

HONOURS PROGRAM: EINDVERSLAG

# Identificatie van kleine watersalamanders op basis van stippattenpatronen



Auteurs: Jesse Daems & Rune De Coninck

Begeleider: Em. prof. dr. Nick Schryvers

ACADEMIEJAREN 2023-2025

# Inhoudsopgave

<b>Acknowledgement</b>	<b>3</b>
<b>1 Achtergrond en motivatie</b>	<b>3</b>
<b>2 Resultaten</b>	<b>4</b>
2.1 Aannames over de foto's . . . . .	4
2.2 Overzicht van de verschillende stappen . . . . .	5
2.3 Pose Estimation . . . . .	6
2.3.1 Gekozen punten . . . . .	6
2.3.2 Training data . . . . .	8
2.4 Isoleren van de buik . . . . .	8
2.4.1 Isoleren van de buik via Pose Estimation . . . . .	10
2.4.2 Isoleren van de buik via Color Segmentation . . . . .	13
2.5 Stipdetectie . . . . .	20
2.5.1 Klassieke methoden . . . . .	20
2.5.2 Haar cascades . . . . .	25
2.6 Rechttrekken stippenpatroon via Pose Estimation . . . . .	27
2.7 Vergelijken van stippenpatronen . . . . .	28
2.7.1 Lijst van coördinaten . . . . .	30
2.7.2 Controle op het aantal stippen . . . . .	30
2.7.3 Selecteren van driehoeken om te matchen . . . . .	31
2.7.4 Matching van de driehoeken . . . . .	33
2.7.5 Verminderen van het aantal valse matches . . . . .	33
2.7.6 Match punten, gebaseerd op de gematchte driehoeken . . . . .	34
2.7.7 Score berekenen tussen twee salamanders . . . . .	35
<b>3 App en Serversoftware</b>	<b>36</b>
3.1 Kwaliteit van de foto . . . . .	36
<b>4 Tegengekomen problemen</b>	<b>37</b>
<b>5 Mogelijke uitbreidingen</b>	<b>37</b>
5.1 Trainable Weka Segmentation . . . . .	38
5.2 Meest prominente stippen . . . . .	38
5.3 Pose Estimation model . . . . .	39
5.4 Quality of Life aanpassingen in de app . . . . .	39
<b>6 Conclusie</b>	<b>39</b>

<b>7 Referenties</b>	<b>40</b>
<b>A Handleiding voor gebruik van de app/website</b>	<b>42</b>
A.1 Instellen . . . . .	43
A.2 Foto indienen . . . . .	44
A.3 Nieuw individu aanmaken/matchen met bestaand individu . . . . .	46

## Acknowledgement

We zouden graag onze begeleider Em. prof. dr. Nick Schryvers bedanken voor al zijn hulp en betrokkenheid tijdens dit project. We vonden het een aangename samenwerking waarbij we veel hebben geleerd.

## 1 Achtergrond en motivatie

Wanneer de kleine watersalamander, wat een amfibie is, wil gaan paren in een voortplantingspoel, moet het vaak een drukke, gevaarlijke weg oversteken. Om te voorkomen dat er veel salamanders sneuvelen, worden er overzetten georganiseerd om de salamanders veilig aan de andere kant van de weg neer te zetten. Dit gebeurt via overzetztploegen, zij nemen emmers, met daarin de amfibieën, mee naar de overkant van de weg om ze daar weer vrij te laten. Op deze manier is er een significante daling in het aantal verkeersdoden. Bij deze overzetten zou het nuttig zijn om bij te houden welke salamanders er zijn en welke er elk jaar terugkeren. Ons project probeert dit op een efficiënte manier in kaart te brengen. De kleine watersalamander kan op een unieke manier geïdentificeerd worden aan de hand van het stippenpatroon, dit kenmerk van de kleine watersalamander proberen wij uit te buiten in dit project om zo salamanderidentificatie mogelijk te maken.

Het doel met dit project is het opleveren van een gebruiksvriendelijke app. De app zal een foto kunnen maken van de buik van de salamander waar de meest duidelijke stippen te zien zijn, en deze matchen met een bestaand individu in de database. Als er geen matches gevonden worden, kan op basis van de genomen foto een nieuw individu geregistreerd worden.

Om dit te bereiken hebben we een reeks technieken verzameld die we willen uitproberen. Het uiteindelijke programma zal deze samenleggen om tot een betrouwbaar resultaat te komen.

Het systeem zal rekening moeten houden met de volgende omstandigheden die de identificatie kunnen bemoeilijken:

- Verschillen in belichting,
- Verschillen in schaal, rotatie en positie,
- Verschillende krommingen van salamanders en
- Variabele resolutie van invoerafbeeldingen.

We kregen honderden voorbeeldfoto's, genomen tijdens overzetmomenten van de laatste jaren. Sommige foto's bleken onbruikbaar te zijn, maar de meerderheid voldeed aan onze vereisten (zie sectie

2.1).

Wat de implementatie betreft hebben we twee delen. Enerzijds is de logica en functionaliteit van het project geïmplementeerd in Python. Dit stuk wordt beschikbaar gemaakt via een RESTful API. Anderzijds is de user interface geïmplementeerd in Dart met het cross-platform framework Flutter, wat ons toelaat om met eenzelfde codebase zowel een app als website te bouwen.

In sectie 3 gaan we verder in op de software.

Voor ons beiden was dit een interessant project. Voor Jesse als informaticus was het waardevol om een dergelijk project op een meer wiskundige en formele manier aan te pakken dan gewoonlijk. Daarnaast was het een goede manier om zijn kennis in softwareontwerp en AI in de praktijk te brengen, en te leren werken met een nieuw frontend-platform. Voor Rune als wiskundige was het project zeer waardevol omdat de wiskunde toegepast kon worden op een probleem in de werkelijkheid. Bovendien kwam er ook veel programmeerwerk bij te kijken, wat de programmeervaardigheden positief beïnvloedde. In het algemeen is het voor ons beiden een waardevol en leerzaam project geweest waarbij we samen onderzoek konden doen om zo tot ons einddoel te geraken.

## 2 Resultaten

### 2.1 Aannames over de foto's

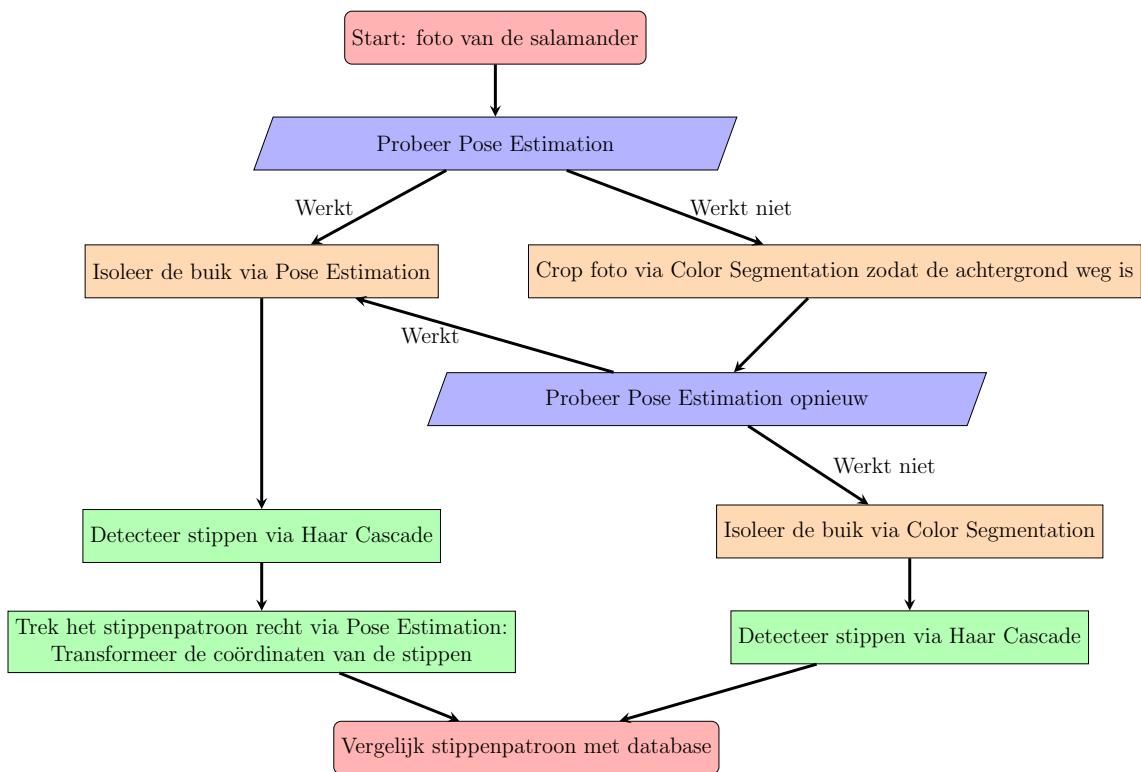
Volgende aannames werden gemaakt in ons project.

- Op elke foto is slechts een enkele salamander te zien.
- De salamander werd langs onder getrokken zodat de buik zichtbaar is op de foto.
- De belichting van de foto's is niet te donker of te helder.



Figuur 1: Salamanderfoto die aan de aannames voldoet.

## 2.2 Overzicht van de verschillende stappen



Figuur 2: Schema met de verschillende stappen van het proces.

We zullen in wat volgt de verschillende stappen van het schema bespreken.

## 2.3 Pose Estimation

Pose estimation is de stap die we als eerste uitvoeren. Het succes van deze stap bepaalt welke stappen er verder in het proces gezet zullen worden (zoals te zien in bovenstaand schema). De resultaten van de Pose estimation worden gebruikt in de latere stappen van het proces.

Het doel van deze stap is, gegeven een afbeelding, de coördinaten van bepaalde punten op het lichaam van de salamander te achterhalen (denk bijvoorbeeld aan de schouders of de staart). Indien sommige punten niet aanwezig zijn op de afbeelding, is het ook belangrijk dat we dit weten.

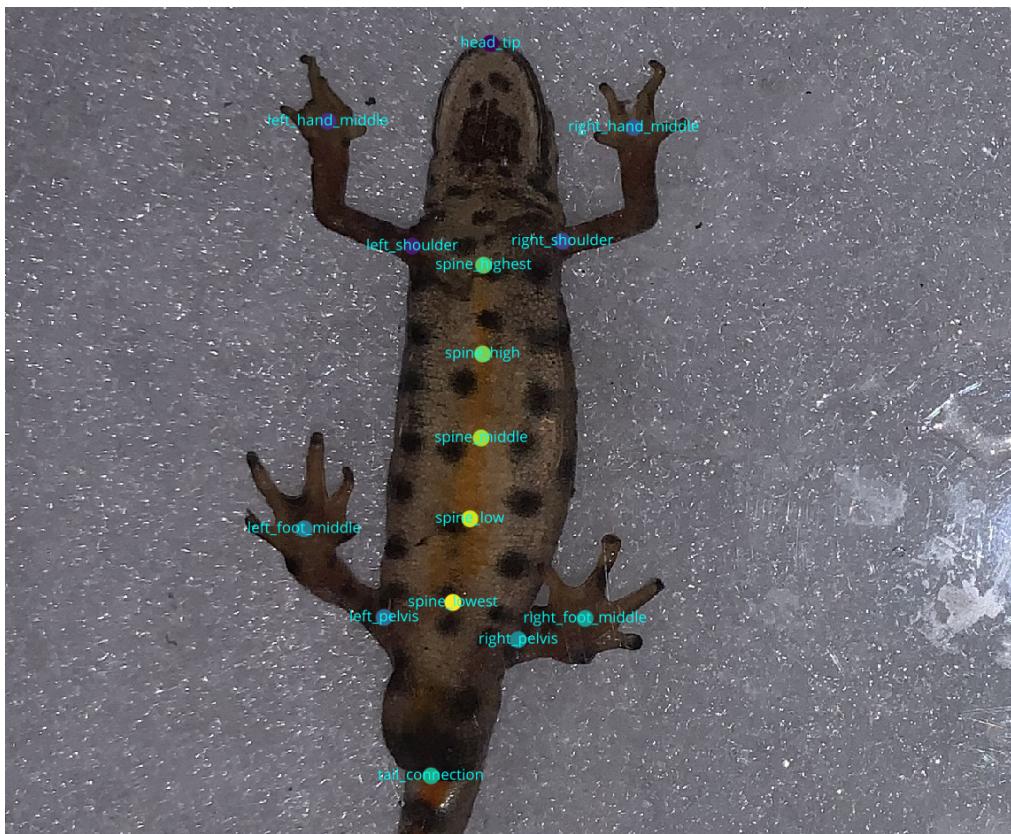
Om dit te bereiken, gebruiken we het DeepLabCut-softwarepakket [7, 8]. Dit pakket laat ons toe een effectief pose estimation-model te trainen aan de hand van een beperkte hoeveelheid trainingsdata: tussen de 100 en de 200 afbeeldingen, waar de punten op voorhand manueel op zijn aangeduid, zijn voldoende. Wanneer we dan een nieuwe afbeelding door het model voeren, krijgen we voor elk vooraf gedefinieerd punt, de geschatte  $x$ - en  $y$ -coördinaat, en een confidence-waarde. Deze waarde gebruiken we om te beslissen of we een geschat punt al dan niet gebruiken. Een lage confidence kan twee oorzaken hebben. Oftewel is de afbeelding te onscherp, of is het punt omwille van andere omstandigheden moeilijk zichtbaar, oftewel is het gezochte punt simpelweg niet aanwezig op de foto. Een hoge confidence betekent dan dat we met hoge zekerheid weten dat het geschatte punt zich op de juiste locatie bevindt.

### 2.3.1 Gekozen punten

Voordat we kunnen beginnen met trainen moet er eerst duidelijk worden afgesproken welke punten er belangrijk zijn om te kunnen lokaliseren. In totaal gebruiken we 16 verschillende punten (zie ook figuur 3):

- **head\_tip:** het midden aan het uiteinde van de kop.
- **left\_shoulder:** het snijpunt tussen de linkerkant van de romp en de rechte tussen de romp en de linker voorpoot.
- **left\_hand\_middle:** het midden van de linkerpoot.
- **right\_shoulder:** het snijpunt tussen de rechterkant van de romp en de rechte tussen de romp en de rechter voorpoot.
- **right\_hand\_middle:** het midden van de rechterpoot.
- **left\_pelvis:** het midden van de verbinding tussen linkerachterpoot en romp, op de kromme gedefinieerd door de linkerkant van de romp

- **left\_foot\_middle**: het midden van de linkerachterpoot.
- **right\_pelvis**: het midden van de verbinding tussen rechterachterpoot en romp, op de kromme gedefinieerd door de rechterkant van de romp
- **right\_foot\_middle**: het midden van de rechterachterpoot.
- **tail\_connection**: het uiteinde van de cloaca aan de kant van de staart.
- **tail\_end**: het uiteinde van de staart.
- **spine\_highest, spine\_high, spine\_middle, spine\_low en spine\_lowest**: een reeks punten op de ruggengraat. De uiteinden **spine\_highest** en **spine\_lowest** zijn respectievelijk gedefinieerd als de punten in het midden tussen de binnenste hoekpunten tussen de romp en de voor- en achterpoten. De punten ertussen zijn equidistant uitgespreid over het lichaam zodat ze in het midden van de romp liggen. Merk op dat we hier dezelfde punten op de ruggengraat gebruiken als in [7].



Figuur 3: Illustratie van de verschillende punten, verkregen via Pose Estimation.

### 2.3.2 Training data

In de eerste versie van het model voor deze punten gebruikten we enkel de 100 hoogste-kwaliteit foto's die we hadden. Dit gaf al indrukwekkende resultaten op andere foto's die in ongeveer dezelfde omstandigheden genomen waren. Het merendeel van deze foto's bevatten echter de salamander in grofweg dezelfde oriëntatie: horizontaal, met de kop aan de rechterkant. Als gevolg kon het model minder goed om met foto's in andere oriëntaties.

In de tweede versie hebben we enerzijds meer foto's toegevoegd, en anderzijds elke foto voor het trainen willekeurig  $0^\circ$ ,  $90^\circ$ ,  $180^\circ$  of  $270^\circ$  gedraaid. We hebben ons beperkt tot deze hoeken om geen extra zwartruimte te moeten toevoegen of de foto's te moeten bijsnijden, met het risico dat sommige salamanders niet volledig meer op de foto zouden passen. Aangezien de salamanders toch meestal niet volledig recht op de oorspronkelijke foto lagen zou deze beslissing de hoeken die uiteindelijk bereikt werden niet te hard beïnvloeden. Deze nieuwe versie was veel effectiever voor foto's in andere rotaties.

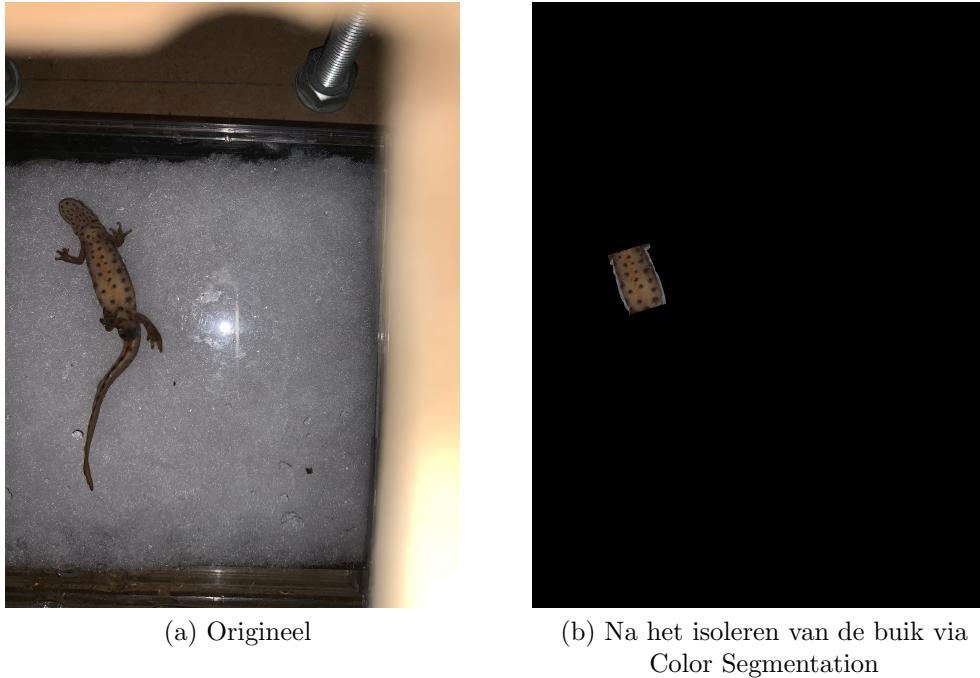
Zoals we aangeven in sectie 5.3, denken we dat het misschien wel nuttig kan zijn om het model ook op de foto's van het huidige jaar te trainen. Dit zou mogelijks de performantie van onze app verbeteren op de meest recente foto's.

## 2.4 Isoleren van de buik

De volgende stap gaat over het isoleren van de buik, hiermee bedoelen we een aantal zaken:

- Weghalen van de achtergrond zodat enkel de salamander nog overblijft.
- Weghalen van de poten, staart en kop van de salamander zodat enkel nog de buik overblijft.

De bedoeling is dus om enkel de buik van de salamander over te houden zoals te zien in Figuur 4. Dit is zeer belangrijk voor de volgende stap waarin we de stippen detecteren. Hoe beter we enkel de buik kunnen isoleren, hoe minder false positives er gedetecteerd kunnen worden als een stip (herinner dat we énkel de stippen op de buik willen detecteren).



Figuur 4: Voorbeeld van het idee wat isoleren van de buik betekent.

Oorspronkelijk hebben we dit geprobeerd via Color Segmentation [6] en dit werkt goed als de foto aan een aantal voorwaarden voldoet:

- De salamander ligt redelijk centraal op de foto.
- Als de salamander horizontaal ligt, dan ligt de kop aan de rechterkant.
- Als de salamander verticaal ligt, dan ligt de kop aan de bovenkant.
- Er is geen geel of bruin licht op de achtergrond van de foto.

Door deze (soms strikte) eisen zijn we overgestapt naar het gebruik van Pose Estimation om de buik te isoleren en dit werkt veel algemener. Dit is ook te zien in Figuur 2, waar we de voorkeur geven aan de methode met Pose Estimation [7, 8], maar indien deze faalt, kunnen we nog altijd terugvallen op de methode via Color Segmentation aangezien deze ook accurate resultaten geeft wanneer de voorwaarden voldaan zijn. Bovendien heeft de methode via Pose Estimation veel meer rekenkracht en tijd nodig dan de methode via Color Segmentation om resultaten te vinden. We verkiezen dus in feite langere rekentijd en accuratere resultaten omdat deze accurate resultaten het aantal false positives verminderen in de stap waar we stippen moeten detecteren. Dit is van cruciaal belang aangezien we geen methode hebben gevonden om met zekerheid false positive stippen te kunnen

verwijderen. Ten slotte hebben we het in sectie 5.3 ook nog over een lichtere versie van het pose estimation model, dat een verbetering zou zijn.

In volgende secties geven we een gedetailleerde uitleg hoe elke methode in zijn werk gaat.

#### 2.4.1 Isoleren van de buik via Pose Estimation

We proberen via Pose Estimation de buik van de salamander te isoleren. Het algoritme dat hier achter zit vertrekt van de gedetecteerde pose van de salamander, wat verkregen wordt uit 2.3. Het zal enkel werken als volgende (vrij zachte) voorwaarden gelden:

- Het hoogste en laagste stuk van de ruggengraat werden goed gedetecteerd.
- Een van de twee kanten van de heup werd goed gedetecteerd.
- Een van de twee schouders werd goed gedetecteerd.

We hebben gemerkt dat deze voorwaarden bijna altijd in orde zijn voor de foto's waar we mee werken. Na deze controle volgt het algoritme een aantal stappen om zo de buik te isoleren. Deze stappen leggen we in volgende secties uit.

##### 2.4.1.1 Selecteer de gedetecteerde punten

Van alle goed gedetecteerde punten (door Pose Estimation) op het lichaam van de salamander houden we enkel degene bij die behoren tot de ruggengraat, schouders of de heup. We verkrijgen dan een foto zoals Figuur 5.



Figuur 5: Salamander waar de interessante punten op zijn aangeduid.

We willen uiteindelijk punten vinden die gelokaliseerd zijn op de romp van de salamander zodat we mooi de buik kunnen uitsnijden door een lus te trekken door deze punten op de romp. Het lokaliseren kunnen we eenvoudig doen door vanuit elk punt op de ruggengraat een lijnstuk te trekken naar de romp van de salamander. Dit betekent concreet dat we moeten weten hoe ver de romp verwijderd is van de ruggengraat en onder welke hoek we deze lijnstukken moeten tekenen, uiteraard zal de hoek voor elk punt op de ruggengraat anders zijn als de salamander er gekromd bij ligt zoals in Figuur 5.

#### 2.4.1.2 Bereken van de afstand tot de romp voor elk punt op de ruggengraat

In deze stap gebruiken we al onze voorwaarden opdat dit algoritme zou werken. We willen voor elk punt op de ruggengraat weten hoe ver het verwijderd is van de romp van de salamander maar we weten niet waar de romp zich bevindt (anders hadden we al meteen de buik kunnen uitsnijden, we kennen enkel de ruggengraat). Wat we wel kennen zijn de afstanden (uitgedrukt in aantal pixels) tussen de linker- en rechterheup en het laagste punt op de ruggengraat. Analoog kennen we ook de afstanden tussen de linker- en rechterschouder en het hoogste punt op de ruggengraat.

Eerst berekenen we de *onderafstand*, dit is het rekenkundig gemiddelde van de afstanden tussen een of beide heupkanten en het laagste punt op de ruggengraat. Ook berekenen we de *bovenafstand*, dit is het rekenkundig gemiddelde van de afstanden tussen een of beide schouderkanten en het hoogste punt op de ruggengraat. Met andere woorden, de onderafstand geeft aan hoe ver de onderkant van de ruggengraat verwijderd is van de heup, wat dus op die plek een benadering is voor de afstand tot de romp. En de bovenafstand geeft aan hoe ver de bovenkant van de ruggengraat verwijderd is van de schouders, wat dus op die plek ook een benadering is voor de afstand tot de romp.

We gaan er bovendien van uit dat de dikte van het lichaam van de salamander niet met een merkbare hoeveelheid verandert. We kunnen nu door lineaire interpolatie benaderen hoe ver de romp verwijderd is van elk deel van de ruggengraat. We demonstreren dit via een simpel voorbeeld.

Stel de onderafstand is 5 en de bovenafstand is 6 en we hebben drie punten op de ruggengraat gedetecteerd. Dan krijgt het punt het dichtste bij de heupen een afstand van 5.25, het volgende punt een afstand van 5.50 en het laatste punt, wat het dichtste bij de schouders is, een afstand van 5.75.

#### 2.4.1.3 Berekenen van de hoek horende bij elk punt op de ruggengraat

Eens we de afstand hebben bepaald voor elk punt op de ruggengraat tot de romp van de salamander, moeten we ook nog de hoek vinden (ten opzichte van de horizontale) horende bij dit punt op de ruggengraat. We werken in poolcoördinaten waarbij we dus een hoek en afstand nodig hebben om

een lijnstuk te tekenen tussen het punt op de ruggengraat en een punt waar we dan in uitkomen. Dit laatste punt moet dan, als we de hoek juist kiezen, op de romp van de salamander liggen én het lijnstuk moet (zo goed als mogelijk) loodrecht staan op de romp van de salamander. Deze aanpak zorgt dat de eindpunten van deze lijnstukken een accurate benadering geeft voor duidelijk uiteenliggende punten op de romp van de salamander. Figuur 6 geeft een illustratie van het idee.



Figuur 6: Aanduiding van de afstanden en hoeken bij elk punt op de ruggengraat.

Om dit idee wel degelijk te realiseren, gebruiken we opnieuw beide aannames. De hoek horende bij de twee uiterste punten op de ruggengraat is dezelfde hoek als de hoek die de lijnstukken maken door de twee gedetecteerde punten op de heup en schouders. Dit is ook te zien op Figuur 6; de lijnstukken door de twee uiterste punten op de ruggengraat zijn evenwijdig met de, niet getekende, lijnen door de punten op de heup en schouders. Echter, de aannames vermijden niet dat er een van de twee heup- of schouderpunten niet goed gedetecteerd kan zijn. Stel nu dat er maar een enkele schouder gedetecteerd is, dan zullen we het lijnstuk (door het punt op de ruggengraat dat het dichtste bij de schouders ligt) zo trekken zodat dit lijnstuk loodrecht staat op de rechte door dit punt op de ruggengraat en het punt op de ruggengraat dat er vlak naast ligt (het op een na dichtste punt bij de schouders). Op deze manier garanderen we dat we altijd de oriëntatie kunnen benaderen door de meest extreme punten op de ruggengraat.

Hierna gaan we systematisch te werk om de hoeken te bepalen bij alle punten op de ruggengraat. Stel we kennen de hoeken al bij de twee linkse punten (op de ruggengraat) en nu willen we de hoek kennen bij het middelste punt (op de ruggengraat) op Figuur 6. Eerst trekken we twee rechten die beide door dit middelste punt gaan. De eerste rechte gaat ook door het punt links van het middelste punt en de tweede rechte gaat ook door het punt rechts van het middelste punt. Deze rechten snijden met een bepaalde hoek  $\theta$ . We kenden de oriëntatie van het punt links van het middelste punt, zeg dat deze ligt volgens een hoek  $\varphi$ . Dan wordt de hoek van het middelste punt gegeven door  $\theta + \varphi$ .

Zo kunnen we een voor een, startende van het punt dat het dichtst aan de heupen ligt, alle hoeken berekenen (omdat we telkens de hoek kennen van het punt er vlak naast en die hoek is net degene die we gebruiken). De afstanden kenden we al uit de vorige sectie en zo kunnen we via poolcoördinaten uiteindelijk aankomen bij Figuur 6. Een groot voordeel hieraan is dat we telkens de vorige hoek gebruiken in de berekening van een nieuwe hoek. Dus als de buik van de salamander plots lokaal ergens extra doorbuigt, dan gaan onze hoeken automatisch ook mee aangepast worden zodat ze de kromming van de salamanderbuik volgen. Dat is goed geïllustreerd in figuur 6, waar we duidelijk zien dat de nieuwe, gevonden punten op de ruggengraat (dit zijn uiteraard de uiteinden van de lijnstukken) effectief de gekromde oriëntatie van de salamander volgen.

#### 2.4.1.4 Detecteren en isoleren van de buik

Eens we door de technische stappen zijn geraakt, is het vervolg vrij eenvoudig. Via poolcoördinaten hebben we nu allerlei punten gedetecteerd op de romp van de salamander. We kunnen dan via interpolatie een gesloten kromme tekenen door deze punten en we krijgen de omtrek van de buik. Eens we de omtrek hebben van de buik kunnen we alles verwijderen dat buiten deze omtrek valt en het resultaat is dat we de buik hebben geïsoleerd.



(a) Kromme door punten op de romp.



(b) Buik uitgesneden via de kromme.

Figuur 7: Eindresultaat isoleren van de buik met Pose Estimation.

We hebben (buiten dat het veel rekenkracht en rekentijd nodig heeft) geen enkel probleem ondervonden met deze methode zolang de pose estimation goed werkt en er aan de voorwaarden voldaan is. Deze methode heeft dus bij uitstek onze voorkeur ten opzichte van de methode via Color Segmentation, waarbij er wel problemen kunnen optreden. Deze methode zullen we nu bespreken.

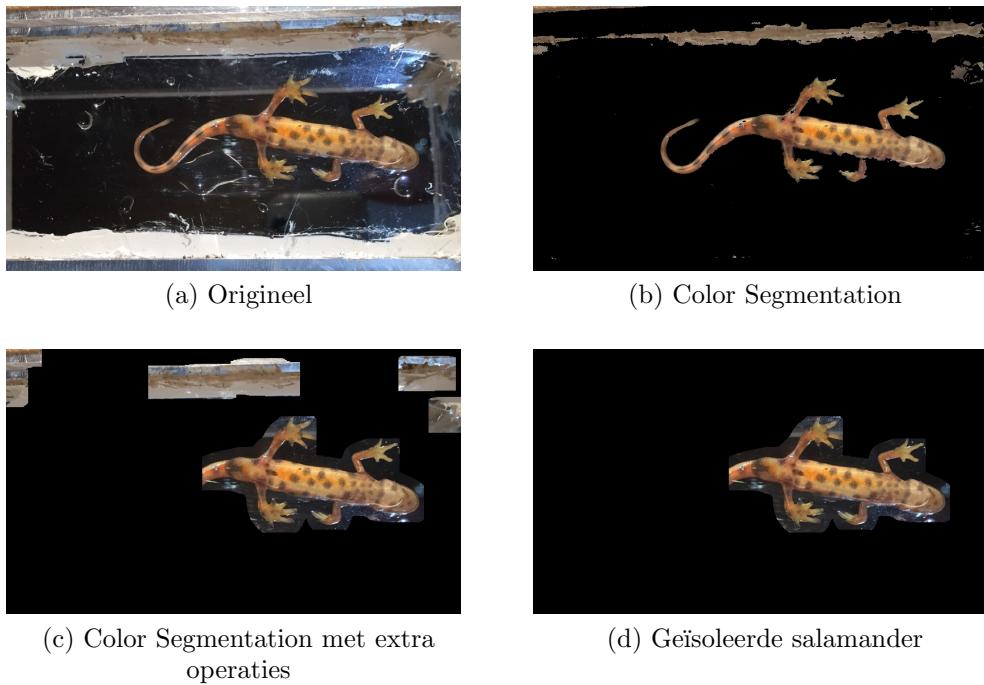
#### 2.4.2 Isoleren van de buik via Color Segmentation

Met Color Segmentation [6] kan je objecten met distincte kleuren op een foto isoleren. Een bereik aan kleuren wordt op voorhand ingesteld en alle pixels die die specifieke kleuren hebben, worden gedetecteerd. De salamanders op onze foto's hebben altijd een geel-oranje-bruine kleur, terwijl de achtergrond voornamelijk een blauw-grijs-zwart heeft. Dit maakt het mogelijk (als er geen

geel-bruine schijn is op de achtergrond van de foto!) om op de meeste foto's het salamanderlichaam te onderscheiden van de rest van de foto. Deze aanpak gaan we enkel uitvoeren wanneer er iets misloopt met de Pose Estimation aangezien de manier om de buik te isoleren via Pose Estimation veel meer zekerheid geeft voor accurate resultaten. Het is wel interessant om op te merken dat, ondanks beide methoden enorm verschillend zijn, ze beide zeer goede resultaten geven wanneer de voorwaarden voldaan zijn.

Uiteraard zijn de stippen van de salamander meestal zwart waardoor deze niet gedetecteerd worden. Bovendien is het enkel overhouden van specifieke kleuren een discreet proces en zal het vaak zo zijn dat het lijkt alsof er stukken zijn gesneden uit de salamander. Om dit tegen te gaan, en dus een mooi continu lichaam te verkrijgen (na het filteren op kleur), gebruiken we een aantal morfologische operaties, gecombineerd met blurringsoperaties. Als bijkomend probleem is het zeer goed mogelijk dat sommige delen van de achtergrond ook een bruine kleur hebben, waardoor deze mee gedetecteerd worden. Deze zullen we later nog wegfilteren, maar om te voorkomen dat deze objecten in de achtergrond verbonden worden met het lichaam van de salamander, gebruiken we technieken zoals dilatie en erosie [10].

Het idee achter Color Segmentation en de extra technische operaties is geïllustreerd in Figuur 8.



Figuur 8: Idee achter de methode om via Color Segmentation de salamander te isoleren.

Om dan ook effectief van de derde foto naar de laatste foto in Figuur 8 te gaan, moeten we van alle gedetecteerde objecten kunnen zeggen welke net de salamander is. Dit doen we in allerlei verschillende stappen waarbij we telkens het aantal objecten verengen tot er maar een overblijft, dit zal dan de salamander moeten zijn. Deze stappen worden in volgende secties uitgelegd.

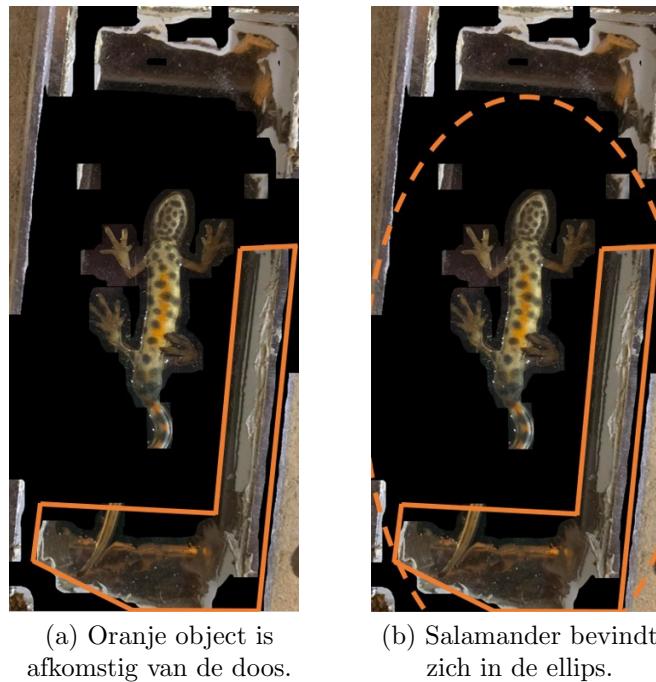
#### **2.4.2.1 Filter de objecten op locatie**

In deze eerste stap verwijderen we alle objecten die aan de rand van de foto liggen en die niet in de buurt liggen van het midden van de foto. Mocht er dan blijken dat we geen enkel object meer overhouden, dan veranderen we de parameters zodat de kans groter is dat objecten behouden blijven. Dit doen we iteratief zodat er altijd minstens een object geselecteerd is.

#### **2.4.2.2 Filter op concentrische objecten**

Een groot aantal van onze salamanders zijn getrokken terwijl de salamander in een doos zat. Dit heeft als gevolg dat vaak de rand van de doos ook gedetecteerd kan worden als object. Om deze objecten te verwijderen gaan we als volgt te werk.

Eerst tekenen we een ellips rond elk object. Als er een ander object (deels) binnen deze ellips zit, dan wordt het object waarrond de ellips in eerste instantie is getekend verwijderd. We gebruiken geen rechthoek aangezien een ellips strakker zit rond het lichaam van de salamander en we willen de salamander uiteraard niet verwijderen. Op deze manier kunnen we altijd de rand van de doos verwijderen aangezien de salamander in de doos ligt en dus ook in de ellips die getekend wordt rond de doos. Een voorbeeld wordt geïllustreerd in Figuur 9.



Figuur 9: Deel van de doos wordt weggefilterd omdat de salamander zich in de ellips ervan bevindt.

#### 2.4.2.3 Filter de objecten op oppervlakte

Tijdens de derde stap van het filteren gaan we alle objecten die tot nu toe nog overblijven rangschikken in volgorde van grootste oppervlakte naar kleinste oppervlakte. We mogen aannemen dat de salamander een van de grootste oppervlakten gaat hebben aangezien de andere grote objecten degene zijn die voorkomen aan de rand van de foto of die deel zijn van de doos waar de salamander in zit en deze objecten zijn ondertussen al weggefilterd. We houden enkel maximaal de drie grootste objecten over (op basis van oppervlakte).

Zoals te zien op Figuur 9 kan het zijn dat er kleine objecten gedetecteerd waren in de buurt van de salamander (in dit geval zijn dat de twee kleine objecten boven de twee voorpoten). Als deze op dit moment in het algoritme nog altijd niet zijn weggefilterd, dan gaan we dat nu doen. We houden het grootste object altijd bij en het tweede en derde grootste object houden we enkel bij als hun oppervlakte 30% of meer is dan de oppervlakte van het grootste object. Met deze voorwaarde verwijderen we dus de kleinere objecten.

#### 2.4.2.4 Selecteren van de salamander

Op dit moment hebben we nog maximaal drie objecten over, waaronder de salamander zelf. We kiezen nu het meest centraal gelegen object als het object dat de salamander zou voorstellen, aangezien we aannemen dat de salamander centraal gelegen is op de foto.

Uiteraard zijn we nu nog niet klaar want we zouden ook graag de poten, staart en kop van de salamander verwijderen zodat we enkel de buik overhouden. Hoe we dit doen wordt beschreven in volgende sectie

#### 2.4.2.5 Wegsnijden van de poten, staart en kop van de salamander

Eerst snijden we langst de rechterkant van het object 20% weg als de salamander horizontaal ligt of snijden we langst de bovenkant 20% weg als de salamander verticaal ligt. Dit zorgt ervoor dat we zeker de kop niet mee hebben in de volgende stappen (herinner dat we als voorwaarde hadden dat de kop aan de rechterkant of bovenkant van de salamander lag, afhankelijk van het horizontaal of verticaal liggen van de salamander). We hebben het object dus kleiner gemaakt, nu gaan we de omtrek van dit object bepalen.

Om te weten of de salamander horizontaal of verticaal ligt, kunnen we een ellips trekken die de vorm van het object dat de salamander voorstelt goed benadert. Van deze ellips kunnen we dan eenvoudig de oriëntatie bepalen en zo komen we te weten wat bij benadering de oriëntatie van de salamander is.

Dan maken we gebruik van het softwarepakket *largest interior rectangle; lir* [11]. Deze software genereert de zo grootst mogelijke rechthoek binnenin de verkregen omtrek. Deze rechthoek geeft dan een benadering voor de buik van de salamander. Om ervoor te zorgen dat we zeker heel de buik over houden (vooral de romp) vergroten we de breedte van de rechthoek nog met 5% langs beide kanten. Ook snijden we ook nog eens 5% van de lengte weg aan de kant waar de staart zou liggen. Op deze manier houden we enkel nog de buik van de salamander over.

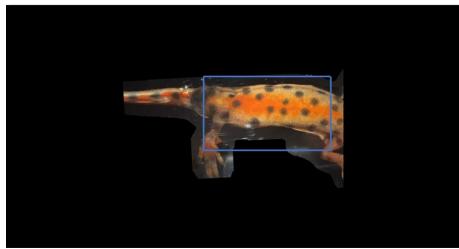
Een voorbeeld van dit proces wordt geïllustreerd in Figuur 10. Figuur 4 kan ook bekijken worden als nog een extra voorbeeld van waar het proces zeer goed werkt.



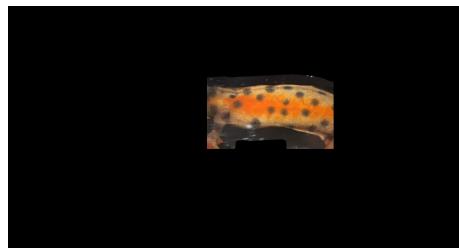
(a) Geïsoleerde salamander.



(b) Rechts 20% wegsnijden.



(c) Rechthoek via lir, met zijden al 5% vergroot of verkleint.

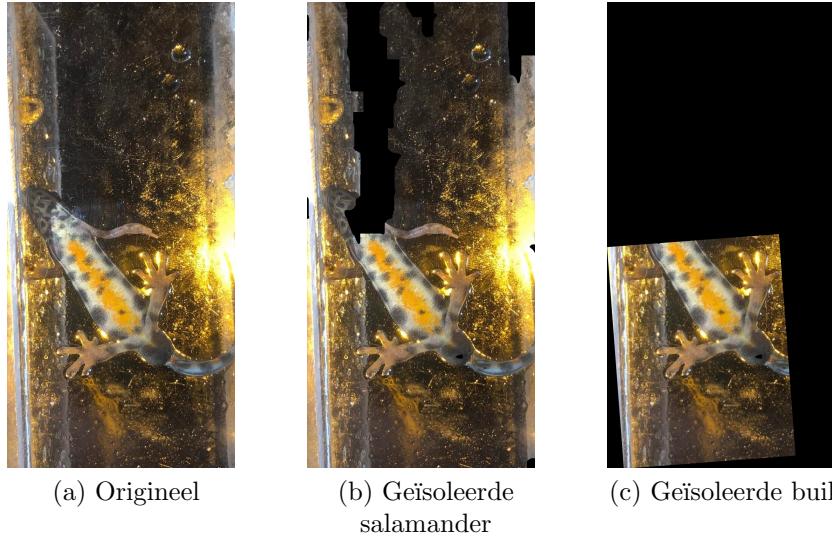


(d) Eindresultaat; de buik van de salamander.

Figuur 10: Wegsnijden van staart, poten en kop door bedachtzaam te snijden en lir.

#### 2.4.2.6 Tekortkomingen

Er zijn veel voorwaarden die voldaan moeten zijn om de methode met Color Segmentation te gebruiken. Wanneer men deze schendt kunnen er slechte resultaten verschijnen. Bovendien past de methode zich ook niet aan als de buik van de salamander gekromd is, terwijl de methode via Pose Estimation dit wel doet. Daarnaast zijn veel van de resultaten ook afhankelijk van de software lir die soms onvoorspelbaar is. Zo kan het zijn, bij foto's waar de staart breed is en perfect in het verlengde van de buik ligt, dat de grootste rechthoek ook een deel van de staart bevat, wat we natuurlijk niet willen. Om het nog erger te maken, kan het zijn dat deze rechthoek heel veel van de staart bevat en dus automatisch uitgerokken lijkt, dit zorgt dat de breedte niet zo groot is als de breedte van de buik van de salamander waardoor we ook kostbare informatie op de buik verliezen. In Figuur 11, 12 en 13 zijn voorbeelden te vinden van waar het mis gaat.



Figuur 11: Geel licht op de achtergrond van de foto zorgt dat Color Segmentation volledig misloopt.



Figuur 12: Bruine modder op de achtergrond van de foto zorgt dat Color Segmentation volledig misloopt.



(a) Origineel



(b) Geïsoleerde buik

Figuur 13: De staart ligt perfect in het verlengde van de buik waardoor de rechthoek van lir zeer uitgerokken is; het loop volledig mis.

## 2.5 Stipdetectie

Na het isoleren van de buik gaan we de stippen detecteren op de buik. We hebben zowel klassieke methoden als Haar Cascades uitgeprobeerd. Zoals we zullen vertellen, zal het zo zijn dat de klassieke methoden niet geschikt waren voor onze foto's. Bovendien geeft de Haar Cascade goede resultaten, dus is deze methode preferabel voor ons.

### 2.5.1 Klassieke methoden

We hebben, zoals vermeld in het projectvoorstel, uitgebreid geëxperimenteerd met allerlei klassieke methoden voor stipdetectie:

- Global Thresholding
- Otsu's methode
- Adaptive Thresholding
- Blob Detection
- Color Segmentation
- Een of meerdere van bovenstaande, gecombineerd met verschillende soorten van blurren en morfologische operaties.

Uiteindelijk bleek dat het succes van deze technieken te afhankelijk was van lichtomstandigheden op en kwaliteit van de foto's. Daardoor konden we deze technieken niet gebruiken voor onze doeleinden. We probeerden veel van de parameters, waar de technieken van gebruikmaken, automatisch in

te schatten en hiermee konden we beperkte vooruitgang boeken, maar dit was alsnog ontoereikend door de grote invloed van false positives (pixels die gedetecteerd worden als stip maar in realiteit geen stip zijn) door verschillende lichtomstandigheden en kwaliteit.

In de secties die nu volgen, zullen we voorbeelden geven van hoe deze methoden hebben gefaald.

#### 2.5.1.1 Global Thresholding

Global Thresholding, zoals ook beschreven in [2] is een methode om features uit een afbeelding (die omgezet is in grijswaarden) te proberen halen. De gebruiker stelt op voorhand een parameter in, dit is *de threshold*. Alle pixels die een grijswaarde hebben die kleiner is dan de threshold worden verandert naar zwarte pixels en alle pixels die een grijswaarde hebben die groter is dan de threshold worden verandert naar witte pixels. Wij probeerden op deze manier de stippen te scheiden van de rest van de salamander maar het grote probleem hierbij is dat de threshold manueel ingesteld moet worden voor elke foto. Daarnaast is deze threshold ook verschillend van foto tot foto. Hierdoor was het onmogelijk om met Global Thresholding verder te blijven werken aangezien we het niet konden automatiseren naar grote schaal.



(a) Origineel



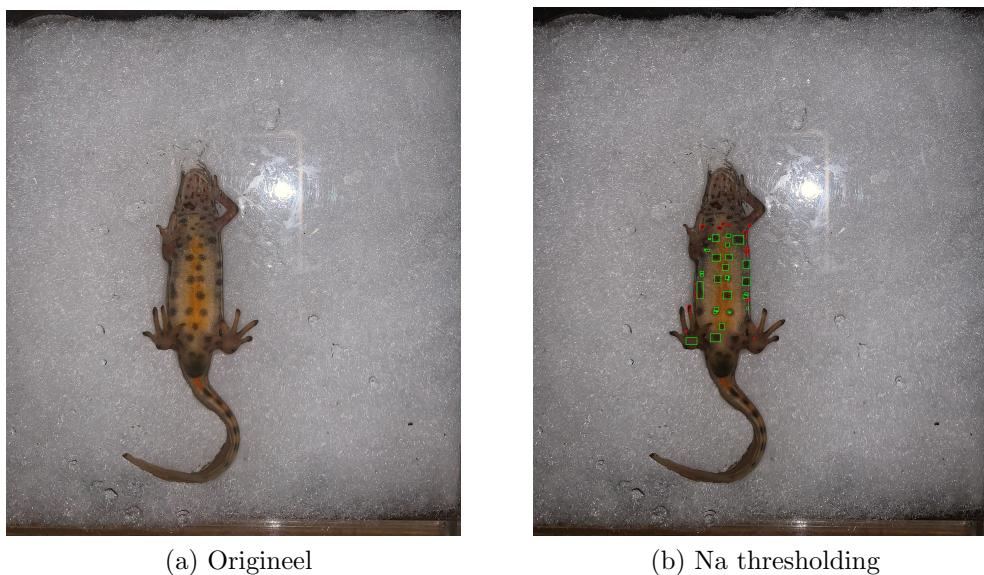
(b) Na thresholding

Figuur 14: Thresholding met een threshold van 60 kan werken om stippen te detecteren.



Figuur 15: Thresholding met een threshold van 60 werkt zeker niet om stippen te detecteren.

Hieronder nog een voorbeeld waarin we makkelijk de stippen konden detecteren na thresholding. Hier werkte het dus wel, maar we hadden alle parameters specifiek afgesteld op de gegeven foto.



Figuur 16: Thresholding werkt na manueel instellen parameters.  
Stippen staan in het groen.

### 2.5.1.2 Otsu's methode

*Otsu's methode* [3] automatiseert het proces van Global Thresholding (het zoekt dus zelf een threshold uit) door onder andere gebruik te maken van de intensiteitsniveaus op de foto. Om goed met Otsu's methode te kunnen werken, zou er een duidelijk verschil in intensiteit en contrast moeten zijn tussen de stippen en de rest van het lichaam, zo kan de methode een threshold uitkiezen die

er tussen ligt waardoor stippen worden gescheiden van de rest van het lichaam. Dit verschil in intensiteit is bij ons niet altijd het geval, waardoor dit ook geen oplossing was voor ons.



(a) Origineel



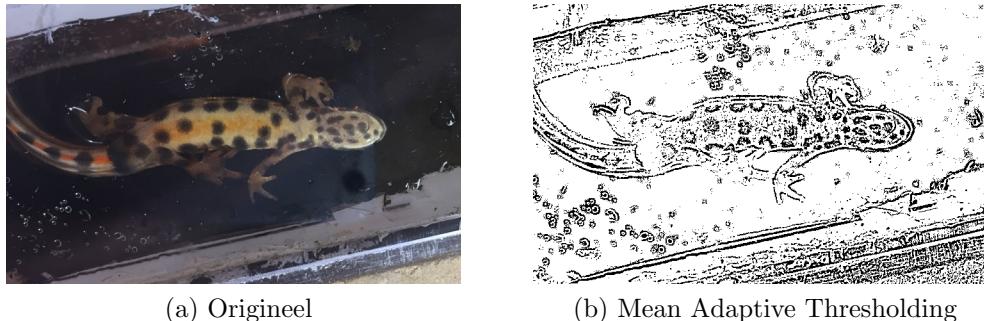
(b) Na thresholding

Figuur 17: Thresholding met Otsu's methode gaat mis.

### 2.5.1.3 Adaptive Thresholding

Een alternatieve manier is gebruik maken van Adaptive Thresholding (soms ook Local Thresholding genoemd) [4]. In plaats van een vaste threshold te kiezen (zoals voorheen), gaan we nu voor elke groep naburige pixels een eigen, lokale, threshold automatisch genereren. Als voorbeeld kan er als lokale threshold voor een pixel het gemiddelde van de intensiteit van de pixels in de buurt genomen worden. Dit zorgt voor veel accuratere resultaten en we hadden hier veel hoop in dat het een goede methode zou zijn voor ons. We leken vooral veel potentieel te zien in Median Adaptive Thresholding. Gaussian Adaptive Thresholding vonden we iets minder accuraat voor onze toepassingen.

Het probleem met deze methodes was dat het soms te gedetailleerd was. Voor onze toepassing wouden we enkel stippen detecteren maar er werden ook stukken van het lichaam gedetecteerd die net iets lichter waren dan de omliggende stukken van het lichaam. Er was veel te veel noise gedetecteerd op de foto's en we hebben dit proberen wegwerken door de omgeving rond de pixel die werd gebruikt te vergroten. Ook hebben we daarna proberen blurren en morfologische operaties toe te passen om zo effectief goede stippen te kunnen detecteren, maar zoals voorheen hebben we ondervonden dat al deze operaties afhankelijk zijn van foto tot foto en dus is deze manier niet werkbaar op grote schaal.



Figuur 18: Mean Adaptive Thresholding detecteert veel te veel details.

#### 2.5.1.4 Blob Detection

Blob Detection [5] is een methode om gegroepeerde objecten (blobs) te detecteren. Men kan zelf de soort van deze objecten instellen zoals minimale en maximale oppervlakte en vorm. Echter waren er een aantal problemen. De stippen op een salamander zijn niet altijd mooie cirkels, soms zijn ze enorm uitgerokken. Ook zijn er stippen aan de romp van de salamander die half af de foto vallen waardoor dit in feite halve cirkels worden op de foto. Hierdoor konden we de blob detector nooit specifiek laten zoeken naar bepaalde eigenschappen. Deze methode werkte dus ook niet voor ons.

#### 2.5.1.5 Color Segmentation

Color Segmentation [6] lijkt op thresholding. Het verschil is dat we bij Color Segmentation gaan schiften op een bereik van kleuren in plaats van op een grijswaarde, zoals bij thresholding het geval is. We houden bij Color Segmentation dus enkel de gebieden over die een kleur hebben die in het bereik van kleuren vallen dat we op voorhand instellen. Ons idee was om de stippen te kunnen detecteren aan de hand van hun donkere kleur, aangezien het lichaam een bruin-oranje kleur heeft. Maar dit leek ook niet te werken, op sommige foto's zijn de stippen zeer donker en op andere foto's zijn de stippen dan weer eerder lichbruin. Hierdoor moeten we de lichtbrune kleur (die dus bij sommige salamanders de kleur van heel het lichaam is) meenemen in het bereik van kleuren dat een stip zou voorstellen. Uiteraard was het onmogelijk om deze methode te gebruiken op grote schaal.



Figuur 19: Verschil in contrast en kleur van stippen bij salamanderfoto's.

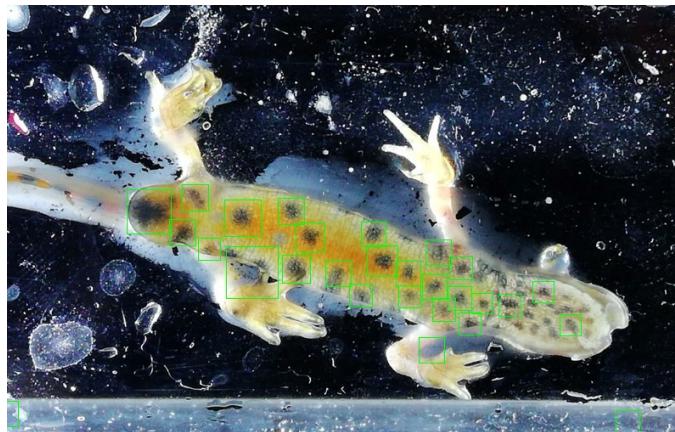
Hiermee sluiten we de klassieke methodes voor stipdetectie af en schakelen we in de volgende sectie over naar de haar cascades, die wel werkte voor onze salamanderfoto's.

### 2.5.2 Haar cascades

Als middenweg tussen deze methoden en de zwaardere neurale netwerken voor object detection hebben we Haar Cascades [9] uitgeprobeerd. Deze techniek bleek verrassend effectief. De cascade wordt getraind met een hoog aantal “positive” afbeeldingen van het te herkennen object (de stip, in ons geval). Merk op dat deze foto's enkel het object moeten bevatten: de achtergrond, poten, staart en kop van de salamander worden op voorhand zo veel mogelijk weggesneden met methoden die we eerder al besproken. Deze worden tijdens het trainen geplaatst bovenop de grotere “negative” afbeeldingen, die zo gekozen worden dat ze absoluut niet de stip bevatten, maar wel achtergronden waarin de salamander zich vaak bevindt.

Met de ingebouwde implementatie in de OpenCV-library kregen we al met een 500-tal positive afbeeldingen en 4 cascade-stages een goed resultaat. Meer stages leidde snel tot overfitting, waarbij het model slecht zou generaliseren naar nieuwe foto's. Om een idee te geven van hoe licht de Haar Cascades zijn in vergelijking met zwaardere Computer Vision-modellen: het trainen van de Haar Cascade duurde bij ons slechts 30 seconden, terwijl het trainen van het DeepLabCut-model voor

pose estimation (zie 2.3) enkele uren duurde.



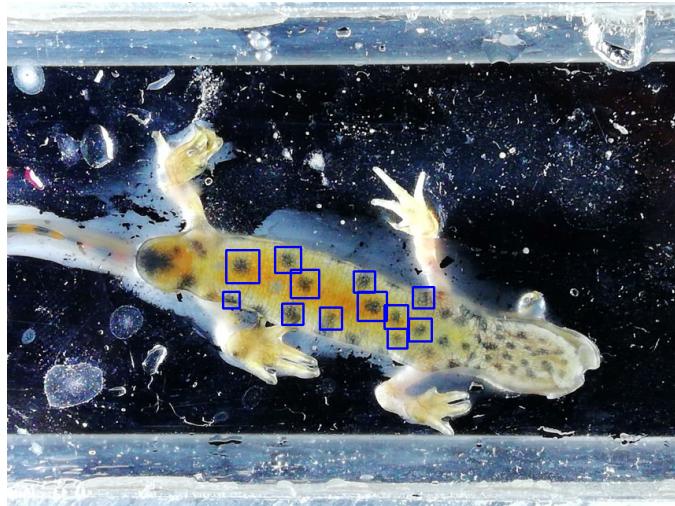
Figuur 20: Stipdetectie met een vroege versie van de haar cascade, met slechts enkele false positives.

#### 2.5.2.1 Beoordeling van versies van de Haar cascade

Om de parameters van de haar cascade te kunnen instellen om een zo goed mogelijk resultaat te bereiken, was het belangrijk om de resultaten van een versie objectief te kunnen beoordelen, om daarna te kunnen vergelijken. Om dit te kunnen doen hebben we een script geschreven dat bij een aantal foto's de automatische detecties gaan vergelijken met manueel aangeduiden stippen. Dit enerzijds met de afstanden tussen de locaties, en anderzijds met de mate van overlap tussen de gebieden gemarkeerd als zijnde onderdeel van de stip.

#### 2.5.2.2 Eliminatie van false positives door achtergrondverwijdering

De weinige false positives die nog voorkwamen bij detectie met de Haar Cascade bevonden zich vaak op de achtergrond van de foto. Door de achtergrond op voorhand te verwijderen konden we de false positives dan ook tot een minimum reduceren. Zoals te zien op Figuur 21, zorgt het isoleren van de buik voor minimale (in dit geval geen) false positives; we detecteerden dus enkel stippen op de buik.



Figuur 21: Stipdetectie met haar cascade, waarbij we eerst de buik van de salamander isoleerden.

## 2.6 Rechttrekken stippenpatroon via Pose Estimation

De gedetecteerde coördinaten tussen verschillende salamanderfoto's kunnen we op dit moment nog niet met elkaar gaan vergelijken. Ze zijn namelijk zeer afhankelijk van de pose van de salamander. Inderdaad, het stippenpatroon van een salamander die krom ligt en een salamander die recht ligt, kunnen we niet vergelijken met elkaar. Om dit te verhelpen maken we gebruik van de punten op de ruggengraat die met pose estimation (zie 2.3) gedetecteerd zijn.

Als eerste stap doen we zelf een interpolatie van de ruggengraat aan de hand van de 5 gekende punten. Een interpolatiefunctie  $y = f(x)$  zou niet altijd werken, aangezien we geen garanties hebben over de pose van de salamander: het is goed mogelijk dat de ruggengraat van de salamander niet kan worden uitgedrukt aan de hand van een functie. Om dit probleem te omzeilen gebruiken we een parameter  $t$ . Zo kunnen we  $x$ - en  $y$ -coördinaten afzonderlijk interpoleren met de geparametriserde functies  $x(t)$  en  $y(t)$ . De betekenis van  $t$  is als volgt: op het gedetecteerde punt **spine\_highest** is  $t = 0$ , de tussenpunten hebben opeenvolgende natuurlijke  $t$ -waarden, en op **spine\_lowest** is  $t = 4$ . Vervolgens bepalen we voor het recht te leggen punt, dat we  $P$  noemen, een aantal gegevens:

- het dichtstbijzijnde punt op de geïnterpoleerde ruggengraat, dat noemen we  $Q$ ,
- de bijbehorende  $t$ -waarde  $t_Q$  zodat  $Q = (x(t_Q), y(t_Q))$ ,
- de afstand  $d$  tussen  $P$  en  $Q$ ,
- en aan welke kant van de ruggengraat het punt ligt. Dit laatste doen we als volgt. Eerst definiëren we de  $t$ -waarden voor de twee dichtstbijzijnde pose estimated punten:  $t_{prev} =$

$\lfloor t_Q \rfloor$  en  $t_{next} = \lceil t_Q \rceil$ . We gebruiken respectievelijk de notatie voor de *floor* en *ceiling*. De bijbehorende punten noemen we  $Q_{prev}$  en  $Q_{next}$ . Vervolgens nemen we het vectorproduct tussen de vectoren  $Q_{next} - Q_{prev}$  en  $P - Q_{prev}$ . Afhankelijk van het teken van het vectorproduct weten we dat het punt “links” of “rechts” lag.

Vervolgens leggen we eerst de ruggengraat zelf recht. Dit houdt in dat we alle ruggengraatpunten  $Q_t$  op de  $x$ -coördinaat van het bovenste punt  $Q_0$  leggen, en voor de  $y$ -coördinaat rekening houden met de afstand op de ruggengraat tussen de punten onderling. Deze afstand  $L$  kunnen we berekenen als de booglengte over de geïnterpoleerde ruggengraat. Samen wordt dit:  $Q_T = (x_0, y_0 + L)$  met

$$L = \int_{t_0}^T \sqrt{(x'(t))^2 + (y'(t))^2} dt.$$

Nu kunnen we voor het nieuwe rechtgelegde punt  $P'$  de  $y$ -coördinaat berekenen aan de hand van  $t_Q$  en de volledige lengte van de ruggengraat. De nieuwe  $x$ -coördinaat is dan weer gelijk aan die van de rechtgelegde ruggengraat, met de oorspronkelijke afstand tot de ruggengraat oftewel opgeteld oftewel afgetrokken, afhankelijk van aan welke kant van de ruggengraat het punt ligt, zoals hierboven bepaald. Uiteindelijk vinden we in coördinaten dat

$$P' = (x_0 \pm d, y_0)$$

## 2.7 Vergelijken van stippenpatronen

Eens we voor elke salamander stippen hebben gedetecteerd en hebben rechtgelegd, kunnen we een stippatroon matching algoritme gebruiken om te controleren of salamanders op verschillende foto’s effectief dezelfde salamanders zijn. Ons algoritme is gebaseerd op [13] en [12] en werkt in een aantal stappen, waarbij we vaak het woord punten gebruiken in plaats van stippen. In wat volgt geven we een kort overzicht van alle stappen, nadien bespreken we alle stappen in detail.

1. Start met een lijst coördinaten van de stippen op de buik van de onbekende salamander.
2. We gaan een voor een de onbekende salamander vergelijken met een salamander uit de database. Voor deze salamander starten we ook met een lijst coördinaten van de stippen op de buik. Hierna voeren we de stappen (a) tot (g) uit voor deze salamander, die we de geselecteerde salamander noemen. Nadien gaan we over naar de volgende salamander in de database en voeren we opnieuw stappen (a) tot (g) uit en blijven dit doen totdat we elke salamander hebben gehad uit de database.
  - (a) Controleer of de onbekende en geselecteerde salamander ongeveer evenveel stippen hebben.
  - (b) Genereer alle mogelijke verschillende driehoeken tussen de stippen uit beide stippatronen en selecteer dan de driehoeken die interessant zijn voor het matchen.

- (c) Voer een matching algoritme uit op de driehoeken zodat we driehoeken van beide foto's aan elkaar kunnen linken.
- (d) Verminder het aantal valse matches.
- (e) Match punten van beide foto's met elkaar, gebaseerd op de gematchte driehoeken. Nu hebben we stippen van beide foto's met elkaar laten overeenkomen.
- (f) Voer eenmalig nog eens de stappen (a) tot (e) uit, waarbij we enkel de punten gebruiken die gematcht zijn in stap (e), om een accurater resultaat te bekomen (dit is nog een extra manier om valse matches weg te filteren).
- (g) Bereken een score om te bepalen hoe goed de resulterende match tussen beide salamanders is.
3. Toon de drie geselecteerde salamanders op het scherm, die de hoogste score hadden uit alle salamanders van de database. Hierbij wordt ook de bijbehorende scores weergegeven (die we later nog zullen uitleggen). Bovendien wordt er ook aangegeven of er een strong, medium, weak of no match optreedt.

In de praktijk zal het echter zo zijn dat we, van de drie beste matches, enkel degene zullen tonen waarvoor  $S \geq 0.4$  ( $S$  is de score, later meer hierover). Dit is om te voorkomen dat de gebruiker in de war zou geraken, door het voorstel van een slechte match als potentiële kandidaat, en zo twee verkeerde salamanders aan elkaar zou toewijzen.



Figuur 22: Eindresultaat waarbij de beste drie matches getoond worden op het scherm.

Zoals we zien op figuur 22, staan de drie beste matches gelokaliseert naast de verticale streep. Het spreekt voor zich dat de meest linkse foto de beste is aangezien het dezelfde foto is als de unknown

image. De tweede beste match heeft ook een hele hoge score en wordt beschouwt als strong match. Dit is enorm goed aangezien (als er aandachtig wordt gekeken naar het stippenpatroon) dit dezelfde salamander is als de salamander op de unknown image, echter in een totaal andere positie en setting. De derde beste optie is eigenlijk geen match, dit betekent dat er ook alle andere salamanders van de database geen match zullen zijn. We zien dus dat er geen false positives optreden.

Het doel is dus om, via het matchen van driehoeken, punten van beide salamanders te matchen. Het is evident dat een groter aantal gematchte punten zal resulteren in een betere match (lees dus ook in een hogere score).

Het grote voordeel aan deze aanpak is dat het matching algoritme onafhankelijk is van rotatie, schaling en spiegeling. Dit is noodzakelijk omdat we nog altijd willen dat een salamander herkent wordt als hij later, in een nieuwe foto, getrokken wordt waarbij de positie anders is. Bovendien maakt het ook niet uit of de salamander op een nieuwe foto krommer ligt, aangezien we dan het “kromme stippenpatroon” al hebben rechtgetrokken in vorige stappen. In wat volgt zullen we in meer detail ingaan op de voorgaande stappen.

### 2.7.1 Lijst van coördinaten

Voor elke salamander bepalen we de stippen op de buik zoals voorheen beschreven in dit werk. Als er meerdere stippen op minder dan een afstand  $\varepsilon$  van elkaar staan, dan houden we er slechts een over. Deze parameter  $\varepsilon$  is onze tolerantie waar we mee werken. Standaard werd deze ingesteld op  $\varepsilon = 0.01$ . Merk op dat we de coördinaten van de stippen stilzwijgend normaliseren zodat deze coördinaten hebben tussen 0 en 1. Elke stip die we dan nog overhouden is in feite een rechthoek (de Haar Cascade slaat de stippen op als rechthoeken), dus voor handiger te kunnen verder werken, slaan we de coördinaten van de middens van de gedetecteerde rechthoeken op. We hebben ervoor gekozen om de afmetingen van de stip niet bij te houden aangezien de stippen op een salamander meestal toch allemaal dezelfde grootte hebben.

### 2.7.2 Controle op het aantal stippen

Het is opvallend dat het aantal stippen van salamanders enorm verschillend kan zijn. Zie bijvoorbeeld Figuur 23. Het is dus een logische stap om eerst te controleren of het aantal stippen wel ongeveer overeenkomt (minder dan tien verschil in aantal stippen) voordat we beginnen aan heel de procedure van het matchen. Op deze manier kunnen we onmiddellijk salamanders negeren die anders toch geen match zouden zijn, dus we besparen hier enorm veel rekentijd wanneer we werken met een grote database. Als er in onderstaande stappen wordt gesproken over een database of alle salamanders, dan bedoelen we hiermee enkel de salamanders die door deze controle zijn geraakt.



Figuur 23: Twee kleine watersalamanders

### 2.7.3 Selecteren van driehoeken om te matchen

Na dat we alle mogelijke driehoeken (dit is telkens een collectie van drie punten) hebben gegenereerd voor elk stippatroon, gaan we voor elke collectie drie punten, die we vanaf nu vertices noemen, ze ordenen zodanig dat de kortste zijde van de driehoek tussen vertices 1 en 2 ligt en de middelste zijde tussen vertices 2 en 3 ligt en de langste zijde tussen vertices 1 en 3 ligt. Dit zal ons later helpen bij het verwijderen van valse matches. We noemen bovendien  $r_2$  de kortste zijde en  $r_3$  de langste zijde, horende bij een driehoek.

Hierna gaan we voor elke collectie drie vertices, i.e. de punten  $(x_1, y_1), (x_2, y_2), (x_3, y_3)$  een aantal waarden berekenen. Het doel is om per salamander een lijst te genereren met als elementen elke driehoek met overeenkomstige berekende waarden. Deze waarden zijn de onderstaande:

$R = \frac{r_3}{r_2}$  verhouding tussen langste en kortste zijde.

$s = \frac{r_3}{\max r_3}$  verhouding tussen langste zijde en de langste zijde van alle driehoeken.

$$A = \frac{1}{2} \cdot (x_1 \cdot (y_2 - y_3) + x_2 \cdot (y_3 - y_1) + x_3 \cdot (y_1 - y_2)).$$

$C = \frac{1}{r_3 \cdot r_2} \cdot ((x_3 - x_1) \cdot (x_2 - x_1) + (y_3 - y_1) \cdot (y_2 - y_1))$  de cosinus van de hoek horende bij vertex 1.

$S = \frac{1}{r_3 \cdot r_2} \cdot ((x_2 - x_1) \cdot (y_3 - y_1) - (y_2 - y_1) \cdot (x_3 - x_1))$  de sinus van de hoek horende bij vertex 1.

$$F = \varepsilon^2 \cdot \left( \frac{1}{r_3^2} - \frac{C}{r_3 \cdot r_2} - \frac{1}{r_2^2} \right).$$

$\text{tol}_r = \sqrt{2R^2F}$  tolerantie die we gaan gebruiken voor  $R$ .

$\text{tol}_c = \sqrt{2S^2F + 3C^2F^2}$  tolerantie die we gaan gebruiken voor  $C$ .

$\log_p = \log(\text{omtrek van de driehoek})$ .

Deze berekende waarden kunnen we nu gebruiken om een aantal gegenereerde driehoeken te verwijderen (deze gebruiken we dus niet om mee te matchen). Als  $R > 8$  of  $C > 0.99$  of  $s > 0.85$ , dan verwijderen we deze driehoek. Deze waarden hebben we experimenteel gevonden en ze zorgen ervoor dat in essentie de grote driehoeken verwijderd worden. Er zullen altijd veel lange driehoeken zijn die bijna lijken op een rechte lijn, omdat ze zo uitgerokken zijn. Deze lange driehoeken matchen te makkelijk met lange driehoeken van andere salamanders, daarom vergroot de kans op valse matches. Met deze eisen verkleinen we dus de valse matches en houden we nog veel driehoeken over die we kunnen gebruiken om te matchen.

Hierna berekenen we ook de oriëntatie van de driehoek, i.e. we stellen ons de vraag of de vertices (van een tot drie) in wijzerzin of in tegen wijzerzin worden belopen. Dit doen we via de *Shoelace formula* [14], dus als  $A > 0$ , dan is het tegen wijzerzin en als  $A < 0$ , dan is het in wijzerzin. Als  $A = 0$  (in de praktijk als  $A \approx 0$ ), zijn de punten collineair en wordt deze ‘driehoek’ verwijderd.

Tot slot onthouden we voor elke driehoek die overblijft de drie punten,  $R$ ,  $C$ ,  $\text{tol}_r$ ,  $\text{tol}_c$ ,  $\log_p$  en de oriëntatie. Bovendien kunnen we opmerken dat, door  $R$  en  $C$  te kennen, we de driehoek op een unieke wijze hebben vastgelegd, onafhankelijk van de plaats, oriëntatie, rotatie en schaling.

### 2.7.4 Matching van de driehoeken

Na het genereren van alle interessante driehoeken van beide salamanders, gaan we over tot het matchen van deze driehoeken. We proberen voor elke driehoek van de onbekende salamander een corresponderende driehoek van de geselecteerde salamander te vinden. Deze stap is het duurste om uit te voeren en zorgt ervoor dat het algoritme een lange tijd nodig heeft om te runnen. Daarom is er gekozen om een andere (snellere) implementatie te gebruiken dan beschreven in [12].

We gaan volgend proces voor elke driehoek van de onbekende salamander uitvoeren; laat  $R_a$  de  $R$ -waarde zijn van een driehoek horende bij de onbekende salamander. Houd deze driehoek nu vast. We gaan nu alle driehoeken van de geselecteerde salamander ordenen op basis van een zo klein mogelijke waarde  $(R_a - R_b)^2$ , waarbij  $R_b$  de  $R$ -waarde van een driehoek horende bij de geselecteerde salamander betekent. Na dit ordenen gaan we dan deze driehoeken een voor een af (we starten dus met driehoeken waarvan de waarde  $(R_a - R_b)^2$  klein is). Vanaf dat een driehoek voldoet aan volgende voorwaarden

$$(R_a - R_b)^2 < \text{tol}_{r_a}^2 + \text{tol}_{r_b}^2,$$

$$(C_a - C_b)^2 < \text{tol}_{c_a}^2 + \text{tol}_{c_b}^2,$$

wordt deze driehoek gematcht met de driehoek van de onbekende salamander. Hierdoor zorgen we ervoor dat enkel de beste driehoek  $b$  gaat matchen met de vastgehoude driehoek van de onbekende salamander. Hierna doen we het proces opnieuw met de volgende driehoek van de onbekende salamander, totdat we alle driehoeken van de onbekende salamander hebben gehad. Voor de duidelijkheid vermelden we ook dat de waarden  $\text{tol}_{r_a}$ ,  $\text{tol}_{r_b}$ ,  $\text{tol}_{c_a}$  en  $\text{tol}_{c_b}$  de waarden zijn die u verwacht (de index  $a$  duidt op de onbekende salamander en de index  $b$  duidt op de geselecteerde salamander).

Wanneer we een match hebben tussen twee driehoeken, zeg maar driehoek  $a$  en driehoek  $b$ , dan kunnen we extra informatie berekenen voor deze specifieke match. We berekenen eerst

$$\log M := \log_{p_a} - \log_{p_b}$$

en noemen  $M$  de schalingsfactor tussen de twee driehoeken. Daarna berekenen we ook de *sense* van de match; als beide driehoeken dezelfde oriëntatie hebben, dan is de match *same sense*, anders is de match *opposite sense*.

### 2.7.5 Verminderen van het aantal valse matches

Na het matchen van de driehoeken, hebben we veel valse matches verkregen. Deze willen we proberen te verminderen zonder echte matches te verwijderen. We gaan kijken naar de verdeling van alle  $\log M$  waarden bij alle matches. Stel dat onze onbekende salamander inderdaad dezelfde is

als een geselecteerde salamander. Dan zal de log  $M$  waarde bij alle gematchte driehoeken ongeveer hetzelfde moeten zijn, want de schalingsfactor  $M$  tussen beide foto's is een constante. Stel dan nu dat de geselecteerde salamander niet dezelfde is als de onbekende salamander, dan zijn er dus een hele hoop driehoeken vals gematcht met totaal willekeurige log  $M$  waarden.

Door deze opmerking over de log  $M$  waarden, kunnen we de valse matches snel scheiden van de goede matches. We kijken naar het gemiddelde en de standaardafwijking van de log  $M$  verdeling en we gaan iteratief de uitschieters verwijderen totdat we niets meer over hebben, een op voorhand bepaald aantal iteraties (bij ons 20) hebben bereikt of als alle overblijvende waarden binnen een bepaalde geschaalde standaardafwijking liggen ten opzichte van het gemiddelde. Voor meer informatie over de specifieke berekeningen van de niet vermelde waarden, wordt u doorverwezen naar [13] en [12].

Bovendien kunnen we ook kijken naar het aantal same sense en opposite sense matches dat nu nog overblijft. Het spreekt voor zich dat als we twee dezelfde salamanders met elkaar proberen matchen, de gematchte driehoeken allemaal same sense of allemaal opposite sense zijn (afhankelijk van de relatieve oriëntatie en spiegeling tussen de salamanders op beide foto's). Maar als we naar twee niet dezelfde salamanders kijken en de matches zijn per toeval; willekeurig gebeurt (en dit zijn dus valse matches), dan gaat het aantal same en opposite sense matches ongeveer gelijk zijn. Hierdoor kunnen we dus ook nog schiften. We tellen het aantal same sense matches en het aantal opposite sense matches en we zullen alle matchende driehoeken verwijderen die in de minderheid zijn. Hierdoor verwijderen we voornamelijk veel valse matches.

### 2.7.6 Match punten, gebaseerd op de gematchte driehoeken

Op dit moment in het algoritme hebben we al gematchte driehoeken. Het doel van het algoritme is echter om overeenkomstige stippen op de buik van de salamanders te matchen. Daarom gaan we de gematchte driehoeken omzetten naar gematchte punten. Omdat het nog altijd kan zijn dat er valse matches zijn onder de driehoeken, gaan we een geavanceerdere methode gebruiken om punten te matchen dan eenvoudigweg zeggen dat een vertex  $a$  van de ene driehoek matcht met vertex  $a$  van de andere (bijbehorende gematchte) driehoek.

Elke twee gematchte driehoeken bevatten exact drie paar vertices (vertex 1 van driehoek  $a$  met vertex 1 van driehoek  $b$ , ...) Op deze manier hebben we in totaal een groot aantal paren van vertices. Deze paren van vertices zijn kandidaat matches. Dus bijvoorbeeld vertex 1 van driehoek  $a$  kan gematcht worden met vertex 1 van driehoek  $b$ . Om ervoor te zorgen dat enkel echte matches van punten gelabeld worden als een match, gebruiken we een stemsysteem. Elke twee gematchte driehoeken geeft één stem aan elk paar vertices. Paren vertices die in meerdere gematchte driehoeken bevatten zijn, zijn wellicht echte matches en krijgen ook meerdere stemmen (omdat meerdere gematchte driehoeken een stem uitbrengen voor dit paar). Na het stemmen ordenen we alle paren

van vertices van meeste stemmen tot minste stemmen. Het spreekt dan voor zich dat de paren van vertices die de meeste stemmen hebben gekregen, worden gelabeld als een paar matchende punten. We starten het labelen dan ook met de eerste paren vertices in onze geordende rij en we gaan ze een voor een af. We stoppen onmiddellijk wanneer er ofwel een label zou worden gegeven aan een paar vertices waarbij een van deze vertices al eerder werd gelabeld met een andere vertex, het aantal stemmen van een paar vertices kleiner is dan het aantal stemmen van het vorige paar gedeeld door twee of als het aantal stemmen van een paar vertices nul is.

Op bovenstaande manier matchen we paren van vertices met elkaar, dit komt dus overeen met gematchte stippen tussen beide foto's.

### 2.7.7 Score berekenen tussen twee salamanders

Tot slot moeten we nog de score  $S$  berekenen tussen beide salamanders zodat we (na het afgaan van alle salamanders uit de database) kunnen bepalen welke salamanders uit de database het beste overeenkomen met de onbekende salamander. Onze score  $S$  bepalen we op volgende wijze:

$$S = 0.4 \cdot \frac{S_1}{V_{\max}} + 0.6 \cdot S_2,$$

waarbij

- $V$  = Totaal aantal stemmen, horende bij de gematchte punten.
- $V_{\max}$  = Totaal aantal mogelijke stemmen; drie keer het aantal gematchte paren driehoeken.
- $f_T$  = Percentage van driehoeken dat heeft gestemd t.o.v. het totaal aantal driehoeken dat overblijft na stap (b); in het overzicht.
- $S_1$  =  $V \cdot f_T$ .
- $S_2$  = Percentage van gematchte punten t.o.v. het aantal punten van de onbekende salamander.

Met andere woorden, hoe hoger de score  $S$ , hoe beter de match. We kunnen dus via deze parameter bepalen wat de drie beste matches zijn. Bovendien wordt er, net zoals in Figuur 22, zowel  $S_1$  als  $S_2$  weergegeven. Meer specifieke informatie over  $S_1$  kan worden teruggevonden in [13], we vermelden dat als  $S_1 \geq 50$  en  $S_2 \geq 75\%$ , er een hele grote kans is dat het om dezelfde salamander gaat.

## 3 App en Serversoftware

Een belangrijk onderdeel van het project was het beschikbaar stellen van de software op een manier die bruikbaar is voor eindgebruikers. Om dit te realiseren hebben we een app, en draaien we de herkenningsssoftware op een publiek toegankelijke server.

Samen laten de app en server toe een salamanderfoto in te sturen, en als antwoord de salamanders terug te krijgen die het beste overeenkomen met de ingestuurde foto. De gebruiker kan dan oftewel één van de gesuggereerde salamanders selecteren, oftewel beslissen dat dit een nieuwe salamander is. De server zal dit resultaat opslaan in de database, en alle informatie - inclusief de ingezonden foto's - blijft beschikbaar.

Langs de serverkant hebben we verdergewerkt in Python. Met Flask stellen we een REST API open. De app is ontwikkeld in Flutter, met de Dart programmeertaal, wat ons toelaat om met eenzelfde codebase zowel een app als website te bouwen.

Voor de werking van de app/website verwijzen we naar Appendix A.

Voor alle achterliggende code verwijzen we naar het volgende:

<https://github.com/SpeedyCodes/salamander-tracking>

<https://github.com/SpeedyCodes/salamander-tracking-frontend>

### 3.1 Kwaliteit van de foto

Om ervoor te zorgen dat de gebruikers van de app geen slechte kwaliteitsfoto's proberen te uploaden en matchen, hebben we hier een controle voor ingebouwd. Wanneer we te maken hebben met een nieuwe salamanderfoto, dan gaan we deze een score geven op basis van hoe goed de kwaliteit van de foto is. Dit kan ofwel "goed", "gemiddeld" of "slecht" zijn.

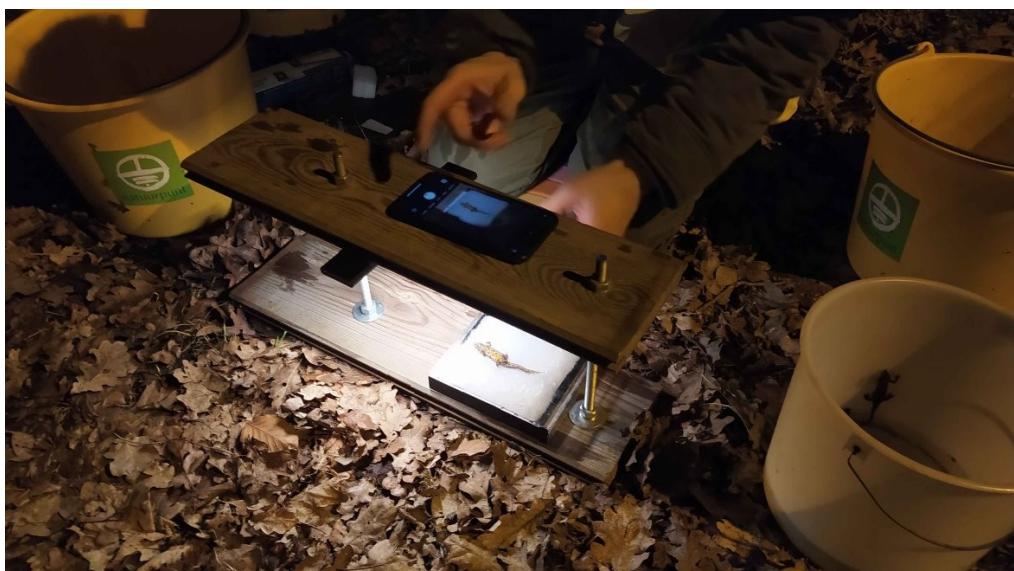
Uiteraard hangt de kwaliteit van de foto af van hoe goed stappen zoals de Pose Estimation, de Color Segmentation en de Haar Cascade het doen, daarom kunnen we de kwaliteit uitdrukken aan de hand van de output van deze stappen. We hebben ervoor gekozen om met volgende definities te werken:

1. Goed: De Pose Estimation werkt goed, zoals gedefinieerd in 2.4.1. De Haar Cascade detecteert minstens drie stippen.
2. Gemiddeld: De Pose Estimation werkt niet, maar de methode via Color Segmentation werkt wel goed. De Haar Cascade detecteert minstens drie stippen.
3. Slecht: De Pose Estimation werkt niet én de methode via Color Segmentation werkt ook niet en/of de Haar Cascade detecteert strikt minder dan drie stippen.

Als de kwaliteit van de foto slecht is, laten we de gebruiker een nieuwe foto maken of uploaden van de salamander in kwestie. Wanneer de kwaliteit gemiddeld is, geven we een waarschuwing dat de kwaliteit van de foto niet ideaal is, maar gaan we er toch mee verder in heel het proces.

## 4 Tegengekomen problemen

We hadden tijdens het project verschillende *soorten* foto's tot onze beschikking, waarbij telkens de belichting of achtergrond anders was. Dit probleem zorgde ervoor dat we stipdetectie niet konden verwezenlijken met klassieke methoden zoals thresholding. In zekere mate beïnvloedt dit ook het Pose Estimation model dat we trainen op foto's van oudere jaren. Om dit probleem te voorkomen, heeft Em. prof. dr. Nick Schryvers achter de schermen gewerkt aan een nieuwe opstelling om zo de salamanders elk jaar op een gelijkaardige manier te fotograferen.



Figuur 24: Opstelling om een salamanderfoto te maken

## 5 Mogelijke uitbreidingen

We geven in deze sectie een lijst met mogelijke uitbreidingen voor dit project. We starten met een paar algemene uitbreidingen en gaan daarna verder met meer specifieke uitbreidingen.

- Toelaten dat foto's die reeds verwerkt zijn, opnieuw verwerkt en gematcht kunnen worden. Dit is vooral handig om aanpassingen aan de code uit te testen met foto's die reeds op de server staan.

- Uitbreiding van de applicatie naar andere salamanders of hagedissen: zolang het dier soortgelijke stippen heeft, zou een proces gelijkaardig aan wat we in dit project gebruikt hebben, moeten kunnen werken om hetzelfde doel te bereiken.

## 5.1 Trainable Weka Segmentation

Trainable Weka Segmentation [15] is een methode om stippenpatronen te detecteren. We hebben besloten om deze methode niet te gebruiken omdat van drie redenen, maar onderzoekers in de toekomst kunnen dit wel uitproberen. Ten eerste, rekening houdend met de vele false positives die voorkwamen bij klassieke stipdetectie door kleine schaduwen en allerlei vuil dat voorkwam op de huid van de salamander, leek de segmentation-aanpak (waarbij de afbeelding wordt ingedeeld in vlakken, wat bij ons zou neerkomen op een groot vlak met de huid, met kleine vlakken erin voor de stippen) ook bijzonder vatbaar voor false positives. Ten tweede bleek de software bijzonder gekoppeld te zijn aan het ImageJ programma: integratie met een ander programma was enkel mogelijk door zelf software te schrijven om de koppeling te maken, wat onnodig tijdrovend had geweest. Ten derde waren de resultaten van de Haar Cascade al voldoende accuraat voor onze doeleinden.

## 5.2 Meest prominente stippen

Onze huidige opstelling om de salamanderbuikjes te fotograferen maakt gebruik van een jewelcase waar de salamanders in worden gestoken. Het zou het kunnen dat door het platdrukken van de salamanders er meer stippen aan de zijkant zichtbaar zijn dan op oudere foto's waarbij we deze opstelling nog niet gebruikten. We zijn ons niet bewust of dit een groot of klein probleem is maar een mogelijke oplossing zou kunnen zijn dat we niet alle stippen gebruiken om te matchen maar enkel de meest prominente stippen.

We hebben al geprobeerd om de  $x$  stippen te gebruiken die het dichtste liggen bij de gemiddelde positie van de stippen. Hierbij hadden we verschillende keuzes van  $x$  gemaakt die variëren tussen 5 en 15. Deze aanpak gaf echter geen betere resultaten. Een meer robustere aanpak met de mediaan (in plaats van het gemiddelde) en de Mahalanobis afstand (in plaats van de Euclidische afstand) kan zeker uitgeprobeerd worden, alsook andere methoden om de meest prominente stippen te detecteren.

We hebben hiernaast ook geprobeerd de buitenste stippen te detecteren via het *convex hull* algoritme [16], waarbij we de stippen op de zijkant van de salamander konden identificeren met de hoekpunten van de regelmatige veelhoek rond de stippen. Hierbij kregen we echter hetzelfde probleem als voorheen, waarbij de methode lijkt te werken op bepaalde foto's en goede resultaten geeft maar niet meer werkt op andere foto's. Verder onderzoek kan hier dus ook zeker worden gedaan.

### 5.3 Pose Estimation model

Er zijn nog een aantal verbeteringen die kunnen doorgevoerd worden met betrekking tot het Pose Estimation model.

- Het pose estimation model opnieuw trainen met de nieuwe foto's van 2025: in de nieuwste setup waarmee foto's genomen wordt, presteert het huidige model (getraind met data t.e.m. 2024) soms minder goed. Hoe beter de training data overeenkomt met de gebruikte setup, hoe beter het model zal presteren.
- Een lichtere versie van het pose estimation model: pose estimation is veruit het meest computationeel intensieve deel van de fotoverwerking. Deeplabcut voorziet de mogelijkheid om een aantal kleinere soorten modellen te gebruiken, die potentieel minder krachtige hardware nodig hebben en/of sneller de pose estimation voltooiën.

### 5.4 Quality of Life aanpassingen in de app

Er zijn nog een aantal aanpassingen die de app een pak gebruiksvriendelijker zouden maken.

- De eigenschappen (naam, datum en locatie) van een individu of sighting aanpasbaar maken in de app.
- Sorteren van de weergave van individuen en sightings op naam (alfabetische volgorde, datum laatst gezien, etc.).
- Locatie en datum automatisch extraheren uit de metadata van nieuwe afbeeldingen indien aanwezig. Voor locatiedata zal dit neerkomen op het zoeken van de reeds gedefinieerde locatie waarvoor de coördinaten van de nieuwe foto binnen een bepaalde straal van de coördinaten van de gedefinieerde locatie liggen.
- Toelaten dat meerdere foto's van dezelfde salamander op hetzelfde tijdstip worden ingestuurd.

## 6 Conclusie

We leveren met dit project een app/website af, waarmee we proberen om individuen van de kleine watersalamander te identificeren op basis van het stippenpatroon op de buik. Dit doen we aan de hand van een aantal stappen. Eerst bepalen we een virtueel skelet van de afbeelding, hiermee kunnen we dan de buik van de salamander isoleren. Stipdetectie gebeurt dan op enkel de geïsoleerde buik waarna we het stippenpatroon gaan rechttrekken en dit gaan vergelijken met andere stippenpatronen. Heel dit algoritme zit verwerkt in een gebruiksvriendelijke app/website. Zoals al eerder aangehaald, zijn er nog veel verbeteringen mogelijk waarbij wij geloven dat onze aanpak al een zeer goede basis kan zijn om op verder te werken.

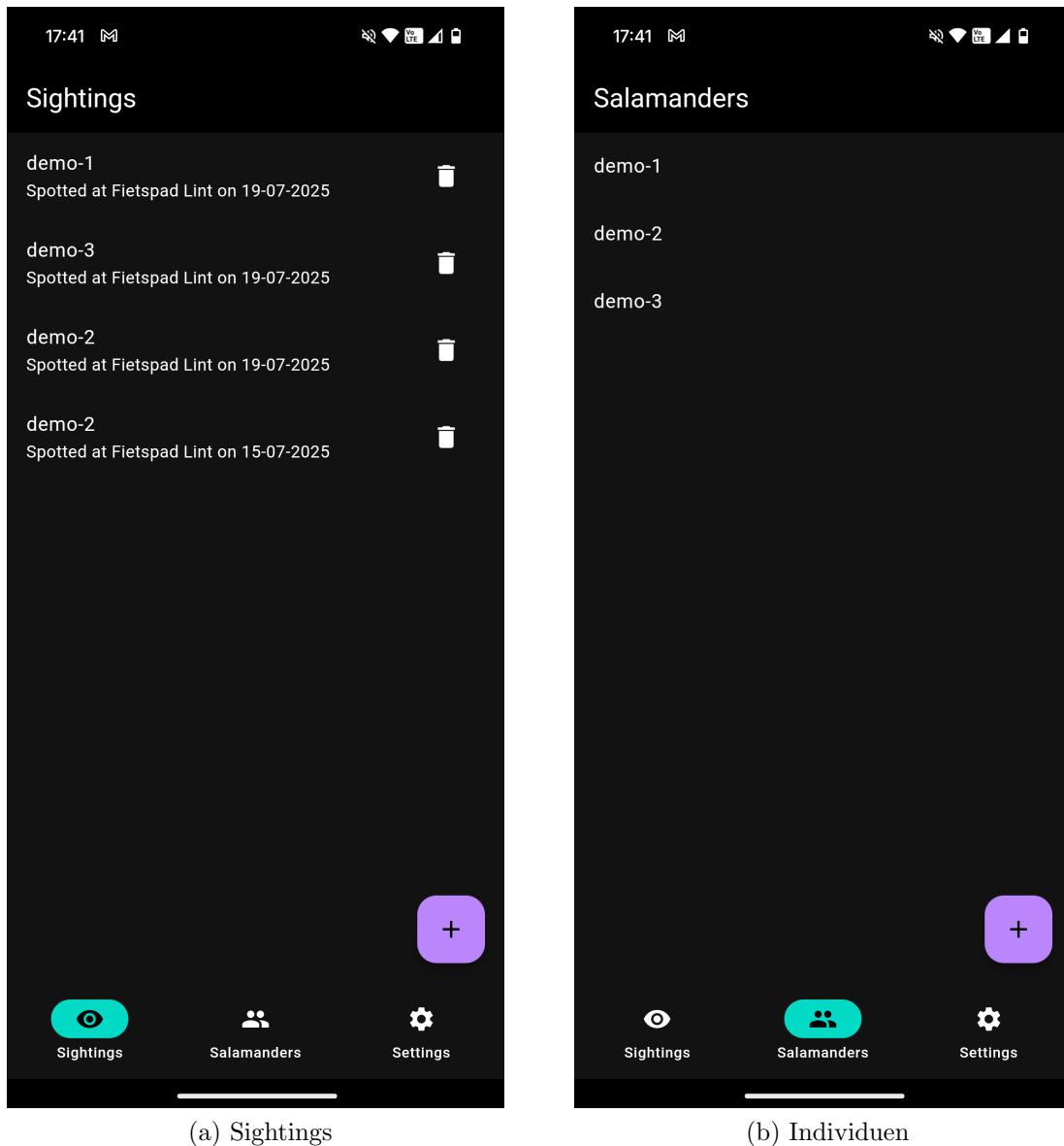
## 7 Referenties

- [1] Ravon. (n.d.). Kleine watersalamander. Geraadpleegd op 29/02/2024.  
<https://www.ravon.nl/Soorten/Soortinformatie/kleine-watersalamander>
- [2] Wikipedia. (n.d.). Thresholding (image processing). Geraadpleegd op 09/04/2024.  
[https://en.wikipedia.org/wiki/Thresholding\\_\(image\\_processing\)](https://en.wikipedia.org/wiki/Thresholding_(image_processing))
- [3] Wikipedia. (n.d.). Otsu's method. Geraadpleegd op 09/04/2024.  
[https://en.wikipedia.org/wiki/Otsu%27s\\_method](https://en.wikipedia.org/wiki/Otsu%27s_method)
- [4] Encord (n.d.). Image Thresholding in Image Processing. Geraadpleegd op 11/07/2024.  
<https://encord.com/blog/image-thresholding-image-processing/>
- [5] LearnOpenCV (n.d.). Blob Detection Using OpenCV ( Python, C++ ). Geraadpleegd op 18/07/2024.  
<https://learnopencv.com/blob-detection-using-opencv-python-c/>
- [6] Medium. (Februari 2021). Color Image Segmentation - Image Processing. Geraadpleegd op 20/07/2024.  
<https://mattmaulion.medium.com/color-image-segmentation-image-processing-4a04eca25c0>
- [7] Mathis, A., Mamidanna, P., Cury, K.M. et al.(2018) DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. Nat Neurosci 21, 1281–1289  
doi: 10.1038/s41593-018-0209-y
- [8] DeepLabCut. (n.d.). Geraadpleegd op 20/03/2024.  
<https://www.mackenziemathislab.org/deeplabcut>
- [9] Jones. M. & Viola. P. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features.  
<https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf>
- [10] Medium. (Augustus 2023). OpenCV: Morphological Dilatation and Erosion. Geraadpleegd op 22/07/2024.  
<https://medium.com/@sasasulakshi/opencv-morphological-dilation-and-erosion-fab65c29efb3>
- [11] Github OpenStitching. (Juni 2024). lir. Geraadpleegd op 06/08/2024.  
<https://github.com/OpenStitching/lir>

- [12] Groth, J. E. (1986). A pattern-matching algorithm for two-dimensional coordinate lists. *The Astronomical Journal*, Volume 91, Number 5.  
doi: 10.1086/114099
- [13] Arzoumanian, Z., Holmberg, J. & Norman, B. (2005). An astronomical pattern-matching algorithm for computer-aided identification of whale sharks *Rhincodon typus*. *Journal of Applied Ecology*, 42 , 999–1011.  
doi: 10.1111/j.1365-2664.2005.01117.x
- [14] Wikipedia. (n.d.). Shoelace formula. Geraadpleegd op 29/07/2024.  
[https://en.wikipedia.org/wiki/Shoelace\\_formula](https://en.wikipedia.org/wiki/Shoelace_formula)
- [15] ImageJ. (n.d.). Trainable Weka Segmentation. Geraadpleegd op 10/04/2024.  
<https://imagej.net/plugins/tws/>
- [16] Wikipedia. (n.d.). Convex Hull. Geraadpleegd op 30/07/2025.  
[https://en.wikipedia.org/wiki/Convex\\_hull](https://en.wikipedia.org/wiki/Convex_hull)

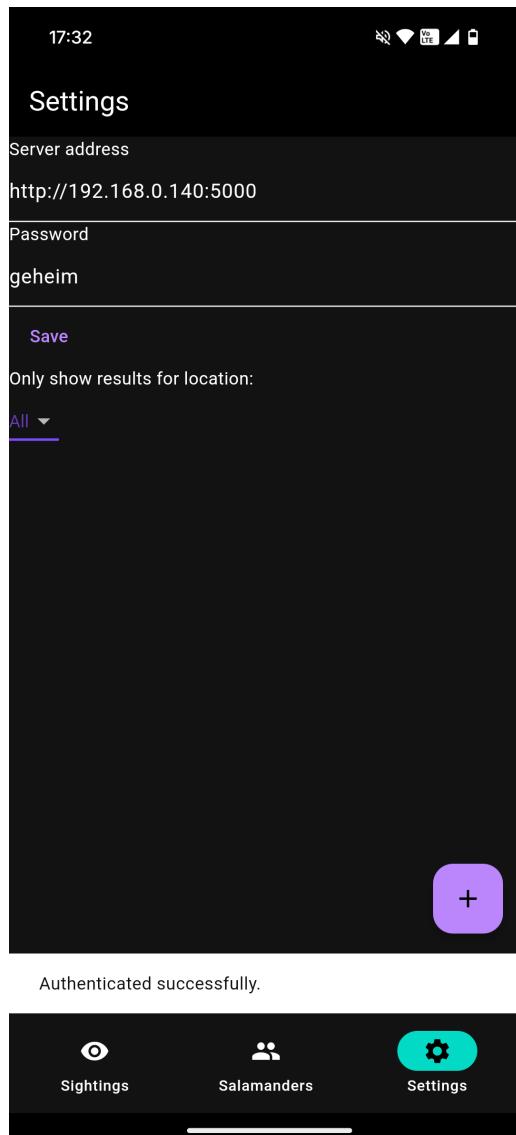
## A Handleiding voor gebruik van de app/website

Op onderstaande afbeelding staat het hoofdscherm: onderaan staan knoppen om te wisselen tussen Sightings (momenten waarop eender welke salamander is gespot), Salamanders (alle individuele salamanders die we in de database hebben), en Settings.



## A.1 Instellen

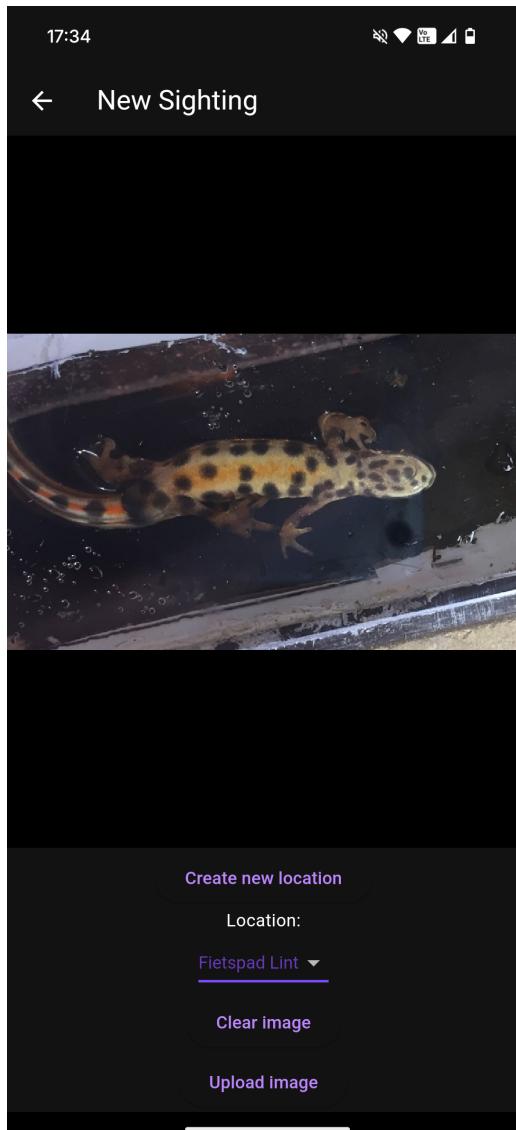
Eerst moet er verbinding gemaakt worden met de server. Op de website gebeurt dit automatisch, maar op de app moet in de instellingen nog het IP-adres of de domeinnaam van de server worden ingesteld, alsook de netwerkpoort. Daarnaast, om zelf foto's te kunnen uploaden, moet ook het wachtwoord worden ingevuld. Beide worden opgeslagen op het apparaat van de gebruiker, dus moeten niet elke keer opnieuw worden ingevuld.



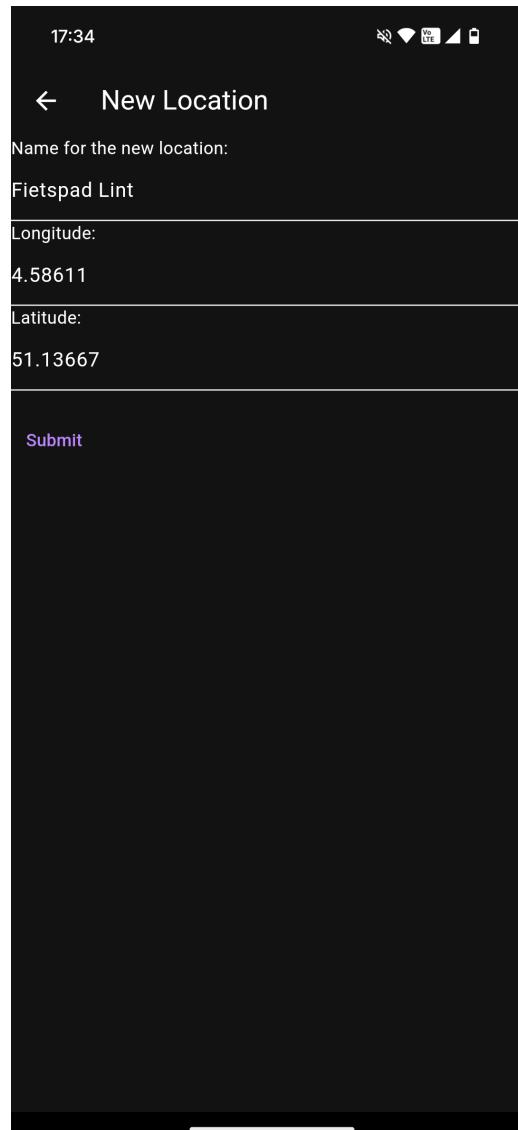
Figuur 25: Na het invullen en indienen van de juiste gegevens is de authenticatie geslaagd.

## A.2 Foto indienen

Om een foto in te dienen (met de paarse + knop) kan oftewel een nieuwe foto worden gemaakt, oftewel een bestaande foto worden geüpload. De foto kan aan een locatie worden toegewezen: deze kan worden aangemaakt als deze nog niet bestaat. Na het uploaden wordt de foto verwerkt door de server. Dit duurt doorgaans een vijf- tot tiental seconden.

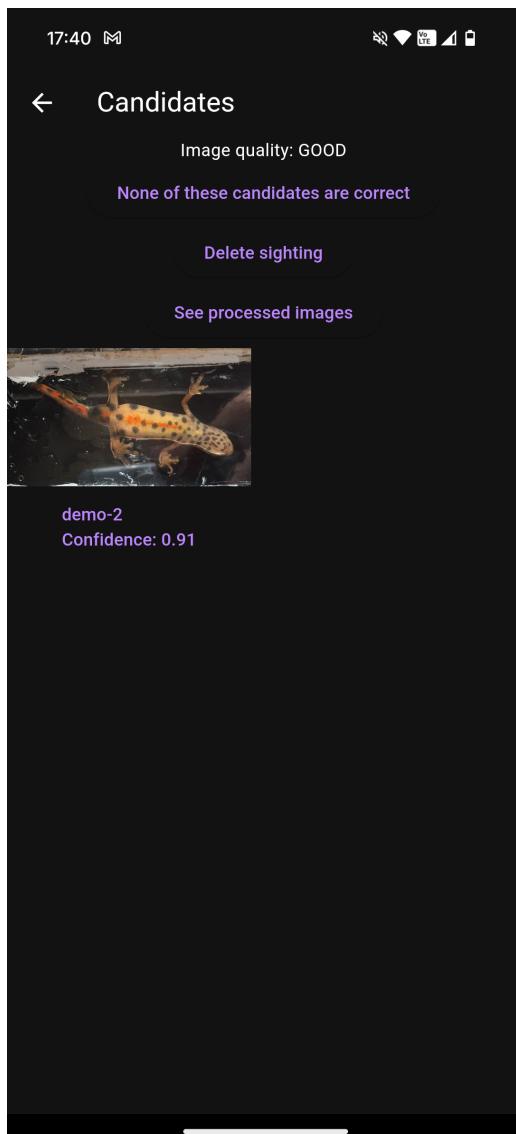


(a) Een foto klaar om te uploaden

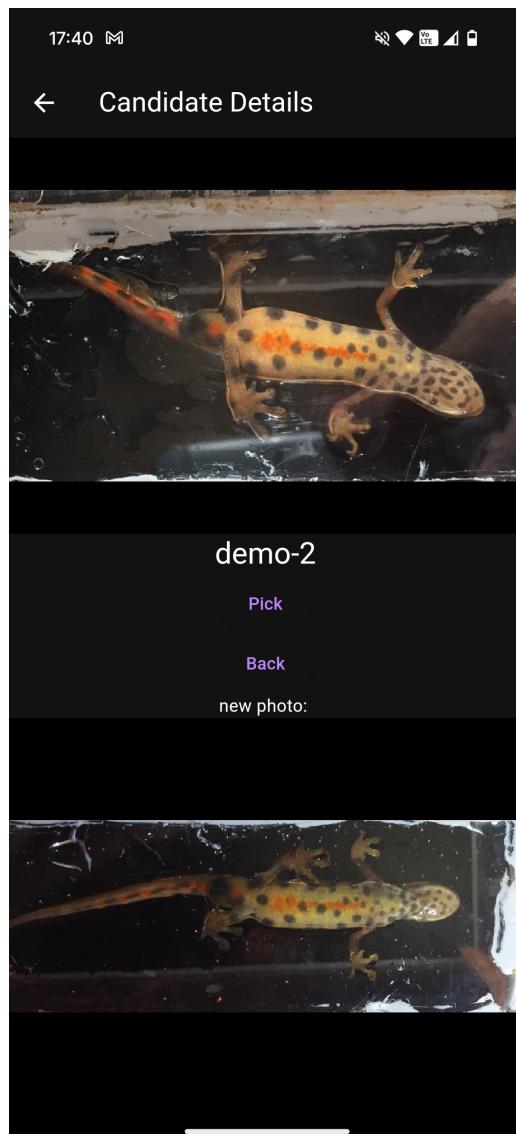


(b) Een nieuwe locatie wordt aangemaakt

Na het verwerken zullen er een aantal bestaande individuen worden gesuggereerd, met telkens een score tussen nul en één dat aangeeft hoeveel vertrouwen het programma heeft dat dit het individu is op de foto. Ook wordt de ingezonden foto getoond, om eventueel manueel vergelijken gemakkelijker te maken. In geval van een foutieve inzending kunnen we de inzending verwijderen, en we kunnen altijd de tussenstappen van het matching-proces zien bij “See processed images”.



(a) Er is één mogelijke match gevonden met 91% zekerheid

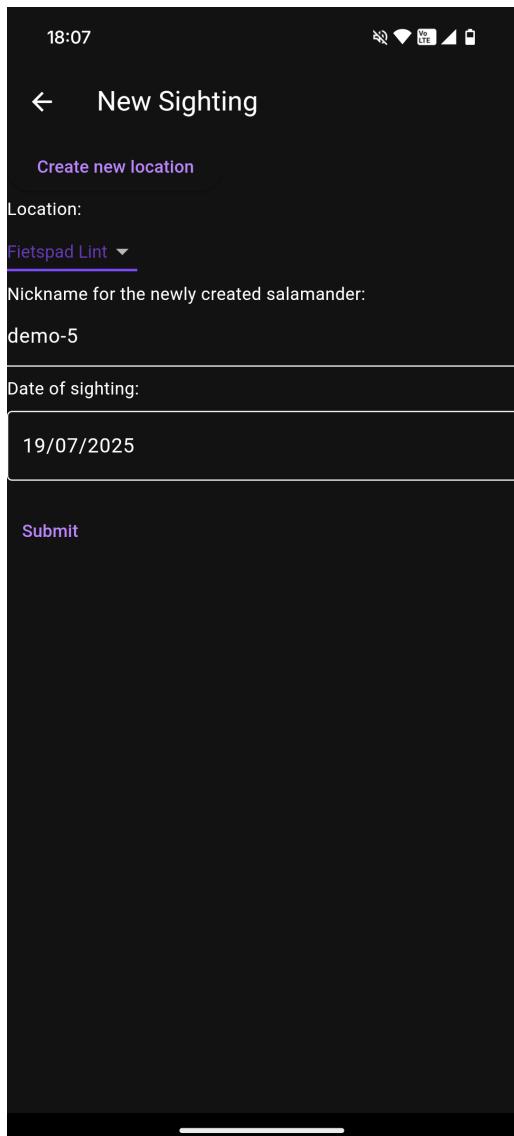


(b) We kunnen manueel nakijken of de patronen overeenkomen

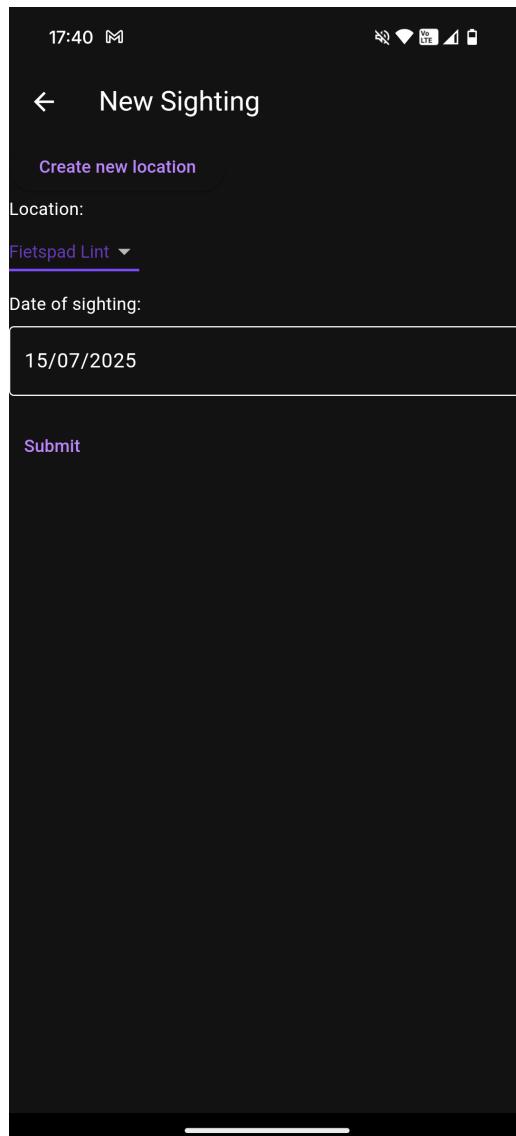
De gebruiker kan dan een van de voorgestelde individuen selecteren, of een nieuw individu aanmaken.

### A.3 Nieuw individu aanmaken/matchen met bestaand individu

Bij het maken van deze beslissing kunnen we ook de datum meegeven waarop de salamander is gespot. Een nieuw individu krijgt ook een naam.

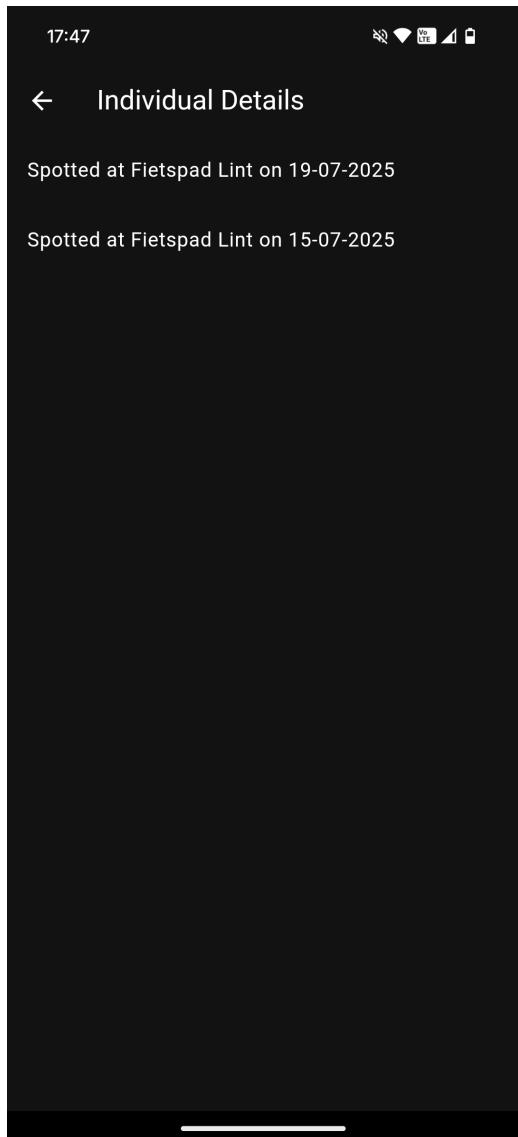


(a) We vonden geen goede match, dus we maken van deze foto een nieuw individu

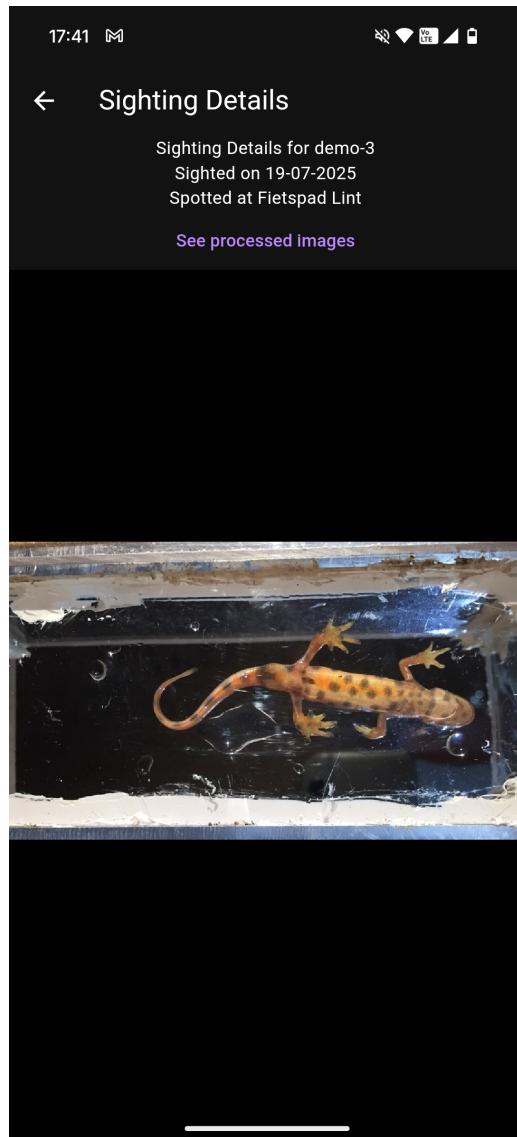


(b) We vonden een goede match, dus we koppelen deze foto aan een bestaand individu

Ten slotte komen we terug op het hoofdscherm. Als we daar nu het individu selecteren waaraan net een nieuwe foto gematcht is, zien we inderdaad dat deze nu twee keer gespot is.



(a) Dit individu is twee keer gespot



(b) Één van de foto's van dit individu