



# MULTIMEDIA : STAY ON TRAILS



Maarten Dequanter

December 7, 2025

# Contents

<b>1 Inleiding</b>	<b>2</b>
1.1 Gebruikte hardware en software . . . . .	2
<b>2 Methode</b>	<b>2</b>
2.1 Netwerk setup . . . . .	2
2.2 Perceptie en AI-segmentatiemodellen . . . . .	3
2.3 Row-wise midpoint heading computation . . . . .	3
<b>3 Werking van de Stay On Trails-applicatie</b>	<b>4</b>
3.1 Globale architectuur . . . . .	5
3.2 Camera-acquisitie en beeldcompressie . . . . .	5
3.3 WebSocket-communicatie en segmentatieserver . . . . .	5
3.4 Visualisatie en auditieve feedback . . . . .	6
3.5 Instellingen, modelkeuze en recording mode . . . . .	7
3.6 Locatieregistratie en telemetrie . . . . .	8
<b>4 Source code</b>	<b>9</b>
<b>5 Conclusie</b>	<b>10</b>

# Abstract

Binnen dit project voor het vak Multimedia werd een applicatie ontwikkeld die slechtzienden ondersteunt bij het wandelen in een park. De beelden van een camera worden in quasi-realtime verzonden naar een cloudserver, waar een specifiek segmentatiemodel wordt gebruikt om de begaanbare zones op het pad te detecteren en een haptische feedback terug te sturen die aangeeft waar men het best kan wandelen. Het model is in staat om het midden van het pad te bepalen en om hindernissen, zoals objecten of andere wandelaars, op het pad te vermijden.

Daarnaast krijgt de gebruiker visuele en auditieve begeleiding: op het scherm wordt een duidelijke richtingspijl weergegeven, aangevuld met gesproken instructies (links, rechts, recht-door) en meldingen, bijvoorbeeld bij de selectie van het AI-model of het opstarten van de applicatie.

Voor analyse- en evaluatiedoelinden worden diverse statistieken zowel weergegeven als opgeslagen in een CSV-bestand, zoals onder andere latency, type netwerkverbinding (wifi of cellular), gps-locatie en aanverwante parameters. Ten slotte is er een recordingmodus voorzien waarbij op vaste tijdsintervallen beelden worden opgeslagen, zodat er in de cloud later een specifiek model kan worden getraind voor een nieuw wandelpad.

# Acknowledgment

Sommige teksten werden verbeterd qua structuur en leesbaarheid via **ChatGPT (GPT-5, OpenAI)**. Eveneens werd deze gebruikt om makkelijk latex formaten toe te passen.

## 1 Inleiding

Dit project kadert binnen het vak *Transmissietechnieken en multimedia*. De beoordeling zal gebeuren op basis van een kort verslag (zie voorbeeld onder canvas bestanden) en een korte presentatie/demo. Enkele voorbeeld projecten (spotify scrobbler schaak analyse app op basis van een foto/video, morse code versturen met de zaklamp en ontvangen met de camera)

### 1.1 Gebruikte hardware en software

**Hardware** Voor de ontwikkeling van de applicatie werd gebruik gemaakt van een Samsung Galaxy A71 (Model: SM-A715F/DS)

**Software** Android Studio Narwhal 3 Feature Drop — 2025.1.3 on Windows 11 Pro (25H2).

## 2 Methode

### 2.1 Netwerk setup

Op figuur 1 is de volledige netwerksetup weergegeven.

Thuis (in Zellik) draait een segmentatieserver in hetzelfde lokale netwerk (LAN) als de signaling server.

De werking is als volgt. De applicatie *Stay On Trails* stuurt de camerabeelden van de smartphone via wifi of een mobiele (5G-)verbinding naar de signaling server over het internet via NAT (Network Address Translation). Dankzij de juiste firewallregels en port forwarding is deze signaling server vanaf elke locatie bereikbaar via publieke poort 9000.

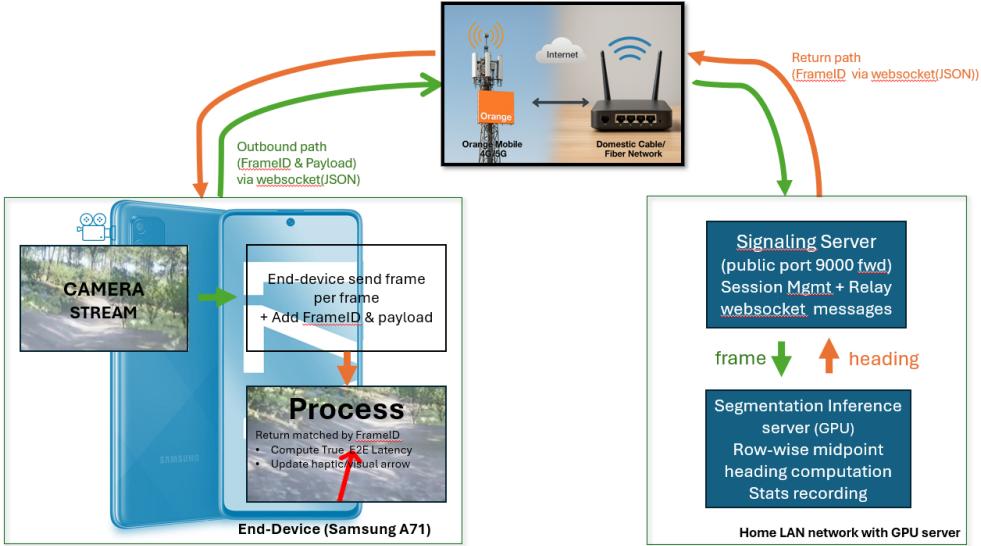


Figure 1: Netwerk setup van smartphone, signaling en inference server.

De signaling server bevindt zich in hetzelfde LAN als de segmentation inference GPU-server, zodat de beelden in quasi real-time door het segmentatiemodel kunnen worden geanalyseerd en de berekende heading (looprichting) terug naar de applicatie kan worden gestuurd.

De applicatie toont deze heading op het scherm met een richtingspijl en spreekt de bijbehorende commando's uit ("links", "rechts", "rechtdoor").

## 2.2 Perceptie en AI-segmentatiemodellen

Voor de patherkennung wordt een Roboflow 3.0 *Instance Segmentation (Fast)*-model gebruikt, gebaseerd op een COCO-compatibele checkpoint (**COCOn-seg**). Dankzij de COCO-pretraining kan het model terugvallen op robuuste, generieke visuele kenmerken, die vervolgens via finetuning zijn aangepast aan een eigen dataset met parkpaden. De gekozen *Fast*-variant is geoptimaliseerd voor lage latency in plaats van maximale nauwkeurigheid, waardoor het model geschikt is voor quasi real-time inferentie op beperkte hardware, zoals een standaard thuis GPU/Game desktop.

## 2.3 Row-wise midpoint heading computation

Op basis van de segmentatiemaskers wordt de looprichting (*heading*) bepaald via een eenvoudige row-wise midpoint-methode. Zie figuur 2. Het binaire mask (pad versus niet-pad) wordt hiervoor horizontaal doorgesneden in meerdere scanlijnen. Voor elke scanlijn wordt de begaanbare zone gedetecteerd en wordt het midden van deze vrije ruimte als midpoint opgeslagen. Door de midpoints van verschillende rijen te middelen, ontstaat een doelpunt dat zich typisch in het centrum van het pad bevindt. Vervolgens wordt de headingvector berekend van het beeldcentrum onderaan naar dit doelpunt en wordt de bijhorende hoek bepaald met de arctangensfunctie. Deze hoek wordt in de applicatie vertaald naar eenvoudige commando's zoals *links*, *rechts* of *rechtdoor*. Deze aanpak is aanzienlijk lichter dan optimalisatie-gebaseerde methoden zoals genetische algoritmen sedighi2004 en is daardoor goed bruikbaar voor real-time sturing in assistieve navigatie.



Figure 2: Visuele feedback via Row-wise midpoint heading computation

### 3 Werking van de Stay On Trails-applicatie



Figure 3: Demo Stay On Trails

De Android-applicatie *Stay On Trails* vormt de clientzijde van het navigatiesysteem voor slechtzienden. De app draait op een smartphone en stuurt camerabeelden in real-time naar een segmentatieserver. Op basis van de door de server teruggestuurde heading geeft de app zowel visuele als auditieve feedback aan de gebruiker en verzendt deze telemetrie voor latentie- en netwerkmeting.

### 3.1 Globale architectuur

De hoofdlogica van de app is geïmplementeerd in de klasse `MainActivity`. Bij het opstarten configueert de app een volledigscherm-interface met drie hoofdelementen:

- een `PreviewView` voor de live camerastream;
- een `ArrowOverlay`-view die de berekende heading als pijl over de video tekent;
- een `TextView` (`info`) die debug- en statusinformatie weergeeft (server, latency, GPS, gekozen model, enz.).

Daarnaast wordt een `WsJpegUploader` geïnitialiseerd, die instaat voor de WebSocket-verbinding met de segmentatieserver, en worden drie `MediaPlayer`-objecten opgezet voor de gesproken richtingscommando's (*links*, *rechts*, *rechtdoor*).

### 3.2 Camera-acquisitie en beeldcompressie

Voor de cameraverwerking maakt de app gebruik van de *CameraX*-bibliotheek. In `startCamera()` wordt een `Preview`-use case gekoppeld aan de `PreviewView`, en een `ImageAnalysis`-use case met resolutie  $640 \times 480$ . De analyzer draait op een afzonderlijke thread en ontvangt frames in het `YUV_420_888`-formaat.

Elk frame wordt als volgt voorbereid voor verzending:

1. De YUV-gegevens worden met de functie `yuv420888ToNv21` omgezet naar een NV21-buffer.
2. De NV21-buffer wordt via `YuvImage` gecomprimeerd naar een JPEG-afbeelding (kwaliteit  $\approx 50\%$ ).
3. De resulterende JPEG-bytearray wordt in `latestJpeg` opgeslagen en doorgegeven aan de uploader.

Het aantal verzonden frames wordt begrensd door de parameter `min_interval_ms` (standaard 200 ms), zodat de bandbreedte en belasting op de server onder controle blijven.

### 3.3 WebSocket-communicatie en segmentatieserver

De klasse `WsJpegUploader` beheert de WebSocket-verbinding met de segmentatieserver (poort 9000). Bij elke oproep van `trySend(...)` gebeurt het volgende:

1. Er wordt gecontroleerd of de minimale tijd sinds het vorige frame (`min_interval_ms`) verstreken is.
2. De uploader verzekert dat de WebSocket-verbinding actief is (`ensureConnected()`).
3. Er wordt een nieuw `frame_id` gegenereerd en een klein JSON-object (`type = "frame_meta"`) verstuurd met onder meer:

- `frame_id` en tijdstempel;
- de huidige netwerkverbinding (*WiFi, Cellular, Other*);
- de laatst gemeten latency en GPS-coördinaten;
- het geselecteerde AI-model of de *recording mode*.

4. Daarna wordt het binaire JPEG-frame verstuurd via dezelfde WebSocket.

De segmentatieserver verwerkt de binnenkomende beelden met het instance-segmentationmodel en voert de *row-wise midpoint heading computation* uit (zie vorige sectie). Het resultaat is een headinghoek in graden, samen met het `frame_id`, die via een JSON-bericht terug naar de app wordt gestuurd.

Bij ontvangst van zo'n bericht wordt in `MainActivity` de callback `onHeading` uitgevoerd. Op basis van het `frame_id` wordt de end-to-end latency berekend door de verzendtijd uit een `pendingFrames`-tabel te halen en te combineren met de huidige tijd.

### 3.4 Visualisatie en auditieve feedback

De heading wordt op twee manieren naar de gebruiker gecommuniceerd:

1. **Visueel:** de custom view `ArrowOverlay` bewaart de laatst ontvangen heading en tekent in `onDraw()` een rode pijl die vertrekt vanuit de onderkant van het scherm naar de doelrichting. Een heading van 90° komt overeen met recht vooruit; waarden kleiner of groter sturen de pijl respectievelijk naar rechts of links.
2. **Auditief:** de functie `headingToDirection()` vertaalt de continue headinghoek naar één van drie discrete richtingen (`LEFT, FORWARD, RIGHT`). Hierbij wordt een tolerantie in graden gebruikt, afgeleid uit de instelling `heading_tolerance_pct`. Alleen wanneer de richting verandert speelt `speakDirectionIfChanged()` het corresponderende audiobestand (`left.mp3, right.mp3, forward.mp3`) af.

De verzamelde statusinformatie (server-URL, laatste heading, latency, GPS-positie, gekozen model, modelconfidence, enz.) wordt door `updateInfoText()` in de `info-TextView` geplaatst. Door op het scherm te tikken kan de gebruiker deze overlay aan- of uitzetten. Zie figuur 4.

- **Inference server :** De cloud server waar de inferentie wordt uitgevoerd
- **Local IP :** Het locale IP van de smartphone
- **Heading :** Het aantal graden dat er moet worden gedraaid om naar het midden van het pad te gaan.
- **Last Send FrameId :** Het ID van het laatst verzonden frame, gebruikt voor de juiste End-to-End Latency te berekenen.
- **Latency :** De End-to-End latency, de tijd tussen het verzenden van het frame en het ontvangen van de corresponderende heading.
- **Loc:** De coördinaten van de huidige locatie volgens geïntegreerde GPS van de smartphone.
- **Tol :** De minimale afwijking van het midden van het pad, vooraleer er een audio feedback wordt gegeven. Dit om auditieve overbelasting te vermijden.

- **Selected Model:** Het huidig geselecteerde segmentatiemodel voor het bepalen van de heading.
- **Model Confidence:** De minimale confidentie voor de detectie van het pad.

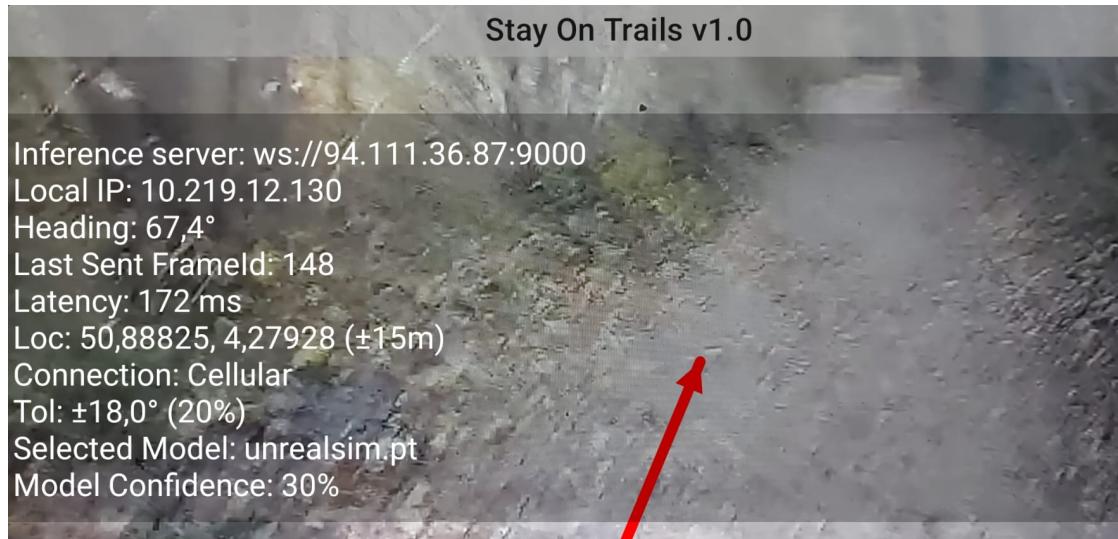


Figure 4: Demo Stay On Trails

### 3.5 Instellingen, modelkeuze en recording mode

De app bevat een aparte `SettingsActivity` met een `PreferenceFragment`, waarmee volgende parameters kunnen worden aangepast, het menu is te zien in figuur 5 :

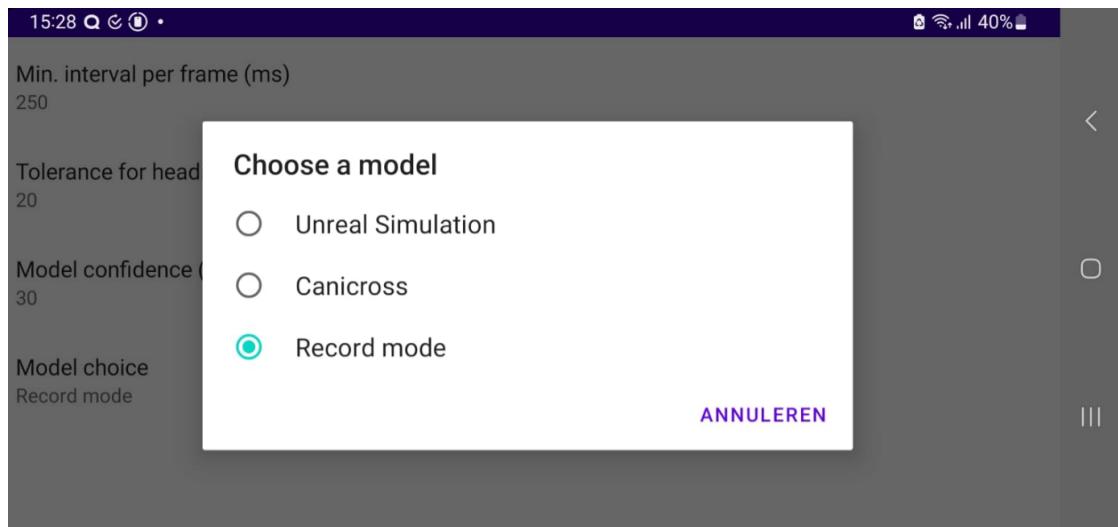


Figure 5: Settings page

- `min_interval_ms`: minimale tijd tussen twee verzonden frames;
- `heading_tolerance_pct`: gevoeligheid voor het onderscheid tussen *links*, *rechtdoor* en *rechts*;
- `model_confidence_pct`: gewenste betrouwbaarheid van het AI-model;
- `model_select`: keuze tussen verschillende segmentatiemodellen (bijv. `unrealsim.pt`, `canicross.pt`) of de *recording mode*.

Via een `SharedPreferences`-listener in `MainActivity` worden wijzigingen in deze instellingen onmiddellijk doorgegeven aan de `WsJpegUploader` (bijvoorbeeld via `updateMinIntervalMs()`, `updateHeadingTolerance()` en `updateModel()`) en, indien nodig, via JSON naar de server gestuurd. Bij het kiezen van bepaalde modellen of de *recording mode* worden bovendien korte audiомeldingen afgespeeld (bijvoorbeeld `unrealsim_model_selected.mp3`, `recordmode.mp3`) zodat de gebruiker weet in welke modus de applicatie zich bevindt.

In de *recording mode* worden de doorgestuurde beelden niet alleen gebruikt voor navigatie, maar ook opgeslagen aan de serverzijde, zodat later een nieuw, pad-specifiek segmentatiemodel kan worden getraind voor dat park.

### 3.6 Locatierегистрация и телеметрия

Met behulp van de *Fused Location Provider* (`FusedLocationProviderClient`) vraagt de app periodiek GPS-updates op. Bij elke update wordt een klein JSON-bericht (`type = "location"`) via de WebSocket verstuurd met latitude, longitude, nauwkeurigheid en tijdstempel. De huidige locatie wordt ook in tekstvorm getoond in de info-overlay. Samen met de latency-per-frame en

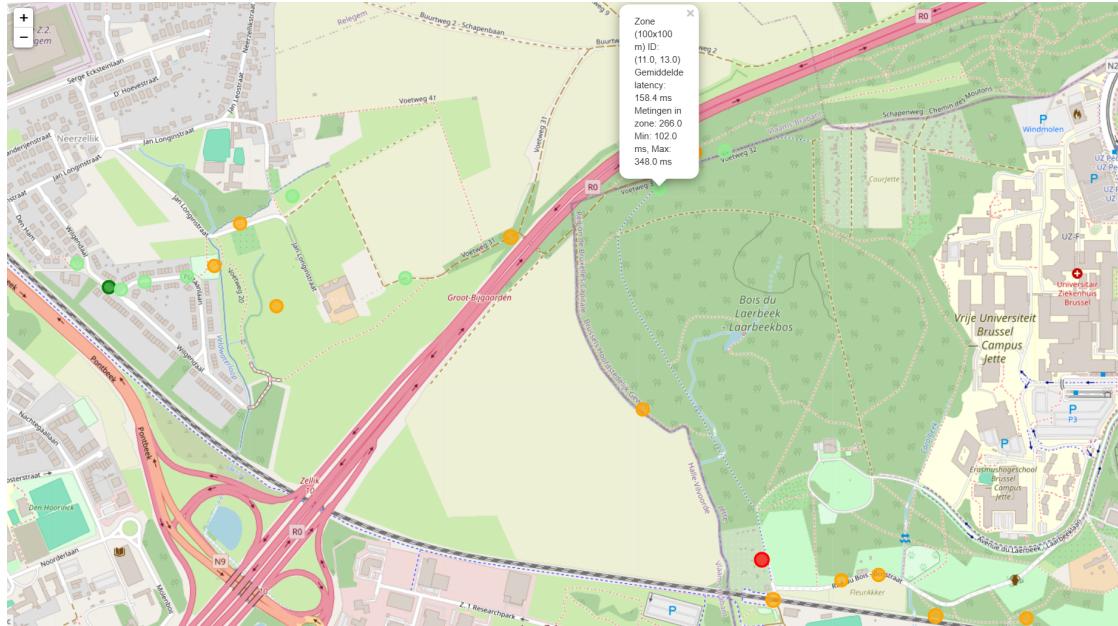


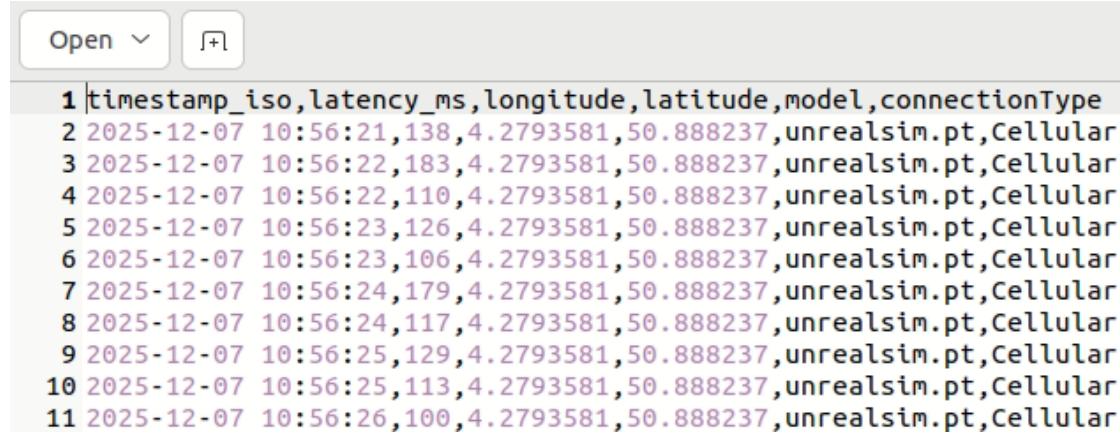
Figure 6: Visualisatie van Latency over een testwandeling naar Laerbeekbos

het type netwerkverbinding (*WiFi* of *Cellular*) levert dit voldoende telemetrie op om de prestaties

van het volledige End-to-End systeem te analyseren, zowel in het labo als in echte parken. Een voorbeeldrapport van een wandeling in Laerbeekbos kan worden gezien in volgende figuur 6. Dit met volgende legende :

- groen: max 140 ms
- lichtgroen : 140–170 ms
- oranje: 170–210 ms
- rood: 210–260 ms

De raw content werd opgeslagen in een CSV bestand op de GPU server zie figuur 7 en eveneens werden de frames opgeslagen in Record modus, zie figuur 8 :



```

1 timestamp_iso,latency_ms,longitude,latitude,model,connectionType
2 2025-12-07 10:56:21,138,4.2793581,50.888237,unrealsim.pt,Cellular
3 2025-12-07 10:56:22,183,4.2793581,50.888237,unrealsim.pt,Cellular
4 2025-12-07 10:56:22,110,4.2793581,50.888237,unrealsim.pt,Cellular
5 2025-12-07 10:56:23,126,4.2793581,50.888237,unrealsim.pt,Cellular
6 2025-12-07 10:56:23,106,4.2793581,50.888237,unrealsim.pt,Cellular
7 2025-12-07 10:56:24,179,4.2793581,50.888237,unrealsim.pt,Cellular
8 2025-12-07 10:56:24,117,4.2793581,50.888237,unrealsim.pt,Cellular
9 2025-12-07 10:56:25,129,4.2793581,50.888237,unrealsim.pt,Cellular
10 2025-12-07 10:56:25,113,4.2793581,50.888237,unrealsim.pt,Cellular
11 2025-12-07 10:56:26,100,4.2793581,50.888237,unrealsim.pt,Cellular

```

Figure 7: CSV values while walking in Laerbeekbos



Figure 8: Saved frames while recording via the app.

## 4 Source code

Bronnen van de applicatie voor Android Studio zijn te vinden op:

- <https://github.com/mdequanter/BlindNavApp>

#### **De bronnen voor de segmentatie en signaling server zijn:**

- Signaling server : <https://github.com/mdequanter/MasterThesis/blob/main/signalingServer.py>
- Inferencing on GPU : <https://github.com/mdequanter/MasterThesis/blob/main/unrealsim/segmentVideoAndroid.py>

## **5 Conclusie**

In dit project werd met *Stay On Trails* een werkende multimediatoeassing gerealiseerd die camerabeelden van een smartphone in quasi real-time naar een segmentatieserver streamt, daar een instance-segmentationmodel uitvoert en de resulterende looprichting terugstuurt naar de gebruiker via visuele en auditieve feedback. Het volledige traject van cameracaptatie en JPEG-compressie over WebSocket-transport tot heading-berekening en haptische/auditieve hints illustreert hoe verschillende multimedia- en netwerkcomponenten kunnen worden gecombineerd tot een concreet, assistief navigatiesysteem voor slechtzienden.

De gekozen architectuur met een lichte Android-client en een zwaardere GPU-server blijkt in de praktijk goed inzetbaar. De row-wise midpoint heading computation levert een robuuste en tegelijk computationally goedkope methode om het pad te bepalen en obstakels te vermijden. Dankzij de instelbare parameters (onder meer *min interval ms*, *heading tolerance* en *model confidence*) kan de gebruiker of begeleider de balans tussen reactietijd, stabiliteit en geluidsbelasting regelen. De recording mode maakt het bovendien mogelijk om nieuwe datasets te verzamelen voor parkspecifieke modellen, wat de nauwkeurigheid in reële situaties verder kan verhogen.

De uitgebreide telemetrie met onder andere end-to-end latency, type netwerkverbinding en GPS-coördinaten toont aan dat de performantie van het systeem betekenisvol kan worden geanalyseerd in functie van de omgeving. Door de metingen te aggregeren per locatie wordt duidelijk waar de latency oploopt en waar de gebruiker dus potentieel minder vloeiende feedback zou ervaren. Wat later van pas zou kunnen worden bij het effectief inzetten van deze applicatie in nieuwe omgevingen.

Samengevat demonstreert *Stay On Trails* dat real-time beeldsegmentatie via een externe server, gecombineerd met slimme visualisatie en auditieve cues, een haalbare en veelbelovende benadering is om slechtzienden veilig op wandelpaden te begeleiden. Tegelijk wijst de analyse van netwerkprestaties en latentieclustering op verdere onderzoekslijnen, zoals het gebruik van lokale inference op het toestel, adaptieve kwaliteitsinstellingen of de integratie met andere sensoren. Daarmee vormt dit project een solide basis voor toekomstige, meer uitgebreide assistieve navigatiesystemen.