The Story of Jigsolve

Web Research: Proof of Concept

Ergens in november/december

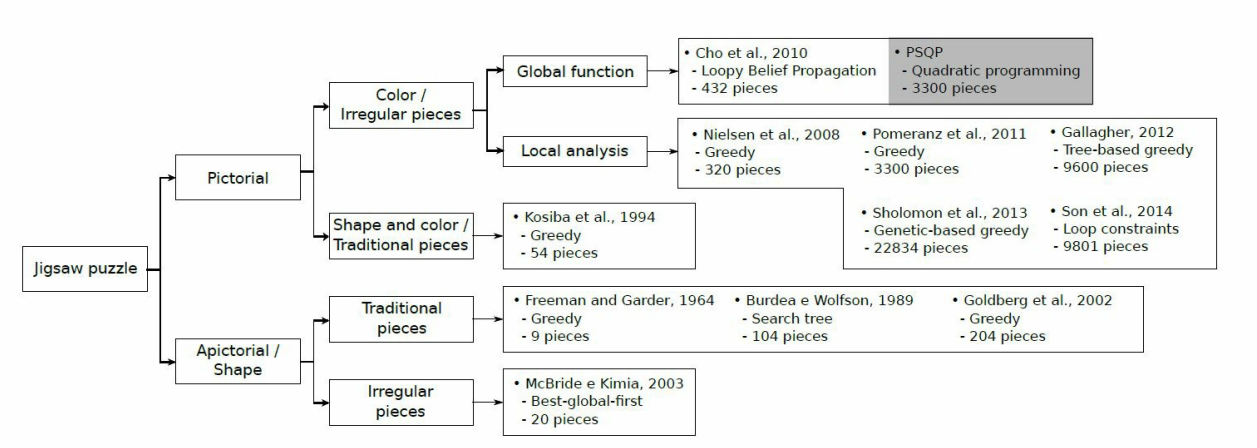
Dit idee is geboren toen ik met mijn vriendin een puzzel aan het maken w as. Het vlotte niet zo goed en na even paste geen enkel stukje meer. Toen kw am het in me op: zou er een app bestaan die me kan helpen met deze puzzel? Na eens snel te kijken op de Play Store vond ik niet direct iets dat er legitiem uitzag, dus dacht ik dat het misschien zelf kon maken.

> De puzzel van die bew uste avond
>
> Status: Niet af, ligt op de kast

Mijn idee w as dus geboren:

> Kan ik een app/programmaatje maken dat mij kan helpen puzzels op te lossen?

Een beetje later die w eek begon ik op te zoeken over puzzels en het oplossen ervan door computers. Ik ontdekte dat er veel verschillende manier w aren om dit probleem aan te pakken. Ze w aren dan ook allemaal (meerdere keren) getest door mijn (w at slimmere) collega's, medepuzzelaars.



> Boom die de verdeling in het domein toont.
>
> Bron: *F. Andalo et al. [PSQP – Puzzle Solving by Quadratic Programming] p.3*

Een hele tijd gaat voorbij en ik vergeet er aan te werken. De feestdagen en kerstvakantie vliegen voorbij en plots is het 9 januari: tijd voor onderzoek!

Het idee was dus om te proberen iets te maken dat het dagelijkse leven van een puzzelaar makkelijker kan maken. Dus ging ik kijken naar oplossingen voor pictoriële stukken. Dit zijn stukken met een afbeelding/grafiek er op. Voor gewoon gevormde puzzelstukken staat op bovenstaande afbeelding maar 1 paper gegeven. Er staat ook bij wat voor algoritme ze gebruiken om de puzzel op te lossen: greedy.

Greedy algoritmen

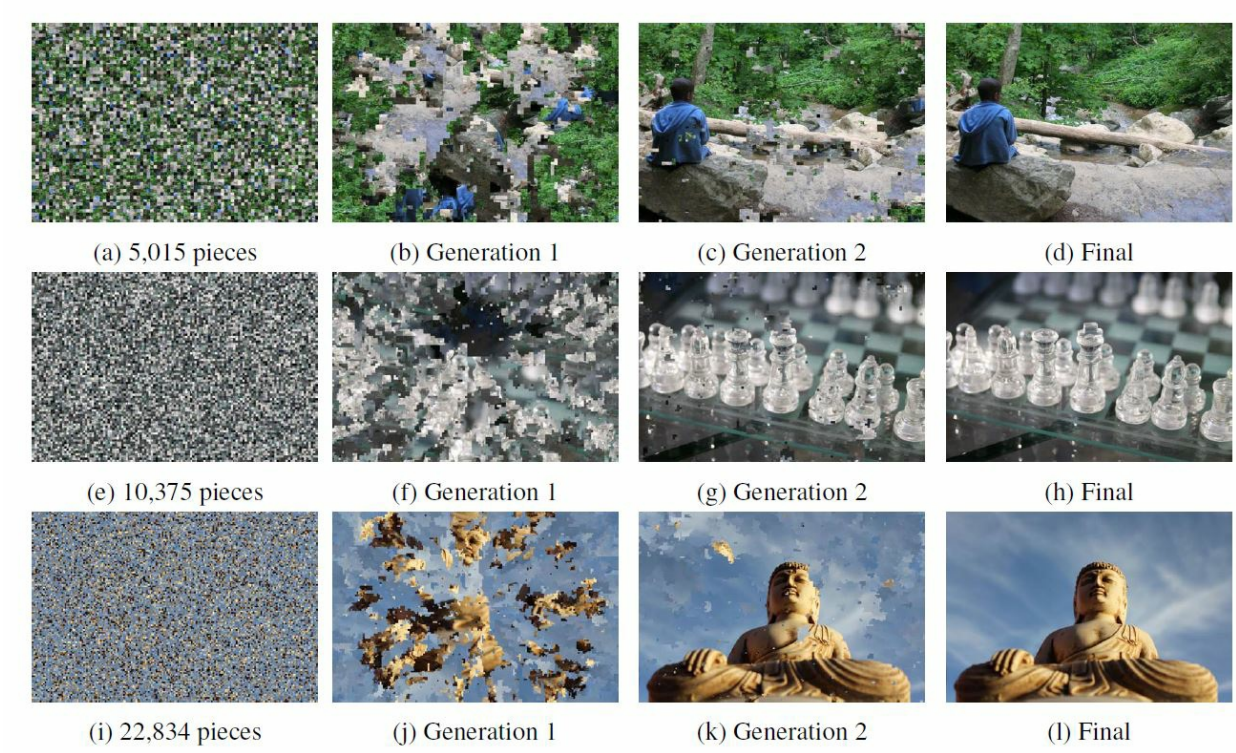
Ik begon de paper te lezen en probeerde het te snappen, maar het lukte niet. Toen besefte ik dat ik niet eens wist wat een greedy algoritme was. Een paar uur later had ik gelezen over wat voor algoritmen het juist zijn, graph coloring en MST's (minimum spanning trees).

Met die informatie las ik de paper nog een keer en het ging al vlotter, maar ik kon het niet volledig vatten: er werd wiskunde in gebruikt die ik ofwel niet kende ofwel te ver weg zat dat het te veel moeite zou zijn om het terug op te frissen.

Zo keek ik verder op het lijstje en zag ik *genetic-based greedy* staan. Dit bleek een combinatie van een greedy algoritme en een genetisch algoritme. Weer iets dat ik nog nooit gezien had!

Genetische algoritmen

Door geleerd te hebben van mijn fouten, begon ik eerst met research naar deze soort van algoritmen. Kort gezegd zijn het algoritme die beginnen met een willekeurige start-set, waarop ze verschillende functies op toe gaan passen (fitness, crossover, mutatie,…) om een nieuwe, hopelijk betere generatie te genereren. Het is dus gebaseerd op de biologische genetica van alle leven op Aarde. Door telkens de beste oplossingen uit die generatie te bevoordelen en te gebruiken voor 'voortplanting' creeert men telkens betere en betere oplossingen die meer en meer de oplossing van de puzzel zullen benaderen.



> Voorbeeld van hoe een genetisch algoritme een puzzel oplost generatie op generatie
>
> Bron: *Sholomon et al. [A Genetic Algorithm-Based Solver for Very Large Jigsaw Puzzles] p.7*

Net zoals bij de vorige paper was dit te ver boven mijn pet en had ik de tijd niet om dit te gaan bestuderen en uit te dokteren. Op zoek naar een iets makkelijkere oplossing dus!

Hopelijk makkelijkere oplossingen

Ik ging eens kijken op github om te zien of er mensen waren die dit eerder geprobeerd hebben. Ik vond deze repo en daar stonden twee oplossingsmogelijkheden. Met lineair programmeren of met quadratisch programmeren. Na dit weer allebei kort op te zoeken en over te leren, begreep ik de basisprincipes wel, maar de paper die ik vond van quadratisch puzzels oplossen *(F. Andalo et al. [PSQP])* was weer te wiskundig en liet deze oplossingen dus weer links liggen.

Een trend in het puzzel-oplossen is dat er steeds grotere puzzels worden opgelost door de verbeteringen die ze maken in hun algoritmen. Nu worden er puzzels van 22 000+ stukken opgelost, terwijl het eerste algoritme een puzzel van 9 stukken kon oplossen (1964).
Dat deed me denken dat ik misschien helemaal naar het begin kon gaan, naar de allereerste algoritmen, omdat deze veel minder stukken aanpakten en dan toch zeker makkelijker zouden zijn? Het vroegste algoritme waaraan ik kon geraken was 1 uit 1989 en dat was voor 24 stuks.
Spijtig genoeg was het hier weer mis. Weer werd er gebruik gemaakt van hoge wiskunde en snapte ik er niets van.

Back to the Roots

Al deze tegenslagen lieten me nog eens nadenken over het probleem en wat ik ging doen voor de oplossing. *'Falen is hip'*, maar ik kon toch niet met alleen een paar papers binnenwandelen op de dag van het examen?

Na even na te denken en met puzzels te spelen, kwam ik op het idee dat ik helemaal geen puzzel uit het niets moet oplossen. Ik was het punt van mijn onderzoeksvraag vergeten: *Kan ik een app/programmaatje maken dat mij kan **helpen** puzzels op te lossen?*

Als mens maak je ook geen puzzel met alleen de stukjes, je maakt ook gebruik van de afbeelding op de doos, de referentie afbeelding.
Daar draait heel de komende oplossing rond. Het matchen en lokaliseren van puzzelstukjes op de referentie afbeelding.

OpenCV

Voor een ander project was ik al eens in contact gekomen met OpenCV en ik wist dat het een heel goeie library is om beeldverwerking te doen. De naam zeg het zelf: Open Computer Vision. Plus, het kon in Python en dat moest ik nog opfrissen voor mijn stage.

Al snel vond ik de functie cv::matchTemplate() en ik was er van overtuigd dat ik hiermee verder kon. Deze functie neemt een patroon (vb. deel van afbeelding) en begint deze vervolgens vanaf linksboven over de bron afbeelding te schuiven. Op het einde van de rit checkt de functie waar de grootste overeenkomst was en geeft die terug als locatie van de match.
Al snel had ik hiervoor een klein scriptje geschreven en kon ik afbeeldingen in afbeeldingen terugvinden. (searchPiece.py)

Intermezzo: Faaldroid

Als intermezzo, waarschijnlijk onder invloed van slaaptekort, besloot ik dat het tof zou zijn om mijn app ook echt in een app te gieten voor Android. Zoals de titel van dit stuk laat uitwijzen, lukte dit niet echt.

Om pythons scripts te builden tot Android-apps waarbij je ook gebruik kan maken van het toestel en het besturingssysteem, gebruikte ik Kivy:

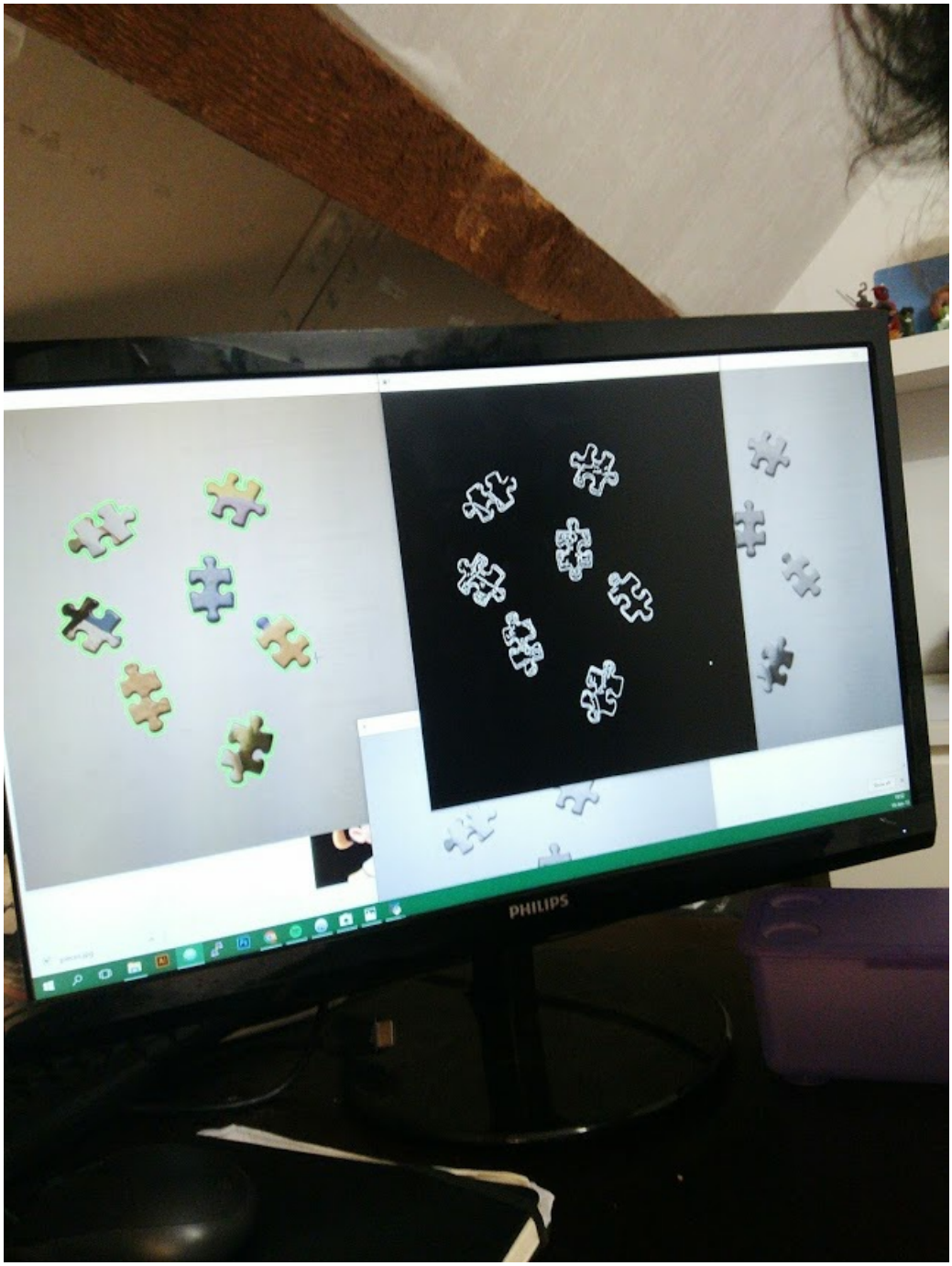
> Open source Python framework for rapid development of applications that make use of innovative user interfaces, such as multi-touch apps.

Rapid development was toch niet waar bij mij en ik besloot (jammer genoeg iets te laat) na eindelijk 'Hello World!' op mijn gsm te krijgen dat deze route het niet waard zou zijn. Zeker omdat ik nog niet alles had.

Puzzelstukken extracten

Een volgend deel van mijn uitwerking was het extracten van puzzelstukken uit een foto. De gebruiker trekt een foto van 1 of meerdere puzzelstukken op een vlakke achtergrond en deze worden dan gezocht in de voorafgegeven referentie afbeelding.

Door gebruik te maken van OpenCV's functie cv::findContours() zou ik alle objecten die op de ingevoerde afbeelding staan, kunnen markeren met een omlijning. Door dan te kijken naar de langste contouren die de functie teruggeeft, zou ik de contouren van de puzzelstukken er uit kunnen filteren.



> Links: de originele afbeelding met de gefilterde contouren van de puzzelstukken
> Rechts/midden: Alle contouren die herkent zijn door cv::findContours()

Met de gefilterde contouren zou ik dan voor elk puzzelstuk op de afbeelding een mask kunnen maken om het puzzelstuk achteraf uit de afbeelding te halen en over te blijven met het puzzelstuk met zwarte achtergrond.

Het vinden van de contouren en het filteren lukte vrij snel. Dan dacht ik dat het maskeren ook vrij makkelijk ging gaan, maar niets was minder waar. Door een gebrek aan goede documentatie voor OpenCV voor Python omtrent maskering heb ik een lange tijd met de handen in het haar gezeten omdat alles wat ik probeerde niet vond. Mogelijke oplossingen in C++ die ik dan vertaalde naar Python werkten niet omdat de versie voor Python niet identiek is aan die van C++.

Laat in de nacht lukte het me dan toch en het bleek een vrij makkelijke oplossing te zijn…

Dit stukje code was de boosdoener:

```
roi = img[y:y2, x:x2]
```

Ik had een paar uur vroeger dit al geprobeerd:

```
roi = img[x:x2, y:y2]
```

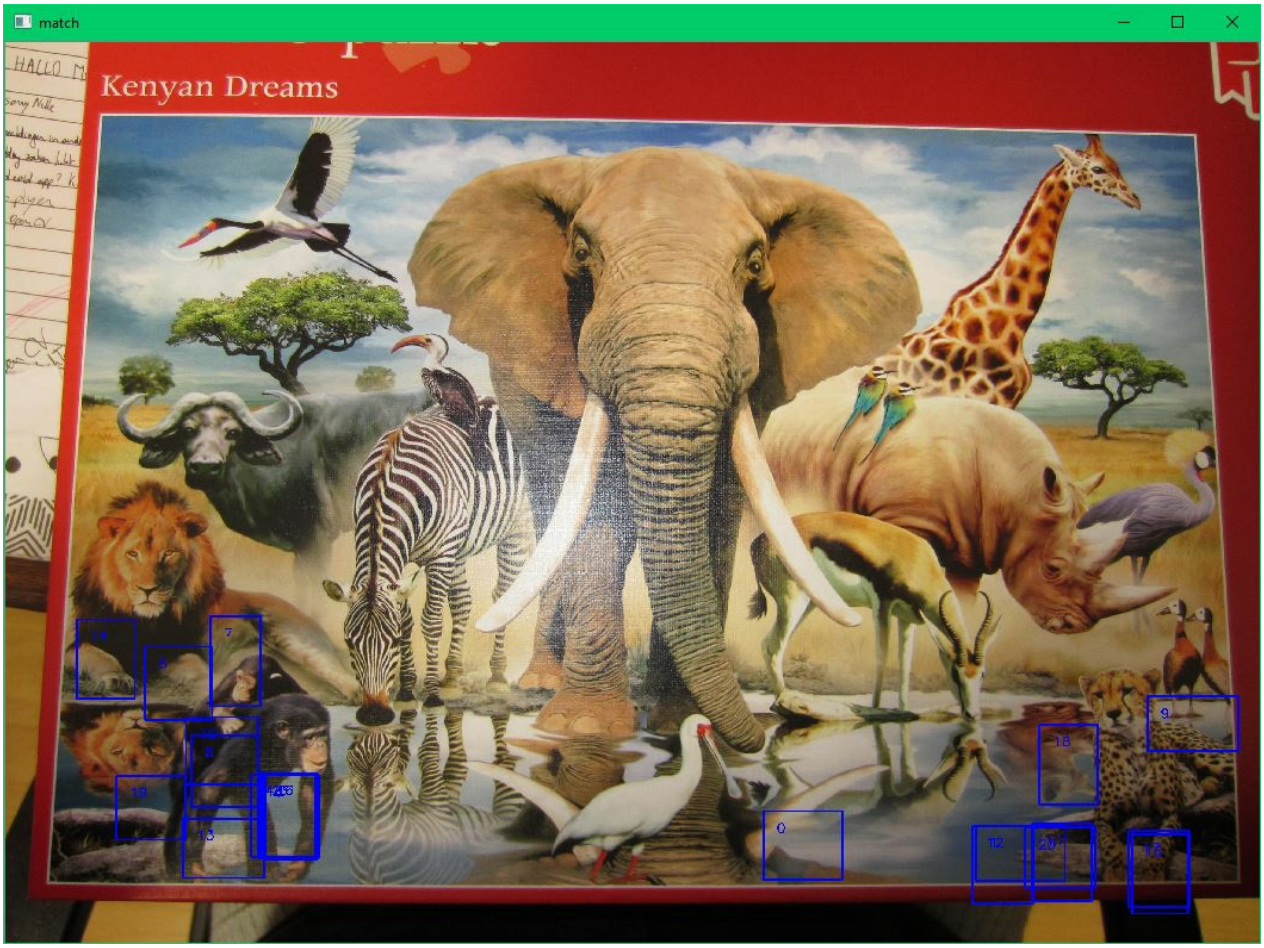
Maar blijkbaar w erkt OpenCV met een omgekeerd coördinatenstelsel…

Uiteindelijk kon ik de puzzelstukken dus w el uit hun afbeelding halen. Uit bovenstaande afbeelding haalde ik volgende puzzelstukken:

Alles samenvoegen

Nu dat ik afbeeldingen kan zoeken in een andere afbeelding en ik puzzelstukken uit andere afbeeldingen kan halen, is het tijd om deze tw ee functionaliteiten met elkaar te laten samenw erken.

Ik dacht door alle puzzelstukken die ik verzamel uit de extractie in de zoek functie te gooien dat het opgelost zou zijn, maar helaas. Heel soms w as het raak, maar hoe algemener het puzzelstuk hoe meer kans op een w illekeurige, foute plek het programma aangaf op de referentie afbeelding.



> Het resultaat van het zoeken van 21 bovenstaande puzzelstukken in de Afrikaanse puzzel
>
> Conclusie: niet zo goed

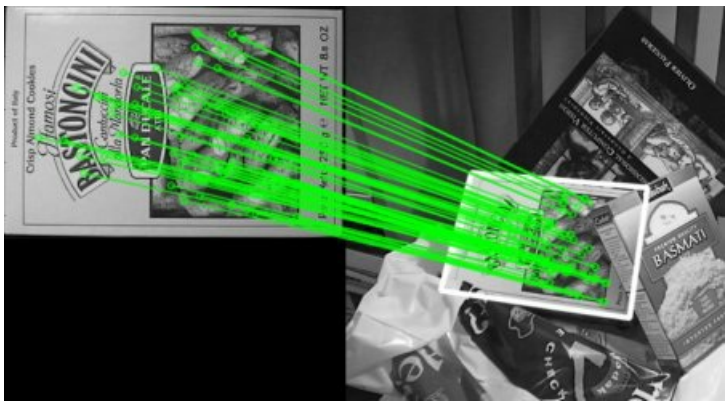
Afgezien van de foute matching, is het ook niet zo duidelijk met zoveel puzzelstukken.

Toen dacht ik w eer even na en schoot me te binnen dat er geen rekening w ordt gehouden met de grootte van de puzzelstukken ten opzichte van de referentie afbeelding. In het echt zouden de puzzelstukken een veel grotere afbeelding vormen en via de camera is het onmogelijk te zien of ze juiste schaal t.o.v. elkaar hebben. Ook zie je dat de referentie afbeelding nog scheef getrokken staat door het perspectief w aarmee de foto getrokken is.

Matchfunctie mk.2: SIFT

Door w at te bladeren op de documentatiepagina's van OpenCV kw am ik het SIFT algoritme tegen. SIFT is een algoritme om objecten te detecteren/matchen en zoals de naam (Scale-invariant feature transform) het al zegt, hoeft er hierbij geen rekening te houden w orden met schaal en eventuele rotatie.

Het algoritme gaat over het te zoeken puzzelstuk en probeert hieruit een aantal merkw aardige punten te halen. Dit doet het ook met de referentie afbeelding en vervolgens w ordt er gekeken of de punten van het puzzelstuk overeenkomen met (een deel van) de punten van de referentie afbeelding.



> Hoe SIFT zou moeten w erken

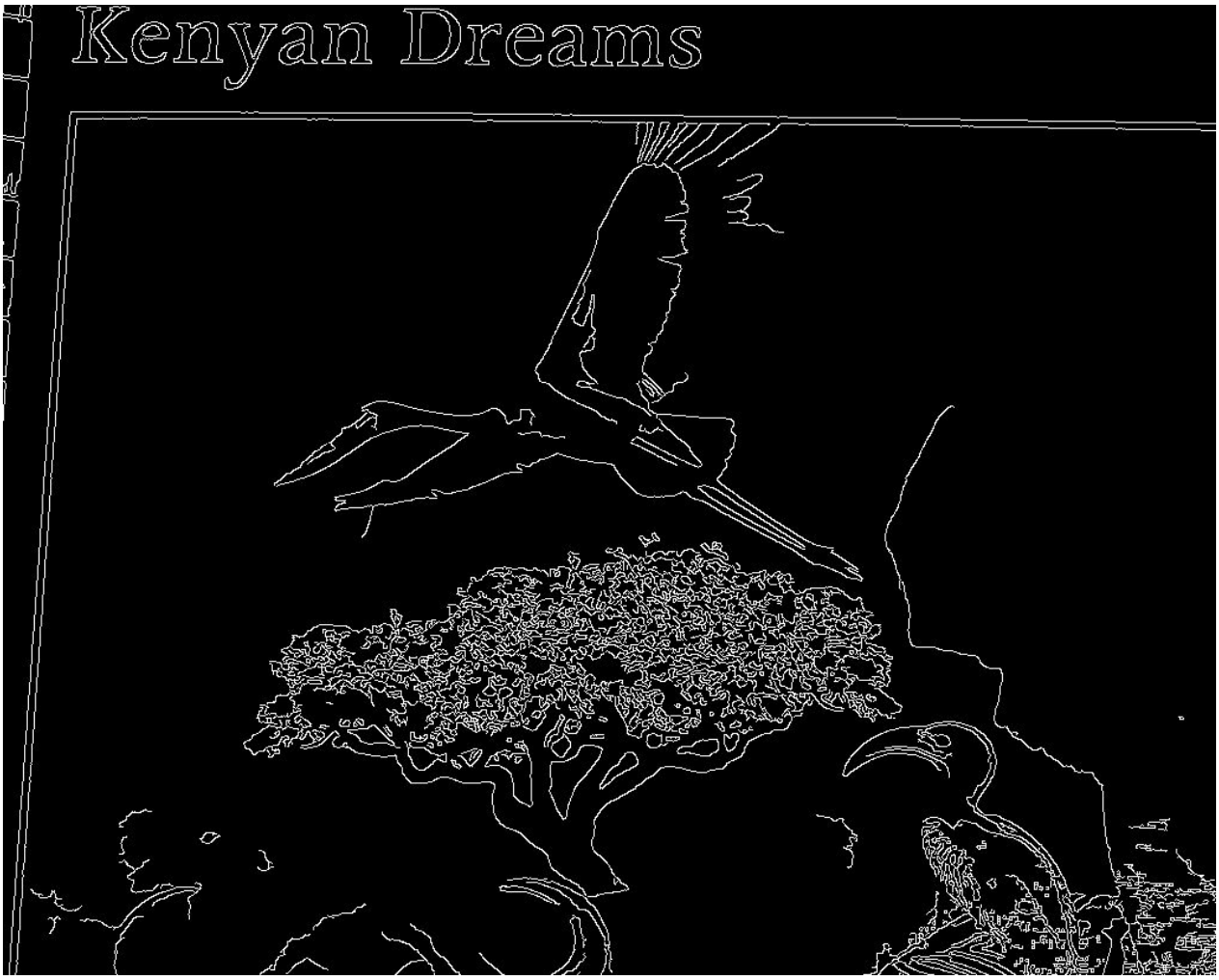
Door iets te w einig tijd en verstaanbaarheid nam ik de uitw erking die op de documentatiepagina stond en paste het aan aan mijn situatie. Voor gew one afbeeldingen (geen puzzelstukken) w erkte het perfect. Het kon bijvoorbeeld heel makkelijk de aap in de linker onderhoek terugvinden.
Wanneer ik het met puzzelstukken probeerde w as het ook mogelijk, maar in veel gevallen w aren er geen gematchte punten tussen de tw ee afbeeldingen.

Ik kon nog 1 ding proberen dat het misschien zou oplossen, of toch op z'n minst zou verbeteren: het rechttrekken van de referentie afbeelding.

Rechttrekken referentie

Dit w as op zich niet zo'n moeilijke opgave, maar w egens tijdsdruk hebben een aantal blogposts op het w eb me toch moeten helpen.

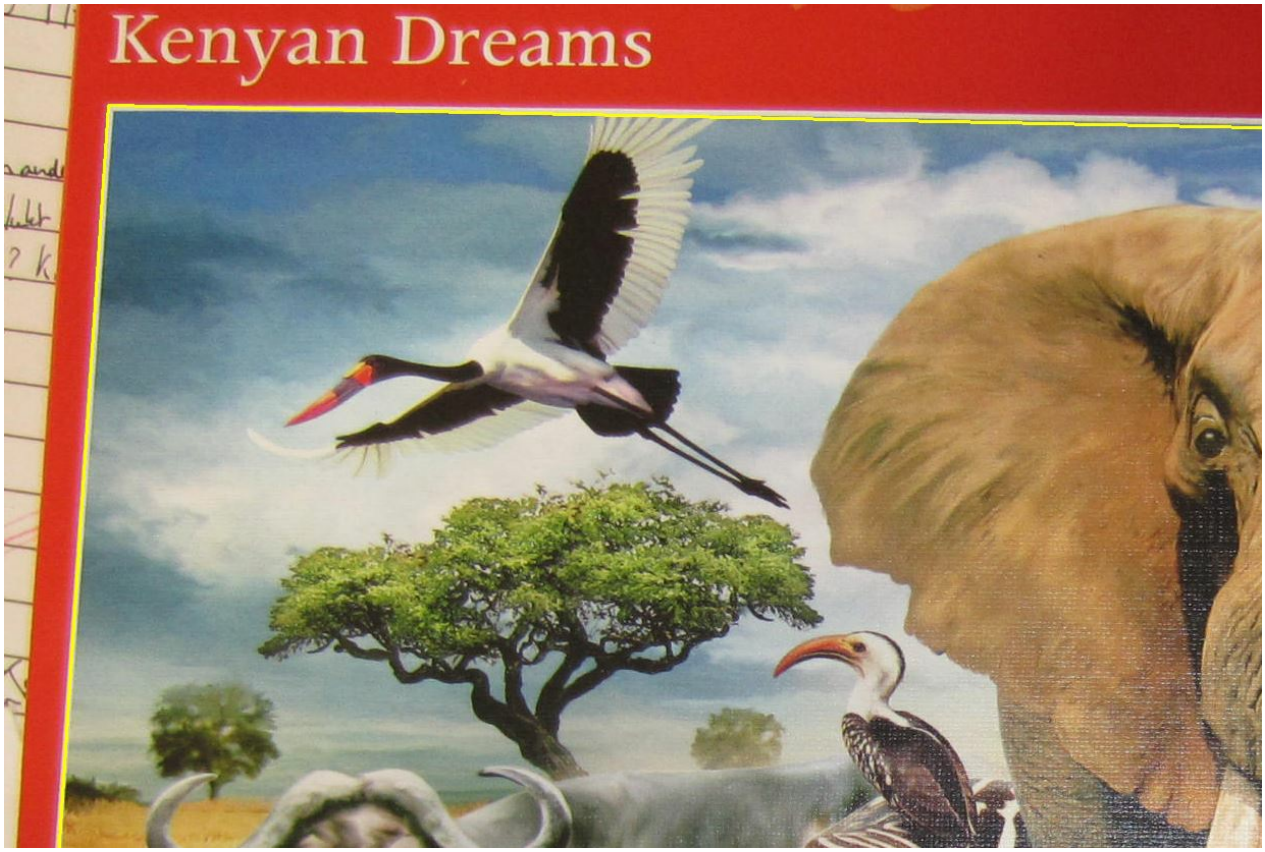
Eerst ga je met een canny edge filter over de afbeeldingen om alle randen in de afbeelding te zoeken.



> Close-up van het resultaat van de canny edge filter

Vervolgens neem je de randen van de canny edge filter en zoek je hierop de contouren met de eerder gebruikte cv::findContours() functie.
Vervolgens filteren w e w eer de allergrootsten er uit. Maar hoe zouden w e nu de grote rechthoek er uit kunnen halen?

Een rechthoek bestaat altijd uit 4 hoeken, dus gaat de contour die deze rechthoek beschrijft ook slecht 4 punten hebben. Als w e daarop nog eens filteren hebben w e de uiteindelijke contour voor de afbeelding:



> De contour is te zien in het geel

Nu moeten w e deze scheve rechthoek alleen nog maar rechttrekken naar de hoeken van de afbeelding en dan hebben de referentie afbeelding recht getrokken!

Hiervoor gebruikte ik de functies van onderstaande blogposts die specifiek over het maken van document scanner gaat.

- https://w w w.pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/
- https://w w w.pyimagesearch.com/2014/09/01/build-kick-ass-mobile-document-scanner-just-5-minutes/

Het resultaat hiervan is dit, een mooie vlakke referentie afbeelding:

Hoe mooi dit allemaal mag zijn uitgedraaid voor de referentie afbeelding, het vinden van puzzelstukken in de afbeelding w erd niet beter.

Conclusie

Ik ben tot op heden nog geen betere oplossing tegengekomen, maar mijn vermoeden is dat het ligt aan de lage resolutie van de foto's, de slechte belichting en de vorm van de puzzelstukken.
Wanneer het gedaan w ordt met digitaal uitgesneden stukken afbeelding en de bron afbeelding is er een groter succes percentage dan w anneer er handgemaakte foto's w orden gebruikt.

Ik vond het zeer tof om eens dieper in iets te gaan en me helemaal toe te w ijden aan de w ereld van het puzzelen.

Eén quote die me bijgebleven is uit het lezen van de papers is de deze van Goldberg uit *A global approach to automatic solution of jigsaw puzzles*:

> The real
> interest in jigsaw puzzle solving is simply that it is a natural and challenging problem that
> catches people's imaginations.

De quote zegt perfect w at ik vond van het opzoeken en onderzoeken over dit probleem. Het is uiteindelijk niet gelukt, maar ik heb er veel uit geleerd en het w as interessant!