tqdm(frames)):
            if selected_samples_per_frame.get(frame_idx) is None:
                continue

            image = cv2.imread(str(frame))

            # verzamelen van alle bounding boxes en klassenamen in de
                huidige frame
            class_ids, bboxes = zip(*all_samples_per_frame[frame_idx])
            bboxes = np.array(bboxes)
            class_labels = [
                per_class_metadata[class_id]["class_name"] for
                    class_id in class_ids
            ]

            # aanmaken van de crops voor elk geselecteerd doelobject
                in de huidige frame
           for _, target_box in selected_samples_per_frame[frame_idx]:
                transformed_image, transformed_bboxes,
                    transformed_class_labels = create_crop_for_frame(
                    image,
                    crop_size,
                    target_box,
                    bboxes,
                    class_labels,
                    is_validation
                )

                create_data_files(
                    labels_path,
                    images_path,
                    class_label_to_model_id,
                    current_sample_idx,
                    transformed_image,
                    transformed_bboxes,
                    transformed_class_labels,
                )

                current_sample_idx += 1
```

Codefragment 8.6: De `create_train_or_val_dataset` functie genereert de crops en labels voor de trainings- of validatiedataset. Het laadt de originele afbeelding, verzamelt de relevante bounding boxes en klassenamen en maakt voor elk geselecteerd doelobject een crop rond de bounding box.

De werking van `create_train_or_val_dataset` is als volgt:

1. Eerst worden de `selected_samples_per_class` (die de geselecteerde doelobjecten voor de huidige set bevatten) geherstructureerd naar een per-frame dictionary `selected_samples_per_frame`.
2. De functie itereert vervolgens over alle frames van de kalibratieopname.
3. Voor elke frame wordt de originele afbeelding geladen indien er samples voor die frame zijn geselecteerd. Ook worden alle bounding boxes en bijbehorende klassenamen van *alle* gelabelde objecten in die frame verzameld uit `all_samples_per_frame`.
4. Daarna wordt geïtereerd over elk *geselecteerd doelobject* dat in de huidige frame aanwezig is (opgehaald uit `selected_samples_per_frame`). Het is rond deze `target_box` dat een crop zal worden gemaakt.
5. Voor elk `target_box` binnen de frame wordt de functie `create_crop_for_frame` aangeroepen. Deze functie, hieronder beschreven, retourneert de beelduitsnede. Het geeft tevens de bounding boxes, en de klassenamen weer van alle objecten die na het croppen nog significant zichtbaar zijn binnen die uitsnede.
6. Tenslotte worden de resultaten opgeslagen via de hulpfunctie `create_data_files`, die de beelduitsnede en de bijbehorende labels opslaat in de juiste mappenstructuur.

De functie `create_crop_for_frame` in Codefragment 8.7 is verantwoordelijk voor het creëren van de crop rond het doelobject. Het transformeert ook de bounding box-coördinaten van alle relevante objecten naar het coördinatenstelsel van de nieuwe crop en past indien nodig augmentaties toe.

```python
    def create_crop_for_frame(
        image: np.ndarray,
        crop_size: int,
        target_box: tuple[int, int, int, int],
        bboxes: np.ndarray,
        class_labels: list[str],
        is_validation: bool = False,
    ):
        x1, y1, x2, y2 = target_box
        cx, cy = (x1 + x2) // 2, (y1 + y2) // 2

        # crop aanmaken rond het doelobject
        half_crop = crop_size // 2
        x_min = max(0, cx - half_crop)
        y_min = max(0, cy - half_crop)
        x_max = min(image.shape[1], cx + half_crop)
        y_max = min(image.shape[0], cy + half_crop)

        transform_steps = [
            A.Crop(x_min=x_min, y_min=y_min, x_max=x_max,
                y_max=y_max),
           A.PadIfNeeded(min_height=crop_size, min_width=crop_size),
        ]

        if not is_validation:
            transform_steps.append(A.HorizontalFlip(p=0.5))
            transform_steps.append(
                A.RandomBrightnessContrast(p=0.2)
            )

        transform = A.Compose(
            transform_steps,
            bbox_params=A.BboxParams(
                format="pascal_voc", label_fields=["class_labels"],
                    min_visibility=0.7
            ),
        )

        # augmenteren van de afbeelding en herberekenen van de
            bounding boxes
        augmented = transform(image=image, bboxes=bboxes,
            class_labels=class_labels)
        transformed_image = augmented["image"]
        transformed_bboxes = augmented["bboxes"]
        transformed_class_labels = augmented["class_labels"]
        return transformed_image, transformed_bboxes,
            transformed_class_labels
```

Codefragment 8.7: De `create_crop_for_frame` functie maakt een crop rond een doelobject in de afbeelding. Het past ook transformaties en augmentaties toe, al naar gelang de crop bedoeld is voor training of validatie. De functie retourneert de getransformeerde afbeelding, de bounding boxes, en de klassenamen van de objecten in de crop.

Voor het uitvoeren van beeldtransformaties en augmentaties werd de open-source bibliotheek `Albumentations` gebruikt (Buslaev e.a., 2018). De bibliotheek staat toe om eenvoudig transformatiepipelines te definiëren en vereenvoudigt sterk het werken met bounding-box coördinaten. Augmentaties kunnen de robuustheid van een model verbeteren door extra variatie aan de trainingsdata toe te voegen. De volgende transformaties en augmentaties werden toegepast:

- **`A.Crop`**: Deze transformatie werd gebruikt om een vierkante uitsnede te maken rond het geselecteerde doelobject.
- **`A.PadIfNeeded`**: Soms kan het zijn dat de crop deels buiten de grenzen van de originele afbeelding valt. Om dit te voorkomen, wordt padding met nullen toegepast om de crop altijd te vullen tot de gewenste grootte.
- **`A.HorizontalFlip`**: Deze augmentatie wordt willekeurig toegepast met een kans van 50% om de afbeelding horizontaal te spiegelen.
- **`A.RandomBrightnessContrast`**: Deze augmentatie past willekeurig de helderheid en het contrast van de afbeelding aan met een kans van 20%.

De transformaties `A.Crop` en `A.PadIfNeeded` werden altijd toegepast, terwijl de transformaties `A.HorizontalFlip` en `A.RandomBrightnessContrast` enkel werden toegepast als een crop bedoeld was voor de trainingset (dus niet voor de validatieset). Het toepassen van de `A.Crop` transformatie paste ook de bounding box coördinaten aan van frame-niveau naar crop-niveau. Tenslotte heeft de `A.BboxParams` parameter ook een `min_visibility` parameter, die ervoor zorgt dat alleen bounding boxes met een zichtbaarheid van minstens 70% worden behouden in de crop. Deze parameter werd hardgecodeerd op 70%, maar kan in de toekomst, indien nodig, verder geoptimaliseerd worden.

8.4.2. Training van YOLOv11 Modellen

Met de aangemaakte trainingsdatasets, kon de volgende stap beginnen: het trainen van YOLOv11-modellen. Zoals eerder vermeld, werden vier verschillende datasets gecreëerd, elk met een ander aantal samples per klasse: 500, 1000, 2000 en 3000. Voor elke dataset werd een afzonderlijk model getraind.

Voor de training werd gebruik gemaakt van het `YOLOv11n.pt` model. Dit is de 'nano'-variant van een reeks modellen die zijn voorgetraind (volgens Khanam en Hussain (2024)) op de COCO-dataset. Het starten met een voorgetraind model is een gangbare praktijk in transfer learning, waarbij het model al een basisniveau van kennis heeft over objectherkenning. Hiermee kan het model sneller convergeren met een kleinere dataset aan domeinspecifieke objecten, dan wanneer training aangevat wordt vanaf nul.

De training werd uitgevoerd met behulp van de functionaliteit van de `ultraly-`

`tics`[3] bibliotheek, binnen het script `scripts/train_object_detectors.py`. De relevante code voor het trainen van de modellen is te vinden binnen de functie `train_model` in Codefragment 8.8.

```
TRAIN_EPOCHS = 100
BATCH_SIZE = 0.9
PATIENCE = 10

def train_model(dataset_path, model_name, crop_size):
    model = YOLO("yolo11n.pt")

    model.train(
        data=dataset_path / "data.yaml",
        epochs=TRAIN_EPOCHS,
        imgsz=int(crop_size),
        device="cuda",
        batch=BATCH_SIZE,
        patience=PATIENCE,
        plots=True,
        save=True,
    )

    model_path = OBJECT_DETECTION_MODELS_PATH /
        f"{model_name}.pt"
    model.save(str(model_path))
}
```

Codefragment 8.8: De `train_model` functie traint een YOLOv11-model op basis van de opgegeven dataset. Het gebruikt een voorgetraind model (`yolo11n.pt`) en past de training toe op de opgegeven dataset voor een bepaald aantal epochs. De `patience` parameter bepaalt hoeveel epochs het model zonder verbetering mag trainen.

Voor de training werden de volgende hyperparameters gebruikt:

- **Epochs:** 100 epochs, wat betekent dat het model de volledige dataset 100 keer doorloopt.
- **Batch Size:** 0.9, wat betekent dat de batchgrootte automatisch wordt berekend om 90% van het beschikbare GPU-geheugen te gebruiken.
- **Patience:** Een patience van 10 betekent dat de training stopt als er gedurende 10 opeenvolgende epochs geen verbetering in de validatieprestaties wordt waargenomen. Aangezien de prestaties van de modellen tijdens de training

[3] https://pypi.org/project/ultralytics/ (laatst geraadpleegd op 2025-05-29)

echter steeds beter werden tot en met de 100ste epoch, werd de training niet vroegtijdig gestopt.

De `train_model` functie werd telkens in een apart proces uitgevoerd voor optimalisatie van het geheugenbeheer. Tenslotte werd elk model opgeslagen in de map `data/models/object_detection`.

8.4.3. Combineren van Objectdetectie en FastSAM Tracking

Met de getrainde modellen was het mogelijk om de objectdetectie te koppelen aan de eerder verkregen FastSAM trackingresultaten uit Sectie 8.2. In deze sectie wordt het proces beschreven om de FastSAM segmentaties te combineren met objectdetectie, met als doel een dataset te verkrijgen die voor elke frame van de evaluatieopnames een lijst met gelabelde, bekeken objecten bevat. Het proces werd geïmplementeerd in de notebook `13_predict_object_detection.ipynb` en kan worden samengevat in de volgende stappen:

1. **Genereren van Vergelijkingssets per Frame**: Voor elke frame van een evaluatieopname werd een 'vergelijkingsset' samengesteld. Deze set bracht de informatie uit twee bronnen samen. De eerste component bestond uit de door FastSAM geïdentificeerde objectsegmenten, die al eerder op basis van de blikdata van de student waren geselecteerd. De tweede component bevatte de objectdetecties die het resultaat waren van de toepassing van het getrainde YOLOv11-model op een specifieke uitsnede (crop) binnen het betreffende frame. Deze uitsnede werd gecentreerd rond het blikpunt van de student.
2. **Matchen van Segmentaties en Detecties:** Binnen elke vergelijkingsset werden de FastSAM-segmentaties gematcht met de YOLOv11-detecties op basis van ruimtelijke overlap (Intersection over Union, IoU), waarbij enkel betrouwbare matches werden behouden. De term 'Intersection over Union' wordt in de volgende sectie verder toegelicht.
3. **Toekennen van een Klasse aan FastSAM Object Tracks:** De klassevoorspellingen van de gematchte YOLOv11-detecties werden geaggregeerd voor elke unieke FastSAM object ID (die een object over meerdere frames volgt). Op basis hiervan werd een definitieve classificatie (één van de 14 kritische objecten of 'onbekend') toegekend aan elke FastSAM-track.
4. **Samenstellen van de Finale Voorspellingsdataset:** De resulterende, nu gelabelde FastSAM-tracks werden per evaluatieopname geconsolideerd tot een finale voorspellingsdataset, die per frame de geïdentificeerde, bekeken objecten specifieerde.

In een latere sectie worden deze voorspellingsdatasets geëvalueerd aan de hand van de grondwaarheid, en worden de resultaten besproken.

Genereren van Vergelijkingssets per Frame

De eerste stap in het combineren van de FastSAM-trackingresultaten met de YOLOv11-objectdetectie was het per frame samenstellen van een 'vergelijkingsset'. Dit vormde een dataset die later gebruikt zou worden om de finale voorspellingsdataset te creëren. De functie `get_comparison_dataset` implementeert de logica voor het genereren hiervan (zie Codefragment 8.9).

De volgende functieparameters werden op voorhand samengesteld:

- `frame_to_gaze_position`: Een dictionary die per frame-index de (x,y)-coördinaat van het blikpunt van de student bevat. Deze data werden reeds eerder berekend in Sectie 5.3.3.
- `frame_to_gaze_segmentation_data`: Een dictionary die per frame-index de resultaten van de FastSAM-tracking bevat (bounding boxes en object ID's van de door de student bekeken segmenten).

Vervolgens worden voor elke frame in de evaluatieopname de volgende stappen uitgevoerd:

1. Indien er een segmentatie beschikbaar is voor de huidige frame, wordt de originele afbeelding geladen.
2. De functie `create_frame_crop` (zie Codefragment 8.10) wordt aangeroepen om een crop te maken rond het blikpunt van de student. Deze functie transformeert ook de bounding boxes van de FastSAM-segmentaties naar het coördinatenstelsel van de nieuwe crop. Hier werd, net zoals bij de creatie van de trainingsdataset, Albumentation gebruikt om de crop te maken. Segmentaties die minder dan 70% zichtbaar waren in de crop, werden weggelaten via de `min_visibility` parameter.
3. De getrainde YOLOv11-modellen werden toegepast op de gemaakte crop, en de voorspellingen werden verzameld. De `model.predict` methode aanvaardt twee belangrijke parameters:
    - `conf`: De drempel voor de vertrouwensscore van de voorspellingen. Deze werd hier ingesteld op 0.5 om in deze fase veel voorspellingen te krijgen. Op deze manier kon later geëxperimenteerd worden met het filteren van voorspellingen bij het matchen van segmentaties en detecties.
    - `iou`: De drempel voor de Intersection over Union (IoU) bij het filteren van overlappende voorspellingen van het model. Wanneer twee voorspellingen een hogere graad van overlap vertonen dan deze drempel, wordt de voorspelling met de lagere vertrouwensscore verwijderd.
4. De resultaten werden opgeslagen in een dictionary met de volgende informatie:

- De FastSAM-segmentatiegegevens (bounding boxes en object ID's).
- De YOLOv11-voorspellingen (bounding boxes, klasse-ID's en vertrouwensscores).

Tenslotte werden alle vergelijkingsdatasets voor de verschillende evaluatieopnames opgeslagen als JSON-bestanden in de map `data/comparison_datasets/<recording_id>.json`.

```python
    def get_comparison_dataset(
        recording_id: str,
        model: YOLO,
        frame_paths: list[Path],
        frame_to_gaze_position: dict[int, tuple[int, int]],
        frame_to_gaze_segmentation_data: dict[int, dict],
    ) -> None:
        comparison_dataset = {}

        for frame_path in tqdm(frame_paths, desc=f"Running inference for {recording_id}"):
            frame_idx = int(frame_path.stem)
            if frame_to_gaze_segmentation_data.get(frame_idx) is None:
                continue

            image = cv2.imread(str(frame_path))
            gaze_position = frame_to_gaze_position[frame_idx]

            transformed_image, transformed_gs_boxes, transformed_gs_object_ids = create_frame_crop(
                image,
                gaze_position,
                frame_to_gaze_segmentation_data,
                frame_idx
            )

            results = model.predict(
                source=transformed_image, conf=0.5, iou=0.5, device="cuda", verbose=False
            )

            predicted_confidences = []
            predicted_bboxes = []
            predicted_class_ids = []
            for box in results[0].boxes:
                conf = float(box.conf[0].cpu().numpy())
                class_id = int(box.cls[0])
                x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()
                predicted_confidences.append(conf)
                predicted_bboxes.append((int(x1), int(y1), int(x2), int(y2)))
                predicted_class_ids.append(class_id)

            comparison_dataset[frame_idx] = {
                "gaze_segmentation": {
                  "boxes": transformed_gs_boxes.astype(np.int32).tolist(),
                    "object_ids": transformed_gs_object_ids,
                },
                "predicted": {
                    "boxes": predicted_bboxes,
                    "class_ids": predicted_class_ids,
                    "confidences": predicted_confidences,
                },
            }

        return comparison_dataset
```

Codefragment 8.9: De `get_comparison_dataset` functie genereert een vergelijkingsdataset voor een evaluatieopname. Deze dataset bevat de FastSAM-segmentaties en de YOLOv11-voorspellingen voor elke frame. De functie maakt een crop rond het blikpunt van de student, past hierop het YOLOv11-model toe en slaat de resultaten op in een dictionary die per frame de segmentatie- en detectiegegevens bevat.

```python
    def create_frame_crop(
        image: np.ndarray,
        gaze_position: tuple[int, int],
        frame_to_gaze_segmentation_data: dict[int, dict],
        frame_idx: int
    ) -> tuple[np.ndarray, list[tuple[int, int, int, int]],
        list[int]]:
        gaze_segmentation_boxes =
            frame_to_gaze_segmentation_data[frame_idx]["boxes"]
        gaze_segmentation_object_ids =
            frame_to_gaze_segmentation_data[frame_idx][
            "object_ids"
        ]

        cx, cy = gaze_position
        x_min = max(0, cx - IMG_CROP_SIZE_HALF)
        y_min = max(0, cy - IMG_CROP_SIZE_HALF)
        x_max = min(image.shape[1], cx + IMG_CROP_SIZE_HALF)
        y_max = min(image.shape[0], cy + IMG_CROP_SIZE_HALF)

        transform = A.Compose(
            [
                A.Crop(x_min=x_min, y_min=y_min, x_max=x_max,
                    y_max=y_max),
                A.PadIfNeeded(min_height=IMG_CROP_SIZE,
                    min_width=IMG_CROP_SIZE),
            ],
            bbox_params=A.BboxParams(
                format="pascal_voc",
                    label_fields=["object_ids"],
                    min_visibility=0.7
            ),
        )

        transformed = transform(
            image=image,
            bboxes=gaze_segmentation_boxes,
            object_ids=gaze_segmentation_object_ids,
        )
        transformed_image = transformed["image"]
        transformed_gs_boxes = transformed["bboxes"]
        transformed_gs_object_ids = transformed["object_ids"]

        return transformed_image, transformed_gs_boxes,
            transformed_gs_object_ids
```

Codefragment 8.10: De `create_frame_crop` functie maakt een crop rond het blikpunt van de student. Het past ook de bounding boxes van de FastSAM-segmentaties aan naar het coördinatenstelsel van de nieuwe crop. Indien een segmentatie minder dan 70% zichtbaar is in de crop, wordt deze weggelaten. De functie retourneert de getransformeerde afbeelding, de aangepaste bounding boxes en de object ID's van de segmentaties.

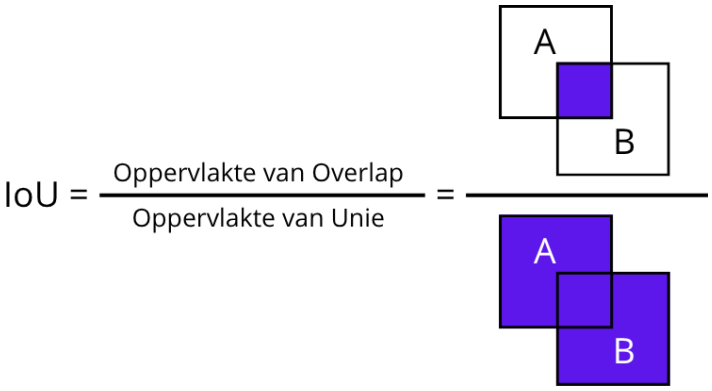
Matchen van Segmentaties en Detecties

De volgende stap was het koppelen van de FastSAM-segmentaties aan de YOLOv11-detecties binnen elke vergelijkingsset. Het doel was om voor elke FastSAM-segmentatie te bepalen welke YOLOv11-detectie, indien aanwezig, het beste overeenkwam. Aangezien de code veel dataverwerkingsstappen bevat, worden hier enkel de belangrijkste conceptuele stappen beschreven.

De functie `get_matched_boxes_per_frame` implementeert de logica voor het matchen van de segmentaties en detecties. Deze functie aanvaardt twee parameters die later binnen een grid-search worden geëvalueerd:

- `min_pred_conf`: De minimum vereiste vertrouwensscore voor een YOLOv11-detectie om als geldig te worden beschouwd.
- `iou_threshold`: De minimum Intersection over Union (IoU) drempel die vereist is om een match te beschouwen als geldig.

IoU wordt gedefinieerd als de verhouding tussen de oppervlakte van de overlap tussen twee bounding boxes en de oppervlakte van hun unie. Zie Figuur 8.6 voor een visuele representatie van IoU.



Figuur 8.6: Conceptuele weergave van Intersection over Union (IoU). De overlap tussen de twee rechthoeken wordt gedeeld door de unie van de twee rechthoeken. Dit resulteert in een waarde tussen 0 en 1, waarbij 1 betekent dat de rechthoeken volledig overlappen. (eigen afbeelding)

De matching-functie omvat de volgende conceptuele stappen:

1. Voor elke frame in de vergelijkingsset worden de FastSAM-segmentaties en YOLOv11-detecties opgehaald.
2. De objectdetectie-voorspellingen worden gefilterd op basis van de `min_pred_conf` parameter.
3. Indien er voor een klasse meerdere detecties zijn, wordt enkel de detectie met de hoogste vertrouwensscore behouden.

4. Indien er geen voorspellingen overblijven na filtering, wordt de frame overgeslagen.
5. Voor elke FastSAM-segmentatie wordt de IoU berekend met elke YOLOv11-detectie.
6. Indien de IoU groter is dan de `iou_threshold`, wordt de detectie beschouwd als een match voor de segmentatie.
7. Voor elke segmentatie wordt de beste match bepaald op basis van de hoogste IoU.
8. Tenslotte wordt de lijst van gematchte segmentaties, detecties en hun IoU-waarden opgeslagen in een dictionary per frame.

Toekennen van een Klasse aan FastSAM Object Tracks

Na het matchen beschikten we per frame over een lijst van FastSAM-segmentaties die succesvol gekoppeld konden worden aan een YOLOv11-objectdetectie. De volgende uitdaging was om op basis van deze per-frame informatie, een definitieve klasse toe te kennen aan elk uniek FastSAM object ID die doorheen de evaluatie-opname werd gevolgd. Een object ID kan immers over meerdere frames worden waargenomen en het YOLOv11-model kan in verschillende frames verschillende voorspellingen maken voor eenzelfde object (of zelfs geen voorspelling maken). De aggregatie dient de ruis die opduikt door verschillende individuele voorspellingen, te reduceren.

Dit proces, inclusief de vorige matchingstap, werd geïmplementeerd in de overkoepelende functie `get_predictions_df`. De belangrijkste conceptuele stappen zijn als volgt:

1. Aggregeren van Voorspellingen per FastSAM Object ID

De eerste stap is het verzamelen van alle YOLOv11-klassevoorspellingen en hun bijbehorende vertrouwensscores per FastSAM object-ID, ongeacht in welk frame ze voorkwamen. Dit aggregeert de per-frame matchingresultaten naar een lijst van voorspellingen voor elke unieke getrackte entiteit.
De functie `get_predictions_per_object_id` voert deze groepering uit en genereert een dictionary waarin elk FastSAM object-ID is gekoppeld aan een lijst van voorspellingen. Voor elke voorspelling wordt de klasse-ID en de bijbehorende vertrouwensscore opgeslagen.

2. Filteren op Minimale Observatieduur

In de latere evaluatiestap werd ook onderzocht of het zinvol was om een parameter te introduceren die bepaalt in hoeveel frames een object moet worden waargenomen vooraleer het wordt geclassificeerd. Dit zou eventueel een effect kunnen hebben op vals-positieve voorspellingen, omdat korte observaties meer vat-

baar zijn voor ruis. Zo worden enkel object tracks die in een minimaal aantal frames zijn waargenomen, behouden. Dit wordt aangedreven door de parameter `min_observed_frames` in de functie `filter_by_min_observed_frames`.

3. Toekennen van de Finale Klasse per Track
Voor elk overgebleven FastSAM object-track, werd een definitieve klasse bepaald door de functie `get_final_prediction_per_object_id`. Hier werd voor elk object ID de klasse gekozen die de hoogste totale som van YOLOv11-vertrouwensscores behaalde over de gehele duur van de track. Dit kan gezien worden als een vorm van stemmingsaggregatie, waarbij de klasse met de meeste 'stemmen' (in termen van vertrouwensscores) wordt gekozen.

Samenstellen van de Finale Voorspellingsdataset

Het resultaat van de vorige stap was een mapping van FastSAM object-ID's, naar hun definitieve voorspelde klasse (één van de 14 kritische objecten) en de geaggregeerde vertrouwensscore. In Sectie 8.2.4 werd al beschreven hoe de metadata van de FastSAM-trackingresultaten opgeslagen werden in zogenaamde 'object-datasets'. In deze stap werden deze datasets, voor elke evaluatieopname, uitgebreid met de voorspelde klasse en de bijbehorende vertrouwensscore voor elk object ID. Het resultaat was een finale voorspellingsdataset (pandas `Dataframe`) per evaluatieopname. Deze was klaar voor evaluatie en bevatten de volgende velden:

- `frame_idx`: De index van de frame in de evaluatieopname.
- `object_id`: De unieke ID van het getrackte object.
- `gs_confidence`: De vertrouwensscore van de FastSAM-segmentatie voor het object in de frame (dus niet van de YOLOv11-detectie).
- `x1, y1, x2, y2`: De coördinaten van de bounding box van de FastSAM-segmentatie in het frame.
- `predicted_class_id`: De voorspelde klasse-ID van het object, gebaseerd op de YOLOv11-detecties. Indien er geen voorspelling was voor het object, werd deze op -1 gezet (dit betekent dat het object als 'onbekend' werd beschouwd).
- `predicted_confidence`: De geaggregeerde vertrouwensscore van de voorspelde klasse, gebaseerd op de YOLOv11-detecties.

8.5. Evaluatie van de Analysepipeline

Na het implementeren van de functionaliteit voor het creëren van de finale voorspellingsdatasets, kon de evaluatie van de volledige analysepipeline beginnen. Deze evaluatie had drie hoofddoelen:

1. Het bepalen van de optimale combinatie van hyperparameterwaarden voor het eerder besproken matchingproces waarbij de finale klasse aan een track wordt toegekend.
2. Het beoordelen van de algehele prestaties van de verschillende getrainde YOLOv11-modellen tegenover de grondwaarheid op basis van precisie, recall en F1-score.
3. Het verkrijgen van een kwantitatief antwoord op de deelonderzoeksvragen betreffende de accuraatheid van het ontwikkelde systeem, specifiek (zoals origineel geformuleerd in Sectie 1.2):
    - In welke mate kan de ontwikkelde software correct bepalen welke kritische objecten studenten hebben waargenomen?
    - In welke mate kan de ontwikkelde software nauwkeurig meten hoe lang studenten naar deze objecten keken?

8.5.1. Evaluatieprocedure

De evaluatie van de analysepipeline werd systematisch uitgevoerd door de voorspellingsdatasets te vergelijken met de eerder gecreëerde grondwaarheidsdataset (zie Hoofdstuk 7). Dit proces werd net zoals de creatie van de voorspellingsdatasets, geïmplementeerd in de notebook `13_predict_object_detection.ipynb`.

Definitie van de Grid Search Ruimte

Om na te gaan welke hyperparameterwaarden het beste resultaat opleveren, werd een grid search uitgevoerd. Een grid search is een manier om de beste combinatie van parameterwaarden te vinden door alle mogelijke combinaties uit te proberen. Deze search werd voor de vier getrainde modellen uitgevoerd over de volgende hyperparameters:

- **training_sample_count**: Een impliciete parameter die het aantal trainingssamples per klasse bepaalt, die gebruikt werden voor het trainen van de modellen.
- **min_pred_conf**: De minimum vereiste vertrouwensscore voor een YOLOv11-detectie om als geldig te worden beschouwd.
- **iou_threshold**: De minimum Intersection over Union (IoU) drempel die vereist is om een match tussen een FastSAM-segmentatie en een YOLOv11-detectie te beschouwen als geldig.
- **min_observed_frames**: Het minimum aantal frames waarin een object moet worden waargenomen alvorens het wordt geclassificeerd.

Zie Codefragment 8.11 voor de gebruikte waarden voor deze hyperparameters, inclusief de overkoepelende logica voor het opstellen van het resultaat van de grid

search.

```
ground_truth_df =
    experiment_utils.get_ground_truth_df(IGNORED_CLASS_IDS)

min_pred_confs = [0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9]
iou_thresholds = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
min_observed_frames = [0, 1, 3, 5, 7]

grid_search_params = list(
    itertools.product(min_pred_confs, iou_thresholds,
        min_observed_frames)
)

grid_search_results = []
for i, model_path in enumerate(models):
    print(f"Evaluating model {i + 1}/{len(models)}:
        {model_path.stem}")

    model = YOLO(model_path)
    model_grid_search_results = evaluate_model(
        model_name=model_path.stem,
        model_class_names=model.names,
        ground_truth_df=ground_truth_df,
        grid_search_params=grid_search_params,
    )
    grid_search_results.extend(model_grid_search_results)
```

Codefragment 8.11: De code voor het uitvoeren van de grid search over de hyperparameters. Eerst wordt de grondwaarheid geladen en worden de hyperparameterwaarden gedefinieerd. Vervolgens wordt voor elk model de evaluatiefunctie `evaluate_model` aangeroepen. Deze functie evalueert de prestaties van het model over alle combinaties van hyperparameters. De resultaten van de grid search worden opgeslagen in een lijst.

Evaluatie van de Modellen

Voor elk model wordt de overkoepelende functie `evaluate_model` aangeroepen, die elke mogelijke combinatie van hyperparameters evalueert. Belangrijk om op te merken is dat, hoewel we hier de verschillende getrainde YOLOv11-modellen evalueren, we op hetzelfde moment dit ook doen voor de gehele pipeline, inclusief de FastSAM-tracking. Aangezien er in totaal 315 mogelijke combinaties zijn, maakt deze functie gebruik van `multiprocessing` om de evaluatie van de modellen te versnellen. Het stuurt elke combinatie van hyperparameters naar een apart subproces, uitgevoerd door de functie `evaluate_grid_combination` (zie Codefrag-

ment 8.12).

```python
    def evaluate_grid_combination(
        model_name: str,
        model_class_names: dict[int, str],
        ground_truth_df: pd.DataFrame,
        min_pred_conf: float,
        iou_threshold: float,
        min_observed_frames: int,
    ):
        # Lege confusion matrix aanmaken
        cm = experiment_utils.create_confusion_matrix(IGNORED_CLASS_IDS)

        # Lege evaluatie DataFrame aanmaken
        full_evaluation_df = pd.DataFrame()

        comparison_sets = (COMPARISON_SETS_PATH / model_name).iterdir()
        for comparison_set_path in comparison_sets:
            recording_id = comparison_set_path.stem

            # Voor elke evaluatieopname, de voorspellingen berekenen
            predictions_df = get_predictions_df(
                model_class_names=model_class_names,
                comparison_set_path=comparison_set_path,
                min_pred_conf=min_pred_conf,
                iou_threshold=iou_threshold,
                min_observed_frames=min_observed_frames,
            )

            # De grondwaarheid voor de huidige opname ophalen
            gt_df_recording =
                ground_truth_df[ground_truth_df["recording_id"] ==
                recording_id]

            # Evalueren van de voorspellingen
            eval_df = experiment_utils.evaluate_predictions(
                predictions_df=predictions_df,
                gt_df=gt_df_recording,
            )

            # Voorspellingsresultaten toevoegen aan de volledige evaluatie
                DataFrame
            full_evaluation_df = pd.concat(
                [full_evaluation_df,
                    eval_df.assign(recording_id=recording_id)],
                ignore_index=True,
            )

            # Confusion matrix bijwerken met de evaluatieresultaten
            experiment_utils.update_confusion_matrix(cm, eval_df)

        # Metrieken van de confusion matrix berekenen (precision, recall,
            F1-score, etc.)
        cm_metrics = experiment_utils.confusion_matrix_metrics(cm)

        # Valideren van de confusion matrix
        validate_confusion_matrix(cm_metrics, ground_truth_df)

        return full_evaluation_df, cm, cm_metrics
```

Codefragment 8.12: De `evaluate_grid_combination` functie evalueert een enkele combinatie van hyperparameters voor een model over de volledige collectie van evaluatieopnames. Het laadt de grondwaarheid en de voorspellingsdataset en berekent de evaluatiemetrieken. De functie retourneert een DataFrame met de evaluatieresultaten, de confusion matrix en zijn metrieken.

De `evaluate_grid_combination` functie berekent drie belangrijke outputs:

- **full_evaluation_df**: Een DataFrame met de evaluatieresultaten voor elk frame van elke evaluatieopname. Dit bevat informatie zoals de voorspelde klasse, de grondwaarheidsklasse en de bijbehorende metriekwaarden.
- **cm**: De confusion matrix die de prestaties van het model over alle evaluatieopnames heen samenvat. Deze matrix toont het aantal correcte en onjuiste voorspellingen per klasse.
- **cm_metrics**: Berekende metriekwaarden (precisie, recall, F1-score, etc.) op basis van de confusion matrix.

Het creëren van deze drie outputs gebeurt in de volgende stappen:

1. Creatie van de Evaluatie DataFrame
De evaluatiedataframe wordt berekend door de functie `experiment_utils.evaluate_predictions`, die de voorspellingen vergelijkt met de grondwaarheid voor elke frame van de evaluatieopname. Het voegt voor elke frame twee zaken toe aan de voorspellingsdataset:

- **true_class_id**: De klasse-ID van het object zoals gedefinieerd in de grondwaarheid.
- **label**: Een categorisch label dat aangeeft hoe de voorspelling zich verhoudt tot die grondwaarheid. Dit label kan de volgende waarden aannemen:
  - **TP (True Positive)**: De voorspelde klasse komt overeen met de grondwaarheidsklasse voor een object in hetzelfde frame. Een correcte detectie en classificatie.
  - **FP (False Positive):** Het model voorspelt een object, maar er is geen overeenkomstige grondwaarheid voor die klasse in het frame; of het voorspelde object komt niet overeen met de grondwaarheidsklasse.
  - **TN (True Negative):** Het model voorspelt correct dat er geen object bekeken werd (voorspelt -1, of 'onbekend') en er is inderdaad geen overeenkomstige grondwaarheid binnen dit frame.
  - **FN (False Negative):** Er is een object in de grondwaarheid aanwezig, maar het model heeft dit object niet gedetecteerd of geclassificeerd.

Ook werd het veld `mask_area` vervangen door zijn waarde in de `grondwaarheid`, om latere analyses mogelijk te maken. Indien het ging om een vals-positief waar geen grondwaarheid voor bestond, werd hier een lege waarde (`NaN`) toegekend.

Hier is het interessant om stil te staan bij twee specifieke gevallen: Wanneer de evaluatie een vals-positief aangeeft waarvoor geen grondwaarheid bestaat, werd hier

een speciale klasse-ID -3 toegekend als 'ware klasse' (met als naam 'geen grondwaarheid'). Wanneer de evaluatie een vals-negatief aangeeft, werd de klasse-ID -2 toegekend als 'voorspelde klasse' (met als naam 'geen voorspelling'). Dit maakte het mogelijk om deze gevallen mee te visualiseren in de confusion matrix.

2. Bijwerken van de Confusion Matrix

De confusion matrix is een vierkante tabel die het mogelijk maakt om de prestaties van het model te visualiseren en metrieken zoals precisie, recall en F1-score te berekenen. De matrix bestaat uit rijen en kolommen die overeenkomen met de verschillende klassen. Elke rij vertegenwoordigt de ware klasse (grondwaarheid), terwijl elke kolom de voorspelde klasse voorstelt.

De functie `experiment_utils.update_confusion_matrix` wordt gebruikt om de confusion matrix bij te werken op basis van de evaluatieresultaten voor een bepaalde evaluatieopname (zie Codefragment 8.13).

```
    def update_confusion_matrix(
        confusion_mat: pd.DataFrame, eval_df: pd.DataFrame
    ) → pd.DataFrame:
        for _, row in eval_df.iterrows():
            true = row.get("true_class_id")
            pred = row.get("predicted_class_id")

            if pd.isna(true): # Als er geen grondwaarheid is
                if pred == UNKNOWN_CLASS_ID:
                    # Dit is een waar-negatief (TN),
                    # en wordt niet meegeteld in de confusion matrix.
                    continue

                # Dit is een vals-positief (FP) voor de voorspelde
                    klasse.
                true = MISSING_GROUND_TRUTH_CLASS_ID

            t = int(true)
            p = int(pred)
            if t in confusion_mat.index and p in
                confusion_mat.columns:
                confusion_mat.loc[t, p] += 1
```

Codefragment 8.13: De `update_confusion_matrix` functie werkt de confusion matrix bij op basis van de evaluatieresultaten. Voor elke rij in het evaluatiedataframe wordt het aantal voorspellingen in de cel met de ware klasse en de voorspelde klasse verhoogd. Indien er geen grondwaarheid is, en het model toch een klasse voorspelt, wordt dit beschouwd als een vals-positief (FP).

3. Berekenen van Confusion Matrix Metrieken
Uit de confusion matrix kunnen verschillende metrieken worden afgeleid die de prestaties van het model samenvatten. Dit gebeurt met behulp van de functie `experiment_utils.confusion_matrix_metrics`.

Voor elke individuele objectklasse (met uitzondering van de speciale klassen voor FP/FN) worden de volgende metrieken berekend:

- **Precisie (Precision)**: Deze metriek meet het aandeel van de correct voorspelde instanties (True Positives, TP) ten opzichte van alle voorspelde instanties binnen die klasse (TP + False Positives, FP). Een hoge precisie betekent dat wanneer het model een bepaalde klasse toekent aan een object, die voorspelling meestal correct is. Met andere woorden, het geeft een indicatie over hoeveel vals-positieven het model maakt. Optimalisatie van precisie kan echter leiden tot een afname van recall, omdat het model mogelijk minder objecten van die klasse detecteert om vals-positieven te vermijden.

$$\text{Precisie}_{\text{klasse}} = \frac{\text{TP}_{\text{klasse}}}{\text{TP}_{\text{klasse}} + \text{FP}_{\text{klasse}}}$$

- **Recall**: Dit meet het aandeel van de correct voorspelde instanties (TP) van een klasse ten opzichte van alle daadwerkelijke instanties in de grondwaarheid (TP + False Negatives, FN). Het geeft aan hoe goed het model in staat is om alle relevante objecten van een bepaalde klasse te herkennen. Een hoge recall betekent dat het model weinig vals-negatieven geeft. Overoptimalisatie van recall kan echter leiden tot een toename van vals-positieven.

$$\text{Recall}_{\text{klasse}} = \frac{\text{TP}_{\text{klasse}}}{\text{TP}_{\text{klasse}} + \text{FN}_{\text{klasse}}}$$

- **F1-score**: Het harmonisch gemiddelde van precisie en recall voor een klasse, die een gebalanceerde maatstaf biedt door zowel de precisie als de recall in overweging te nemen.

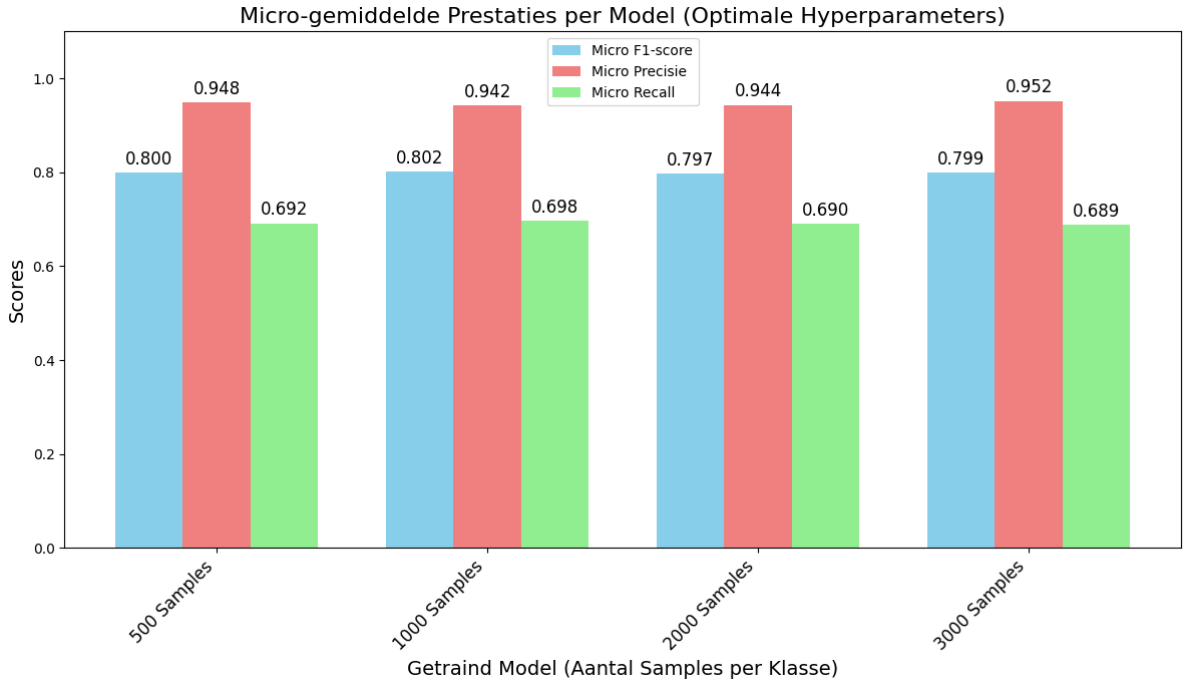
$$\text{F1-score}_{\text{klasse}} = 2 \times \frac{\text{Precisie}_{\text{klasse}} \times \text{Recall}_{\text{klasse}}}{\text{Precisie}_{\text{klasse}} + \text{Recall}_{\text{klasse}}}$$

- **Support**: Het aantal daadwerkelijke instanties van een klasse in de grondwaarheidsdataset (de som van een rij in de confusion matrix voor die klasse).

Naast deze per-klasse metrieken worden ook **micro-gemiddelde** precisie, recall en F1-score berekend. Bij micro-averaging worden eerst de globale totalen van True Positives, False Positives, en False Negatives over alle klassen heen gesommeerd. Vervolgens worden de metrieken op basis van deze globale totalen berekend (net zoals hierboven beschreven, maar dan met de micro-totalen). Dit geeft een algemeen overzicht van de prestaties van het model over alle klassen heen.

8.5.2. Resultaten van de Hyperparameter Grid Search

Op basis van F1-score werd voor elk model de beste combinatie van hyperparameters bepaald. De precisie, recall en F1-score voor het beste resultaat per model worden weergegeven in Figuur 8.7.



Figuur 8.7: De precisie, recall en F1-score van de beste modellen per getraind YOLOv11-model. De beste hyperparameters werden bepaald op basis van de hoogste F1-score over alle evaluatieopnames. De resultaten zijn weergegeven voor de vier getrainde modellen, gesorteerd op de hoeveelheid samples die gebruikt werden voor het trainen.

Merk op dat de resultaten van elk model hier heel nauw bij elkaar liggen, wat misschien onverwacht lijkt. Dit wijst echter potentieel op het feit dat de beperkende factor in de prestaties van de analysepipeline niet de YOLOv11-objectdetectie is, maar eerder de FastSAM-tracking en segmentatie stap.

Volgens de grid search werd het beste model getraind op 1000 samples per klasse, dat de volgende resultaten behaalde:

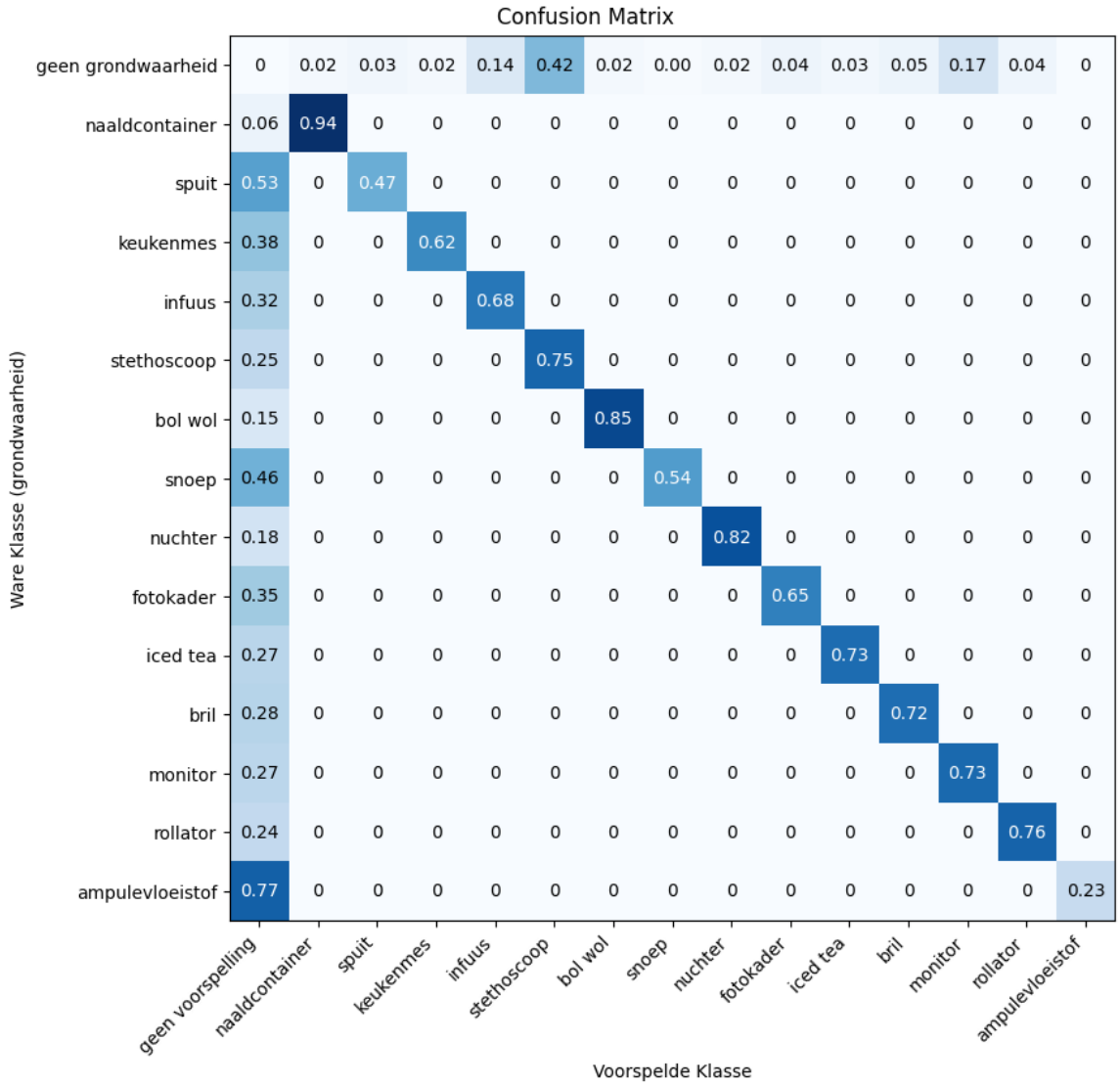
- **Precisie:** 0.9424
- **Recall:** 0.6976
- **F1-score:** 0.8017

De beste combinatie van hyperparameters voor dit model is:

- **min_pred_conf:** 0.85
- **iou_threshold:** 0.2

- **min_observed_frames:** 3

De confusion matrix voor dit model is weergegeven in Figuur 8.8. We zien dat er binnen de objecten in de grondwaarheid geen enkele vals-positief te bespeuren valt. We zien echter wel dat er een aantal vals-positieven zonder grondwaarheid zijn (de rij 'geen grondwaarheid' in de matrix). Een hypothese is dat een groot deel van deze zogezegde vals-positieven eigenlijk objecten zijn die correct gedetecteerd zijn, maar die niet in de grondwaarheidsdataset werden opgenomen. Dit wordt in een latere sectie verder onderzocht.



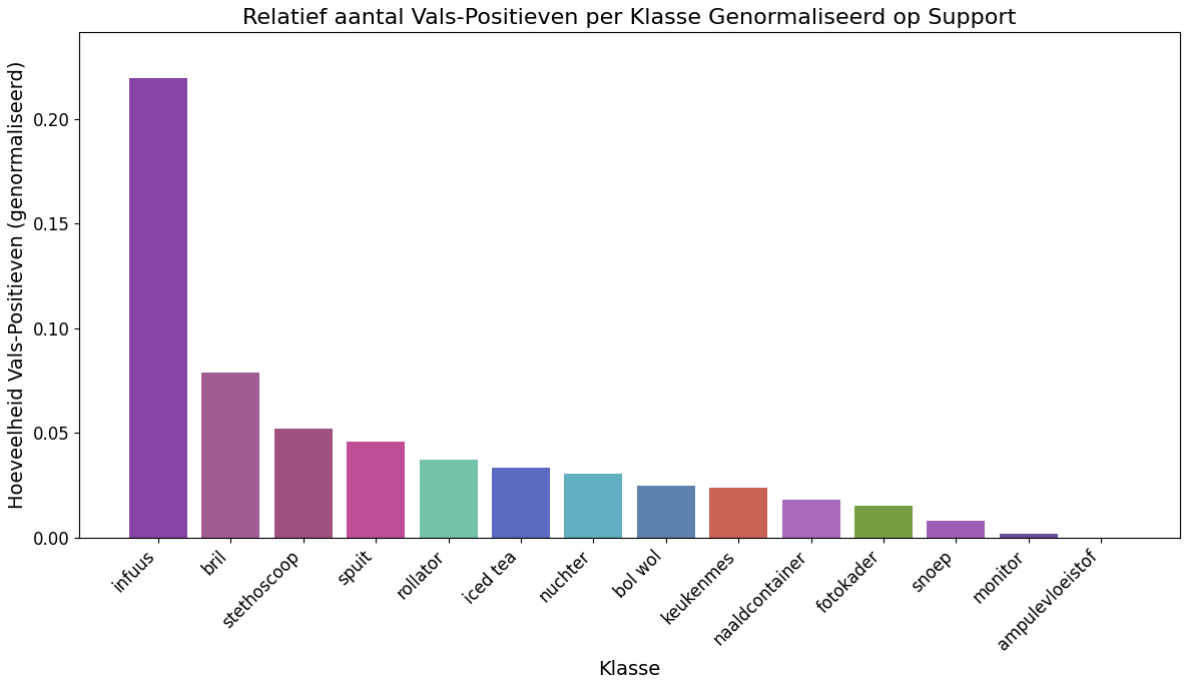
Figuur 8.8: De confusion matrix voor het beste model, getraind op 1000 samples per klasse. Hier zijn de totaalwaarden van elke cel genormaliseerd door te delen door de som van de waarden in zijn rij. Elke rij in de matrix vertegenwoordigt de ware klasse (grondwaarheid), terwijl elke kolom staat voor de voorspelde klasse. Merk op dat er twee specifieke klassen zijn toegevoegd, namelijk 'geen grondwaarheid' wat wijst op vals-positieven en 'geen voorspelling' wat wijst op vals-negatieven. De diagonaal van de matrix toont de correcte voorspellingen, terwijl de andere cellen de fouten van het model weergeven.

8.5.3. Verdere Analyse van het Beste Model

Om een beter inzicht te krijgen in de prestaties van de analysepipeline en om mogelijk problemen te identificeren, werden de resultaten van het beste model (getraind op 1000 samples per klasse) verder geanalyseerd.

Analyse van de Vals-Positieven

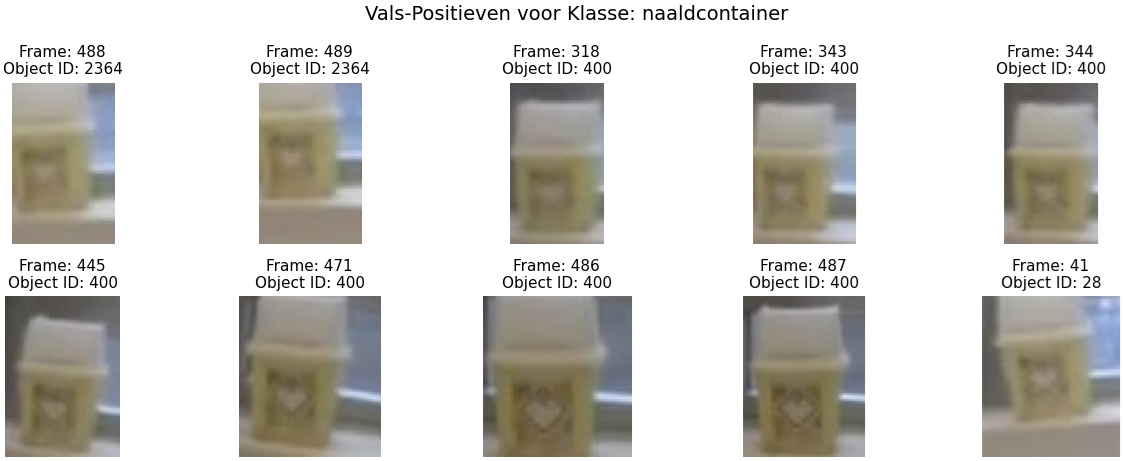
Eerst werden de vals-positieven per klasse geanalyseerd door middel van een staafdiagram om een algemeen overzicht te krijgen van de verdeling van vals-positieven over de verschillende objectklassen (zie Figuur 8.9). De aantallen werden genormaliseerd ten opzichte van de support van die klasse om een eerlijke vergelijking te verkrijgen. Opmerkelijk is dat het 'infuus' object veruit de meeste vals-positieven geeft, wat wijst op een meer systematisch probleem met de detectie van dit object.



Figuur 8.9: Staafdiagram van het aantal vals-positieven per klasse voor het beste model, genormaliseerd ten opzichte van de support van die klasse.

Om verder inzicht te verkrijgen in de aard van de vals-positieven, werden per klasse de uitsnedes van elk vals-positief object manueel bekeken. Hier is het belangrijk om te vermelden dat de uitsnedes betrekking hebben op de FastSAM-segmentaties, en niet op de YOLOv11-detecties. Voor voorbeelden van deze uitsnedes binnen de klasse 'naaldcontainer', zie Figuur 8.10. Zoals men kan zien, zijn alle vals-positieven in de werkelijkheid toch correcte detecties, maar werden ze niet opgenomen in de grondwaarheidsdataset. Een mogelijke verklaring hiervoor is dat de segmentatiemaskers van de FastSAM-tracking niet altijd volledig overeenkomen met de grondwaarheid. Dit zorgt ervoor dat sommige objecten als 'bekeken' aan-

geduid worden in de geautomatiseerde analyse, terwijl ze niet opgenomen werden in de grondwaarheid omdat er geen overlap was met het blikpunt van de student.



Figuur 8.10: Vals-positieve voorbeelden voor de klasse 'naaldcontainer' van het beste model. De afbeeldingen tonen de uitsnedes van de FastSAM-segmentaties die als vals-positief werden beschouwd, vergeleken met de grondwaarheid. Het framenummer en de klasse-ID van de FastSAM-segmentatie zijn weergegeven in de titel van elke afbeelding.

Voor elke klasse werden op deze manier de uitsnedes van de vals-positieven manueel bekeken om te bepalen of ze in werkelijkheid ook vals-positief waren. Uit het totaal van 262 zogezegde vals-positieven, bleken er echter slechts drie 'echte' vals-positieven te zijn (met uitzondering van het infuus, dat hierna wordt besproken). Interessant was dat deze drie vals-positieven allemaal betrekking hadden op de klasse 'fotokader', en ook tot dezelfde FastSAM object ID behoorden (zie Figuur 8.11). Dit wijst op een fout in de tracking van dit specifieke object, veroorzaakt door het onjuist toekennen van een object ID aan een segmentatie die niet tot hetzelfde object behoort. Inderdaad, bij manuele inspectie van de resultaten bleek dat de meeste frames met dit object ID eigenlijk betrekking hadden op het effectief bekeken fotokader. Echter, wanneer de student het hoofd draaide, bleef de FastSAM-tracking andere objecten toewijzen aan dit object ID, waardoor vals-positieven ontstonden. Dit kan eventueel worden opgelost in de toekomst na een meer uitgebreide analyse van dit probleem in de FastSAM-tracking.

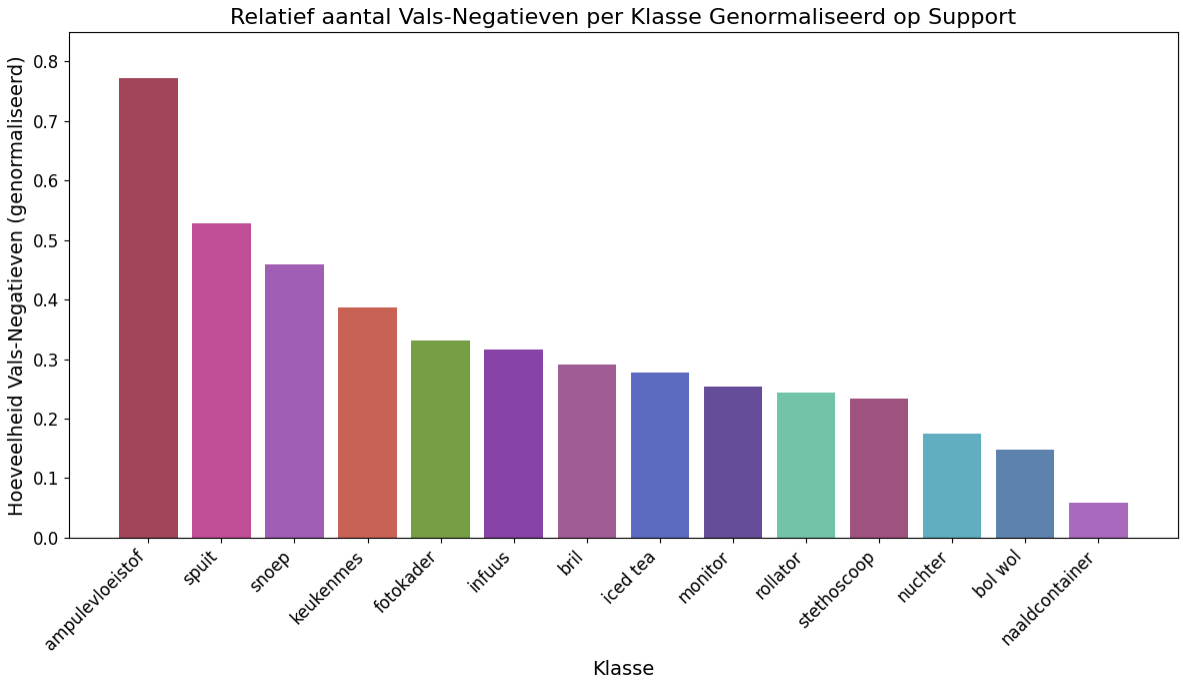
Figuur 8.11: Vals-positieve voorbeelden voor de klasse 'fotokader' van het beste model. Hier zien we drie 'ware' vals-positieven, en drie vals-positieven die in werkelijkheid correcte detecties waren.

Bij de vals-positieven verdient het infuus een aparte vermelding. Het model heeft in totaal 102 vals-positieven voorspeld voor deze klasse. In Figuur 8.12 zien we enkele willekeurige uitsnedes van deze vals-positieven. Echter, bij manuele inspectie bleken de uitsnedes van deze voorspellingen toch het infuus te bevatten. De definitie van 'vals-positief' steunt bij dit object op de definitie van wat begrepen wordt onder 'bekeken'. In de grondwaarheid werden enkel de zak en de druppelteller van het infuus opgenomen; terwijl de analyse ook het infuus detecteert wanneer de student enkel naar de infuuspaal kijkt. Als men enkel wil weten of de student naar de infuuszak of druppelteller kijkt, dan zijn dit inderdaad vals-positieven. De waarneming valt te verklaren door het feit dat FastSAM het volledige infuus, inclusief de paal en de voet, detecteert. Vervolgens detecteert het YOLOv11-model de infuuszak en de druppelteller, waarvan de bounding-box overlapt met de FastSAM-segmentatie. Bij een lage minimum vereiste IoU drempel, worden deze segmentaties dus ook geclassificeerd als 'infuus'. Deze bevinding wijst op problemen bij de detectie van objecten die samengesteld zijn uit meerdere componenten.

Figuur 8.12: Vals-positieve voorbeelden voor de klasse 'infuus' van het beste model.

Analyse van de Vals-Negatieven

Om het effect van het type object op de prestaties van het model te onderzoeken, werden vals-negatieven per klasse (van het beste model) geanalyseerd (zie Figuur 8.13). De aantallen werden opnieuw genormaliseerd ten opzichte van de support van die klasse om een eerlijke vergelijking te verkrijgen. Uit deze analyse bleken de klassen 'ampule vloeistof', 'spuit' en 'snoep', de meeste vals-negatieven op te leveren. Dit is logisch, aangezien deze objecten klein zijn, waardoor ze moeilijk te detecteren bleken. De 'ampule vloeistof' en de 'spuit' waren heel doorschijnend, met een witte achtergrond. Hoewel het snoepje heel klein was, werden toch meer dan 50% van de gevallen correct gedetecteerd.



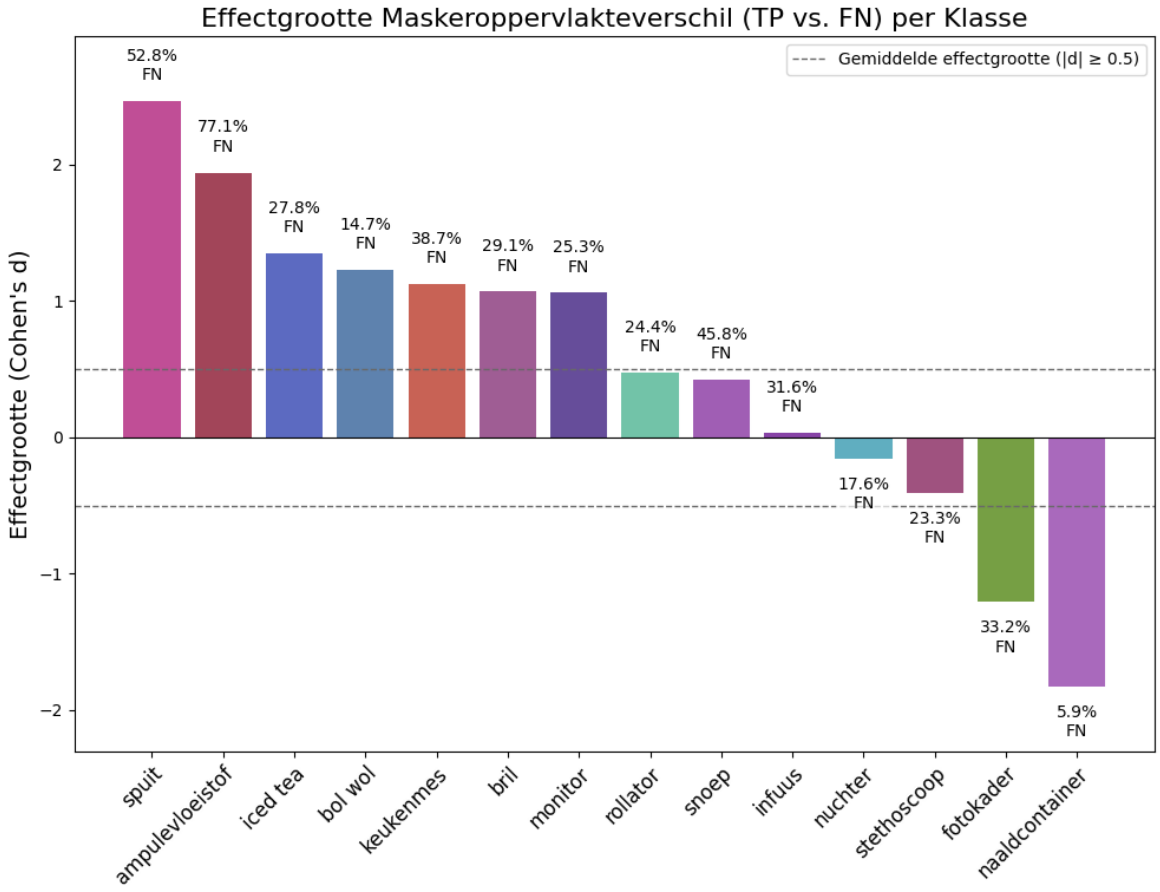
Figuur 8.13: Staafdiagram van het aantal vals-negatieven per klasse voor het beste model, genormaliseerd ten opzichte van de support van die klasse.

Het verwerven van een beter inzicht in de aard van de vals-negatieven vereiste echter een meer diepgaande analyse.

Analyse van de Maskergroottes van Vals-Negatieven en Waar-Positieven

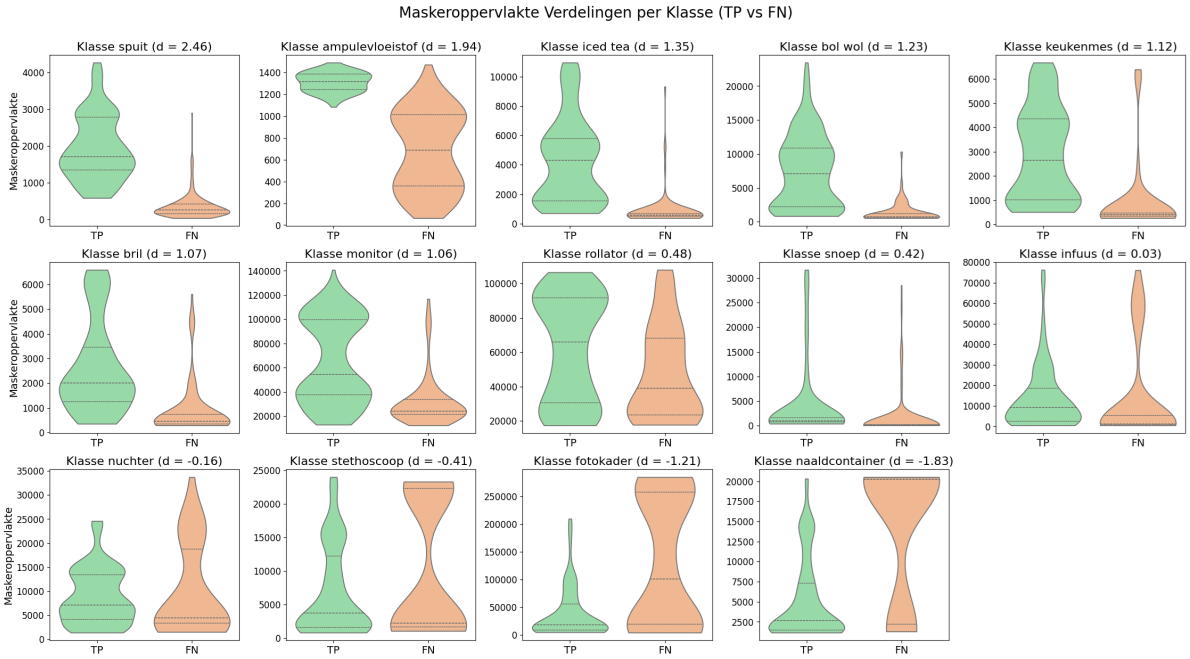
Als hypothese kunnen we stellen dat de analysepipeline vooral kleine objecten mist. Om dit te onderzoeken, werden de maskergroottes van de waar-positieven (TP) en vals-negatieven (FN) per klasse geanalyseerd. Voor elke klasse werd 'Cohen's d' berekend, een maatstaf voor de effectgrootte die aangeeft hoe groot het verschil is tussen de gemiddelde maskergrootte van de TP en FN (zie Figuur 8.14). Boven elke staaf in de figuur staat ook het gehalte aan vals-negatieven voor die klasse, tegenover de support ervan. Dit geeft een indicatie over de betrouwbaarheid van de ef-

fectgrootte, aangezien een hoge effectgrootte bij een laag aantal vals-negatieven mogelijk niet representatief is. We zien dat de meeste klassen een gemiddelde tot hoge effectgrootte tonen, wat betekent dat de TP gemiddeld grotere maskers bevat dan de FN.



Figuur 8.14: Cohen's d voor de gemiddelde maskergrootte van de waar-positieven (TP) en vals-negatieven (FN) per klasse. Boven elke staaf staat het percentage vals-negatieven ten opzichte van de support van die klasse. Een positieve waarde duidt erop dat de TP gemiddeld grotere maskers bevat dan de FN, terwijl een negatieve waarde het omgekeerde aangeeft. Een waarde van 0.2 wordt vaak beschouwd als een klein effect, 0.5 als een gemiddeld effect, en 0.8 als een groot effect.

Om een beter inzicht te krijgen in de distributie van de maskergroottes binnen zowel de TP als de FN, werden voor elke klasse vioolplotten gemaakt (zie Figuur 8.15). Deze plots tonen de verdeling van de maskergroottes voor zowel de TP als de FN, waarbij de breedte van elke viool de dichtheid van de waarden aangeeft. Hier zijn de klassen van links naar rechts en van boven naar beneden geordend volgens hun 'Cohen's d' waarde.



Figuur 8.15: Vioolplotten van de maskergroottes voor de waar-positieven (TP) en vals-negatieven (FN) per klasse. De breedte van elke viool geeft de dichtheid van de waarden aan, terwijl de hoogte de verdeling van de maskergroottes weergeeft.

Als aanvullende analyse werd ook manueel gekeken naar de uitsnedes van de vals-negatieven per klasse (op basis van de bounding box van de grondwaarheid). Met deze drie analyses werden de volgende inzichten verkregen voor elke klasse die een 'hoge' effectgrootte vertoonde ('Cohen's d' > 1):

- **spuit en ampule vloeistof**: Beide objecten vertonen een zeer hoge effectgrootte. Dit is logisch, aangezien deze objecten klein en sterk doorzichtig zijn met een witte achtergrond. Binnen de vioolplots van beide klassen zien we dat er voor de FN's een duidelijke grens is waaronder de maskeroppervlakten vallen. Voor de spuit is dit rond de 600 pixels, en voor de ampule vloeistof rond de 1200 pixels.
- **iced tea, bol wol, keukenmes, en bril**: In de vioolplot van deze klassen zien we dat de FN's duidelijk geconcentreerd zijn onder de 1000 pixels, terwijl de TP's een veel bredere spreiding hebben.

Daarnaast zijn er ook enkele klassen die een lage effectgrootte vertonen ('Cohen's d' tussen -0.5 en 0.5):

- **rollator**: De rollator vertoont hier geen systematisch probleem. De 'Cohen's d' waarde is hier waarschijnlijk een toeval. Een beter inzicht in de vals-negatieven voor dit object vergt een manuele inspectie van de opname, in combinatie met een overlay van de FastSAM-segmentaties, het blikpunt en de grondwaarheid.

- **snoep**: De vioolplot toont dat het aantal vals-negatieven toeneemt naarmate de maskergrootte kleiner wordt. Toch vertoont het snoepje een hoog aantal TP's met een maskergrootte rond de 1000 pixels, in tegenstelling tot de hierboven besproken klassen. Dit valt waarschijnlijk te verklaren door het feit dat dit object een hoog kleurcontrast heeft met de achtergrond (groen snoepje op een witte tafel). Het FastSAM model heeft het dan ook gemakkelijker om dit object te segmenteren, zelfs wanneer het klein is.
- **infuus, nuchter en stethoscoop**: Deze klassen vertonen geen systematisch probleem ten opzichte van de maskergrootte. Een beter inzicht in de vals-negatieven van deze objecten zal net zoals de rollator een manuele inspectie vereisen.

Tenslotte vertonen de fotokader en de naaldcontainer een hoge negatieve effectgrootte ('Cohen's d' < -1). Bij de fotokader valt dit te verklaren door het feit dat binnen één van de opnames, de student het object van heel dichtbij bekeek. Dit zorgde ervoor dat het object niet meer binnen de crop van 640x640 viel, waardoor het objectdetectiemodel geen voorspelling kon maken. De negatieve effectgrootte van de naaldcontainer is waarschijnlijk niet representatief, aangezien deze klasse slechts 5.9% van de support heeft als vals-negatieven.

Een consistent resultaat is dat de meeste klassen een dunne, lange staart vertonen in de vioolplots voor de FN's. Dit valt, net zoals bij de vals-positieven, te verklaren door het feit dat de segmentatiemaskers van de FastSAM-tracking niet altijd volledig overeenkomen met die van de grondwaarheid. Soms is een segmentatiemasker echter opgeschoven of kleiner dan die van de grondwaarheid, waardoor het blikpunt van de student net niet meer binnen de segmentatie valt. Dit fenomeen resulteert dus ongeacht de maskergrootte in vals-negatieven.

Met de analyses die in dit hoofdstuk werden uitgevoerd, worden de onderzoeksvragen in het volgende hoofdstuk beantwoord.

9

Conclusie

Deze bachelorproef had als doel een methode te ontwikkelen om de observatievaardigheden van studenten in het 360° Zorglab van HOGENT op een geautomatiseerde, objectieve manier te evalueren. Computervisie-modellen werden geïntegreerd met eyetrackingdata van Tobii Glasses, om zo de huidige, subjectieve evaluatiemethode te vervangen door een datagestuurde aanpak. Hiertoe werd een proef-of-concept applicatie ontwikkeld en werden verschillende computervisie-modellen geëvalueerd op hun vermogen om objecten te detecteren en te segmenteren in de eyetrackingdata.

9.1. Beantwoording van de Onderzoeksvragen

De centrale onderzoeksvraag was: *Hoe kunnen computervisie-modellen geïntegreerd worden met eyetrackingdata van Tobii Glasses om observatieprestaties van studenten in het 360° Zorglab automatisch te analyseren?* Er werd aangetoond dat een dergelijke integratie haalbaar is en bovendien veelbelovende resultaten oplevert. De deelvragen kunnen als volgt beantwoord worden:

Welke barrières (cognitief, technisch of didactisch) ervaren trainers en studenten bij de huidige, handmatige observatiemethodes?

Deze vraag werd binnen deze proef niet expliciet onderzocht via een nieuw, formeel gebruikersonderzoek. Desondanks konden de inherente barrières worden geïdentificeerd. Dit gebeurde op basis van een combinatie van de initiële probleemstelling, een grondige literatuurstudie (Sectie 2.1) en inzichten verkregen uit overleg met de co-promotor. Deze laatste is zelf een ervaren docent in de verpleegkunde en is de opdrachtgever van dit onderzoek. De voornaamste barrières die hieruit naar voor kwamen, waren de subjectiviteit in de beoordeling, de tijdrovende

aard van zelfrapportage en directe, observatie evenals het gebrek aan objectieve data over kijkgedrag.

Welke kenmerken moet een geautomatiseerde analysemethode hebben om de huidige beperkingen van handmatige observatie te verhelpen?

In een ideale wereld zou een volledig geautomatiseerde analysemethode in staat zijn de eyetrackingopnames van studenten volledig te analyseren zonder enige menselijke tussenkomst. Echter, gezien de huidige stand van de technologie, is dit nog niet haalbaar. Daarom werd er gekozen voor een semi-automatische aanpak, waarbij de trainer nog steeds een actieve rol speelt in het initialiseren van de analyse. In de ontwikkelde pipeline voert de trainer de 'kalibratiestap' uit, waarbij de kritische objecten worden gelabeld. De ontwikkelde methode omvat daarom als ondersteunend kenmerk, een softwareapplicatie (Hoofdstuk 5) met een gebruiksvriendelijke interface. Deze stelt trainers in staat eyetrackingopnames te beheren, objecten te definiëren en deze efficiënt te labelen. Aangezien computer-vision een veld is dat voortdurend evolueert, was het ook belangrijk dat de ontwikkelde software voortdurend kon bijgestuurd worden. De software is daarom modulair opgebouwd (volgens Sectie 5.3.1), zodat nieuwe componenten gemakkelijk kunnen worden toegevoegd.

In welke mate kunnen de modellen en de ontwikkelde software:

1. *correct bepalen welke kritische objecten studenten hebben waargenomen?*
2. *nauwkeurig meten hoe lang studenten naar deze objecten keken?*

De beste resultaten werden behaald door de combinatie van een YOLOv11-objectdetector getraind op 1000 samples per klasse, gecombineerd met FastSAM-tracking (zie Sectie 8.5.3). Deze combinatie behaalde een micro-gemiddelde F1-score van 0.8017, met een precisie van 0.9424 en een recall van 0.6976. Dit duidt erop dat de software met hoge precisie objecten kan identificeren (weinig fout-positieven), maar dat er nog ruimte voor verbetering is in de recall (het detecteren van alle daadwerkelijk bekeken objecten). De analyse van vals-positieven toonde aan dat een aanzienlijk deel hiervan correct gedetecteerde objecten waren die niet in de grondwaarheid voorkwamen. Dit maakt de werkelijke precisie mogelijk hoger.

Hoewel de micro-gemiddelden een goed overzicht geven van de prestaties van het model, hangen de resultaten in de praktijk sterk af van de aard van de kritische objecten. Zo bleek het model het bijzonder moeilijk te hebben met het detecteren van kleine objecten (ampule, snoepje, spuit), zeker wanneer ze een laag contrast hebben met hun omgeving. Dit leidde tot een lage recall voor deze objecten. Ook werden er problemen vastgesteld bij het detecteren van objecten die meerdere componenten bevatten, zoals het infuus. Wanneer men wil weten of een student naar een specifiek deel van een object heeft gekeken, zoals de infuuszak,

zal het model dit object ook detecteren wanneer er gekeken wordt naar een ander deel, bijvoorbeeld de infuuspaal. Dit kan, afhankelijk van de context, gezien worden als een vals-positief wat de precisie van het model verlaagt voor dat specifieke object.

De nauwkeurigheid van de duurmeting (2) is direct afhankelijk van de frame-per-frame correctheid van de objectidentificatie (1). Waar objecten correct worden geïdentificeerd, kan de duur accuraat worden afgeleid door het aantal frames te tellen. Fouten in identificatie (vals-negatieven of -positieven) leiden echter direct tot onnauwkeurigheden in de duurmeting.

9.2. Bijdrage en Meerwaarde

De voornaamste bijdrage van deze bachelorproef ligt in de ontwikkeling van een werkend Proof-of-Concept. Deze demonstreert de haalbaarheid van het geautomatiseerd analyseren van eyetrackingdata in een dynamische omgeving zoals het Zorglab. Het was de bedoeling om een platform te creëren dat gemakkelijk uit te breiden en te onderhouden valt, zodat toekomstige onderzoekers en ontwikkelaars deze basis kunnen gebruiken voor verder onderzoek. De methodologie voor het creëren van een grondwaarheidsdataset en de evaluatie van de analysepipeline levert een referentie voor toekomstige projecten binnen dit domein.

Hoewel het hier niet gaat om productieklare software die direct in het Zorglab kan worden ingezet, zet dit onderzoek wel een grote stap richting een geautomatiseerde evaluatiemethode. In de toekomst zullen trainers en studenten kunnen profiteren van een meer doelgerichte en objectieve evaluatie van observatievaardigheden. Het opleiden van elke student vereist een unieke aanpak, wat momenteel een hoge werkdruk met zich meebrengt voor docenten. De geautomatiseerde evaluatie kan deze werkdruk verlichten door objectieve data te leveren die gericht zijn op de specifieke vaardigheden (of blinde vlekken) van elke student. Ook de maatschappij heeft voordeel bij een hogere zorgkwaliteit, door studenten beter voor te bereiden op de praktijk. Deze bachelorproef legt de fundering voor een toekomst waarin technologie en zorgonderwijs hand in hand gaan, en waar de hierboven vernoemde voordelen gerealiseerd kunnen worden.

9.3. Reflectie en Discussie

De behaalde resultaten zijn veelbelovend, met name de hoge precisie (circa 0.94, en in de werkelijkheid hoger) van het beste model. Een hoog aantal vals-positieven zou echter de bruikbaarheid van de software in het gedrang kunnen brengen, door een inaccuraat beeld te schetsen van de werkelijk bekeken objecten. De lagere recall (circa 0.70) geeft aan dat niet alle bekeken objecten consistent werden gedetecteerd en dat er een marge is voor verbetering. Dit was deels te verwachten,

gezien de complexiteit van de taak (variërende objectgroottes, belichting, occlusie, snelle hoofdbewegingen).

Een onverwachte conclusie was dat er praktisch geen verschil was in de prestaties van de verschillende YOLOv11-objectdetectors, ondanks de variërende datasetgroottes. Dit gaf een indicatie dat de segmentatie van FastSAM de beperkende factor was, en niet de objectdetectie op zich. Om een beter beeld te krijgen van de impact van FastSAM, zou het interessant zijn om bij de vals-negatieven te onderzoeken of deze objecten toch gedetecteerd werden door de YOLOv11-objectdetector, maar niet door FastSAM. Daarnaast heeft FastSAM twee parameters die in dit onderzoek niet werden geoptimaliseerd: `iou` en `conf`. Een hogere `iou` en een lagere `conf` zouden kunnen leiden tot meer getrackte objecten, waardoor de recall eventueel kan worden verhoogd. Langs de andere kant is er een kans op een lagere precisie, omdat er mogelijk meer fout-positieven worden gegenereerd.

De analyse van vals-positieven en -negatieven bracht interessante inzichten. Veel zogenaamde vals-positieven bleken toch correcte detecties te zijn die door inconsistenties tussen de FastSAM-output en de grondwaarheid niet als dusdanig werden geregistreerd. Dit wijst op een potentieel probleem in de methodologie voor het creëren van de grondwaarheid, waarbij 'bekeken' objecten werden geselecteerd op basis van de overlap tussen het blikpunt en de handmatig gelabelde segmentaties.

Een andere invalshoek is dat er potentieel een probleem is met de definitie van 'bekeken' objecten. Deze definitie van 'overlapping', creëert echter een grijze zone. Het is denkbaar dat een student visuele informatie van een object verwerkt, zelfs als het precieze blikpunt net buiten de grenzen van het gedetecteerde segment valt; bijvoorbeeld door perifeer zicht of een lichte onnauwkeurigheid in de eyetracker die de gedefinieerde marge overstijgt. De huidige binaire aanpak (wel/niet bekeken) kan deze nuances moeilijk vatten en leidt mogelijk tot een onderschatting van daadwerkelijk geobserveerde objecten. Toekomstig onderzoek zou kunnen overwegen om deze definitie te herzien, bijvoorbeeld door te werken met de afstand tussen het blikpunt en de segmentatie in plaats van een binaire overlap.

9.3.1. Toekomstig Onderzoek en Aanbevelingen

Het is belangrijk te benadrukken dat deze resultaten geen volledig beeld geven hoe het systeem zal presteren onder alle mogelijke praktijkomstandigheden of met een ongelimiteerde variëteit aan objecten. Het experiment dat uitgevoerd werd in het Zorglab was geen naturalistische setting, maar een gecontroleerde omgeving met een beperkt aantal objecten. Om de relevantie van het systeem verder te valideren, is het belangrijk om tests uit te voeren in een realistische setting, waarbij studenten in een echte, relevante zorgsituatie worden geplaatst. Bovendien werd bij de evaluatie van de analysemethode en het trainen van de modellen

uitsluitend gebruik gemaakt van de kalibratieopname waarbij de objecten zich tegen dezelfde achtergrond bevonden als tijdens de evaluatieopnames. Hoewel er ook een kalibratieopname met een afwijkende achtergrond werd gecreëerd, viel het analyseren van de impact hiervan buiten de scope van deze proef. Toekomstig onderzoek zou zich kunnen richten op het expliciet trainen en testen van de modellen met variërende achtergronden om de robuustheid van het systeem tegen contextuele veranderingen te beoordelen en te verbeteren.

Daarnaast is de ontwikkelde analysemethode, hoewel veelbelovend, momenteel redelijk complex. De combinatie van FastSAM en YOLOv11 is niet vanzelfsprekend en zorgt voor veel potentiële foutenbronnen. Toekomstige modellen kunnen eventueel gebruik maken van een enkele, end-to-end benadering die zowel objectdetectie als segmentatie in één stap uitvoert. Hoewel YOLOv11 ook tracking ondersteunt, is het nog niet mogelijk om segmentaties te verkrijgen binnen een trackingopdracht. Daardoor kan de overlap tussen het blikpunt en het object niet worden berekend.

Een andere interessante overweging is de beschikbaarheid van een grafische kaart (GPU) in het Zorglab. Momenteel beschikt het Zorglab niet over een GPU, waardoor de ontwikkelde software niet kan worden gebruikt. Bij het Zorglab worden niet enkel computervisie-modellen onderzocht, maar ook andere AI-toepassingen zoals spraakherkenning. Ook deze toepassingen zouden kunnen profiteren van de rekenkracht van een sterke GPU. De GPU die gebruikt werd in dit onderzoek, was een NVIDIA RTX 4090, wat snelle iteratie mogelijk maakte. Met tragere GPU's zou het trainen van de modellen en het uitvoeren van de analyses veel langer duren. Aangezien AI-toepassingen steeds sterker worden en meer toepassingen bieden, is het aan te raden om in de toekomst te investeren in een krachtige GPU.

Op vlak van hardware is het ook belangrijk om stil te staan bij de resolutie van de camera van de eyetracker. Bij de analyse werd vastgesteld dat het model moeite had met het detecteren van kleine objecten. Het kan dus interessant zijn om de evolutie op de markt van de eyetrackers nauwlettend op te volgen. Hierbij kan men kijken naar hogere resoluties of betere camera's die mogelijk meer details kunnen vastleggen. Een andere optie voor het verbeteren van beeldkwaliteit is het toepassen van motion-deblurring technieken, die in eerdere onderzoeken al potentieel toonden (Cederin & Bremberg, 2023).

Zoals eerder vermeld, is de ontwikkelde software modulair opgebouwd, zodat nieuwe componenten gemakkelijk kunnen worden toegevoegd. Studenten kunnen bij volgende bachelorproeven baat hebben bij de ontwikkelde labeling tool, om zo meer tijd te kunnen besteden aan het ontwikkelen van nieuwe analysecomponenten. Indien deze componenten goed presteren, kunnen ze worden toegevoegd aan de bestaande software. Hierbij dient een analysemodule te worden ontwikkeld waarbij een trainer eyetrackingopnames kan selecteren en eventueel ver-

schillende analysemethoden kan combineren. Een voorbeeld hiervan is een analysemethode voor gezichtsherkenning, die kan bepalen of een student naar een specifieke persoon kijkt. Tenslotte beschikt de software momenteel niet over een visualisatiemodule die de resultaten van de analyse(s) op een gebruiksvriendelijke manier presenteert.

A

Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

Samenvatting

Een goed observatievermogen is belangrijk voor zorgverleners om nauwkeurige diagnoses te kunnen stellen en passende zorgplannen op te stellen. In het 360° Zorglab van HOGENT worden studenten via simulaties getraind met behulp van Tobii Glasses, die oogbewegingen registreren. Ondanks de waardevolle data die deze eyetrackingtechnologie genereert, ontbreekt er software om de verzamelde videodata te analyseren en visualiseren. Dit onderzoek richt zich op het ontwikkelen van een proof-of-concept softwareoplossing die objectherkenning en segmentatiemodellen integreert met de data van Tobii Glasses. Het onderzoek wordt uitgevoerd volgens een agile methodologie, waarbij het ontwikkelingsproces is opgedeeld in iteratieve sprints. Door toepassing van computer vision-technieken zoals YOLOv8, Grounding DINO en Segment Anything Model, wordt in elke sprint stap voor stap een oplossing uitgewerkt die trainers inzicht geeft in welke kritische objecten studenten tijdens simulaties observeren. Het resultaat is een efficiëntere evaluatie en gerichte feedback aan studenten, wat leidt tot verbeterde leerresultaten en tijdbesparing voor trainers. Dit project biedt een duidelijke meerwaarde voor het Zorglab en zet een stap vooruit in het gebruik van eyetracking voor onderwijs.

A.1. Inleiding

Observatievaardigheden van zorgverleners
zijn belangrijk om nauwkeurige diagnoses te stellen en effectieve zorgplannen te

ontwikkelen. In het 360° Zorglab aan HOGENT worden studenten getraind via simulaties waarbij hun oogbewegingen worden geregistreerd met Tobii Glasses. Een belangrijk aspect van deze training is dat studenten leren om kritische objecten, zoals een colafles op het nachtkastje van een diabetespatiënt, op te merken.

Hoewel de Tobii Glasses bruikbare eyetracking data leveren, ontbreekt er momenteel geschikte software om te analyseren of studenten daadwerkelijk naar deze objecten hebben gekeken. Dit gebrek aan dataverwerking en visualisatie maakt het voor trainers lastig om de observatieprestaties van studenten efficiënt te beoordelen en te verbeteren. Zonder een geautomatiseerde manier om te detecteren welke specifieke objecten studenten wel of niet hebben waargenomen, wordt het geven van directe feedback een tijdrovend proces.

Om dit probleem aan te pakken, richt dit bachelorproefonderzoek zich op het beantwoorden van de volgende onderzoeksvraag:

"Hoe kan objectdetectie- en segmentatiesoftware geïntegreerd worden met eyetrackingdata van Tobii Glasses om observatieprestaties van studenten in het 360° Zorglab automatisch te analyseren en te visualiseren?"

Deze onderzoeksvraag wordt uitgewerkt aan de hand van de volgende deelvragen:

1. Welke bestaande objectdetectie en segmentatie modellen zijn hiervoor geschikt?
2. Welke preprocessing- en fine-tuning methoden zijn nodig om Zorglab-specifieke data effectief te gebruiken met deze modellen?
3. Hoe kan een softwareoplossing ontwikkeld worden voor een gebruiksvriendelijke analyse en visualisatie van eyetrackingdata?
4. In welke mate kunnen de modellen en de ontwikkelde software een nauwkeurige evaluatie van kritische objectwaarnemingen garanderen?

Het doel is om trainers in het Zorglab te ondersteunen bij het analyseren en visualiseren van deze data, zodat ze snel inzicht krijgen in de observatieprestaties van studenten en kunnen vaststellen of belangrijke objecten tijdens zorgsimulaties zijn waargenomen.

A.2. Stand van zaken

A.2.1. Bestaande Implementaties

Recente ontwikkelingen in deep learning hebben de toepassingen van eye-tracking aanzienlijk versterkt, vooral op het gebied van objectdetectie in dynamische en complexe omgevingen.

Cho e.a. (2024) introduceerden het ISGOD systeem, dat oogbewegingen en objectdetectie integreert voor kwaliteitsinspectie in productieomgevingen, waarbij real-time analyse mogelijk gemaakt wordt ondanks variabele posities en bewegingen.

Cederin en Bremberg (2023) onderzochten automatische objectdetectie en tracking in eye tracking analyses en verbeterden de nauwkeurigheid door motion deblurring technieken toe te passen, zoals DeblurGAN-v2 gecombineerd met objectdetectoren en trackers.

Daarnaast combineerde Kulyk (2023) objectdetectie met eye-tracking data in een virtuele kunsttentoonstelling om bezoekersinteresses en visuele aandachtspunten te identificeren.

A.2.2. Machine-learning Modellen

Naast geïntegreerde eye-tracking systemen, maken de onderzochte studies gebruik van diverse objectdetectiemodellen.

Kulyk (2023) maakte gebruik van het Faster R-CNN netwerk, een Convolutioneel Neuronaal Netwerk (CNN) dat bekend staat om zijn hoge nauwkeurigheid bij objectdetectie.

Cederin en Bremberg (2023) breidden hun onderzoek uit door naast Faster R-CNN ook andere CNN-gebaseerde architecturen zoals Feature Pyramid Network (FPN), Spatial Pyramid Pooling Network (SPP-Net), You Only Look Once (YOLO) en Single Shot MultiBox Detector (SSD) te evalueren. Ze onderzochten ook transformer gebaseerde modellen zoals DEtection TRansformer (DETR) en DINO, die recentelijk aanzienlijke verbeteringen hebben laten zien in het omgaan met complexe en dynamische scènes.

Liu e.a. (2023) introduceerden in 2023 Grounding DINO, een voorgetraind model dat objecten in afbeeldingen kan detecteren op basis van tekstprompts. Dit model zou eventueel objecten binnen het Zorglab kunnen detecteren zonder fine-tuning op specifieke data, wat de gebruiksvriendelijkheid van de software zou verbeteren.

Dit bachelorproefonderzoek zal verder gaan dan alleen objectdetectie door ook segmentatiemodellen te verkennen die mogelijk via finetuning en tekstprompting specifiek kunnen worden aangepast aan de use case van het 360° Zorglab.

Een veelbelovend model hiervoor is Meta's
segment Anything Model 2 (Ravi e.a., 2024),
dat in de medische context succesvol is toegepast door Zhu e.a. (2024) binnen Medical SAM 2 (MedSAM-2) voor zowel 2D als 3D medische beeldsegmentatie via één enkele prompt.

Wang e.a. (2023) ontwikkelden GazeSAM, dat eye-tracking data gebruikt als input-prompt voor SAM om real-time segmentatiemasks te genereren.

Daarnaast biedt het state-of-the-art werk van Bagchi e.a. (2024) met het ReferEverything framework een model voor het segmenteren van concepten in videodata via natuurlijke taal-
beschrijvingen, wat relevant kan zijn voor het verwerken van de dynamische videodata van Tobii Glasses. Momenteel is de code van ReferEverything nog niet beschikbaar, maar de onderzoekers hebben aangegeven dat deze binnenkort open-source zal worden gemaakt.

A.3. Methodologie

De bachelorproef zal worden uitgevoerd
volgens een agile aanpak, waarbij iteratieve cycli (sprints) en overlappende fasen worden gebruikt om flexibiliteit en continue verbetering mogelijk te maken. Deze aanpak zorgt ervoor dat verschillende onderdelen van het project parallel kunnen verlopen en snel kunnen worden aangepast op basis van tussentijdse bevindingen. Nieuwe taken worden doorheen de sprints toegevoegd aan een backlog binnen een Trello-bord, en worden doorheen de sprints opgepakt en afgerond.

Het doel voor de PoC is om een user-flow te ontwikkelen die het mogelijk maakt om de volgende zaken uit te voeren:

- 1. Het ophalen van eyetrackingdata van de Tobii Glasses
- 2. Het definiëren van kritische objecten die geobserveerd moeten worden.
- 3. Het analyseren van van de data met
  objectdetectie- en segmentatiemodellen.
  Deze stap wordt hierna de data-architectuur genoemd.
- 4. Het al dan niet manueel wegfilteren van vals-positieve objectdetecties.
- 5. Het visualiseren van de resultaten van de analyse via een metriek die de blik-punten van de studenten koppelt aan de gedetecteerde objecten.

De voorkeur gaat naar het gebruik van voorgetrainde modellen om de nood aan hertrainen te minimaliseren en zo de gebruikerservaring te verbeteren. Indien nodig kan de PoC uitgebreid worden met extra functionaliteiten zoals het finetunen van modellen met simulatiespecifieke data.

A.4. Tijdsplanning

De bachelorproef is gepland van 10 februari 2025 tot en met 23 mei 2025, met een totaal van 7 sprints van 2 weken. De activiteiten binnen elke sprint omvatten, in geen specifieke volgorde:

- Literatuurstudie
- Dataverzameling en labeling
- Modelselectie en implementatie
- Schrijven van Python Notebooks voor experimenten
- Ontwikkelen van de data-architectuur binnen de PoC
- Ontwikkelen van de user interface voor de PoC
- Uitbreiding van de PoC met nieuwe functionaliteiten
- Unit-testen schrijven voor de PoC
- Meetings met de co-promotor voor feedback en verfijning van de oplossing
- Documentatie van het proces en resultaten
- Uitschrijven, aanpassen van het proefschrift

De onderstaande tijdsplanning geeft een overzicht van de beoogde deliverables en mijlpalen per sprint.

- **Sprint 1**
  - Relevante modellen zijn verzameld en uitgetest binnen een Python Notebook.
  - Er zijn huisgemaakte data verzameld via geleende Tobii Glasses uit het Zorglab.
  - Potentiële segmentatie- en detectiepipeline architecturen voor de PoC zijn uitgestippeld.
  - Er is een user interface binnen de PoC om opnames op te halen van de Tobii Glasses.
- **Sprint 2**
  - Er is zorglab-specifieke data aangevraagd.
  - De abstract en inleiding van het proefschrift zijn geschreven.
  - Het literatuurstudie onderdeel van het proefschrift is uitgewerkt.

- **Sprint 3**
  - Er is een demo gegeven aan de co-promotor voor een kandidaat architectuur van de PoC.
  - De feedback van de co-promotor is verwerkt en gedocumenteerd.
  - Er is een metriek ontwikkeld voor het meten van waar een student naar heeft gekeken en voor hoe lang.
  - Er zijn verschillende data-architecturen geëvalueerd.
- **Sprint 4**
  - De segmentatie- en detectiepipeline is geïntegreerd binnen de user interface van de PoC
  - De vergelijking tussen verschillende data-architecturen is toegevoegd aan het proefschrift.
  - Er is een meeting gebeurd met de co-promotor voor feedback op de PoC.
- **Sprint 5**
  - De objectdetectie- en segmentatiemodellen zijn geïntegreerd binnen de PoC.
  - Zorglab-specifieke data is gelabeld en gebruikt voor het valideren van de PoC.
- **Sprint 6**
  - De visualisatie-interface is ontwikkeld en geïntegreerd binnen de PoC.
  - De gebruiker kan nieuwe video's analyseren en de resultaten bekijken.
  - De PoC is gevalideerd in het Zorglab.
- **Sprint 7**
  - Het proefschrift is gefinaliseerd en ingediend.

A.4.1. Tools en Technologieën

- **Programmeertaal en Frameworks**: Python, PyTorch, TensorFlow voor machine-learning modellering en implementatie.
- **Programmeerstack van de PoC**: FastAPI voor de backend, HTMX en Jinja2 voor de frontend.

- **Data Verwerking en Visualisatie**: OpenCV voor videoverwerking, Matplotlib voor datavisualisatie.
- **Ontwikkelomgeving**: Visual Studio Code,
  Python Notebooks voor experimenten, Git voor versiebeheer. Poetry voor dependency management.
- **GPU**: At-home Nvidia RTX 4090 en eventuele GPUs van HoGent.
- **Eyetracking**: Tobii Eyetracking Glasses, en Tobii Pro Glasses 3 Controller App.
- **Planning**: Trello voor projectmanagement.

A.5. Verwacht resultaat

Het verwachte resultaat van dit bachelorproefonderzoek is de ontwikkeling van een functionele PoC software die objectherkennings- en segmentatiemodellen integreert met de videodata van de Tobii Eyetracking Glasses. Deze software zal in staat zijn om nauwkeurig te bepalen welke kritische objecten door studenten zijn waargenomen tijdens simulaties, ondersteund door visuele representaties van de oogbewegingen. Verwacht wordt dat modellen na fine-tuning met Zorglab-specifieke data, een hoge detectienauwkeurigheid zullen bereiken.

Trainers en lesgevers in het 360° Zorglab, zullen baat hebben bij een efficiëntere en gedetailleerdere evaluatie van de observatieprestaties van studenten. De ontwikkelde software biedt een meerwaarde door het mogelijk te maken gerichte feedback te geven op specifieke waarnemingen, waardoor het leerproces wordt geoptimaliseerd. Bovendien draagt de PoC bij aan de verdere automatisering van het beoordelingsproces, wat leidt tot tijdbesparing. Het onderzoek zal ook inzicht bieden in de effectiviteit van verschillende machine-learning modellen binnen de context van eyetrackingdata, wat kennis oplevert voor toekomstige toepassingen en verbeteringen in het Zorglab.

Bibliografie

Bagchi, A., Bao, Z., Wang, Y.-X., Tokmakov, P. & Hebert, M. (2024, oktober 30). *ReferEverything: Towards Segmenting Everything We Can Speak of in Videos*. arXiv: 2410.23287 [cs.CV]. Verkregen 15 november 2024, van https://arxiv.org/abs/2410.23287

Buslaev, A., Iglovikov, V. I., Khvedchenya, E., Parinov, A., Druzhinin, M. & Kalinin, A. A. (2018). Albumentations: Fast and Flexible Image Augmentations. *Information*, *11*(2). https://doi.org/10.3390/info11020125

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A. & Zagoruyko, S. (2020). *End-to-End Object Detection with Transformers*. arXiv: 2005.12872 [cs.CV]. Verkregen 2 mei 2025, van https://arxiv.org/abs/2005.12872

Cederin, L. & Bremberg, U. (2023). *Automatic object detection and tracking for eye-tracking analysis* (Bachelor's thesis). Uppsala University. https://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-504416

Cho, S.-W., Lim, Y.-H., Seo, K.-M. & Kim, J. (2024). Integration of eye-tracking and object detection in a deep learning system for quality inspection analysis. *Journal of Computational Design and Engineering*, *11*(3), 158–173. https://doi.org/10.1093/jcde/qwae042

Clarke, A. D. F., Mahon, A., Irvine, A. & Hunt, A. R. (2017). People Are Unable to Recognize or Report on Their Own Eye Movements [PMID: 27595318]. *Quarterly Journal of Experimental Psychology*, *70*(11), 2251–2270. https://doi.org/10.1080/17470218.2016.1231208

Geng, C., Huang, S.-J. & Chen, S. (2018). Recent Advances in Open Set Recognition: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *43*(10), 3614–3631. https://doi.org/10.1109/tpami.2020.2981604

Girshick, R., Donahue, J., Darrell, T. & Malik, J. (2014). *Rich feature hierarchies for accurate object detection and semantic segmentation*. arXiv: 1311.2524 [cs.CV]. Verkregen 2 mei 2025, van https://arxiv.org/abs/1311.2524

Hafiz, A. M. & Bhat, G. M. (2020). A survey on instance segmentation: state of the art. *International Journal of Multimedia Information Retrieval*, *9*(3), 171–189. https://doi.org/10.1007/s13735-020-00195-x

He, K., Gkioxari, G., Dollár, P. & Girshick, R. (2018). *Mask R-CNN*. arXiv: 1703.06870 [cs.CV]. Verkregen 2 mei 2025, van https://arxiv.org/abs/1703.06870

heyoeyo. (2024, augustus 13). *Comment on "Are there any method for reducing gpu memory overhead?"* [Comment on GitHub issue]. Verkregen 20 februari 2025, van https://github.com/facebookresearch/sam2/issues/196

Hu, R., Niels, R., Khedr, H., Arun, CharlesCNorton, Bronchart, I., Ryali, C., Rädle, R., Dymchenko, S., Danwand, Garcia, D. & jhj0517. (2025, maart 16). *ilianbronchart/sam2*. Verkregen 16 maart 2025, van https://github.com/ilianbronchart/sam2

Hu, R., Niels, R., Khedr, H., Arun, CharlesCNorton, Ryali, C., Rädle, R., Dymchenko, S., Danwand, Garcia, D. & jhj0517. (2024, december 25). *facebookresearch/sam2*. Verkregen 20 februari 2025, van https://github.com/facebookresearch/sam2

Huang, H., Wang, B., Xiao, J. & Zhu, T. (2024). Improved small-object detection using YOLOv8: A comparative study. *Applied and Computational Engineering*, *41*, 80–88. https://doi.org/10.54254/2755-2721/41/20230714

Khanam, R. & Hussain, M. (2024). *YOLOv11: An Overview of the Key Architectural Enhancements*. arXiv: 2410.17725 [cs.CV]. Verkregen 2 mei 2025, van https://arxiv.org/abs/2410.17725

Kirillov, A., He, K., Girshick, R., Rother, C. & Dollár, P. (2019). *Panoptic Segmentation*. arXiv: 1801.00868 [cs.CV]. Verkregen 2 mei 2025, van https://arxiv.org/abs/1801.00868

Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., Dollár, P. & Girshick, R. (2023). *Segment Anything*. arXiv: 2304.02643 [cs.CV]. Verkregen 2 mei 2025, van https://arxiv.org/abs/2304.02643

Kulyk, D. (2023, juli). *Combining object detection and eye tracking to identify points of interest for VR art exhibition visitors*. Verkregen 2 mei 2025, van http://essay.utwente.nl/95999/

Lindeberg, T. (2012). Scale Invariant Feature Transform. *Scholarpedia*, *7*(5). https://doi.org/10.4249/scholarpedia.10491

Liu, S., Zeng, Z., Ren, T., Li, F., Zhang, H., Yang, J., Jiang, Q., Li, C., Yang, J., Su, H., Zhu, J. & Zhang, L. (2023). *Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection*. arXiv: 2303.05499 [cs.CV]. Verkregen 2 mei 2025, van https://arxiv.org/abs/2303.05499

Oquab, M., Darcet, T., Moutakanni, T., Vo, H., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, D., Massa, F., El-Nouby, A., Assran, M., Ballas, N., Galuba, W., Howes, R., Huang, P.-Y., Li, S.-W., Misra, I., Rabbat, M., Sharma, V., … Bojanowski, P. (2024). *DINOv2: Learning Robust Visual Features without Supervision*. arXiv: 2304.07193 [cs.CV]. Verkregen 2 mei 2025, van https://arxiv.org/abs/2304.07193

Pauszek, J. R. (2023). An introduction to eye tracking in human factors healthcare research and medical device testing. *Human Factors in Healthcare*, *3*. https://doi.org/10.1016/j.hfh.2022.100031

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G. & Sutskever, I. (2021). *Learning Transferable Visual Models From Natural Language Supervision*. arXiv: 2103.00020 [cs.CV]. Verkregen 2 mei 2025, van https://arxiv.org/abs/2103.00020

Ravi, N., Gabeur, V., Hu, Y.-T., Hu, R., Ryali, C., Ma, T., Khedr, H., Rädle, R., Rolland, C., Gustafson, L., Mintun, E., Pan, J., Alwala, K. V., Carion, N., Wu, C.-Y., Girshick, R., Dollár, P. & Feichtenhofer, C. (2024). *SAM 2: Segment Anything in Images and Videos*. arXiv: 2408.00714 [cs.CV]. Verkregen 2 mei 2025, van https://arxiv.org/abs/2408.00714

Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. arXiv: 1506.02640 [cs.CV]. Verkregen 2 mei 2025, van https://arxiv.org/abs/1506.02640

Remington, L. A. (2012). Chapter 4 - Retina. In L. A. Remington (Red.), *Clinical Anatomy and Physiology of the Visual System (Third Edition)* (Third Edition, pp. 61–92). Butterworth-Heinemann. https://doi.org/10.1016/b978-1-4377-1926-0.10004-9

Rey-Otero, I., Morel, J.-M. & Delbracio, M. (2015). *An analysis of the factors affecting keypoint stability in scale-space*. arXiv: 1511.08478 [cs.CV]. Verkregen 2 mei 2025, van https://arxiv.org/abs/1511.08478

Rublee, E., Rabaud, V., Konolige, K. & Bradski, G. (2011). ORB: an efficient alternative to SIFT or SURF. *Proceedings of the IEEE International Conference on Computer Vision*, 2564–2571. https://doi.org/10.1109/iccv.2011.6126544

Son, J. & Jung, H. (2024). Teacher–Student Model Using Grounding DINO and You Only Look Once for Multi-Sensor-Based Object Detection. *Applied Sciences*, *14*(6). https://doi.org/10.3390/app14062232

Tobii AB. (2023, april). *Tobii Pro Glasses 3 Developer Guide*. Verkregen 13 mei 2025, van https://go.tobii.com/tobii-pro-glasses-3-developer-guide

Tobii AB. (2025a). *Tobii Pro Glasses 3* [Productpagina van Tobii Pro Glasses 3]. Verkregen 2 mei 2025, van https://www.tobii.com/products/eye-trackers/wearables/tobii-pro-glasses-3

Tobii AB. (2025b). *Understanding Tobii Pro Lab Eye Tracking Metrics* [Uitleg over eyetracking-metrics in Tobii Pro Lab]. Verkregen 2 mei 2025, van https://connect.tobii.com/s/article/understanding-tobii-pro-lab-eye-tracking-metrics

Tobii AB. (2025c). *Visualizations for Tobii Pro Lab* [Overzicht van visualisatiemogelijkheden in Tobii Pro Lab]. Verkregen 2 mei 2025, van https://connect.tobii.com/s/article/Visualizations-for-Tobii-Pro-Lab

Wang, B., Aboah, A., Zhang, Z. & Bagci, U. (2023). *GazeSAM: What You See is What You Segment*. arXiv: 2304.13844 [cs.CV]. Verkregen 2 mei 2025, van https://arxiv.org/abs/2304.13844

Zhang, H., Li, F., Liu, S., Zhang, L., Su, H., Zhu, J., Ni, L. M. & Shum, H.-Y. (2022). *DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection*. arXiv: 2203.03605 [cs.CV]. Verkregen 2 mei 2025, van https://arxiv.org/abs/2203.03605

Zhao, X., Ding, W., An, Y., Du, Y., Yu, T., Li, M., Tang, M. & Wang, J. (2023). *Fast Segment Anything*. arXiv: 2306.12156 [cs.CV]. Verkregen 2 mei 2025, van https://arxiv.org/abs/2306.12156

Zhu, J., Qi, Y. & Wu, J. (2024, augustus 1). *Medical SAM 2: Segment medical images as video via Segment Anything Model 2*. arXiv: 2408.00874 [cs.CV]. Verkregen 15 november 2024, van https://arxiv.org/abs/2408.00874