Using Alpha trees for traffic sign recognition

S. R. Elhorst (S3264254)

University of Groningen
09/02/2020

Abstract

*In this paper the α-tree (alpha tree) is tested on recognition of traffic signs while maintaining decent speed. α-trees clusters pixels based on the difference in color of the selected tree depth, resulting in a tree of segmentation's. By grouping clusters based on alpha level, eventually a node with traffic sign might be found (if it is present in the image). Every step the segmentation's are checked using a decision tree whether they contain a traffic sign. When looking at the results, the speed of the given implementation is $O(nlogn)$ which is expected from the alpha tree. However the program could be further optimized such that it does not require more time of calculation than a car actually passing the traffic sign. 1024*768 images can be calculated within a few seconds, however for bigger images the implementation is not fast enough when looking for traffic signs. When looking at the recognition it is clear that leakage is a big problem. Leakage is a problem where a chain of pixels causes a merge of clusters that should not and does not merge clusters that should. Using the decision tree a correct segmentation rate of 15% is achieved due to this leakage problem. However when using a Gabor filter this correct segmentation rate is drastically increased to 55%. The segmentation makes use of a simple decision tree, using a different technique would easily increase the correct segmentation. Nevertheless it can be concluded that using a Gabor filter will increase the correct segmentation of an α-tree. The α-tree has the potential to recognize traffic signs but requires more research into preventing leakage (besides Gabor filtering), better efficiency such as parallelization and preventing unfortunate coloring.*

Keywords: Traffic sign recognition; α-tree; Union-Find; Leakage; CIE L*a*b*; Decision tree; Gabor-filter;

1. Introduction

Traffic signs are an important source of information for drivers to drive save. They generally give us warnings, regulations or other information, which are essential to recognize for drivers. However, since drivers are humans, they tend to sometimes forget or miss a traffic sign. Therefore, developing software which is able to recognize traffic signs on the go, can increase the safety of drivers and its surroundings. Most automated traffic sign recognition systems have the problem that they either contain some serious flaws, or are not capable of recognizing real life scenario's.

Usually the recognition systems are very slow, where they make use of for example: the binary partition tree, minimum spanning tree or shortest spanning tree in [Jas11] [WCT] [VV08]. Although these do solve the problem, a driver has usually already passed the traffic sign before recognition. This means that the system is still building a tree and classifying while the traffic law should have been enforced already. Most researches into traffic sign recognition show examples of decent classifiers. However, they only classify the traffic sign itself or only classify a part of the image, where the traffic sign is assumed to be found [YX]. This paper only focuses on the α-tree (alpha tree), neural networks will not be discussed.

In 2011 Jasper Smit [Jas11] did research into traffic sign recognition using various trees, such as: the binary partition tree, shortest spanning tree and salience tree. In his paper the classification rate was decent, but not good enough, for real-time use (around 60%-70% correct classification). Furthermore, the speed of recognition was not quick enough to classify a real-time traffic sign system, assuming images bigger than 1024*768 pixels. The bigger the size of an image the better the quality and thus better classifying results. These problems were a big issue then, nowadays better hierarchical trees are developed and the hardware has improved significantly.

In 2012 Soille. P. developed the alpha tree (usually denoted as α-tree) [OG12], which showed that it was much faster than other hierarchical trees for image segmentation. The α-tree creates a hierarchical structure of an image by merging pixels that have a value that does not differ by more than α. The α value represents the depth of the tree.

When comparing the α-tree to other trees it is visible that the α-trees give much better results [Jas11]. As already mentioned α-trees are an alternative to neural networks [SL] and Adaboost [XB] which are the most common techniques for recognition nowadays.

This document describes Soille. P's α-tree applied to traffic sign recognition. Testing the speed, quality and recognition capabilities of the α-tree itself when applied to traffic signs. Furthermore it explains how Gabor filtering can solve the leakage problem that often occurs when using an α-tree [ZXb].

2. Theory

This section covers the theory that is used within this paper. Understanding the α-tree is key for this research since it is the main theory that is used for traffic sign recognition. The other chapters within this section elaborate on certain parts of the α-tree, such as: optimization, car speed, 4-connectivity, union-find, CIE Lab coloring and decision trees. But furthermore it explains a big issue within alpha trees, namely leakage and how it can be improved using Gabor filtering. But first the binary partition tree is explained, since this gives a better insight into understanding the α-tree capabilities.

2.1. Binary Partition Tree

The binary partition tree is an algorithm that partitions an image into different representations. Creating a tree of segmentation's, where each leave represents a pixel and each node represents a region of pixels. When following the path to the root, eventually the entire image is found. This technique can be used to slice the tree at a certain level of coarseness to find interesting information (Depending on where you are looking for). The binary partition tree in general gives better segmentation than the α-tree [Jas11] however it is much much slower. This makes it inconvenient to use for traffic sign recognition since it has a limited time of calculation.

2.2. Alpha Tree

The α-tree [OG12] can be seen as a dendrogram which creates a hierarchical tree that is mostly used for segmentation of images. It is based upon a measure of dissimilarity between adjacent elements of for example the image intensity. This distance can be used to construct a hierarchy of partitions. The goal is to segment the image into partitions based on certain criteria to find useful information within an image. As the α-tree states it is a tree structure that has the advantage that it creates multiple fine coarse levels instead of a single output. The root of the tree is the entire image, and the leaves are finer levels of the image. Each alpha level the tree segments are merged together to form a higher level root in the tree. As mentioned earlier the α-tree calculates the dissimilarity between adjacent elements such as intensity. Intensity is an absolute measure that is easy to implement when merging because differences between connected-component values is easy to determine. Colored images however have three different values namely: red, green, blue and cannot be compared easily as humans can not distinguish a distance in color using RGB [Pas01]. Chapter 2.3 will go into detail about this problem.

Determining the connected components and thus merging a leave to a node is done using a so called α connected component. Where α represents the difference in intensity value, 0-CC represents the original image, 1-CC connects components that differ in intensity by no more than 1. Figure 1 shows the results of connecting components by a certain α value. The alpha tree is build by allowing a higher difference in α each layer, where 0-CC represents the entire image and the highest value CC are all the segmented areas combined (root of the entire tree).

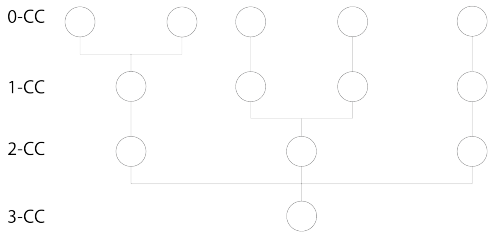

Figure 1: *α connected component alpha tree*

A big advantage of the α-tree is that it is very fast when comparing it to other hierarchical segmentation techniques. This can be seen in the research by Jasper [Jas11] where the α-tree outperforms the binary partition tree significantly. However the segmentation results are a bit less in comparison to the binary partition tree. In Jasper's research different hierarchical tree structures are tested on various examples and resolutions. In his research the binary partition tree, the fast recursive shortest spanning tree and the salience tree (= α-tree) are tested. When looking at his results one can conclude that it is too slow for traffic sign recognition. Recall, for traffic sign recognition speed is also an important factor, the chapter Car Speed goes into detail about this.

The goal of the α-tree is segmentation of groups of pixels, once the tree is build the next step is to find the ground truths equal to the traffic signs. These ground truths can then be tested to see if they contain a traffic sign. The perfect result would be a cluster that contains exactly the entire traffic sign which can be put in the recognizer. A good result would be 11 which contains a perfect yield warning sign.

2.2.1. 4-Connectivity

The standard α-tree has 4 connectivity, meaning that when looking for dissimilarity only the vertical and horizontal angles of distance will be calculated. Another connectivity type

can be diagonally which results into 8 Connectivity. Where the distance is measured for each pixel around the currently selected pixel.

2.3. Car speed

For traffic sign recognition only a limited time is available before an action needs to be performed. Some traffic signs require an instant reaction, such as: a yield warning sign or a stop sign. Others do not require an instant reaction, such as: a sign that is applicable in x amount of meters. The distance to a traffic sign determines the quality. Assuming a standard image of 1920*1080 pixels at a distance of 1km will result into a few pixel that is impossible to recognize. Thus a few assumptions about speed vs pixels are made in this project (based upon example images provided in the appendix). Traffic signs that are further away than 200 meters will be considered low quality. Traffic signs between 200-50 meters are considered decent quality. Traffic signs that are less than 50 meters away are considered high quality. Table 1 shows an overview of the time to recognize a traffic sign. As visible the speed and distance determine the time for recognition. If the distance to the traffic sign is higher, more time is available to recognize a traffic sign. In general the amount of time available to classify a traffic sign is fairly low (between 1 - 10 seconds).

2.4. Union Find based search

A huge part of the α-tree is the union find based search [Tar75]. Each level the α-tree searches for potential clusters that can be merged, where each individual pixel that belongs to that cluster should be updated. Union based find gives each individual pixel a link to a parent, when following this parent a new parent might be found and then again a new parent might be found etc. This chain of parents can be seen as a path, when following this path eventually the root parent is found. Lets say there are two clusters A and B, if the root parent of A is found the parent of root A can be changed to the root parent of B. Resulting in all pixels of cluster A pointing to root parent of B and thus being able to merge two clusters. Figure 2 gives a visualization of the union find algorithm. Where a union is created between A and B, instead of A pointing to itself it will point to the root of B.

2.4.1. Path compression

In order to improve union find based search, path compression can be used. Path compression [THCon] shortens the path of a pixel from parent to parent to parent etc. If there are a lot of clusters that are merged a pixel might point to a parent that is 100 parents deep. With path compression each α level step, the path is shortened by immediately pointing the pixel to the highest root parent of the cluster. This prevents the system to get stuck in long paths, which drastically increases the speed of the algorithm. Figure 3 shows an example of path compression, the black arrows represent the

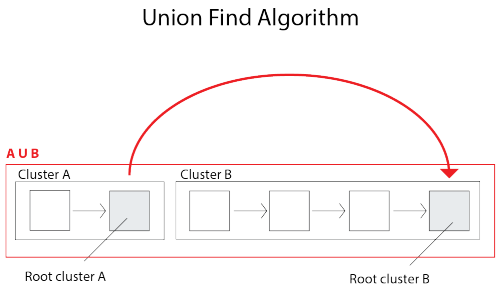

Figure 2: *Union Find*

link to the next parent before path compression and red represents the link after path compression. As visible it makes the linking shorter which leads to a much faster system.

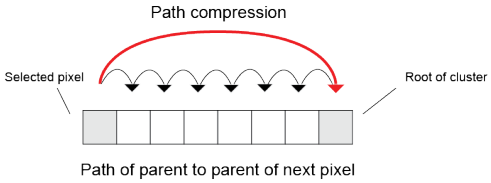

Figure 3: *Path compression*

2.4.2. Ranked based search

Another method that can be used is ranked based clustering [RET84]. Where the clusters are clustered based on the distance of the path of parents. If pixel A for example has a path of 10 parents and cluster B has a path of 100 parents. The root parent of B will point to cluster A instead of the other way around. This will result in a lower depth of the parent path and thus faster speed. Figure 4 shows this principle, as visible Cluster B has a shorter path, when cluster A and B are merged A will point to B instead of B to A.

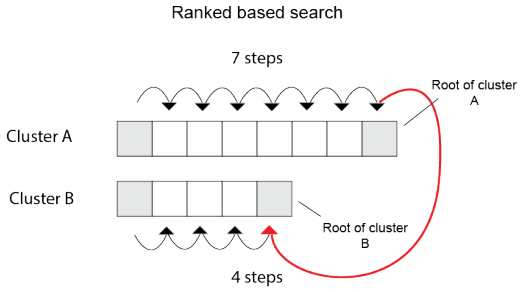

Figure 4: *Rank based search*

Distance to traffic sign (m)	Speed (km/h)	Speed (m/s)	Time for recognition (s)
100	100	27.78	3.6s
200	100	27.78	7.2s
100	100	27.78	3.6s
200	50	13.89	13.8s
10	50	13.89	0.72s
10	20	5.56	1.80s

Table 1: *Time to recognize*

2.5. CIE Lab coloring

The standard coloring for computers is RGB (Red, Green and Blue) values. However humans are not capable of easily recognizing this color space [Sch07] for that reason CIE L*a*b* is used. CIE Lab is a coloring space that is designed for humans in order to determine perceptual difference in color. This color is used for the distance measure within the α-tree, since each level of the α-tree the clusters are merged based on this distance. CIE L*a*b* is a 3d color space that is equal in all directions allowing for the usage of Euclidean distance.

2.6. Leakage

One of the issues with the α-tree is the leakage problem [ZXa], [ZXb]. Leakage is a problem where connected components near edges are merged that should not and does merge certain edges which should not. This primarily occurs near a transition usually an edge with a decreasing pixel value which separates the two groups or serrated edges. This causes the α-tree to be relatively small (Low amount of nodes) giving worse results when looking at segmentation.

Figure 5 and 6 shows an example of a segmentation problem that occurs within the α-tree. Figure 5 is a high resolution image of an ordinary road. When zoomed in figure 6 it is visible that there is a problem, the red from the traffic sign overlaps with a balcony of a building behind it. Furthermore in figure 6 the edge between the red traffic sign and white traffic sign is a serrated edge where the top right of the traffic sign slowly decays into white on the left. This ultimately merges these two traffic signs too early while it should not. This is an example of a segmentation problem that often occurs within the α-tree.

Figure 7 shows the problem in a simple form with 1-CC (connected component with a difference less than 1). Lets assume that 0 represents white and 5 represents grey. The bottom part of the image is very grey and the top part is white. When using 1-CC these two area's will be grouped since there is a ladder from right to left which adds these area's together when using an α value of 1. This is the main issue with the α-tree, which also occurs in figure 6. It is difficult to see but when looking closer there is a ladder going from right to left between the two traffic signs. Where the red color slowly turns into white as seen in figure 8. This leads to bad segmentation since these two traffic signs are merged too early while they are separate traffic signs. When iterating through all the nodes in the α-tree to find the traffic signs, at no point will these traffic signs be recognised because there is no node which only shows the traffic sign itself because these traffic signs are merged too early.

Figure 5: *Example image of traffic sign recognition problems*

Figure 6: *Traffic sign of figure 5 zoomed*

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	1	2	3	4	5
5	5	5	5	5	5
5	5	5	5	5	5
5	5	5	5	5	5

Figure 7: *The leakage problem in simple form, 0 and 5 gets merged at 1-CC while actually these two should be merged at 5-CC*

Figure 8: *The leakage problem on figure 6*

When zooming in figure 5 even further this issue gets visible even more. Figure 9 shows the pix-elation of the image and the ladder that is talked about in figure 8. Furthermore the balcony on the background also gets merged with the traffic sign resulting in one big merge of all these elements. Usually these problems occur on a 1 pixel width that have a huge impact on the segmentation result. Giving a tree that is first of all too small (since less area's are created) and furthermore has wrong segmentation leads to a bad classification rate.

Figure 9: *Figure 6 zoomed in even further, the red marked area shows the leakage problem as in figure 8*

2.7. Gabor Filter

Gabor filtering is a linear filter which filters specific frequency in a given direction [gab]. It does this using a Gaussian kernel function. The Gaussian is calculated using the following formula:

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) cos\left(2\pi\frac{x'}{\lambda} + \psi\right)$$

$$where \begin{cases} x' = xcos\theta + ysin\theta \\ y' = -xsin\theta + ycos\theta \end{cases}$$

Figure 10 shows the Gaussian using Gabor filtering. Since Leakage is a problem that involves a small chain of pixels of usually 1 pixel width. A Gabor filter should be able to remove this effect. In the Gabor filtering λ determines the wavelength of the sinusoidal function, θ determines the orientation, σ is the standard deviation of the Gaussian distribution, γ represents the spatial aspect ratio and last ψ which determines the phase offset. The settings used are based on P.Kruizinga and N.Petkov [KP99] that were used for [ZXb] where [ZXb] only focused on the orientation and wavelength. Since we are only looking for direction and wavelength we do not have to change any of the other parameters. Therefore we primarily use angles of 90, 30, -30, 60 or -60 degrees. In some cases a different settings might be useful since the images are sometimes a bit tilted to the right or left.

2.8. Decision tree

A decision tree is an easy classifier that classifies based on a tree of decisions. It can be seen as a tree structure where each decision determines a different path until the nodes are reached. Since this project is primarily about testing the α-tree, a simple recognizer is required. The decision tree should be able to recognize warning signs / yield signs based on a few variables such as: Color, shape and size. This determines whether the given cluster from the α-tree is a traffic sign. This paper only focuses on recognizing the yield sign and warning signs as in figure 11. The decisions will be based on the shape, in this case a triangle with red and white. Furthermore the system will only look at traffic signs that are bigger than 200 pixels too speed up the recognition since a lot of clusters have to be checked and also increases the accuracy because the input images are of better quality.

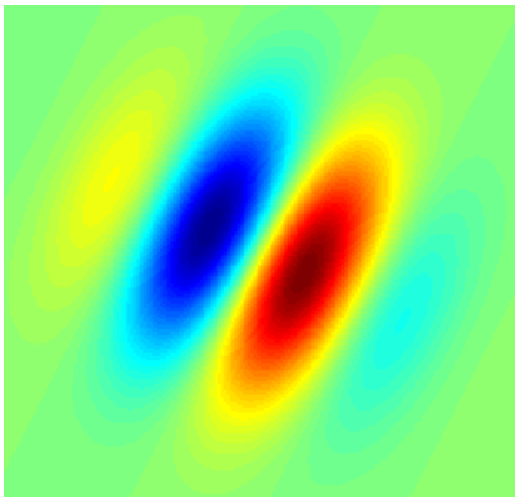

Figure 10: *Gabor Filtering [gab]*

Figure 11: *Dutch yield warning sign*

2.9. OpenCV 2

OpenCV2 [ope] is an open source library that is specialized in computer vision and recognition. For recognition purposes this library is very useful to find the α-tree solutions. OpenCV2 provides build in functions to recognize shapes in combination with a decision tree it can determine whether a traffic sign is apparent in an image. The α-tree provides an image for each cluster in an image each level, where a decision tree in combination with OpenCV2 determines if a traffic sign is found. Furthermore it is used to calculate the Gabor filtering, again OpenCV 2 has a build in function to calculate this.

3. Experiments

This section covers the experiments, where the experiments are split into speed of the α-tree, recognition and segmentation results using a Gabor filter vs not using a Gabor filter. The main goal of the paper is to test the result difference of Gabor filtering vs not using Gabor filtering. Furthermore, it looks at the speed of the α-tree. Last but not least it tests the recognition on real life examples as well as non real life examples. The recognition part of this project is a lower priority than the Gabor filtering and speed of the α-tree. Therefore the recognizer is a simple decision tree. Below describes the program flow during the process.

3.1. Program flow

Figure 12 shows the flow of the α-tree building process. The source code for this program is provided with a link in the appendix. The program starts of by reading the image selected and initializing the α-tree. Here the dissimilarity vectors are created which calculates all the dissimilarities between each pixel using the euclidean distance applied to the CIE L*a*b* colors. After all the dissimilarities are determined a 0-CC alpha step is done, grouping all the pixels that have the exact same color resulting in an initialized α-tree.

The second step is to finish the α-tree. Each step all the dissimilarities, which have the same value as alpha, are looped through. This results into the merging of clusters of that alpha level. After the clusters are merged, which can be considered a new cluster, the recognition process starts. First, all clusters are separated and the amount of pixels for each cluster is calculated. For each cluster it is checked if it is a triangle, based on this decision the flow either continuous to the next decision or selects the next cluster. If the cluster is a triangle the size is checked. If the cluster is smaller than 200 pixels it is skipped to the next cluster. If the cluster is bigger than 200 pixels the last checkpoint is reached which determines if there is enough red and white. During the experiments a percentage will be used to determine whether it is a yield warning sign. This percentage is calculated the following way:

$$\%\text{ white pixels in cluster} = \frac{\#\text{ of white pixels in cluster}}{\#\text{ of pixels in cluster}}$$

$$\%\text{ red pixels in cluster} = \frac{\#\text{ of red pixels in cluster}}{\#\text{ of pixels in cluster}}$$

Once all checkpoints / decisions are positive it will be considered a yield or warning sign. The program will continue to find more traffic signs or continue on with the next step in the α-tree until the bottom is reached. The bottom of the α-tree is reached once the dissimilarities matrix is empty. When this threshold is reached the program will terminate and show the total expired time.

3.2. Experiment goals

This section describes the main goals of the experiments, namely: finding the speed of the α-tree, the recognition and furthermore the segmentation results.

3.2.1. Speed

The first test is based on the speed, how fast does the system build the α-tree? The result is separated in initialization of

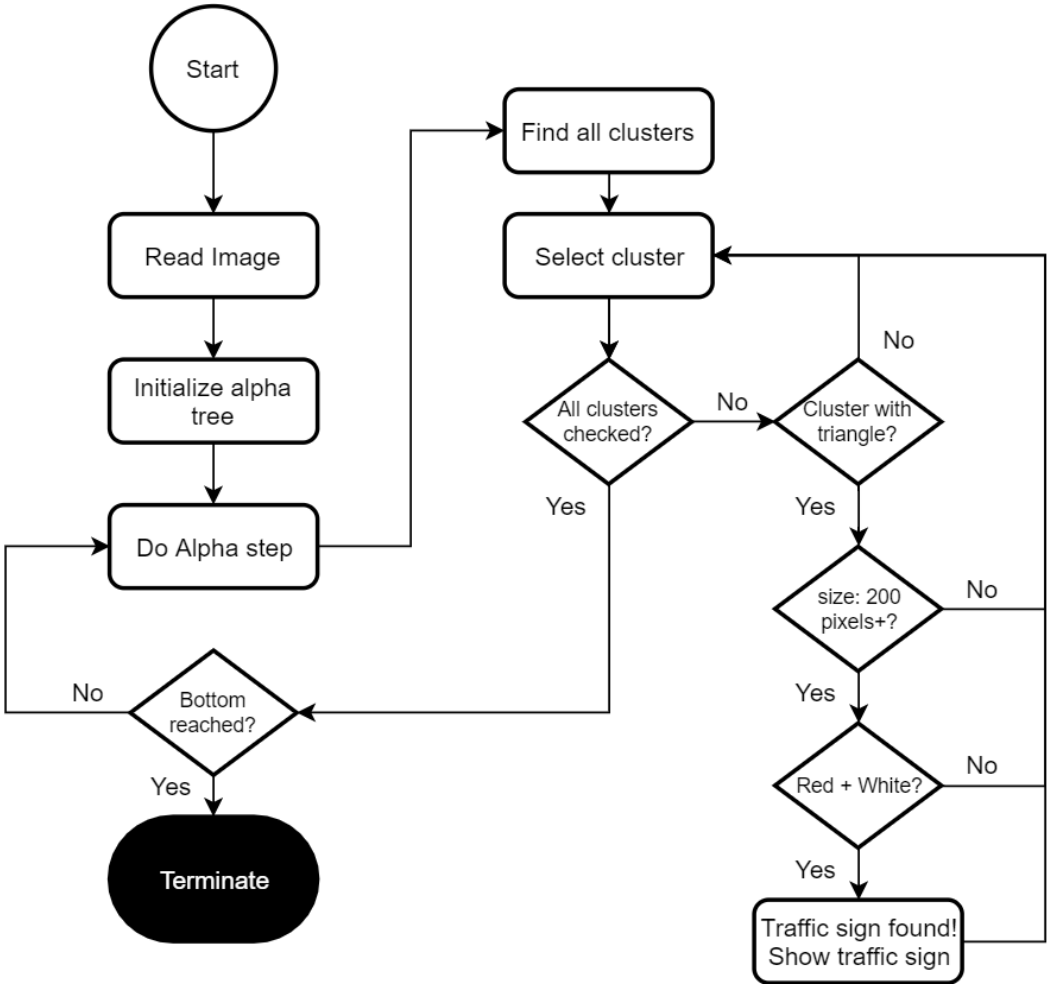

Figure 12: *Program flow α-tree*

the dissimilarities and finishing the α-tree steps. It is important to notice that the current α-tree used for this research makes use of one single core and is not yet optimized for multi core. Meaning that the program is not paralleled to run over multiple cores. However an estimation is given by dividing the total amount of time by the amount of cores. This does give a general idea, but is still an estimation of using multi-core calculation. For the experiment two different CPUs are used, namely: **Intel core I7-3770 3.4Ghz** and **Intel core i5-9600K 3.7Ghz**. Using one core significantly decreases the speed of the α-tree, the estimation speed is divided by 8 (the amount of cores within the Intel core I7-3770 processor) and 6 (the amount of cores within the Intel core i5-9600K).

3.2.2. Gabor Filtering vs No Gabor Filtering

The second test determines the segmentation results, does the α-tree give better results when using a Gabor Filter? According to [ZXb] it should give better segmentation results. Furthermore the decision tree is tested to get the classification rate, does the program recognize actual traffic signs? In the recognition phase the program tests real-life examples, extreme cases as well as easier cases.

4. Results

Below describes the results found during the experiments. It is separated between speed and recognition, furthermore the result of the Gabor filter is given.

4.1. Speed

One of the elements that is important for the α-tree is testing the speed. Table 2 and Table 3 shows the result of the tests where the example figures are provided in the appendix. The code written for the α-tree is written as small as possible to lower the amount of calculations and increase the speed. Furthermore rank based search and path compression are applied every step within the α-tree

to lower the size of the paths. The algorithm itself is only a few lines to decrease the amount of I/O calls etc. Only one thing is not optimized for the α-tree namely parallelization. It is not possible to use multiple cores since this requires more research, it is important to keep this in mind when looking at the table.

The results are tested using the Intel Core i7 3770 3rd generation and the i5 9600K 9th generation CPU. The i5 9600K was released on Quarter 4 2018 and is one of the newest commercial CPUs. As visible the I5 9600K is about 5x times faster than the I7 3770 which was released on Quarter 1 of 2012. This difference is really important since it shows that the speed of CPUs are increasing which speeds up the α-tree building process significantly. It would be unfair to base the results on this older i7 3770 CPU since much faster CPUs have been released since then. This means that all conclusions and results will be based on the result of the i5 9600K CPU. There are still possibilities to improve the speed by using an I9 9900K processor with 8 cores (newest CPU Intel has released), but that will at best improve the speed by 8% according to CPU benchmark [cpu]. When looking at the results this 8% gain does not make a difference. The only big difference might be the increase in amount of cores (8 cores vs 6).

When looking at the performance it is visible that the newest generation can handle smaller images (1024*768) at a decent speed, meaning that the recognition can be done before surpassing a traffic sign. This is of course dependent on the implementation speed, it is assumed that the implementation used is not the most efficient one. First of all parallelization could be used to improve the speed significantly. It took the i5 processor around 32 seconds to build (would be 5.37s using 6 cores) the α-tree for an 1024*768 image. That is acceptable, however for bigger images such as 2048*1536 it is not possible since it would take around 1 minute. The car has most likely already passed the traffic sign and furthermore during a car drive multiple traffic signs must be analyzed, thus the system cannot be stuck with one traffic sign for 1 minute.

As already mentioned this execution time is based on an implementation that is not build for the best optimization. The implementation is $O(nlogn)$ which is expected from the Alpha tree and thus far there are no solutions to prevent $O(nlogn)$.

4.2. Recognition

After an alpha step is done the clusters will be checked whether they contain a traffic sign or not. Each alpha step an image is created for each cluster and it is checked if that image contains a traffic sign. This is done using OpenCV2 library which can automatically detect shapes, colors and size. First the size of the traffic sign is checked, recognition of traffic signs that are a few pixels is pretty much impossible. To speed up the program and not waste time on these small clusters, all clusters that are smaller than 200 pixels will not be checked by the recognition program. This means that all clusters that are less than approximately 14*14 pixels will not be taken into consideration.

The second step is to check if the cluster is a triangle. If this is the case the last part that gets checked are the colors. Does it contain primarily red and white coloring. Once this is checked and the results are all true. The cluster is considered a traffic yield / warning sign.

The first test is done on figure 11. The system was successfully able to recognize this traffic sign. Figure 13 shows the result of finding the traffic sign using binarization of OpenCV2. As visible the system is able to find the traffic sign using this standard image.

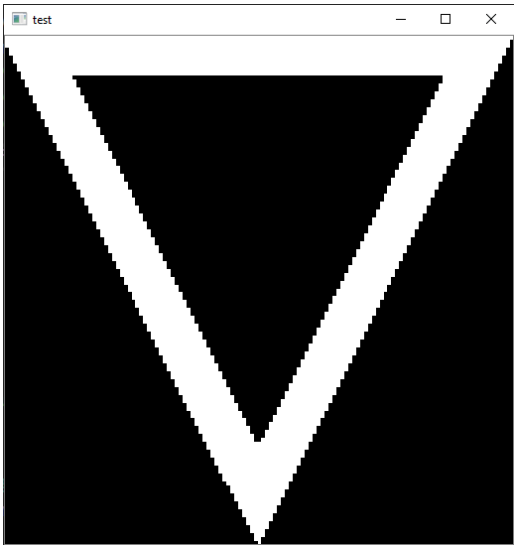

Figure 13: *Recognition using OpenCV2 binarization*

However this image is ideal, meaning that it has the same color everywhere and no background noise. The goal is to see if real life examples can be examined in order to apply the α-tree to real life examples. Figure 14 shows an example of a real life example traffic sign that could possibly be recognized. This example is extracted from a bigger image to see the results of the recognition. When using the α-tree applied to this image a big problem occurs as visible in figure 15. The middle part of the traffic sign is removed which is strange since these should be combined. Looking closer to the image it is visible that the α-tree combines the red part of the traffic sign with the background. This should not happen since we only want the traffic sign as result. When taking a look at the α-tree's principle this does make sense since the distance between red and white

Image size (pixels)	Initialization time	α-tree time	Total time	Estimated time 8 cores	Figure
100*100 = 10.000	0.16s	0.29s	0.45s	0.06s	20
200*200 = 40.000	0.62s	1.64s	2.26s	0.28s	21
300*300 = 90.000	1.43s	4.35s	5.78s	0.72s	22
400*400 = 160.000	2.68s	16.93s	19.61s	2.45s	23
1024*768 = 786.432	12.58s	119.04s	131.62s	16.45s	24
2048*1536 = 3.145.728	49.19s	1438.78s	1487.97s	186.00s	25

Table 2: *Speed of α-tree based on image size using Intel Core I7 3770 one single core*

Image size (pixels)	Initialization time	α-tree time	Total time	Estimated time 6 cores	Figure
100*100 = 10.000	0.06s	0.11s	0.17s	0.03s	20
200*200 = 40.000	0.25s	0.59s	0.84s	0.14s	21
300*300 = 90.000	0.54s	1.37s	1.91s	0.32s	22
400*400 = 160.000	0.97s	4.9s	5.87s	0.98s	23
1024*768 = 786.432	4.76s	27.43s	32.19s	5.37s	24
2048*1536 = 3.145.728	19.3s	435.13s	454.43s	75.74s	25

Table 3: *Speed of α-tree based on image size using Intel Core I5 9600K one single core*

is less than the background, at least that is what it seems. Because when looking closer at the image we can see a chain of pixels that form a ladder between red and the background. When the α-tree gets build the color distance between red and background are actually less than the white coloring, meaning that the α-tree combines the red with the background, before adding the middle white part. This is a problem since it is not possible to recognize a traffic sign if it is combined with the background. This problem is brought up earlier by [ZXb] as Leakage and has been explained earlier.

Figure 14: *Real life example of traffic sign*

4.3. Gabor Filter

A proposed solution to the leakage problem is Gabor filtering, as X.Zhang and M.Wilkinson [ZXb] stated. A Gabor filter changes the pixel intensity using a Gaussian, by filtering the pixel intensity by the surrounding pixels using a certain direction. Since we are looking for yield/warning

Figure 15: *Recognition using α-tree*

signs these directions are 30, 60 and 90 degrees. This might increase correct segmentation and classification.

The next part describes the segmentation results using the alpha tree with and without a Gabor filter applied to the image. Figure 17 shows the result of recognition using no Gabor filter vs using a Gabor filter. 30 degree Image on the top left represents warning signs (which have a 30 degree angle) and 60 degrees represent the yield sign (Which is the inverted warning sign, meaning 60 degrees). An x means that there is no traffic sign found in the image using alpha tree, this is usually due to leakage or the segmented area shows other objects as well. Figure 16 shows two examples of wrongly segmented traffic signs, even though they do contain the traffic signs the cluster is wrong since it contains some pixels that do not belong to the traffic sign. The - represents partly correct segmented, this means that the outside of the traffic sign is found, if we would favor everything inside this object the traffic sign can be considered found. The v

represents a correctly segmented traffic sign, which is self explanatory.

Figure 16: *Wrongly segmented traffic signs*

Figure 17 shows the result of segmentation, as visible the Gabor filter performs much better than without a Gabor filter. If we consider partly correct segmented as correctly segmented we have a 15% correct segmentation vs 55% correct segmentation. That seems like a low score but it has to be considered that the images used are extreme cases. For example number 1 from 60 degree images has the red balcony behind it as seen in Figure 9, which is almost impossible to find using an α-tree.

Figure 18 and Figure 19 shows how effective the Gabor filtering is. As visible in Figure 18 the leakage problem is almost completely gone which results into perfect segmentation of the traffic sign.

4.4. Unfortunate coloring

Another problem that occurred is unfortunate coloring. This sounds a bit vague but is pretty straightforward. The distance between color is an absolute measure that forms a problem for specific scenario's. The Yield warning sign is a good example, in general the absolute distance between white and red using CIE lab coloring is very high. While the distance between red and any other color than white is smaller in most cases. This causes the α-tree to always merge with the background before merging with the inside part of the traffic sign. This can happen to a variety of colors and can cause wrong segmentation results.

5. Conclusion

Using α-trees to breakdown images is an easy technique that has a lot of potential. When looking at traffic sign recognition for simple examples the α-tree is easily able to find traffic signs and can quickly find results. In general the segmentation results of the α-tree is acceptable (depending on where you are looking for). But Leakage is a problem that often occurs within it, which leads to wrong segmentation. Gabor filters are a very good technique to solve this issue and increase the segmentation results significantly. However when an extreme case occurs where we want to separate two parts with almost identical color even the Gabor filter cannot solve this issue.

When looking at the speed using the code provided the α-tree performs decently. Currently the α-tree calculations grows with $O(nlogn)$ for each pixel. Meaning that the bigger the image, the slower the α-tree becomes. Small imagery(1024*786 pixels) are not a problem and could be done within 6 seconds. However for bigger images greater than 1000*1000 pixels, the used implementation becomes too slow for traffic sign recognition. As explained this speed is not acceptable since a car passes a traffic sign much quicker. Even though parallelization could be used to increase the speed significantly it is still not fast enough to comprehend this issue. It must be noted that the code used in this paper is not optimized on speed, thus it might be possible that other implementations give better results. So it cannot be concluded that the α-tree is responsible for this speed issue.

For this project a simple decision tree classifier is used to determine shape, color and size. The decision tree has a decent classification rate when using standard images where the traffic signs are perfectly aligned. However when looking at real life examples the classifier has around 55% correct (Using a Gabor filter), which is decent for a simple classifier. A good classifier would most likely have a much higher correct classification rate. But since we are comparing no Gabor filter vs Gabor filter, the Gabor filter outperforms the α-tree without gabor filter (15% vs 55%). It can thus be concluded that the leakage problem (or chaining) within α-trees can be solved to a certain extend using a Gabor filter. Gabor filtering will give better segmentation results when using an α-tree.

6. Future research

There is a lot of research that can still be done for the α-tree, such as: parallelization, solving leakage problem further (besides using a Gabor filter), but also preventing the wrong merging of certain connected components. Furthermore decreasing the calculation time [PKA] is something that should be considered, since it grows $O(nlogn)$. As visible in the results the bigger the image the slower the α-tree. A possibility would be to seperate certain parts of the image and only focus on a small part of the image to increase the speed. Other examples of increasing the speed would be to down sample the image but that might result into worse classification.

Another implementation could be to improve the use of α-tree steps, it could for example be possible to skip certain steps when there is a small amount of change. Furthermore the first few steps usually change a few pixels of the connected components. In general we are looking for bigger objects where it might be possible to skip certain steps. Last but not least the direction of connecting components can be influential, traffic signs are a good example of that. When we take the red pixels of the traffic sign it would be weird to merge it with another color outside of the red part. Since everything of that traffic sign should be inside of it. But the

Figure 17: *Gabor filter result*

Figure 18: *Applying a Gabor filter in 3 directions: 90, 30, -30*

Figure 19: *Segmentation after applying a Gabor filter*

the distance between red and background is smaller than red and white on the inside it will always merge with the outside.

[JS]

References

[cpu] Intel i5-9600k vs intel i9-9900k cpu benchmark. https://www.cpubenchmark.net/compare/Intel-i5-9600K-vs-Intel-i9-9900K/3337vs3334. Accessed: 10/12/2019. 8

[gab] Gabor filtering. https://en.wikipedia.org/wiki/Gabor_filter. Accessed: 07/02/2020. 5, 6

[Jas11] Jasper S.: Road sign recognition using the binary partition tree. *University of Groningen* (July 2011). 1, 2

[JS] Johannes Stallkamp Marc Schlipsing J. S. C. I.: The german traffic sign recognition benchmark: A multi-class classification competition. *Institut fr Neuroinformatik, Ruhr-Universitt Bochum, Germany; Department of Computer Science, University of Copenhagen, Denmark.* 12

[KP99] Kruizinga P., Petkov N.: Nonlinear operator for oriented texture. *IEEE transactions on image processing 8, 10, p. 1395-1407 13 p.* (1999). 5

[OG12] Ouzounis G.K Soille P.: The alpha-tree algorithm. theory, algorithms, and applications. *Publications Office of the European Union, Luxembourg* (December 2012). 1, 2

[ope] Opencv2 library. https://opencv.org/. Accessed: 10/09/2019. 6

[Pas01] Paschos G.: Perceptually uniform color spaces for color texture analysis: An empirical evaluation. *IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 10, NO. 6* (June 2001). 2

[PKA] Pankaj K. Agarwal Lars Arge K. Y.: I/o-efficient batched union-find and its applications to terrain analysis. *Department of Computer Science, Duke University, Durham, NC 27708, USA. Department of Computer Science, University of Aarhus, Aarhus, Denmark.* 10

[RET84] Robert Endre Tarjan J. v. L.: Worst-case analysis of set union algorithms. *Journal of the Association for Computing Machinery, Vol. 31, No. 2* (April 1984). 3

[Sch07] Schanda J.: Colorimetry. *Wiley-Interscience* (2007). 4

[SL] Sermanet P., LeCun Y.: Traffic sign recognition with multi-scale convolutional networks. *Courant Institute of Mathematical Sciences, New York University.* 2

[Tar75] Tarjan R. E.: Efficiency of a good but not linear set union algorithm. *Journal of the Assoclatlon for Computing Machinery, Vol 22, No. 2* (April 1975). 3

[THCon] Thomas H. Cormen Charles E. Leiserson R. L. R. C. S.: Introduction to algorithms. *MIT Press United States* (1990 (first edition)). 3

[VV08] Veronica Vilaplana Ferran Marques P. S.: Binary partition trees for object detection. *IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 17, NO. 11* (November 2008). 1

[WCT] Wei-Chih Tu Shengfeng He Q. Y. S.-Y. C.: Real-time salient object detection with a minimum spanning tree. *Graduate Institute of Electronics Engineering, National Taiwan University; Department of Computer Science, City University of Hong Kong; School of Information Science and Technology, University of Science and Technology of China.* 1

[XB] Xavier Bar Sergio Escalera J. V. O. P.-P. R.: Traffic sign recognition using evolutionary adaboost detection and forest-ecoc classification. *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS.* 2

[YX] Yongchao Xu Edwin Carlinet T. G. L. N.: Hierarchical segmentation using tree-based shape spaces. 1

[ZXa] Zhang X Wilkinson M. H. F.: Improving narrow object detection in alpha trees. *De Gruyter, University of Groningen.* 4

[ZXb] Zhang X Wilkinson M. H. F.: Preventing chaining in alpha-trees using gabor filters. *University of Groningen.* 2, 4, 5, 7, 9

7. Appendix

Below shows the code that is used for testing and results. Furthermore the images which are primarily used are shown in the used images section of the appendix.

7.1. Open-source Code

Below shows a link to the open-source code that is used for the α-tree and traffic sign recognition. It is freely available and requires the installation of OpenCV2, the α-tree can be used without the use of OpenCV2 for testing purposes.

https://github.com/thegendolz/In-company-research

7.2. Used Images

Figure 20: *100x100 image α-tree build test*

Figure 21: *200x200 image α-tree build test*

Figure 22: *300x300 image α-tree build test*

Figure 23: *400x400 image α-tree build test*

Figure 24: *1024x768 image α-tree build test*

Figure 25: *2048x1536 image α-tree build test*