Image-based localization of pictures using Google Streetview images

S.N.A. Vlaszaty
bas.vlaszaty@os3.nl

T.R.L. Timmermans
tim.timmermans@os3.nl

February 13, 2013

Supervisor:
dr. J.G.M. Schavemaker, TNO
john.schavemaker@tno.nl

Abstract

This paper describes a method to perform image localization based on image matching technology. This is achieved by building a ground-truth database using Google Streetview images, and ordering this database in a tree structure. From this tree structure, similar images can be found very quickly. To prove this, an experimental concept was created capable of localizing images within the main center of Amsterdam. Using different matching technologies, a match can be found in less then 8 seconds. Based on the performance of this proof of concept, a similar system covering the whole Netherlands might be possible using this method.

Contents

1 Introduction

With the rising popularity of smartphones and the increasing use of social media like Facebook and Twitter, more an more pictures are available on the internet. In some cases when an incident has occurred photos are placed online before the emergency services are notified. These photos can provide valuable information about the incident, such as the location. The location in this case can be an exact GPS coordinate, but also an estimation of the location within a given range.

In addition to law enforcement, automatic location detection can also apply to the following services: automatic adding geographical information to a photo, tracking people or extending the photo plugin on social networks with the automatic tagging of photos. The accuracy of the location may very per application. To add geo information to a photo, an accurate location is needed, but for the automatic (hash)tagging of photos the name of the city or street (#amsterdam, #raamgracht) is enough.

Unfortunately, in most cases geographical information about the picture is not available. Even if an image contains geodata, it is still not completely reliable, as there is no way to verify the location data of the used device was up to date at the time the picture was taken. Occasionally, it is possible to manually determine the location by the recognition of known buildings and landmarks. However, in most cases this is not possible and also not completely reliable. A solution based purely on the actual image data would not suffer from problems like this. Image recognition however, is notoriously difficult and computationally expensive.

Recently, TNO performed research into a technique to determine if two photos represent the same object or scene. However, this technique is not suitable to use on a large scale, because it is too computationally expensive.

This paper proposes a method to efficiently determine the geographic location of a photo using scene matching. It will do so by extracting descriptors from Google Streetview[1] images and match them with the query image. Accordingly, the main research question is:

How to efficiently look up the geographical location of a picture using a large set of image descriptors using images from Google Streetview.

Images from Google Streetview have known geographical information and the service has a high coverage in the Netherlands[2], which makes Streetview suitable as ground-truth database. The research is limited to the city of Amsterdam and a experimental concept is developed to demonstrate the results. Due to the limited timeframe of the project, the matching algorithm is not able to automatically detect whether a match is found or not. This implies that the results from the algorithm need to be analyzed manually. The concept needs further work in order to implement such a system in a fully automatic environment, but does meet the requirements in case a decision maker controls the environment.

[1]`http://www.google.com/streetview`

[2]`http://gmaps-samples.googlecode.com/svn/trunk/streetview\_landing/streetview-map.html`

2 Related work

A great deal of research exists in the area of computer vision and image recognition. As computing power becomes more readily available, it becomes possible to process larger datasets. One field where very large datasets are used, is image recognition. The technology used for general image matching can also be used for different image-based solutions to real-life problems.

In the TNO report "*Beeldmerktechnologie naar de Praktijk*" [2] the possibilities for image based localization are explored. Since our research is supervised by TNO, the technology described in this paper has been a major influence on our research. Technologies suggested in the report include using a feature database to look up specific element from the query image, and using a geometric matching algorithm on a subset of the database to pick the most accurate match. The report also includes an estimate for the required equipment, as well as an estimation of the monthly costs involved in offering such a service.

The work done by "*What makes Paris look like Paris*" [5] uses the images from Google Streetview to automatically find visual elements that are distinctive for a certain geographical area. By collecting these distinctive elements, a classifier is built to determine if an image contains elements that can be associated with a specific city. Besides finding a geographical location, they can also find similarities between different cities and connecting visual appearances based on visual elements in a picture.

The work done in "*IM2GPS: estimating geographic information from a single image*"[7] uses a scene matching approach. In this approach, areas in an image are classified as ground, sky or vertical structures. Using these classifications the structure of images can be reliably compared. In this research a dataset containing 6 million GPS-tagged images from Flickr was used. The structural matches were used in a simple algorithm to estimate the geolocation as a probability distribution on a global scale.

In "*Scalable Recognition with a Vocabulary Tree*" [9], a recognition scheme is proposed to recognise pictures of different objects. It uses a vocabulary of local region descriptors. Using a hierarchical quantization a tree-like data-structure is created with leaves containing the vocabulary. This results in a very scalable solution, with the research showing tests with up to 1 million images. In this solution lookup times are barely influenced by larger databases.

3 Methods

This section describes the complete chain used in our concept for localization based on image matching. The first step is to collect images for building the ground-truth database from Google Streetview. Subsequently, descriptors are extracted from those images. For this the OpenCV library[4] is used. Next, the extracted descriptors are used to build a data structure that is searchable quick and efficient. From this data structure the actual matching and localization can be performed.

How is an acceptable match defined? The requirements for a successful match can be different from application to application. Depending on the application, an acceptable result can be identifying the correct city. In other applications, a match can be a street, a location within 20 meters or even the exact camera position and viewing angle. For some applications, the algorithm needs to return one location that has to be correct, whereas for other purposes having the correct result in the first 10 suggestions is accurate enough. When implementing a system, these requirements need to be defined, in order for the system to be able to produce the desired results.

In the experimental concept, three different match methods were implemented, that have different uses, performance and accuracy:

- Statistical match, based on the occurrence of descriptors in images.
- Geometrical match, to check whether the descriptors in images are in the right place relative to the other descriptors.
- Area match, which tries to pinpoint certain hotspots where a lot of possible matches are located

To demonstrate this method a experimental concept is developed. Python is used as programming language, along with the OpenCV library.

3.1 Ground-truth database with geotagged images

To perform image-based localization, an unknown image has to be matched against a database of known images with geographical information, also known as ground-truth or referential database. This means that it is important to have access to a "good" source of data., that meets the following requirements:

- The location (GPS coordinates) where an image was taken must be known and reliable. Image-based localization is not possible if the location of the database images is uncertain or unknown.
- The images must cover the whole search area, or at least as much of the area as possible. If there is no image of a location in the database, finding a good match is not possible.
- Lighting conditions greatly influence an image. Matching a daylight image to an image taken at night will result in sub optimal results. Images in the database should be taken in daylight, preferably in somewhat cloudy conditions as shadows can disturb an image.

- The images should preferably face the buildings head-on. If a picture is taken from an angle, all the lines will obviously be skewed. For the database it is preferable to have neutral images of the face of the building.
- Images should be available in a decent resolution. The best matches are achieved when the query image is around the same size as the ground-truth images. Images more than a factor 2-3 smaller will not yield good results. Larger query images are not a problem as they can be downscaled.
- The data source should preferably be freely available to use.

Considering these requirement the decision is made to use Google Streetview as a source for our data. Google Streetview images are taken by cars equipped with a 360° camera that takes a picture every so many meters. Each Streetview Panorama is an image that provides a full 360 degree view from a single location (equirectangular projection). Each image contains an accurate GPS location of where the it was taken. Current GPS typically provides an accuracy of about 3 meters[1].Also the images are taken by day, and cover almost every street in Amsterdam. This results in the images meeting all the requirements mentioned before.

Streetview Crawler

To download all the Google Streetview panoramas in Amsterdam, a simple tool was created to automate this task. The tool is based on previous work from "*What Makes Paris Look like Paris?*"[5]. The tool calculates all possible coordinates which fall within the city of Amsterdam. For each found coordinate a request is made to the Google Streetview Javascript API[3] which returns a positive result if a panorama is available for the provided location within a radius of 50 meters.

A panorama consists of tiles with 512x512 pixels and is available in different zoomlevels, from 0 to 5 which defines the resolution. At zoomlevel 5 a panorama consists of 338 tiles, that is a resolution of 13312x6656 pixels. The created program to download the panoramas downloads 91 tiles per panorama with a zoomlevel of 4, which is a resolution of 6656x3584 pixels.

Extracting images from the panoramas

As described in the previous chapter, a Google Streetview panorama is a equirectangular projection[4]. The panorama is therefore not useful to use in our database, because all the objects on the panorama are distorted which result in poor matches. The panorama also contains a lot information that is not unique for the image, like the sky and the street.

A solution for that is to cutout only the relevant parts of the panorama which is unique. To stay within the limitation of the used hardware, two perspective cutouts of each panorama are made with a resolution of 1024x768 pixels, as shown in figure 1.

The perspective cutouts need to be converted to a rectilinear image, which is done by using the algorithm used in the work from [5].

[3]`https://developers.google.com/maps/documentation/javascript/streetview?hl=nl`

[4]Panorama that represents a 360° horizontal and 180° vertical field of view

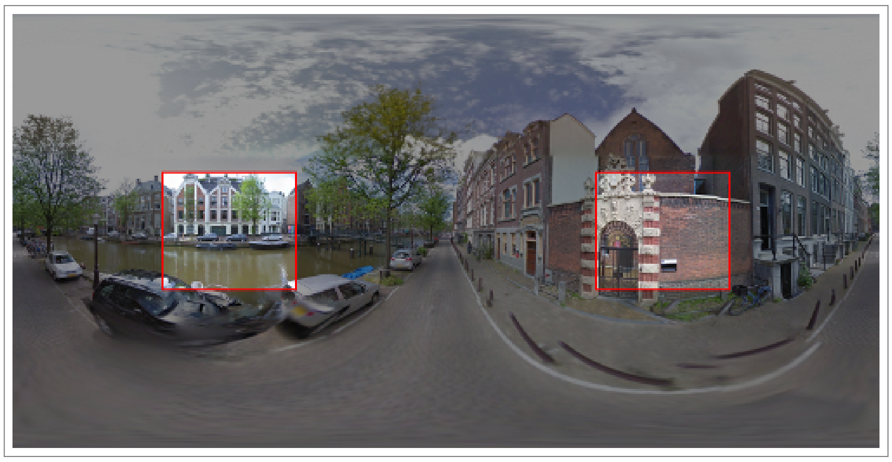


Figure 1: Created cutouts from a Google Streetview panorama

As can be seen in figure 1, much of the detail in the panorama is lost, because only two cutouts are made. In most cases that is not a problem, because 8.699 panoramas are made every 100 meter[5]. Detail that is lost in a panorama is mostly still covered in another panorama. But there are cases where unique objects are complete discarded and not stored in the image database, which possibly leads to a decrease in matching accuracy.

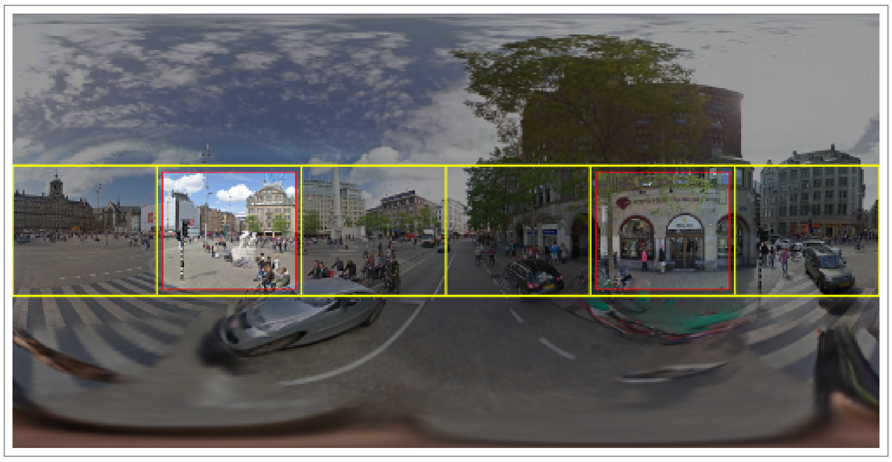


Figure 2: Created cutouts from a Google Streetview panorama

As shown in the above panorama (figure 2), the loss of unique objects can be overcome by creating cutouts from more and larger view angles. The objects drawn in red are the cutouts created in the current setup. The objects drawn in yellow are the cutouts that need to be made to cover all the unique elements in the panorama.

Although not all objects are covered by the two cutouts that are currently made, like the "Royal Palace on the Dam" (most left in figure 2) they can still

[5]See section 4.1 on page 12 for the calculation.

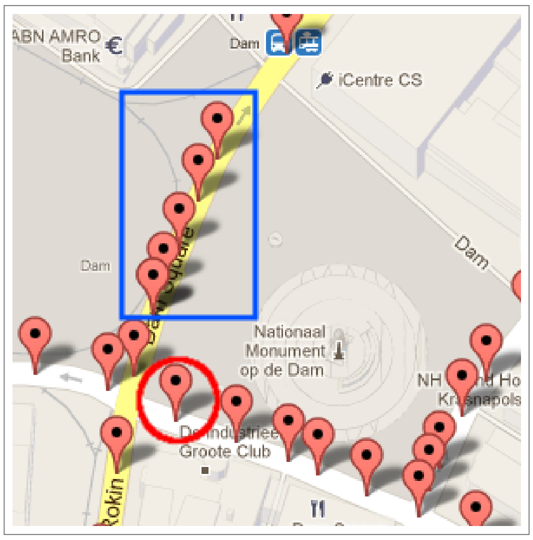

ABN AMRO Bank
Dam
iCentre CS
Dam
Dam
Dam Square
Nationaal Monument op de Dam
NH Grand Hotel Krasnapolsky
De Industrieele Groote Club
Rokin

Figure 3: Visual crawler result for "the Dam" in Amsterdam

be in the image database, due the fact that multiple panoramas are created in that area, as shown in figure 3. Each point on the map represents the location where a panorama is made. The circle drawn in red, is the location of the panorama in figure 2, which does not cover the "Royal Palace on the Dam" and the "National Momument the Dam", but they are still covered by the cutouts made from the panoramas drawn in the blue rectangle.

3.2 Descriptor extraction

In order to perform image matching, descriptors are extracted from the images. A descriptor is a 128 dimensional vector describing a small area of the image that looks "interesting" to the computer. Also included is metadata about the feature itself, like the location where the descriptor was found and the size of the descriptor.

There are various types of features that are being used in the field of computer vision. In this case SIFT [8] (Scale Invariant Feature Transform)[6] features are used. As the name suggests, SIFT descriptors are invariant to the scale of the descriptor. They are also invariant to scaling, and robust to illumination changes, noise in the images and minor changes in viewpoint. These characteristics, along with the fact that SIFT descriptors are relatively easy to extract and highly distinctive, make SIFT descriptors very useful for tasks like object matching.

For the extraction of the features from the images, the OpenCV library is used. OpenCV is an open source library for computer vision tasks, first developed by Intel. Tasks like extracting SIFT features from an image are performed by a simple call to OpenCV.

All the images are run through OpenCV to extract the SIFT features. In this case most images generate about 1200-1400 features. The descriptors from

[6]`http://www.scholarpedia.org/article/Scale_Invariant_Feature_Transform`

each pictures are then stored in a 8 bit unsigned integer matrix. These matrices can be written to disk in blocks of for example 500 images. This is done using the save function from numpy. This function saves the matrices to disk in binary format. These files are later used as the dataset from which the descriptor tree is built.

3.3 Descriptor tree

There are many challenges in any system that involves querying large data sets. One of the biggest challenges is storing the data in such a way that it is quickly accessible, independent of the size of the database. Storing the data in a tree structure has these advantages. Lookups from a tree can be done extremely quickly,

The complexity of a lookup from a balanced tree is $O(logx(n))$ where x is the number of children each node has, and n is the total number of nodes. This means that the average number of hops needed to arrive at the correct leaf, given $n = 100.000$ and $x = 2$ (Binary search tree), would be $log2(100.000) = 16.6$. This performance will only be achieved in a perfectly balanced tree. In practice the tree will in most cases not be perfectly balanced, so then the average number of hops required rises slightly. In contrast, a lookup from an unordered list has complexity $O(n)$, which means that in the previous example it would take $n = 100.000$ hops to locate the correct entry. The latter is better known as a brute force search.

As lookup times are extremely important in a system like this, a tree is used for structuring the data. To be able to build a tree, a way of grouping data is needed. If the data is 1 dimensional this is trivial, and you can just split the data by value, but when more-dimensional data is involved, in our case 128 dimensional, splitting this data is not trivial at all. The idea implemented in the proof of concept is an implementation where nodes are split by using k-means clustering[6] on the descriptors. This results into a number of clusters, who form the child nodes, and are to be split again. Due to the nature of the k-means algorithm, the number of clusters created is fixed, and is set at the moment the tree building process is started. The idea behind using k-means clustering is that this way similar descriptors end up in the same part of the descriptor tree. This way it is easy to find a number of descriptors similar to the query descriptor when looking it up.

This building process results in a tree of nodes and leafs. Each node in the tree contains the center point of the descriptors located in its subtree, and references to its children. Each leaf in the tree contains a number of descriptors that are located closely together in the 128 dimensional descriptor space. Each of those descriptors contain a reference to the image they originate from.

3.4 Matching

With the descriptor tree built as described earlier, it is possible to perform lookups from the tree. When performing a lookup, descriptors are extracted from the query image. Then each descriptor is looked up in the tree. From the root, the distance from the descriptor to the centers of the children of the root is calculated, and the child with the center closest to the query descriptor is selected. From this node, the same process is applied untill a leaf is found.

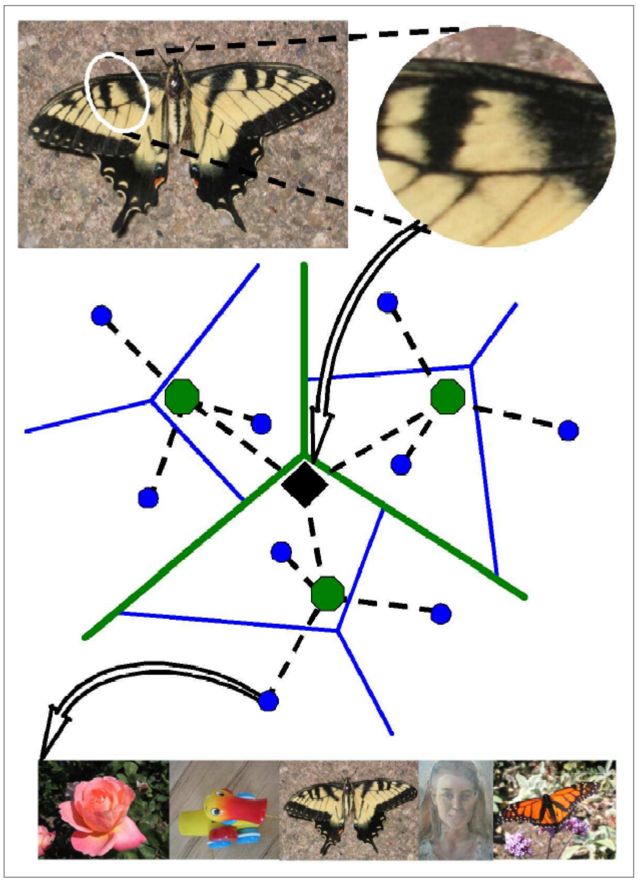


Figure 4: Schematic visualisation of the descriptor tree. A small part from an image ends up in a certain leaf in the tree. This leaf contains pointers to different source images of the similar descriptors. Source: [9]

When the leaf is found, all descriptors in that leaf contain references to their source image. All the source images found in the leaf are possible matches for query descriptor. By giving the highest score to the source image of the leaf descriptor closest to the query descriptor, the best match get the highest rating. Each descriptor lookup returns a list of a candidate match, along with a score achieved from this leaf. For each query descriptor, this process is repeated. The scores returned from all descriptor lookups are added together, which results in an list of candidate images, and the sum of their scores. From this list the best scoring candidate images are the most likely candidates. When these candidates are found, an attempt is made to find the matching picture and location. For this, three seperate matching methods have been implemented.

Statistical matching

Statistical matching is purely based on the results acquired from the descriptor lookups. The candidates and their scores are analyzed. If the highest score is significantly higher than all the others, this candidate is very likely to be the correct match. However, this only occurs when the image is almost identical to an image in the ground-truth, and due to difference in camera angles and lighting this rarely occurs. In order to find more accurate matches, other techniques are needed.

Geometrical matching

The previously discussed ways of matching are based on finding similar descriptors that occur in the images. Geometrical matching not only looks at the occurrence of similar descriptors, but also looks at their relative positions. This is done by creating a homography between the matching descriptors, using OpenCV. A homography, also called a projective transformation, is a matrix that describes the change in perspective when the point of view of the observer changes. Matching two images this way is a computational expensive operation, therefor it is not possible to perform this kind of matching to a entire database. Furthermore, simply calculating a homography does not mean that a good match has been found, a homography can be created from any two sets of points.

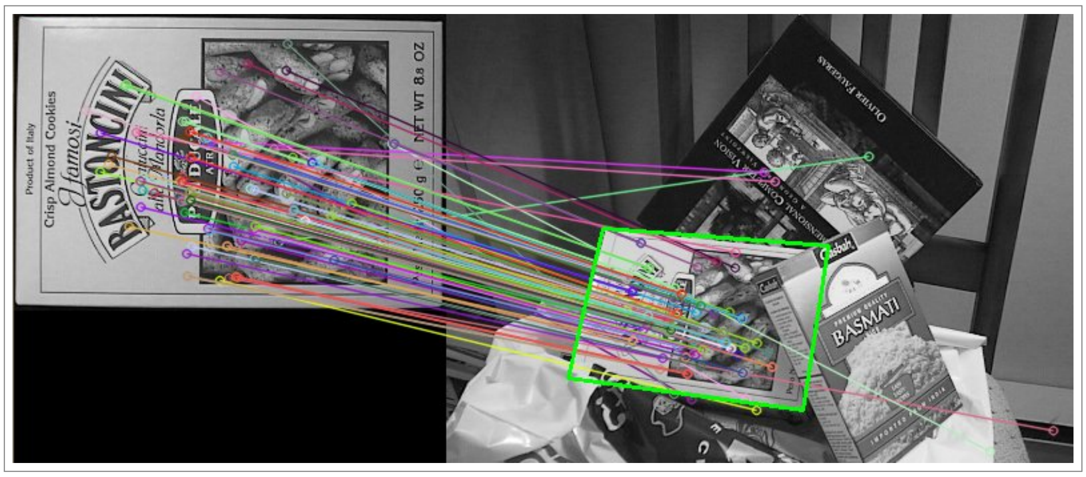


Figure 5: Image showing a projective transformation from one image to another. Based on matching pairs of descriptors, a homography is made from the first image to the second. Source: image from http://docs.opencv.org/doc/tutorials/features2d/feature_homography/feature_homography.html

The quality or likelyhood of the transformation has to be determined. The quality of a homography is determined by calculating the SVD (Singular Value Decomposition)[7]. The higher the ratio between the first and the last singular value, the less likely the projective transformation is. This value is used to rank the geometric matches.

This matching procedure is performed on the best candidates from the descriptor lookup. For each candidate a homography to the query image is made, and a quality score is calculated. A list of these scores for each candidate image is returned, where the highest score represents the best match.

[7]`http://mathworld.wolfram.com/SingularValueDecomposition.html`

Area matching

In the case of localization, determining the exact matching image is not always necessary. The most important piece of information is the location where the query image might be taken. On this note an area matcher is implemented. The area matcher takes a set of candidates from the descriptor lookup. It uses the locations of the candidates to identify clusters. Then the clusters are rated by adding up the scores of the candidates. This creates a map with likely locations for the query image.

4 Results

This chapter describes the achieved results that are gathered from the created experimental concept. Based on these results, an estimation is made for the scalability of the application, the performance of the application is measured and finally the results from the concept are discussed.

4.1 Scalability

Amount of storage needed for covering the Netherlands

In the experimental concept a database is used with a total of 5276 panorama images. Two cutouts are made from each panorama, which means that a total of 10552 images are used to cover the main center of Amsterdam.

The results from the work "*Google Street View Images Support the Development of Vision-Based Driver Assistance Systems*"[10] are used to make an estimate about the number of panoramas in the Google Streetview database that are currently used to cover the Netherlands. The results are gathered using a Streetview Crawler program that is able to detect and download Streetview panoramas made on a particular road.

In the table below, the number of found panoramas at a certain mileage for 6 cities in different countries, is given.

City	Distance [km]	#Panoramas
San Fransisco, USA	3023.8	268,127
Penghu Islands, Taiwan	514.8	44,736
Port Elizabeth, South Africa	2105.1	175,369
Belo Horizonte, Brazil	1426.3	128,459
Alcal de Henares, Spain	409.9	32,645
Gold Coast, Australia	2509.8	219,558

Table 1: Number of panoramas found in 6 different cities for a certain distance. Source: [10]

From these results, the mean is calculated for panoramas in Google Streetview per kilometer, which is 86.99. In March 2010, Google had indexed 75.600 kilometers of the Dutch road network, which is nearly a full coverage. Based on this information, an estimate can be made that there are 6,576,444[8] panorama photos (13,152,888 cutouts) in the Google Streetview database to cover the Dutch road network.

Currently, both the cutouts and the original photos are saved, but only the image descriptors are needed for the actual matching. 1850.99 MB[9] of storage is needed to save the image descriptors of 10552 images with a resolution of 1024x768 pixels. In the proof of concept two cutouts are made from each panorama, that means that for storing the image descriptors for all Streetview Photos in the Netherlands 2307.22 GB[10] of storage is needed.

At the moment all descriptors from each picture are saved, but not all of them are useful for the matching. By implementing an algorithm which can reduce the amount of descriptors the required storage space can be reduced.

Tree building

As described in section 3.3, the k-means algorithm is used to cluster the data for building the tree. The data are vectors of 128 dimensions.

From the 10552 images used to cover the main center of the city of Amsterdam, 15156817 descriptors are detected. In the created application a default, single threaded, implementation of the k-means algorithm is used on a single machine. For clustering $\tilde{1}5$ million data points that is sufficient, but not suitable for clustering billions of data points.

The work ”*Parallel Clustering Algorithms with Application to Climatology*” [3] proposes a method to parallelize the k-means clustering algorithm. Performance results show that 1 billion data points of 128 dimension vectors can be clustered within 50 minutes when using 2048 MPI cores[11]

In the current setup, 13,152,888 images generate a total of 18,892,713,839 descriptors[12]. That amount of data points can only be clustered when using a distributed setup.

Hardware

The k-means algorithm needs all the data in the first step of its algorithm to determine the center points. That means that all the data needs to be loaded when building the tree, which is 2253.15 GB of data.

Currently, the image descriptors are stored in the leafs of the tree. A huge optimization could be to only store a pointer of the descriptor in the leaf, which points to an entry in a database where the descriptor is located.

Clustering the data and building the initial tree is computational expensive. For clustering $\tilde{1}9$ billion data points at least 2,253.15 GB of memory and much processing capacity is necessary for clustering that amount of data. It is not possible the make an estimate about the hardware that is needed to cluster that amount of data, since that is not tested. After the clustering the initially loaded data can be discarded from the memory and a single CPU is powerful enough to handle the tree lookups.

[8] $75600 * 86.99 = 6576444$

[9] File size on disk after exporting the descriptors to file.

[10] $(1850.99/10552 * 13152888)/1000$

[11] `http://users.eecs.northwestern.edu/~wkliao/Kmeans/index.html`

[12] $15156817/10552 * 13152888 = 18892713839$

A cloud based implementation, like Amazone EC2[13], is a good solution when processing power is needed occasionally. The hardware needed to host a platform, which covers the Netherlands is highly dependent on the software implementation. The current implementation of the software can highly be optimized, which reduces the amount of hardware that is required.

4.2 Performance

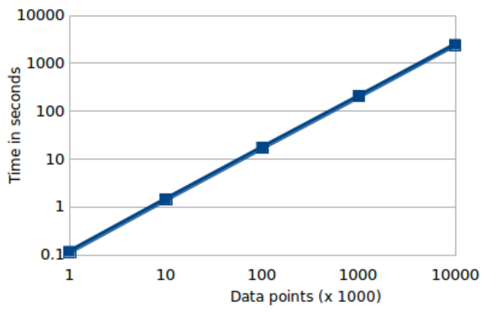
The hardware and software that is used during the testing of the application is described in appendix A.

Tree building

Figure 6 shows the performance of the tree building. Expectedly, it shows a linear relation between the number of descriptors, and the required time.

The tree building time also includes the time for clustering the data using the k-means algorithm.

Figure 6: Time (in seconds) needed to build the tree



Lookup time

Figure 7 shows the lookup time of a single descriptor from the descriptor tree versus the bruteforce lookup time. The bruteforce lookup performance is linear, where the tree lookup is performed in $O(log(n))$. This difference in complexity provides the scalability of the descriptor tree solution.

Image lookup

Figure 8 shows the performance of matching an image. The query image generates 1605 descriptors. The lookup times for those descriptors rises slightly with the size of the tree. The most time however is spent on the matching of the candidates. This time fluctuates slightly based on the found candidates.

Table 2 shows the time each of the matchers take. As expected the geometrical matcher proves to be by far the most expensive. The overhead of the matching is the creation of classes and other routines in the application.

[13]`http://aws.amazon.com/ec2/instance-types/`

Figure 7: Tree lookup vs Brutefoce lookup of a single data point

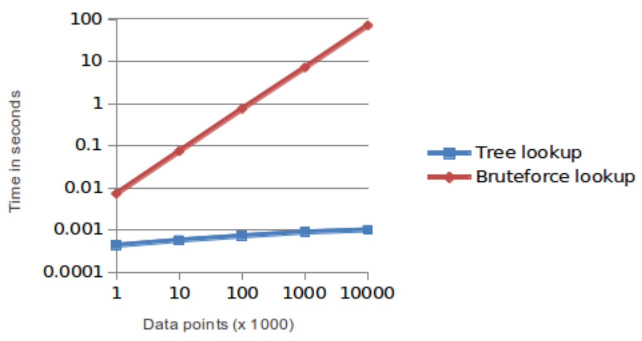
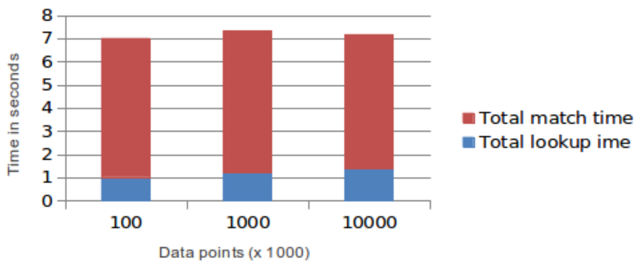


Figure 8: Total time spent for image lookup



Matcher	Time spent (%)
Geometric	97.4314
Statistical	0.0005
Area	0.0392
Overhead	2.5288

Table 2: Time spent in the different matchers

Memory consumption

Figure 9: Used memory in Megabytes

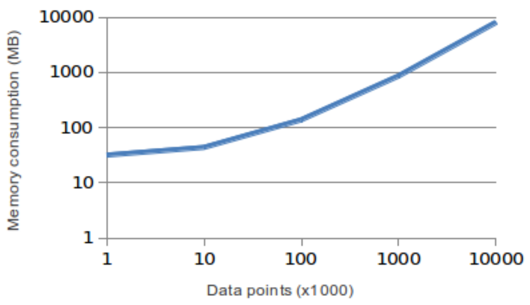


Figure 9 shows the amount of memory that is needed to built a tree. When building a tree with 10 million descriptors 8046MB is used, which is a problem. The filesize on disk of 10 million descriptors is approximately 1300MB. Unfortunately, due the short time frame of the project the exact cause of the problem is still uncovered. A part of the problem is with the used programming language.

4.3 Experimental Concept

In this chapter the experimental concept is described, which is created during the research to demonstrate the achieved results. The concept is an application were images can be queried against a database with geotagged images to find the location where the image is taken. The result of a query is displayed as a visual representation of the data from the matchers.

A link to the source code of the concept can be found in appendix A.3.

Overview

When launching the application, a tree with descriptors from the images in the source database is built and finally a HTTP daemon is spawned, which prevents the application from shutting down and keeps the tree in memory. Each request for a query is handled by the webserver. A new lookup is initiated by a POST request with the contents of the query image. The application first detects and extracts the descriptors from the query image and derives the most similar descriptors from the tree. These results are used in the match modules and the final result is send back to the client as a JSON string. The results are parsed at the client side by a small JavaScript framework into a visual presentation of the received data.

Results

As described in section 3.4 three matching methods are implemented. This section shows some of the results gathered from the experimental concept and shows the effect of the different match methods on various images.

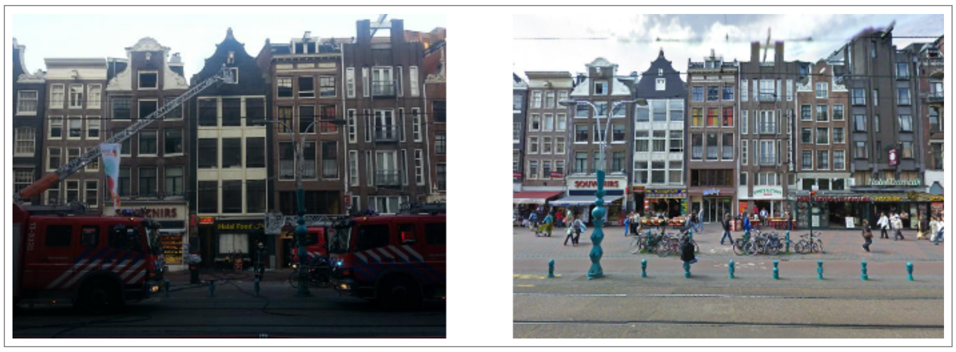


Figure 10: Positive result found based on geometric match.

As shown in figure 10, a positive match is found. This match is found based on the geographical match. In this case the first result from the tree lookup was a positive match, but due the small differences in the scores that could not be systematically concluded. Table 4 shows the first 4 match results from the geometric match method. The difference in score between a positive and negative match is huge and could systematically be determined.

Score	Location	Match
0.009262	Damrak (52.375182,4.895838)	Yes
0.001603	Damrak (52.375132,4.895774)	Yes
0.000008	Laagte kadijk (52.370059,4.912094)	No
0.000003	Anne Frankstraat (52.369447,4.909435)	No

Table 3: The first 4 match results from the geometric match method

In the second example an image with a low resolution and inferior quality was used to query against the database. A match could not be determined from the statistical and geometric match, but the area match gave positive results. The query image is shown on the left side in figure 11. On the right side the positive area result is shown. The query image is taken from a camera with a not ordinary view angle, which makes the matching more difficult. The image is taken on "the Dam" in Amsterdam from the "National Monument on the Dam". The area match in figure 11 shows 4 markers in the yellow circle around the "National Monument on the Dam" and the midpoint of the circle is "the Dam", which is a positive match. On figure 12 the streetview result from the area match is shown.

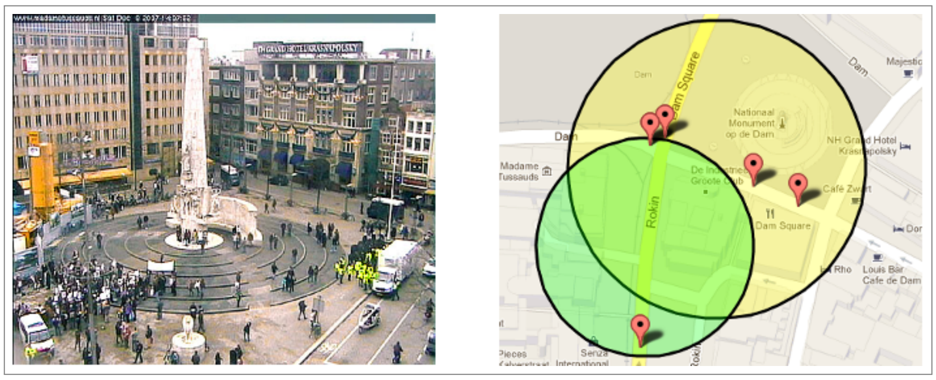


Figure 11: Positive result found based on the area match.

Score	Location	Match
3462	Dam (52.372633,4.893341)	Yes
2145	Nieuwezijds Voorburgwal (52.371575,4.890850)	No
924	Rokin (52.368234,4.892693)	No
808	Rokin (52.372377,4.892954)	No

Table 4: Found groups in the area match method.

The above table shows that a positive area match result can systematically be determined by selecting the group with the highest score if the other matchers give no suitable results.

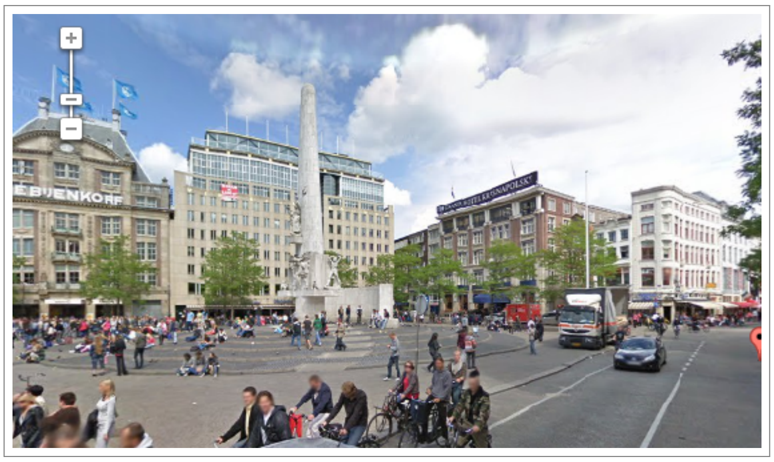


Figure 12: Streetview result from the area match in figure 11

The following example is a positive result based on the statistical match, which allocates a percentage to each result based on the score from the tree lookup. If the scores are proportional distributed a distinct match can not systematically be determined. Substantial deviations only occur when the query image to a large extent is similar to an image in the ground-truth database. That is only applicable on a limited amount of query images, because the conditions when taking a photo like lightning and view angle are almost never corresponding.

However, in this example a query image is found which does meet these requirements. The query image and the corresponding result is shown in figure 13.

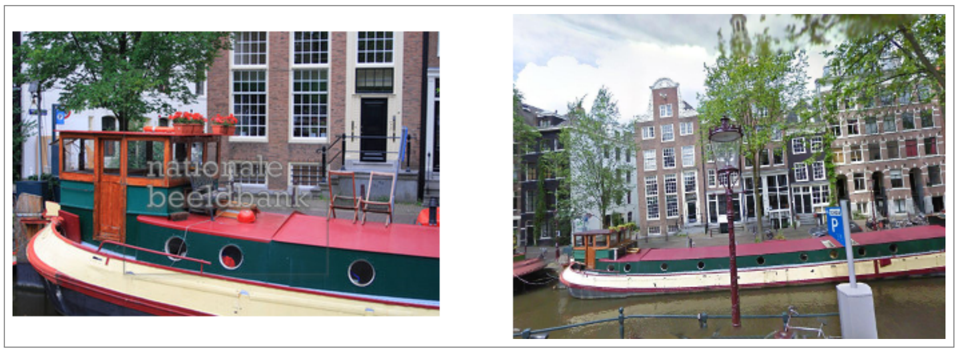


Figure 13: Positive result found based on the statistical match.

As described before, the statistical matcher calculates a percentage based on the score from the possible candidates, which in this case is 403.17 percent for the first match result. The results from the statistical match are shown in table 5. Based on the calculated percentage it can easily be determined that the first match is a positive match. This can be confirmed to compare these results with the results from the geometric match. The advantage of the statistical match is that it is extremely fast in comparison with the geometrical match, as shown

Score	Location	Match
431	Raamgracht (52.369656,4.899106)	Yes
131	Oudezijds Voorburgwal (52.373931,4.898180)	No
130	Oudezijds Voorburgwal (52.373266,4.897867)	No
124	Raamgracht (52.369808,4.899166)	Yes

Table 5: First 4 match results from the statistical match.

in section 4.2.

5 Conclusion

In this research an attempt was made to design and build a system to perform image localization based on looking up an image from a large database of descriptors. As a real world example of a useful implementation of such a system could be a situation where the police find an image of an accident on Twitter, and no information about the location is present.

In this situation, having a system that can quickly search trough a database and suggest possible locations, mean that emergency services can be sent much quicker. The proof of concept presented in this paper shows that such a system can be built. Despite the limited time frame, the system is capable of matching an image against a database of about 10.000 images in less then 8 seconds. Of these 8 seconds, a little more then 1 second is spent looking up the image descriptors from the descriptor tree. Because of the logarithmic complexity of the lookups in the descriptor tree, this means that a descriptor tree containing all images of the Netherlands (˜19 billion descriptors) would only take a little more then two seconds to look up an image. With the time spent after this lookup being constant, a system that yields results within 10 seconds is a very real possibility.

Building the tree however is still a challenge. The processing time required is linear, which is as good a performance as can be expected. As for the memory used, at this point it is possible to keep the descriptor tree in memory. However, if the descriptor count rises, this might not be possible anymore. In this case the tree might have to be stored in a database. Doing this will in turn affect tree building performance, as well as the lookup performance of the system. These things will have to be reevaluated for the implementation.

A solution could be found in realizing a distributed version of the application. If the workload could be split effectively, the performance could be maintained even with much larger datasets. This was however outside the scope of this research.

As for the accuracy of the proof of concept system, no automated tests were run, so no hard figures are available. Devising a way to reliably measure accuracy would be very helpful in further developing the system. To be able to do this, the success and failure conditions of the system need to be defined for the desired application.

6 Further Work

On average 1436 descriptors are detected in each cutout. SIFT has no limit on the number of discovered descriptors in an image. Also descriptors detected in non unique elements, like the sky, trees, water and streets are currently saved. An algorithm is needed which can determine which SIFT descriptors in an image are important and the ones less important.

The created concept can not automatically determine the result of a match. Manually analyzing the result of the matchers is required to determine the result of a match. However, in the created match algorithms scores are given to each result, which define the quality of the found candidates. It is possible to extend the system, to automatically determine the result of a match based on those scores.

If the system is able to determine the result of a match, it is also possible to create automated tests with random images from external sources, like Flickr[14] or Google Images to evaluate the system based on the standard precision and recall measures.

The code for the concept is written in Python, which currently consumes a lot of memory when building the descriptor tree. A low level programming language, like C, is needed to control memory allocation and lower the memory consumption.

Currently it is possible that not all objects on a Google Streetview panorama are being cutout and saved to the database. The solution in this paper is not completely reliable. A procedure could be developed which cuts all possible relevant objects from the panorama, as shown in figure 2. Instead of saving each cutout to the database, query the cutout against a set of images belonging to that area to determine if it already exists in the database. By implementing such a method, the coverage of the ground-truth database can be improved, without causing overhead.

The time spent in the matchers can be reduced by limiting the number of possible candidates. Currently 20 possible candidates are returned, which could be lowered to 10. This reduces the match time by 50

[14]http://www.flickr.com/

References

[1] Global positioning system standard positioning service performance standard, 9 2008.

[2] Beeldmerktechnologie naar de praktijk, 12 2012.

[3] Halil Basgin. Parallel clustering algorithms with application to climatology. 12 2007.

[4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[5] Carl Doersch, Saurabh Singh, Abhinav Gupta, Josef Sivic, and Alexei A. Efros. What makes paris look like paris? *ACM Transactions on Graphics (SIGGRAPH)*, 31(4), 2012.

[6] J. A. Hartigan and M. A. Wong. A K-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.

[7] James Hays and Alexei A. Efros. Im2gps: estimating geographic information from a single image. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[8] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society.

[9] David Nister and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 2161–2168, Washington, DC, USA, 2006. IEEE Computer Society.

[10] Jan Salmen, Sebastian Houben, and Marc Schlipsing. Google street view images support the development of vision-based driver assistance systems. pages 891–895, 2012.

A Implementation details

This appendix describes the used software and hardware for the created implementation.

A.1 Software

This section briefly describes the used software to build and test the experimental concept.

- OpenCV 2.4.3
- Python 2.7.3
- Numpy 1.6.1
- Scipy 0.9.0
- Ubuntu/Linaro 3.2.0-35-generic x86_64

A.2 Hardware

This section briefly describes the used software.

- Intel(R) Xeon(R) CPU E5-2690 2.90 GHz (8 cores)

A.3 Source code

The source code created for the Google Streetview crawler and the experimental concept is published online as-is at the following location: `https://www.os3.nl/2012-2013/students/ttimmermans/rp1`. Be aware that the software is in an experimental state and that it is provided as-is.