Masters thesis

Automatic Interpretation of Nijntje-images

D.R. Hulswit

Student number 9705937

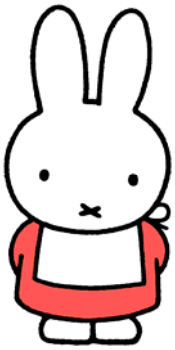
Department of Computer Science
Human Media Interaction

University of Twente

16 August 2006

Supervisors

Ir. Dennis Reidsma
Ir. Rutger Rienks
Dr. Roeland Ordelman



University of Twente
Enschede - The Netherlands

Summary

In this masters thesis the design and implementation of a system to interpret images from the comic Nijntje is discussed. Nijntje is very simple comic, drawn by Dick Bruna. The goal was to build a system that can recognize the objects present in an image from a Nijntje book, even when they are partly occluded.

The approach to recognition that was taken is based on shapes, the building blocks of the objects in the Nijntje comic. First the image is analyzed and a semantic representation of the shapes in the images is created. This representation is then used to interpret the contents of the image.

The system that was built is divided into two separate components: the image analysis stage and the reasoning stage. In the image analysis stage all image processing is done (including steps such as finding the shapes in an image and classifying the forms of those shapes). The steps have been implemented in the ParleVision [UT06] framework.

To be able to describe the objects in the Nijntje universe (at least those from the available books) a description system was built, which was then used to create an ontology of the objects. Using the knowledge obtained while creating this ontology, description logics systems were evaluated as possible candidates to build the reasoning system in. Finally, it was concluded that they are not expressive enough to describe all constraints for sufficiently complex objects.

A custom recognition system was built to implement the reasoning stage, using an iterative approach. The first iteration ignored occlusion, and aimed to do recognition on completely visible objects. The implemented interpretation process works by searching an interpretation tree of hypotheses. Due to time spent on controlling resource consumption spinning out of control, no time was spent on further iterations to add recognition for occluded objects.

The final system can recognize the objects in images with only one ore two complex objects reasonably well, provided the number of shapes is not too high, and the objects themselves are fully visible. Occlusion usually prevents the objects from being recognized, while large numbers of shapes cause the implementation to consume huge levels computing time and memory.

Finally, it is concluded that the purely semantic approach that was chosen is not suitable for the recognition of Nijntje images. Because of this problem, the goal of being able to recognize partly occluded objects that are present in an image is not attained.

Table of Contents

1 Introduction

The interpretation of the content of images in general is a very complicated matter. Images can contain a lot of ambiguous information, and a lot of intelligence is usually needed to interpret them. Currently, no system exists that can automatically determine the contents of unconstrained images (e.g. snapshots of 'penguins walking on the beach').

This assignment seeks to design and build a system to interpret images from the comic Nijntje, drawn by Dick Bruna. This comic is very minimalist, objects and animals are usually drawn with as few shapes as possible. Also, not many of them are present within an image. Because of these characteristics, it is hypothesized that not much ambiguity is present in a Nijntje image compared to unconstrained images, and that interpretation of such images is possible.

1.1 The assignment

The assignment consists of designing and implementing a system that can recognize all depicted objects in an image containing a Nijntje-comic, and which can identify all relations between these objects (spatial relations, 'holds-on-to', etcetera). This system should be constructed in such a way that the biggest problems in recognizing objects (partial occlusion of objects, objects composed of different parts) can be solved in a satisfying way. The recognition process will pass through stages such as feature-extraction (find which lines and shapes are present), object-identification and finally identification of relations between objects.

An important part of the assignment is the construction of an ontology of the Nijntje universe, in which all types of objects that can be found are described. These descriptions must contain enough information so that they can be used to recognize objects effectively. The challenge here is finding a representation that can also be used when only parts of an object are visible.

1.2 Relevance

The recognition system described in the assignment can be used for several purposes. First, it can be used in an educative game, in which little children can use a stack of physical Nijntje-objects (like wooden puzzle-parts with an image of a Nijntje object printed on top of it) to compose their very own Nijntje-scene. An image can be captured of this scene using a camera. The interpretation system can be used to interpret the scene, after which the system can comment on it, or tell a story involving the recognized scene.

Furthermore, this assignment can give hints how to recognize stylished comics and shapes in general. These are used routinely on the World Wide Web. However, this kind of visual information is largely inaccessible for blind people, who are dependent on textual information. When no alternative content is present for the comics, or that content is too concise, the described system could try and interpret the visual content, offering an alternative for the visual information.

The described system must support large variations in form and composition of objects. For example, animals in the Nijntje world can be composed of different parts, such as different types of clothing, wearing hats, etcetera. The system must be able to recognize them all, even when they are partly occluded. This makes this system very suitable for the recognition of industrial parts, where the parts themselves can vary in form and/or composition. There would be no need for building separate models for each of the possible variations, a single model and a description of the variations would suffice.

1.3 Approach

1.3.1 Determining characteristics of Nijntje images

The first step in finding out how to approach the interpretation problem was to analyze a small number of images, to determine their common characteristics. This analysis led to the following observations:

- Objects in images of the Nijntje comics are composed of very simple shapes, separated from each other by black lines. Furthermore, the number of different colors used is very low, and are always chosen from a set of 7 colors (including black and white).
- Simple objects are usually composed of the same types of shape, in the same arrangement. There are only a few types of other objects that can have a more complex structure (rabbits, sheep, etc.).
- Objects can be adjacent to each other, and can also overlap each other.

Furthermore, if two multi-shape objects are present in an image, and these two objects are adjacent, there is no easy way to automatically determine, without user intervention, which shape belongs to which object. Also, all black lines separating the shapes always have roughly the same thickness.

1.3.2 Literature

Object recognition is one of the major goals of the computer vision research field. Especially the general variation is important; being able to recognize the contents of an unconstrained image (an image which can contain literally anything) has a lot applications.

In general, a system for detecting objects takes the following steps [POPE94]:

- Feature detection: detect features of the image (edges, textures, etcetera).
- Perceptual organization: group the features into probable objects.
- Indexing: using the identified features, select the most likely models from a database of objects.
- Matching: Find the best match between the models and the image features.
- Verification: Decide whether the matches suggest if the modeled object is present in the image.

In the feature detection stage, the so-called low-level vision processes are executed. The input image is analyzed and segmented into a number of labeled primitives. The high-

level vision parts then interpret the image using the primitives, using knowledge about the domain of the image.

Usually, the data generated from the low-level vision processes are compared to models of objects that can be present in the scene. There are several different approaches [SUET92]:

- Feature Vector Classification.
- Fitting Models to Photometry.
- Fitting Models to Symbolic Structures.
- Combined Strategies.

The first two strategies are not very well-suited for this domain as they mostly rely on the assumption that the objects that have to be recognized can be easily separated from the other objects present in the image. In Nijntje images however, this can't be done easily, as there is no way of telling directly if two shapes belong to different objects.

Also, the images have a very simple structure. I have chosen to use the combined approach. In the image analysis Feature Vector Classification is used to recognize individual shapes with known forms by calculating a feature vector describing a shape, and matching to a list of known vectors in a database. After that, all information about the image is then converted to a symbolic description, which is then analyzed to find instances of objects from the domain.

1.3.3 Approach

I have chosen to break up the system into two stages: the image analysis stage and the reasoning stage. The first stage will handle image manipulation tasks, such as identifying shapes, classifying their form, etcetera. The second stage will use the information generated by the first to construct a coherent interpretation. Each of the both stages uses different techniques and works with different types of information. Because of this, it was logical to split them up.

Shapes are used as the primitive elements in object descriptions, because they are the basic elements of which objects in the Nijntje universe are composed.

Structure

In the image analysis stage, the input image is broken up into the different shapes it comprises. These shapes, together with other relevant information (location, form and position) are then passed to the reasoning stage. This first stage corresponds with the feature detection step.

In the reasoning stage, the image contents are then compared to models of the objects from the Nijntje world. The output of this stage is a consistent interpretation of the image contents. This corresponds with the other steps of the recognition process.

1.4 Related work

Despite extensive search, not much related work has been found. In [NEUM04] description logics are used to interpret images, but the primitive elements used there are

complete objects in themselves. In Nijntje images, this can't be done, and we must recognize the objects themselves from their parts. Also, the interpretation tree approach displayed in [GRIM90] is suited for recognizing parts of an outline of a single occluded shape, but not for objects composed of multiple shapes.

1.5 Structure of this thesis

In this thesis the design, implementation and performance of the built interpretation system is described.

Because the two stages are very different from each other in the strategies they use, they are described in two different sections; first the image analysis stage in chapter 2, and then the reasoning stage in chapter 3. Finally, in chapter 4 conclusions are drawn about the entire process and its results. Also, some recommendations are given.

2 The image analysis stage

2.1 Introduction

As mentioned before, the first step in image recognition is feature detection, in which features such as edges and textures are identified. In the Nijntje image interpretation system, this first step is implemented by the image analysis stage. This stage is discussed in this chapter.

2.2 Available images

In total, five books with Nijntje images were available: 'Het huis van Nijntje' [BRUN97], 'Nijntje in de dierentuin' [BRUN96a], 'Nijntje aan zee' [BRUN96b], 'Nijntje en Nina' [BRUN99] and 'Nijntje danst' [BRUN02]. Each of these books contained about 14 images.

2.3 Image contents

Nijntje is a cartoon. What it has in common with most (simple) cartoons is that the image is composed of shapes with a single color, separated from other shapes by black lines. The color black is mostly used for freestanding, small shapes like eyes, mouths and lines. Large black shapes are not very common. Besides black and white, only the colors red, blue, brown, green and yellow are used.

The primary elements of the image are the colored shapes. The black lines in between them can be reconstructed using the boundaries of the shapes. Only lines that do not separate two shapes, and black shapes themselves, carry additional information that cannot be easily inferred from the outlines of the shapes.

Furthermore, some elements in the Nijntje cartoons have predictable and somewhat rigid forms. Examples of these are heads of rabbits and the coach-work of cars. If these shapes can be identified by the image analysis, that information can be used as a hint to guide the interpretation process.

Lastly the composition of elements always follows the same general rules. For example, a head usually has two eyes and a mouth, and a rabbit is composed from a head, clothing, hands and feet.

2.3.1 Relevant information from an image

During the initial review of the available images the following types of information were deemed necessary for interpretation (at least for an initial implementation):

- The shapes in the image.
- The lines that do not separate shapes.
- Adjacency information (which shapes and lines are adjacent to which other shapes).
- Is-within information (which shapes and lines are enclosed within another shape).
- The color of shapes.
- The position of shapes and lines within the image.

2.4 The steps of analyzing image contents

To analyze the content of an image, the following steps can be taken (these steps are displayed in the diagram on the next page):

1. Convert the acquired image to the set of known colors. This includes compensation for color variations between images, and image noise caused by scanning an image.
2. Find the individual shapes in the image.
3. Gather information about the individual shapes.
   - Identify which shapes are adjacent to other shapes.
   - Find out which shapes lie within another shape.
   - Identify shapes with known forms.
   - Find blacks lines and shapes that do not separate shapes.

Step 2 needs some extra work, because black shapes are used in some images. These black shapes have no visible boundaries between them and lines adjacent to them. An extra step is needed to separate these black shapes from those lines.

All of these steps have been implemented in ParleVision, a framework for the development of video software. In this framework, several image analysis services are already available, drastically lowering the implementation time of individual analysis steps.

2.4.1 Acquiring images

To analyze an image, it has to be digitized first. The actual acquiring process of images falls outside of the scope of this thesis. However, the problems associated with the acquiring process will still have to be considered.

Usually, an image from a book is digitized with a scanner. The scanning process isn't perfect and introduces little color variations. This is due to hardware noise, dust, little spots on the paper, etcetera. Also a scanner performs best when the scanned item lies flat on the glass plate of the scanner. When scanner books however, paper close to the binding of the cover is somewhat lifted from the plate, causing loss in clarity and sharpness. These problems will all have to be compensated for.

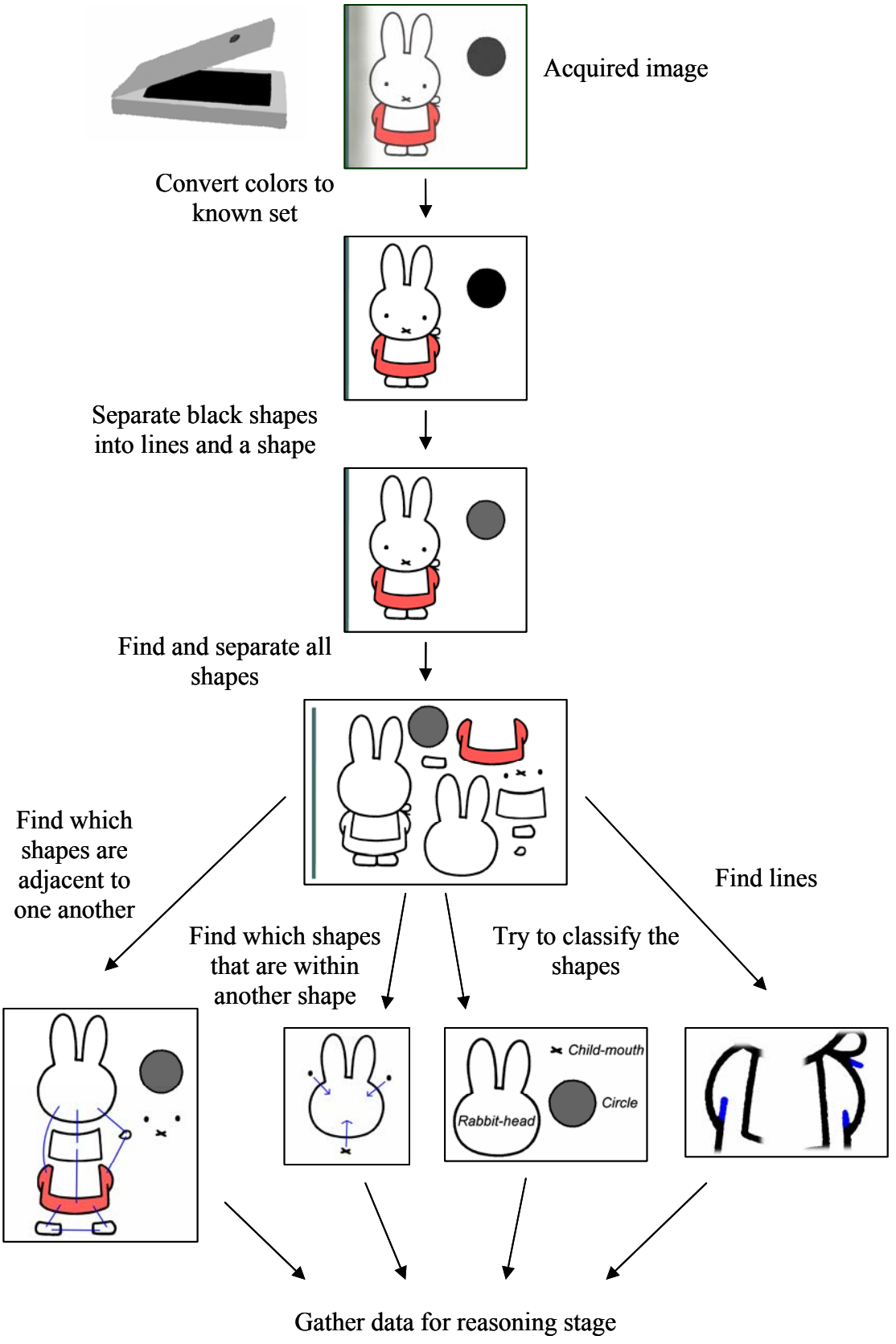
2.4.2 Converting colors to known set

In the Nijntje series, only five different colors are used besides black and white. These are red, brown, blue, green and yellow. Images however, are typically scanned in with 24-bit color depth (with 16 million possible colors). To reduce the number of colors in the image to the set of used colors in Nijntje, a color reduction step needs to be taken. This process is called color quantization.

Color quantization is normally done in 4 stages [HECK82].

1. Sampling the original image for color statistics.
2. Choosing a color map based on those statistics.
3. Mapping the colors to their representative in the color map.
4. Quantizing and drawing the new image.

Image analysis steps

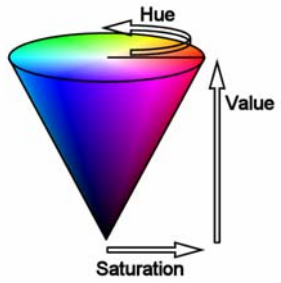


This organization of steps is usually used when the result colors are not known, but must be determined dynamically (for example, when reducing the number of colors within an image). The situation is a little different here because we already know the result colors.

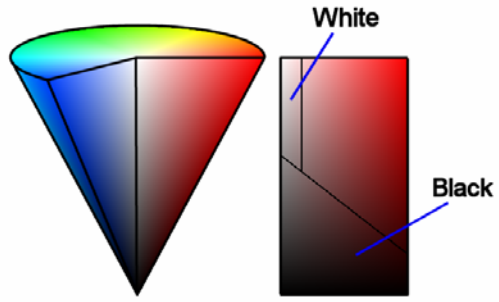
The color quantization step was not regarded as a primary research area, so some shortcuts were taken. For example, a per-pixel algorithm was chosen to convert the image, instead of an algorithm that took large patches of roughly the same color into consideration.

A few approaches were tried: first a representative RGB-color was chosen for every Nijntje color, and every pixel in the input image was mapped to the Nijntje color with the smallest distance in the RGB-color space. This however, did not prove to be a very robust approach; too many pixels were mapped to the wrong color.

The next approach was to convert the input pixels the to the HSV color space. In the HSV color space colors are represented by 3 values, namely hue, saturation and value (the latter being roughly akin to brightness).



HSV Color space

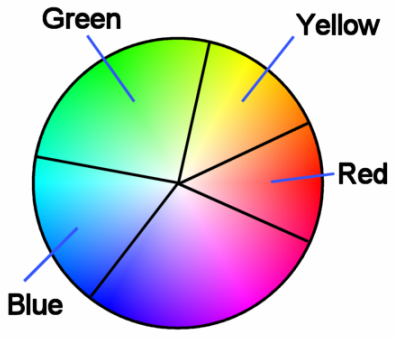


Boundaries for black and white

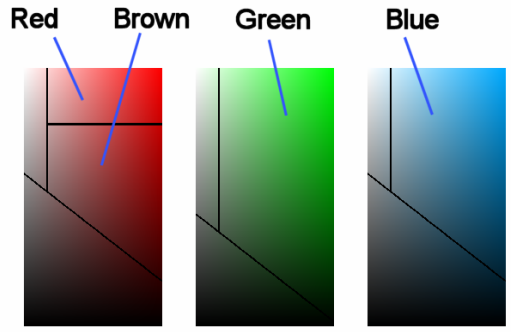
Black and white can be identified by the value and saturation of a pixel. A low value (or low saturation and somewhat higher value) identifies a pixel as black. A white pixel is identified by just having a low saturation.

In the HSV color space, colors can easily be identified by their hue. For every color, the range of hues was determined by converting a number of images from the set of available images to HSV, and manually inspecting the range of hues for every color.

It was found that brown and red shared their hue range, but could be separated from each other using the saturation.



Color hue boundaries



Individual color value/saturation boundaries

The only problem left was the color green, which turned out to be drawn darker than the other colors. Because of that, green was often quantized as black. The solution for this was to first execute color detection. When the color green was detected, the threshold for black was lowered dynamically.

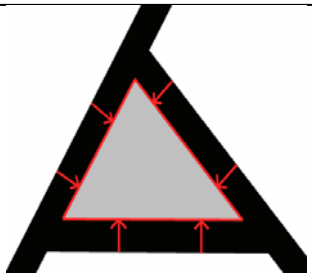
Conversion done in this way is not perfect; some pixels will still be identified as another color. It was chosen not to address this problem in this step, but to defer it to the step where the shapes are identified.

2.4.3 Separating black shapes into shapes and lines

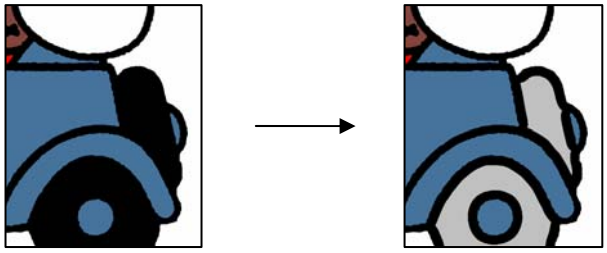
In a Nijntje image, large black shapes and lines cannot be distinguished from each other very easily. Trying to do this is a fundamental different process from finding the other (non-black) shapes, and therefore an additional step was introduced for it.

Black shapes are not very common in the available books, so a quick solution was chosen. This solution consists of taking all the black shapes, and peeling off one line thickness by a process called erosion.

Every black pixel that remains after this erosion process is recolored with a special color that identifies black shapes; black lines keep their original color, which, after this step, is used to identify lines.



Peeling off 1 line thickness



Example of finding black shapes and recoloring them

2.4.4 Finding all shapes

The next step is analyzing the structure of the image; identifying the lines and the shapes the image is composed of.

The first approach to this problem was based on the lines; every line would be extracted from the image and then approximated with vectors. This process is called vectorization.

Vectorization algorithms normally work on binarized images (images in which every pixel is either on or off). They can be divided into five categories:

- Thinning — Thinning works by peeling off layers of pixels of contours, until a line of 1 pixel thick is left [STEF71].
- Contour-following — This process works by tracing the outline of contours in an image, lines are detected when parallel contour outlines are found [HAN94].
- Graph-based — These algorithms work by sampling an image, and then construct the lines from those samples [MONO93].
- Pixel tracking — This method works by tracking pixels within a line (usually in a zigzag pattern, and following the line in that way [DORI99].
- Line tracking — Here, when a piece of line is found (the seed segment), it is extended and followed around curves [SONG00].

Nijntje images usually have lots of sloped and bent lines, circles and junctions. The graph-based and pixel tracking methods are not very robust in such an environment. Furthermore, the thinning process is very sensitive to pixels that have been misclassified by the color quantization step.

This only left the line tracking and contour following algorithms as viable. However, at the moment of analysis the solution to separate black shapes from lines was not thought of yet. Both algorithms cannot cope with large black shapes.

Because all categories of vectorization were eliminated, another approach was needed to analyze the structure of the image. The next approach tried was a shape-based approach.

Finding all the shapes present in the image proved to be fairly easy; the OpenCV library [INTE06] that was already in use in the ParleVision framework provided functions to find all contours in a binary image.

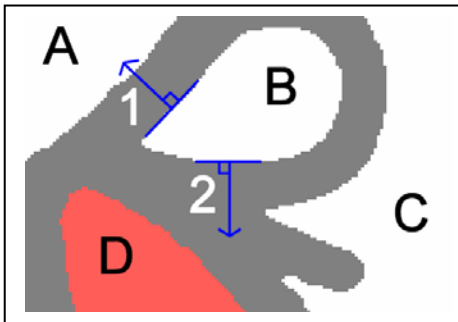
However, this function is not usable in an image with multiple colors. The quick solution proved to be to loop through all the colors, and to generate a binary image containing only the pixels of that color turned on. The contour-finding function is then applied, yielding a list of all shapes of the current color. This process will find all shapes.

A disadvantage of this approach is that it doesn't give in much information about the adjacency of shapes, and it gives no information at all about lines which do not separate two shapes. Separate processes are needed to extract that information.

The problem that pixels are converted to the wrong color during the color conversion (due to image noise or at the border between two colors) is addressed at this stage. This is done by recoloring shapes whose size falls below a certain threshold to the next processed color. This approach uses a lot of CPU time, and becomes increasingly costly as the noise increases.

2.4.5 Determining shape adjacency

The process of finding all the shapes in an image that was just described does not give any information about what shapes are physically adjacent to each other. This information, however, is quite important when searching for objects.



Tracing to find adjacency

As stated earlier, shapes in a Nijntje-image are separated from each other by black lines (displayed in grey). The easy way to find which shapes are adjacent to each other is to identify all parts of black lines that form lines. If there are two different shapes on either side of a line, those shapes are adjacent to each other.

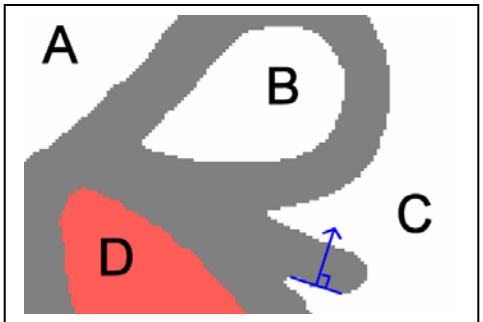
This procedure can be executed by tracing the contours of all the black shapes in the image that form lines, as found by the shape-finding procedure. At specific sample points, the direction of the contour is determined, and a line perpendicular to that direction is traced into the black shape.

If within a certain threshold distance (set to the maximum line thickness plus a small margin) no other shape is found the trace is stopped.

In the example image, trace 1, starting from the contour around shape B, eventually finds shape A. As a result, shape A and B are marked adjacent. Trace 2, however, ends while not having found another shape, and is discarded.

2.4.6 Finding lines that do not separate shapes

The procedure for finding lines which do not separate two shapes is very similar to the procedure for finding which shapes are adjacent. As with the adjacency-finding procedure, the outer contours of all black shapes are traced, and at sample points a line is traced perpendicular to the contour direction.



Tracing to find lines

A line that does not separates two shapes is found when the shape is found at the end of the trace is the same shape as the one on the starting side. In the example image, the tracing line starts at the contour of shape C, and on the other side of the traced line, shape C is found again. Every streak of two parallel contours found in this way is translated into a shape.

2.4.7 Finding which shapes are within another shape

A very simple (but not very efficient) way to find out which shape completely lies within another shape is to draw all the shapes in such a way that the outer shape is always drawn first and any holes within that shape are filled up. If it is detected that a shape is drawn over another shape, that shape must lie within the earlier drawn shape.

Figuring out a suitable draw ordering is very simple; the y-coordinate of the highest pixel in the shape can be used for ordering. This is because the highest pixel of a shape enclosing another shape must lie higher than the highest pixel of the enclosed shape.

The procedure thus becomes the following:

1. Order all the shapes in the image on the highest y-coordinate.
2. Draw the shapes in the order with decreasing highest y-coordinate. If the first drawn pixel has already been filled in by another shape, the current shape lies within the shape that drew that pixel earlier.

This procedure discovers only the direct lies-within relationships, so after this the transitive closure of the found relation must be constructed (if A lies within B and B lies within C, then A must also lie within C).

2.4.8 Classification of shapes

Some shapes in Nijntje image always have roughly the same form. Examples of these kinds of shapes are the head of Nijntje, the mouths of children and adults, squares and circles.

Information about the form of a shape can provide a good initial direction in the reasoning process; if a shape in an image is known to be a rabbit-head, it stands to reason that there is a rabbit in the image.

The shapes that have known forms aren't always exactly alike. They can differ in size, or be rotated (e.g. Nijntje with tilted head).

Techniques

This problem has been studied extensively, and is known as shape retrieval: given a shape, find shapes that are similar to it.

There is a great number of methods and algorithms available on this subject [SAFA00]. Here, we must use a method that can operate on a shape description, as texture and color descriptions are either not very important or not very discriminative.

Methods can be characterized in two ways: what kind of information they use (the boundary, or the entire region), and how they operate (by using geometry or by applying transformations).

I have chosen the Fourier description algorithm [OTTE89]; which is a boundary-based algorithm that applies a transformation. It works by first constructing a list of numbers (a so-called vector) representing the boundary of a shape. That vector is transformed using the Fourier Transformation. This results in a vector containing complex numbers, called the Fourier descriptor of the shape.

By normalizing the representation of the boundary of the shape (dividing by the size of the shape), size invariance is achieved. This means that two shapes that have different sizes but are otherwise identical have the same Fourier descriptor.

The Fourier Transformation transforms a list of numbers to the frequency domain, and results in a list of complex numbers. Each of these complex numbers describe the

amplitude and phase of a certain angular frequency. The absolute value describes the amplitude, and the complex argument describes the phase. The information about the rotation of the shape is only expressed in the phase and thus in the complex arguments.

So, by discarding the phase information in the Fourier descriptor (by keeping only the absolute value of the complex numbers in the descriptor), the descriptor becomes rotation-invariant (shapes that differ only in rotation will get the same descriptor).

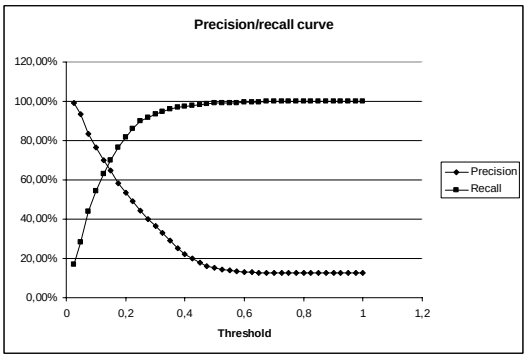
Choosing a shape representation

The shape representation describes how the boundary of a shape is translated to a list of numbers that can be used to calculate a Fourier descriptor. The shape itself is described by a pair of functions x(t) and y(t), which represent the respective x and y-coordinate of the outline of the shape at a certain point, relative to the coordinates at the center of the shape. Using coordinates relative to the center of the shape maked the representation independent of its location within the image, and so translation-independence is achieved.

There are multiple representations possible, some examples are: [OTTE89]

- The position function ($z(t) = x(t) + iy(t)$).
- The derivative of the position function (the tangent function). ($z'(t) = x'(t) + iy'(t)$)
- The centroid distance function ($r(t) = \sqrt{x(t)^2 + y(t)^2}$).
- The area function (the area in triangle defined by boundary point, next boundary point and center point), ($A(t) = \frac{1}{2}\left|x(t)y(t+w) - x(t+w)y(t)\right|$)
- The tangent angle function ($\theta(t) = \arctan(\frac{y(t) - y(t-w)}{x(t) - x(t-w)})$)

None of these representations were described as the 'best' in literature. Also, no information was available how many numbers should be used (the more numbers are used, the more noise-sensitive the descriptor becomes). To see which representation is most suited for this problem a list of viable representations was compiled, and tests were run to determine which one had the best performance.



Example precision/recall curves

To execute these tests, a list of shapes of known forms was compiled, and descriptors were generated for those shapes using the tested representation. The distances between all descriptors were calculated.

For every shape the set of shapes whose descriptors lie within a threshold distance of the descriptor of the original shape was determined. The shapes in these set were designated the retrieved shapes. A shape is considered a match when it has the same known form as the original shape. Using this information, precision and recall rates were

computed[1]. This procedure was executed for different thresholds, and put into graphs.

With these graphs, the point was determined at which the recall line intersects with the precision line. This point represents a specific tradeoff between finding as many matching shapes as possible, and the least possible false-positives.

The following functions were tested:

- The position function z(t), with descriptor sizes 32 and 64.
- The centroid distance function r(t), with descriptor sizes 32 and 64.
- The area function A(t), with descriptor sizes 32 and 64.
- The tangent function z'(t), with descriptor sizes 16, 32 and 64.
- The tangent angle function $\theta(t)$, with descriptor sizes 32 and 64.

(The size of the descriptor had to be a power of two due to the used Fourier Transformation algorithm).

It turned out that the tangent function (the derivative of the position function) with 16 numbers in the descriptor had the precision/recall intersection with the highest percentage (around 70%). The difference with some other representation/descriptor size combinations was very low however. A representation had to be chosen though, so I chose the tangent function with 16 elements.

Shape database

To be able to classify the shapes, a database of known shapes was needed, as well as a quick method to classify a shape using that database. To build this database, shapes with forms that seemed recognizable were collected from the available images. These were manually organized into classes. For every class some example shapes were chosen, and their Fourier descriptors were calculated.

These descriptors were then clustered by the K-means clustering algorithm [SHEH04]. This algorithm results in a list of vectors (called centroids), each of which is the center of their cluster of similar shape descriptors. A shape can be classified by calculating its Fourier descriptor, and finding the centroid that has the least distance to that vector. If the distance is lower than a threshold, the shape is assigned the class associated with that centroid.

The K-means algorithm operates on vectors in the Euclidean space, and calculates the vectors that are the center (called the centroid) of the clusters found by the algorithm. The algorithm works in rounds. A rounds starts with first determining for each centroid the cluster of vectors for which that centroid is the closest one (using the Euclidean distance). In the second step, every cluster is recalculated as the average of all the vectors in its cluster. The algorithm stops when the positions of the centroids stabilize.

The outcome of K-means algorithm depends very much on the initial positioning of the clusters. The KKZ centroid placement algorithm [KATS94] has been used for this,

[1] The precision is the number of retrieved matching shapes as percentage of the total number of retrieved shapes, while the recall is the number of retrieved matching shapes as percentage of the total number of matching shapes.

because it helps the K-means algorithm to generate clusters with relatively good cluster separation [HE00]. Also, it is an automatic approach, which doesn't take any parameters to tune. With this algorithm, the first centroid is randomly chosen from the vectors that will be clustered. The other centroids are then chosen by taking the vector farthest away from all existing centroids.

A parameter for the K-means algorithm is the cluster count, which determines the number of centroids. The output was analyzed for runs with 50 centroids (based on the number of classes found doing a quick manual classification) and with 70 centroids. The latter could discriminate more classes of shape forms, so it was chosen as the final number of clusters.

The last step was to review the types of shape contained in every cluster. If a cluster contained only shapes of a single type, the centroid of that cluster was associated with that type.

Finding the form of a shape

The process of finding out the form of a shape now boils down to first calculating the Fourier descriptor from the representation of the boundary of the shape.

Then, the distance of that descriptor to all the centroids in the database is calculated, as well as which centroid has the least distance to the shape descriptor. If the distance to that centroid is less than the threshold and the centroid has an associated type, the shape is also associated with that type.

This threshold is needed, because when radically new shapes are found, there is always a centroid that is closest-by; but the new shape probably won't have be of the type of the shapes within that cluster. The maximum distance of any vector used in the clustering process to its centroid was used as the threshold value. In this way, every known shape would be classified.

Performance

The images in the five books that were available together contained around 1400 shapes. By manually organizing these shapes into classes, a total of 39 different shape classes were found (29 classes that uniquely identified a part of an object, and 10 geometric forms, such as squares, circles, etcetera).

After the clustering process, a total of 23 shape classes could be identified by their descriptor. In total, 241 shapes could be recognized as member of one of those classes, which is about 17%.

For 17 of the 23 classes, all shapes that belonged to that class were recognized (in total 87 shapes). For the remaining six classes, some, but not all of the shapes belonging to them were recognized (in total 154 shapes).

2.4.9 Gathering the data for the reasoning stage

The reasoning stage is expected to take a very long time to execute. Moreover, its output is not visual in nature. Given this, it is not practical to integrate it into ParleVision, which

is primary a video-processing application. It is better to output the information gathered in the image-analysis stage into a text-file, and to use that as input for the reasoning stage.

2.5 Results

The image analysis system as a whole performs reasonably well. Almost no spurious shapes due to image noise are detected anymore because of the compensation measures that were taken in the shape finding process. However, because the color quantizer has fixed color boundaries, it is very sensitive to differences in lighting conditions. This was not a problem in the set of available scanned images, but in other (less controlled) situations this might prove problematic.

The shape finder works quite well, although the used OpenCV functions join adjacent contours of the same color separated by a diagonal black line of only one pixel thick.

Shape form classification also works reasonably. It has problems with small shapes, because they are either not large enough to provide 16 outline samples, or little deformations have too great an impact (those deformations that may just be a few pixels in size can e relatively big compared to the size of the shape). In larger, more complex shapes the small number of samples might cause problems (because the small number of sample points doesn't give a good enough representative of the entire outline). No classifications failures were explicitly diagnosed with this problem as a cause, but the reason of that is probably lack of searching.

All other components performed as expected.

2.5.1 Genericity

At the moment, most of the processes in the image interpretation stage have been specially engineered for Nijntje comics. The most notable example of this is the existence of the set of Nijntje colors, which every processor that deals with colors uses. Also, it is assumed that shapes have a single color (color gradients are not supported).

It is probably very easy to adapt the processors to recognize another comic which also use a fixed set of colors, or to any situation where only a limited number of colors occur in large patches. The only process that will need much work will be the color quantizer, because that process is specifically geared to the Nijntje colors. However, due to the focus on shapes, it is probably nearly impossible to adapt the processes to real world images like photos and videos.

3 The reasoning stage

3.1 Introduction

The second part of the assignment consisted of designing and building a system that interprets the contents of an image, using the information generated in the image analysis stage. The analysis, design and implementation of this system are described in this chapter.

3.2 Analysis of the objects in the Nijntje world

The first step in building an interpretation system is finding out what exactly has to be interpreted. An important goal is to find patterns that can be used to take shortcuts. This had already been done very roughly to determine what kind of information could be useful for interpretation needed to be extracted in the image analysis stage.

A more thorough analysis was carried out in two phases. In the first phase, a very rough description was made of the objects present in the available Nijntje books. All relevant information was written down, but without paying attention to consistency with earlier descriptions. During this process, the structure needed for the descriptions was slowly worked out. In the end, a system was designed that could describe most of the objects that had been found in the books. The system was designed to be machine-readable, so it could be used very easily (see appendix A for this system).

In the second phase, the images from the available books were described again, using the newly designed description system. This resulted in consistent descriptions for most of the objects in the books.

3.2.1 Analysis outcome

In total, 71 images from five books were analyzed. In these images, a total of 183 different objects were found. After the second round of describing the objects in the images, around 80 types of objects were described fully.

Using the descriptions, three books were annotated. The images were analyzed by the image analysis stage, and the shapes were manually matched to the descriptions of the present objects.

In total, 183 objects were visible in the images. Of those, 73 objects (around 40%) were partially occluded. For 53 of those objects another, completely visible, object of the same type could be found. Comparing those occluded objects to their non-occluded counterparts, on average around 60% of the original outline was still visible.

3.2.2 The description system

The first thing that stands out when looking at the Nijntje images is that objects (cars, plates, dresses, etc.) are composed of shapes. The structure of those shapes (how they are connected, their form) is usually very similar for objects of the same type. Therefore, it seems logical to describe objects as a set of shapes that have to adhere to a list of constraints.

Such a set of shapes, together with their constraints, is called a template. A template has a fixed number of fields, to which a shape can be matched. A field can be mandatory or optional. Furthermore, a template has constraints that must be satisfied by the shapes matched to the fields. To store information about shapes, every shape has several attributes describing the color, form, position, etcetera. A template can also contain attributes, which are calculated using the attribute values of the shapes matched to its fields.

In the description system, two different types of constraints can be distinguished: relational constraints and attribute constraints.

- A relational constraint is used to mandate adjacency or is-within relations between two fields. Using these constraints, the main structure of an object is defined. This can be used by the interpretation system to find the entire object in a structured way. The relational constraints cannot be used for controlling relative positions of shapes, because positional information is only carried by attributes.
- Attribute constraints can operate on a single shape (for example, limiting the acceptable colors for a shape), or can establish a relation between attributes of shapes matched to two fields. Spatial constraints are also handled by attribute constraints, using the attributes containing the coordinates of the bounding-box.

The main structure of the object described by a template is determined by its relational constraints. Every field must be connected to every other field by a path of relational constraints. A template cannot describe an object that consists of two components that are not connected to each other. This enables the interpretation process to use the spatial structure of a template to find matching shapes efficiently. For example: suppose there are two fields with an adjacency constraint, and a shape has been matched to the first field. Then the system only has to consider shapes adjacent to the already matched shape as matches for the second field.

Another example:

Template FLOWER

Fields	**Type**
Stem	Shape
Left-leaf	Shape
Right-leaf	Shape
Flower-leafs	Shape

Constraints:
The Stem is adjacent to the Left-leaf, Right-leaf and the Flower-leafs.
The Stem is green.
The Left-leaf is green.
The Right-leaf is green.
The Left-leaf is left of the Stem.
The Right-leaf is right of the Stem.
The Flower-leafs is above the Stem.

When a set of shapes can be matched to a template, in such a way that all mandatory fields have a matching shapes, and all constraints are satisfied, the combination of that template and all the shape-to-field matches is called a template-instance. A template-instance describes a consistent object in an image.

The problem with this structure is that it can only describe simple objects, more complex object such as rabbits and pigs cannot be efficiently described. For example, rabbits usually have the same head (with little variations in eyes and mouths). If this structure was used, the head would have to be included in every rabbit-description. It is more efficient to build a separate description for a rabbit's head, and include it in the description of an entire rabbit.

This was done by adding a new template field type, a 'composite' field. This field can be matched by an instance of another template from a set of allowed templates. Using this field, a rabbit template can be composed from a head field (which can be matched by instances of the templates Rabbithead, Rabbithead-with-helmet, etc.) and a body field.

Template RABBIT

Fields	Type
Head	Composite (RABBITHEAD, RABBITHEAD-WITH-HELMET, etc.)
Body	Composite

Constraints:
Head is adjacent to Body.
Head is above body.

There is one more field type, 'shape-group'. This field can be matched to by a number of shapes, which all share a common characteristic (for example, the same color). This is used in places where multiple shapes that have something in common are used in an object, but the number of shapes is not fixed. An example of this is a necklace, where the number of beads may differ from image to image.

3.2.3 Described information

Several types of information are used for describing Nijntje images.

Shape form

Some shapes can have a known form that can be classified in the image analysis stage. This is very useful for interpretation, because it limits the list of objects to which that shape can belong.

Colors

A shape matched to a template field can be constrained to a specific color from the Nijntje color set. Furthermore, constraints can be set that shapes matching to two different fields must have the same color.

Adjacency/within

When an adjacency constraint is set on two shape-fields in a template the two shapes matching those fields must be physically adjacent. In this simple form, it is mostly used

to express constraints such as: a hand-shape and a wrist shape must be adjacent, and an eye-shape must be within a head-shape.

When any of the fields in the constraints are composite fields, the matter is complicated a bit. The naïve interpretation of what an adjacency constraint means in that field would be: at least one shape of the composite field must be adjacent to another shape in the other field.

This however, leads to 'unnatural' results in the Nijntje world. A particularly bad example shows itself in the Rabbit template, which contains a head field and a body field. Our naïve interpretation could result in a satisfied head-body adjacency constraint in an image in which the head is only adjacent to the left hand (and thus, a Nijntje that holds her head in her hand would be recognized as a rabbit…).

This has been solved by constraining which fields are considered for adjacency. This is done by defining candidate lists in the templates, in which all allowed shape fields are enumerated. For problematic adjacency relations, the candidate list must be defined in the templates that can be used. In the head-body case, all the head templates would have a candidate list, as would all the bodies and dresses. In the simple dress template this candidate list would contain the field for the dress itself. For templates of dresses with a collar the candidate list would contain the collar fields, but not the dress field.

Further spatial relations

The last type of information that is described is the spatial relation that must hold between the shapes matched to fields. The image analysis stage returns the left-top and right-bottom coordinates of the bounding box of every shape, as well as the center of the bounding box.

Examples of constraints that can be set for these spatial relations:

- Shape A is completely left of shape B
- Shape A is right of the middle of shape B
- Shape A is either completely under or completely above the middle of shape B.

3.3 Evaluation of existing systems

3.3.1 Description logics

The first approach to constructing the system for the interpretation stage was to look at existing systems that could execute the recognition task. Description logics (DL) seemed a pretty good option for this, considering the paper of Neumann and Möller [NEUM04]. In this paper a description is made of the scene that has to be interpreted, called the geometric scene description (GSD). The information that is contained in a GSD is roughly equivalent to the data the image analysis stage provides (shapes, color and spatial information), so using Description Logics seemed a promising approach.

Description Logics is a knowledge representation formalism, realized by a subset of First Order Predicate Calculus. Almost every description logic implements another subset, which is usually chosen in such a way that decidability of consistency checking and other

inference services is preserved. Examples of DL systems are RACER [HAAR01], FLEX [QUAN96], Back [HOPP93] and Kris [BAAD91].

The fundamental concepts in Description Logics are individuals, concepts and roles. A concept describes a set of individuals, and a role describes a relation between pairs of individuals. For example, with the persons 'John' and 'Tom' being brothers, the description logics description would have 'John' and 'Tom' as individuals, both of them would be member of the concept Person, and the role BROTHER(John, Tom) would hold.

The engine can be queried if an individual is member of a concept, or if two individuals are members of a relation. It tries to infer this from the information about the image and the rules it has (which are derived from the description of the objects in Nijntje).

The natural mapping to the information delivered by the image analysis would be to create an individual for each shape, and define concepts (like IS-GREEN) and relations (like IS-LEFT-OF) based on attributes and relations of the shapes. The description logics engine cannot automatically create new individuals: the set of individuals that is initially inputted remains static during processing. Because of this, interpretation results will have to be represented using concepts and relations.

3.3.2 Problems with Description logics

The need for that kind of representation presents a problem. Templates with multiple fields can have an adjacency path from one field to another that have intermediate shapes. The flower template can be used as an example.

Template FLOWER

Fields	Type
Stem	Shape
Left-leaf	Shape
Right-leaf	Shape
Flower-leafs	Shape

Constraints:
The Stem is adjacent to the Left-leaf, Right-leaf and the Flower-leafs.
The Stem is green.
The Left-leaf is green.
The Right-leaf is green.
The Left-leaf is left of the Stem.
The Right-leaf is right of the Stem.
The Flower-leafs is above the Stem.

If we want to know if a shape (an individual in the DL) is part of an instance of the flower template, we typically query the DL engine if the individual representing that shape is part of a concept that represents membership of an instance of the flower template.

This concept will have to be defined in terms of other concepts and relations. For the left leaf, this concept would have to specify that there is an adjacency-relation with another shape that is the stem, and the stem would at least have to have an adjacency-relation to another shape that is the right leaf. In a DL, this kind of relation can only be specified using role composition. This, however, is a rare feature among description logics. More than half of the examined systems did not support it.

The systems that did support it (Flex, Back and Kris) presented another, insurmountable problem: they did not support naming an individual in an expression (like ∃x IS-GREEN(x) introduces the name x, which then can be used multiple times in the rest of the expression). The systems did support an expression like: 'there must be a green shape adjacent to the stem, and there must be a round shape adjacent to the stem'. But, due to the lack of the naming feature, the following could not be incorporated: '... and the green shape and the rounds shape may not be the same shape'.

Thus, the examined description logics were too limited in their implementation of first order logic to implement the interpretation system.

3.3.3 Other systems

Another system related to description logics, but somewhat more powerful was LOOM. The only implementation that was readily available (and directly testable) was that of its successor, PowerLOOM [USC06]. However, this system repeatedly crashed during some early usability tests. Because of that, its capabilities have not been analyzed further.

At the time, I became convinced that the interpretation problem was a hard one, with large time and memory requirements. Logic systems usually do not offer a way to dynamically steer evaluation of possibilities into the 'right' direction, and also didn't offer much possibilities for controlled relaxation of constraints; which is necessary for recognition of occluded objects.

3.4 Building a new system

Because no examined logic system directly suited the needs of the interpretation system, it was decided to build a new system that was custom geared to the problem. This also had the added advantage that heuristics could be added to speed up the process.

The new interpretation system uses the information generated in the image analysis stage, and will then try to construct a coherent interpretation from this information. This interpretation is done using the description of the objects in the Nijntje world that have been built earlier.

3.4.1 Goals

The function of the interpretation part of the system is to determine which objects from the Nijntje world are visible in an image, and which shapes those objects are composed of. This has to be done using the structured information that the image analysis part of the system has generated. Because the interpretation process is often ambiguous in nature (multiple interpretations can be possible), it is not expected that the system will be able to do a perfect job.

Because of this, the first goal of the system was outputting objects that could be present inside an image (and of which shapes they are composed of), together with an estimate of the chance that the interpretation is correct. Not all possible interpretations have to be outputted, only those with the highest rating. The final stage then tries to select a set of objects that together form a coherent interpretation of the entire image.

The system needed to be quite flexible, because during the design process the templates describing the Nijntje world would change very often (to reflect better insights into the recognition task, and to correct programming errors). Also, adding templates for new objects found in other Nijntje books would have to be supported, as well as adapting the system for another comic.

3.4.2 Design approach

Designing a system that immediately supported the entire description system, and could handle occlusion was infeasible for me, lacking the experience in solving these kinds of problems.

I therefore opted for an iterative approach; first selecting a small set of features and implement those, and then adding more features in later iterations.

First iteration

In the first iteration the following features from the description system were selected to be implemented:

- Shape fields and composite fields
- All color and position-based positioning constraints

Not implemented were:

- Shape groups (fields matching multiple shapes).
- Holes (holes in shapes).
- Combined shapes (shapes grouped together).
- Custom attributes
- Occlusion.

The last item of this list is the most notable absentee. It was chosen not to include it in the first iteration because of the very high difficulty of recognizing a partially visible object.

3.5 Initial design

3.5.1 Steps in recognition

As mentioned earlier, there are five steps in the recognition process. The image analysis stage executes the feature detection. In the reasoning stage of this system, the perceptual organization, the indexing, matching and the verification are done. A quick overview of the kind of processing that is usually done in each of these steps follows.

Perceptual organization

Perceptual grouping means the grouping of multiple features into a group that likely contains a single object. In the first iteration, no attempt was made to implement this.

Likely, this step needs some integration with image analysis, to be able to classify the outline of a group of shapes.

Indexing

Given the data generated by the image analysis stage, it might be possible to narrow the set of templates a shape may belong to. This creates an advantage over trying all templates.

Some of the objects in the available books have parts that have recognizable forms. These shapes can be used to select relevant templates. For a lot of other shapes, no effective hint can be derived from the information generated by the image analysis stage about which types of object they might be part of. The only way to interpret those shapes is to try all models.

Matching

There are two approaches to matching [GRIM90]: the global features approach which uses global properties of an object, like surface area or volume, and the local approach, which maps local features to corresponding features in the model.

The global approach isn't suited for this domain, because the objects that have to be recognized are very flexible. Determining which (numerical) features to extract from an image and mapping those features to objects is a nearly impossible task because of that flexibility.

That leaves the local approach. There are two sub-approaches here: searching the correspondence space and searching the pose space (and combinations of those two methods). Searching the pose space means finding the transformation of a model which best fits the image data, and uses tools like the Hough transform [HOUG59]. Because of the flexible nature of the objects in the Nijntje images, this approach isn't very appropriate; it is more suited for rigid objects. This leaves us with searching the correspondence space.

When searching in correspondence space, a mapping is made from image elements to model elements. The correspondence space characterizes all the possible matches of image features and model features. The search in this space boils down to finding the points that identify correct interpretations, without having to explore all the possibilities. The size of the correspondence space is exponentially related to the number of image elements, which can add up to an impossibly large number if the image is complex enough.

The search through this space can be done by building an interpretation tree. In this tree, every node is a list of mappings of image elements to model elements. Every child-node contains all mappings of a parent node, plus an extra mapping. This tree can be pruned by the use of constraints (unary constraints operating on a single image-model element mapping, and binary constraints operating on a pair of image-model element mappings). Using this pruning approach, a large portion of the correspondence space does not have to be searched.

Verification

The verification step takes the list of raw matches the matching step has resulted in, and decides whether the matches are correct interpretations. There is not much research in this area [POPE94], and most of it focuses on systems which use edges as image primitives (for example, require a certain percentage of a model being matched). However, none of this is relevant to this system. A custom method will have to be developed.

3.5.2 Available information

The information available for the interpretation system is the information gathered by the image analysis system, and the templates describing all the objects in the Nijntje world.

The output of the image analysis system consists of:

- The list of shapes in the input image (described by the bounding box and color).
- The names of the forms of shapes that have been recognized as having a known form.
- A list of shapes that are adjacent to one another.
- A list of shapes that are nested within each other.

The description of all the objects that can be found in Nijntje images consists of a list of templates.

3.5.3 Design

The system will use a hypothesis generation and testing approach. It will start with hypotheses matching a single shape to a template field, and will gradually extend those hypotheses until they contain a description of a complete object. This roughly corresponds with the interpretation tree search approach [GRIM90]. The template constraints are used to prune the interpretation tree when possible, in an effort to reduce search complexity.

A hypothesis maps in a very natural way to the template-based description that has been made of the objects in the Nijntje world. It matches a list of shapes from the image to the shape fields in a template instance.

As an example, the following hypothesis describes a part of a flower. It uses the flower template, and matches shape A to the 'flower-leafs' field of that template.

Hypothesis A: uses template Flower

Field	Matched shape
Stem	N/A
Left-leaf	N/A
Right-leaf	N/A
Flower-leafs	Shape A

A hypothesis is tested by evaluating all the constraints in the template that can be checked (obviously, constraints involving fields that have no matching shapes can't be checked).

In some cases, the information needed to check a constraint is missing (for example, a shape that the image analysis system can't always recognize wasn't recognized). In that case, the constraint is ignored.

New hypotheses are generated by taking the current hypotheses, and matching an additional shape to a template field doesn't have a matching shape yet. After this, the hypothesis is tested. This process is repeated until all consistent hypotheses have been processed. A ranking system is employed to determine in which order hypotheses must be processed.

Hypotheses in which all the mandatory fields of their templates have a matching shape, and in which all constraints are satisfied are considered 'complete': a complete object has been recognized. No further verification is done on these hypotheses, because all constraints have already been checked.

As final step, a list of hypotheses is selected from the list of 'complete' hypotheses, which together describe the contents of the image in a consistent way: every shape belongs to only one hypothesis.

This, of course, is only a quick overview of the system. In the next sections, the design will be discussed more in-depth.

3.6 Initial implementation

3.6.1 Description system

The description system developed to describe the objects in the available image was very usable to fulfill its main task: enabling humans to easily describe objects with very few constraints. For this purpose, a lot of different types of constraints were included in the system.

For the interpretation system however, its structure was a little problematic to process directly:

- Maintaining all the (complex) rules would have made the probability of programming errors very high.
- Genericity: The description system used some domain-specific concepts (color, coordinates). If the system is applied to another domain, new rules would have to be integrated into the system, which would pose a complex task.

Because of these problems, another description system was created. In structure, it was almost the same as the original system, the only change being that information like color and coordinates was represented with generic attributes. These attributes could contain either integer numbers or text.

The relational constraints have been carried over unchanged, but all attribute constraints were converted to constraints that compare an attribute to another attribute, or to a fixed value. This reduced the number of different constraint types greatly, but the new system was still able to express most of the concepts in the original system.

3.6.2 Complexity control

An important aspect of the interpretation system is complexity control. The naïve method of generating all possible hypotheses and testing them can easily result in tens of millions of hypotheses. The hardware that can handle that easily is simply too expensive for this to be a viable option. Complexity control is therefore present in a lot of elements of the system, ranging from hypothesis inconsistency propagation to the choice of the next template field to match.

3.6.3 Initialization

In the initialization phase of the system, the templates that describe the objects in the Nijntje world are read in, as well as the data generated by the image analysis system. In this phase the root hypotheses will be generated. These hypotheses form the roots of the interpretation tree that will be explored during the processing stage, and match only a single shape.

The following steps are executed in this stage (the numbers between the parentheses refer to the steps in the diagram on the next page):

Building of the initial hypotheses (step 1)

In this first step, all possible hypotheses that contain only one shape are constructed. This is done by enumerating all templates that have at least one shape-field. Subsequently, for every one of those shape-fields found in those templates, a hypothesis is generated by matching every shape in the image to that field.

For example, if there is a template TemplateA with the two fields Field1 and Field2, and two shapes in the image, ShapeA and ShapeB, hypotheses will be generated for the following combinations: ShapeA – Field1, ShapeA – Field2, ShapeB – Field1 and ShapeB – Field2.

Checking of the initial hypotheses (step 2)

Some constraints can be checked immediately; namely the unary constraints (the ones that operate on only one shape). In the second step these constraints are checked, and hypotheses that do not satisfy them are discarded.

Rating of the initial hypotheses (step 3)

During the last step, the rating of all new hypotheses is calculated, and the hypotheses are inserted into the list of unprocessed hypotheses. After this step, the root hypotheses are ready, and the processing phase can start.

Reasoning stage initialization

Template RABBIT-HEAD

Fields	Type
Head	Shape
Left-eye	Instance of EYE
Right-eye	Instance of EYE
Mouth	Instance of MOUTH

IS-WITHIN(Left-eye, Head)
…
HAS-X-POSITION(Left-eye, Right-eye, LEFT)
…

Template CHILD-MOUTH

Fields	Type
Mouth	Shape

HAS-SHAPE-TYPE(Mouth, CHILD-MOUTH)

Template ADULT-MOUTH

Fields	Type
Mouth	Shape

HAS-SHAPE-TYPE(Mouth, ADULT-MOUTH)

Example descriptions

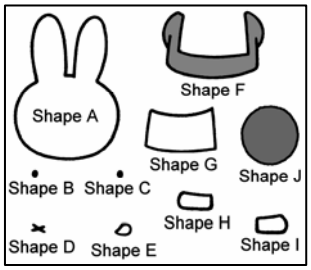


Image analysis shape-data

Descriptions of all elements in the Nijntje world

1. Build initial hypotheses (describing only one shape)

Instance of RABBIT-HEAD	
Head	Shape A

Instance of LEFT-FOOT	
Foot	Shape F

Instance of CHILD-MOUTH	
Mouth	Shape B

Instance of LEFT-FOOT	
Foot	Shape G

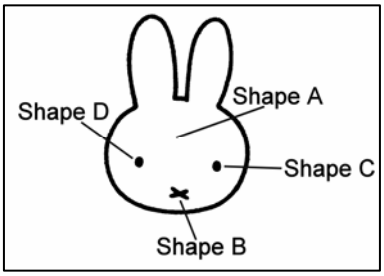
Example initial hypotheses

2. Check constraints of the new hypotheses

3. Rate new hypotheses

Hypotheses database

Processing Hypotheses, part 1

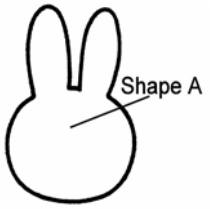


Shape A: shape is "*Rabbit-head*"
Shape B: shape is "*Child-mouth*"
Shape B is within Shape A
Shape C is within Shape A
Shape D is within Shape A

Example image and data

Start → 1. Find highest rated unprocessed hypothesis

Instance of RABBIT-HEAD	
Head	Shape A
Left-eye	Unfilled
Right-eye	Unfilled
Mouth	Unfilled



Example hypothesis

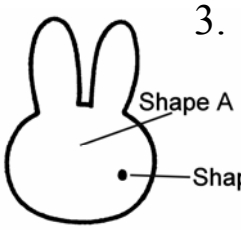
Unfilled	Candidate template fields	Candidate shapes
Mouth	CHILD-MOUTH:Mouth	Shape B, Shape C, Shape D
	ADULT-MOUTH:Mouth	Shape B, Shape C, Shape D
Left-eye	OPEN-EYE:Eye	Shape B, Shape C, Shape D
	CLOSED-EYE: Eye	Shape B, Shape C, Shape D
Right-eye	" " "	" " "

Unfilled fields, compatible template fields and candidate shapes

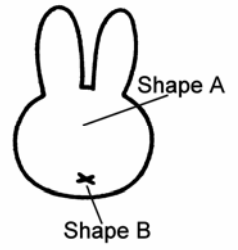
2. Find unfilled fields in the hypothesis, which templates and template-fields can be used to fill them, and which shapes can match those.

Continued on the next page

Processing Hypotheses, part 2



3. Build new hypotheses for every combination of unfilled template field and shape by adding that combination to the original hypothesis



Instance of RABBIT-HEAD	
Head	Shape A
Left-eye	**Instance of OPEN-EYE**
	Eye: Shape C

Hypothesis generated by adding combination CHILD-MOUTH:Mouth and Shape B

Instance of RABBIT-HEAD	
Head	Shape A
Mouth	**Instance of ADULT-MOUTH**
	Mouth: Shape B

Hypothesis generated by adding combination ADULT-MOUTH:Mouth and Shape B

No constraint failures ✓

4. Check all constraints of the new hypotheses

✗ Fails because shape B is of type "*Child-mouth*", instead of the required type "*Adult-mouth*"

5. Output complete hypotheses (those satisfying all constraints)

6. Promote complete hypotheses

7. Rate the new hypotheses

Hypotheses database

Instance of RABBIT-HEAD	
Head	Shape A
Mouth	**Instance of CHILD-MOUTH**
	Mouth: Shape B
Left-eye	**Instance of OPEN-EYE**
	Eye: Shape D
Right-eye	**Instance of OPEN-EYE**
	Eye: Shape C

Example complete hypothesis

3.6.4 Processing hypotheses

The hypothesis extension process is done in the following way: given a hypothesis, the next field to fill must have a direct spatial relation constraint (either adjacency or is-within) to at least one field that has already been filled in the original hypothesis. The list of candidate shapes is then limited to the shapes that have that specific relation to the shape matched to the already filled field.

This way of extending hypotheses is more efficient that to extend all fields, because in this way those spatial constraints are automatically satisfied, and no hypotheses can generated in which those constraints are violated, effectively pruning the search tree.

The following steps are executed in this stage (the numbers between the parentheses refer to the steps in the processing diagram):

Picking the next hypothesis (step 1)

The next hypothesis to process is chosen in a very straightforward manner; the hypothesis with the highest rank that has not been processed yet is chosen as the next one.

Finding candidates for unfilled fields (step 2)

The first step in extending a hypothesis is finding out which fields have not been filled yet, and can now be filled. This is done by examining all the spatial constraints, and to identify which of them reference one filled field and one unfilled field. The unfilled fields of these constraints are the fields that will be filled.

Finding candidate shapes for these fields is now very easy: they are all the shapes that satisfy the spatial constraint through which the unfilled field was found. If no candidate shape was found, and the unfilled field was not optional, the hypothesis is inconsistent, and is marked as such.

A special optimization has been implemented in this step. If any of the unfilled fields is a mandatory field, then all optional fields are ignored. Mandatory fields usually have more constraints then optional fields, so it is advantageous to check them first before trying the optional fields.

Extending the hypothesis (step 3)

When filling a single shape field, extending a hypothesis is quick: for every candidate shape a copy of the original hypothesis is constructed, and the candidate shape is matched to the shape field. If there are N candidate shapes, this results in N new hypotheses, one for every candidate.

For composite fields, it is a little more complicated, because a shape cannot be matched directly to that field, only template instances can. In this case, for all the templates allowed to match to the complex field, a copy is made of the hypothesis, with a new, empty instance of that template matched to the complex field. Then, for every shape field in the new template instance new hypotheses are built matching the candidate shapes. For composite fields, the process is repeated until only shape fields are left.

If a candidate field list is present, it is used to filter the shape fields that have been found. If a shape field does not appear in the list, no hypotheses are created using that field.

An example:

Instance of RABBIT-HEAD	
Head	ShapeA
Left-eye	N/A
Right-eye	N/A
Mouth	N/A

If the spatial relation 'Mouth is within Head' is processed, we will look at the composite field Mouth, that has not been filled yet. This field can be matched by instances of template CHILD-MOUTH and template ADULT-MOUTH. Assume that shapeB is the only candidate shape.

In turns, empty instances of template CHILD-MOUTH and template ADULT-MOUTH are matched to the Mouth field. Then, in both templates the Mouth field of the freshly added templates is matched by ShapeB.

This results in the following two new hypotheses being produced (unmatched fields omitted):

Instance of RABBIT-HEAD	
Head	ShapeA
Mouth	**Instance of CHILD-MOUTH** Mouth: ShapeB

Instance of RABBIT-HEAD	
Head	Shape A
Mouth	**Instance of ADULT-MOUTH** Mouth: ShapeB

After producing the new hypotheses, a check is done whether the new hypotheses have been generated earlier (via another path). For example, the hypothesis containing a head with a left eye matched and the hypothesis containing a head with a right eye matched will (if both hypotheses are correct of course) both generate the same hypothesis with the left and the right eye matched. Duplicates of earlier generated hypotheses are discarded.

This requires that all processed hypotheses are kept in memory until the end of the interpretation process; however, it prevents repeated expansion of hypotheses.

Checking constraints (step 4)

The first constraint check on a hypothesis is done using an open-world assumption. This is because some attribute values of a hypothesis can change as more fields are filled in. Every attribute is represented with a range of valid values it can assume, considering the extensions that can be done on the hypothesis. The constraints on the template are used to limit those ranges. If an attribute has no valid values left, the hypothesis is inconsistent.

Example: Assume we have attribute *left-a*, with valid range [5..10] and *right-b*, with valid range [2..7]. If the constraint *left-a* ≤ *right-b* holds, the maximum of *left-a* must be

smaller than or equal to the maximum of *right-b*. This limits the range of *left-a* to [3..4]. In the same way, the valid range of *right-b* is limited to [5..7] by this constraint.

Inconsistency propagation (also step 4)

If a hypothesis is inconsistent, the parents of the hypothesis are examined. If the inconsistent hypothesis previously was the last consistent one to fill a specific field of a parent hypothesis, then every attempt to fill that field has failed. If that field was mandatory, the consequence is that the parent hypothesis is also inconsistent.

If that inconsistent parent hypothesis has any consistent children left, they are also marked as inconsistent. This because child hypotheses are extensions of their parents, if a field cannot be filled in a parent it also cannot be filled in any child (but only because of the open-world assumption used in checking the constraints).

The other parents of child hypotheses that are marked inconsistent in this way are also examined, and so on. This process continues until all newly inconsistent hypotheses have been found.

As a further optimization, when a hypothesis that has just one single field filled in is found to be inconsistent, all hypotheses are examined. Any hypothesis that contains the same mapping is also marked inconsistent.

Completeness check (step 5)

The completeness check on a hypothesis is the first step of the verification. It is done in two steps. First, all fields that are not filled in the hypothesis are checked. If any of those fields is mandatory, the hypothesis is not complete.

The second step consists of re-evaluating all the template constraints, but this time with the closed-world assumption. This is because the hypothesis must form a consistent template instance by itself. The check is done by calculating the exact value for every attribute, and checking the constraints with those values.

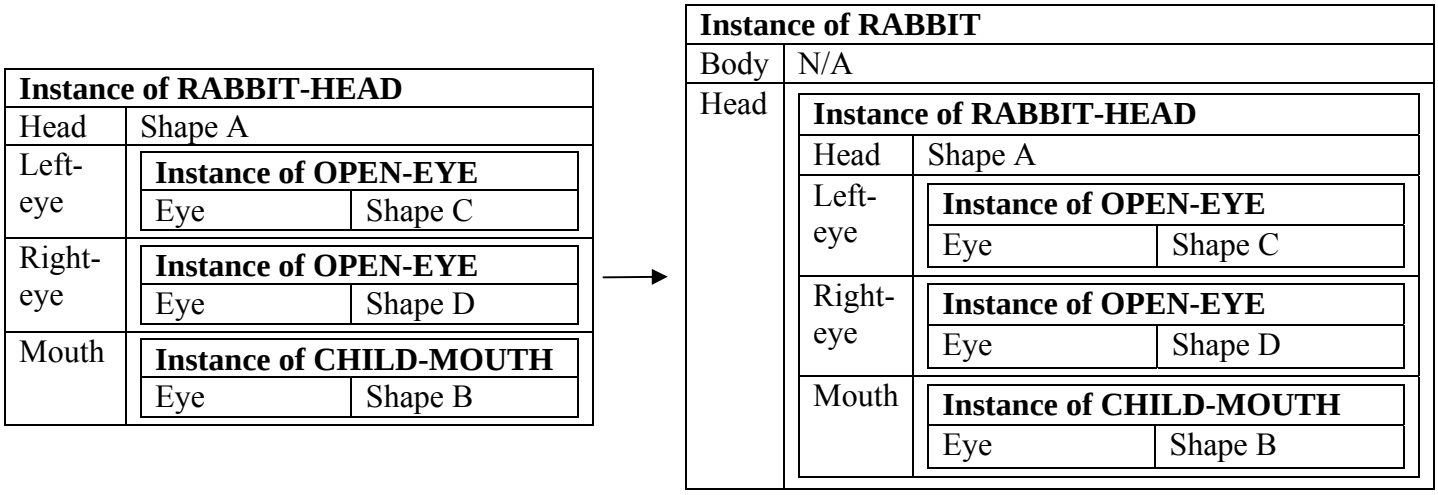
If all constraints are satisfied, the hypothesis is complete and describes a complete object.

Hypothesis promotion (step 6)

At the initialization phase, hypotheses are only generated for templates which have at least one shape field. This leaves out the templates which have only complex fields, like the Rabbit template.

If a hypothesis is complete, every template in the Nijntje universe is searched for complex fields to which that hypothesis can be matched. If such a template is found, a new hypothesis is generated which uses that template, and the original hypothesis is matched to the complex field.

For example, if a hypothesis describing a rabbit's head is complete, a new hypothesis describing a rabbit is generated, with the head-hypothesis matched to the head-field of the rabbit-template. Example:



This way of promoting hypotheses to 'higher-level' templates makes sure that those templates will also be tried eventually; but only if some evidence is present that the object they describe is present in the image.

Rating of the new hypotheses (step 7)

In the last step of processing a hypothesis, the rating of the newly created hypotheses is calculated, after which they are inserted into the list of unprocessed hypotheses.

3.6.5 Processing order

The order in which hypotheses are processed proved to be important because of the complexity control strategies that could be employed.

At first, a depth-first exploration was implemented. After breaking it off after half an hour it was revealed that the hypotheses the system was examining were all derived from the very first hypothesis the system started with, which was one which could not be expanded into a complete hypothesis.

The breath-first exploration that was implemented next had the advantage that all the different paths of the search tree are explored at the same time. This enabled the inconsistency propagation optimizations at a very early stage, limiting the number of hypotheses that needed to be processed by a great deal, in comparison to depth-first exploration.

If a hypothesis has a few short exploration paths that together prove the hypothesis to be inconsistent, breath-first exploration will find these relatively quickly. Depth-first exploration can get stuck in other paths that can take very long to explore fully.

In addition to depth-first and breadth-first exploration, a ranking system was implemented to determine which hypotheses were more interesting to process (because they had more chance to lead to a complete hypothesis, instead of being a dead end).

However, it proved very difficult to find heuristics that were a good predictors whether a hypothesis would eventually be invalidated or not. The correlation between most of the defined measures (number of hypotheses generated from the hypothesis, number of

constraints satisfied) and the success of a hypothesis were as low as to the extent of making them impractical for use.

The only measure that was found to be relevant was the number of different initial hypotheses a single shape could be matched to (as mentioned before, the initial hypotheses match only a single shape). If a shape was used in only one initial hypothesis (for example, because it had a very specific form) that hypothesis usually wasn't invalidated, and led to the generation of a complete hypothesis, at least when the object wasn't too occluded.

When a specific shape can only be matched to one single type of object, it has another advantage: it can safely be assumed that one of the complete hypotheses containing that shape is completely correct. Therefore, the set of shapes that all of those hypotheses have in common can be erased from the list of hypotheses that still have to be processed. This increases the importance of processing the hypotheses containing that specific shape before all other hypotheses.

3.6.6 Difficulties

The problems encountered during implementation all had the same symptom: a huge number of generated hypotheses. To avoid processing hypotheses twice, and to correctly track the parents and children of hypotheses, processed hypotheses could not be discarded, but had to be retained. If the number of generated hypotheses became too large, all available system memory would be used up storing all those hypotheses. Furthermore, the execution time would be too long to be practical.

The first problem was that several templates were indistinguishable from each other. An example is the mouth templates of rabbits. The image analysis system could in practice not recognize the form of a mouth because the shape was too small, so the interpretation system could not distinguish between the mouths of adult or child rabbits.

Because of this, every rabbit hypothesis had two versions: one with the mouth-shape recognized as the mouth of an adult, and one with the mouth-shape recognized as the mouth of a child. Every time when a shape or template could not be distinguished from another one the number of generated hypotheses at least doubled.

The resolution of this problem was to combine the templates that could not be distinguished into one single template.

A second problem was that small templates (four fields or less) didn't have very many constraints, and the constraints they did have were not selective enough. This means that a lot of (invalid) hypotheses were generated using those templates, and a lot of time could go by until they were finally invalidated. No effective way was found to deal with this problem within the restraints of the current system.

3.6.7 Final interpretation system output

The result of the hypothesis processing is the list of all complete hypotheses: hypotheses for which all mandatory fields have been filled, and all constraints have been satisfied.

This has its advantages, but also its disadvantages. If an object is occluded it can be expected that certain fields cannot be filled, because the shapes that should have been matched to them are simply not visible. Furthermore, constraints that are too strict can hamper recognition (as an example, the constraint that the mouth in a head is below the eyes will prevent the recognition of the head if it is projected upside-down, or even tilted a little).

The advantage of this approach however, is that there is a target to aim at. In the current system, there are no reliable heuristics which can tell us if a hypothesis contains a correct interpretation. When a hypothesis satisfies all its constraints the chance of it being a correct interpretation is much greater then when multiple constraints are violated.

Selection of final hypotheses

Outputting a long list of hypotheses, without giving any hint which one is the most likely is not a viable option. For starters, a shape can be used in multiple hypotheses. Especially templates with a low number of fields and a low number of constraints match a lot of the shapes present in an image, without any indication of correctness.

To see if there were any measures which could predict the correctness of the complete hypotheses the output of the reasoning system on the test set of 12 images discussed earlier was reviewed. Several measures were calculated for each of those hypotheses (number of mappings, number of constraints, etc.), and correlated to the number of correct interpreted mappings in the hypothesis.

The chance that an individual mapping of a shape to a template field is correctly interpreted rises with the size (total number of mappings) of the hypothesis. This is probably caused by the increased number of constraints, on average the total number of constraints on a hypothesis increases with 5 per added mapping. When a hypothesis contains 6 or more mappings, usually more than 65% of the mappings are correct, rising to 90% or more with hypotheses with 10 or more mappings.

In general, hypotheses with larger number of mappings have larger chances of being correct (at least when the number of mappings exceeds 5). Therefore the largest hypothesis is picked from the list of completed hypotheses, and offer that one as final hypothesis (if multiple hypotheses have the same size, the one with the most constraints is chosen). Other complete hypotheses containing a shape from that selected hypothesis are then scrapped, and then the next largest hypothesis is selected. This process is continued until no hypotheses are left.

Feedback

The selected hypotheses are written to a text file, which can be read by the image data gatherer in the ParleVision system. The reasoning stage itself implemented as a command-line process, while the vision analysis stage is implemented in ParleVision, which operates in a graphical environment. This makes the latter much more suited to present the results of the reasoning stage in a graphical way (moreover, the reasoning stage doesn’t get enough information to draw the shapes at all).

3.6.8 Results

Setup

To test the reasoning system, the system has been run with a series of images for which a reference interpretation was available. To test only the reasoning part of the system the test images were cleaned, so no scanning artifacts would be present in the output of the image analysis stage.

Two different sets of images were used:

- Development test set (12 images).
- All (unique) images from 3 books (38 images).

No tests have run with images that had not described yet. It might be worthwhile to run them with some other books, to see how the system reacts on objects it hasn't seen before.

These tests were all run on an AMD Athlon 2600 XP machine, with 1 GB of memory.

Performance on the test set

To quickly evaluate the performance of the reasoning system during development, a set of 12 images was compiled, each depicting a single object (see appendix C). This set was used to test various parts of the system. The images were cleaned up a little to remove the scanning artifacts, so the vision analysis stage wouldn't have difficulties with them. These images were also used for development, so these tests to not completely represent the real-world performance of the system. Also, no occlusion is present in the images.

The shapes that the image analysis stage found in the test images were hand-matched to the template, to provide a reference-interpretation. For 2 images (number 5 and 12) no reference could be made, because the objects were not described (due to unimplemented features). They were left in the set to see how the system reacts on objects it does not have descriptions of.

The following statistics were generated (the numbers are an average per image):

Running time:	26 seconds
Generated hypotheses:	30.010
Processed hypotheses:	3.488
Outputted hypotheses (correct)	289 (12)
Finally selected hypotheses (correct)	3.4 (0.6)

The number of correct outputted and selected hypotheses has been calculated for only those images for which a description was available for the present object.

Of the 10 images where the system should be able to recognize the object in the image successfully, 6 were recognized completely. The other 4 images had several problems: a tilted head clashed with template constraints, the image analysis stage did not classify the coach-work of the car correctly, with the kangaroo the wrong shapes were matched, and with the mug the correct hypothesis wasn't selected in the final selection step.

Further tests

To further evaluate the working of the system, it was run on the images of the three books that had been annotated. The images were first cleaned of all scanning artifacts; so the image analysis would not yield spurious shapes that could mess with the reasoning process.

In total, 38 images were run through the interpretation system. When the system took more than 20 minutes to complete the reasoning process, it was aborted (primarily to limit the time needed to run the tests, but also to terminate tests that had used all available system memory). In total, 8 runs were aborted this way. Each of them had generated more than 800.000 hypotheses, and they had used from 400 to 1080 megabytes of memory.

The following statistics were generated (the numbers are an average per image). The tests that have been broken off are not included in this overview.

Runtime:	148 seconds
Generated hypotheses:	123.006
Processed hypotheses:	7.868
Outputted hypotheses (correct)	360 (11)
Finally selected hypotheses (correct)	4.2 (0.52)

The application completed the interpretation process in 30 of the 38 images. Of those 30 images, for 22 of the images to top selected hypothesis was the right object (that object was really present in the image).

In the 30 images, a total of 69 objects were described. The reasoning application managed to find 12 of them completely, and parts of another 15. The number of hypotheses nominated as final results was 123, so 22% of those were actually partially correct. The final results are ordered on rating; the hypothesis with the highest rating was (partially) correct in 80% of the images.

If we consider the limitations of the system, then in 16 of the 38 images the largest object should be found by the system (the object has a template description and is not occluded). In two of the images, the system had to be aborted. In 13 images the top selected object was in fact present in the image, but only in 7 of them the object was found in its entirety.

Evaluation

The system can detect fairly well which objects are present within an image; at least when the objects are large and they are not occluded. But for general application of the system, the results of the tests that have been run are not a reason for much hope. This has several reasons:

- The current implementation is very CPU-power hungry. 148 seconds is a long time to wait for an interpretation, even when ignoring the tests that were broken off. Also, the memory requirements are very high.
- The selection mechanism is only useful for selecting the first hypothesis; usually the largest. For smaller hypotheses, there are a lot of candidates. The system

cannot effectively differentiate between them, and usually selects the wrong hypothesis.

- The system cannot handle tilted objects, because of the strict positional constraints used in the description system. When interpreting images from books, this is not a problem very often, but for other applications, it probably will be.

Partly occluded objects aren't recognized in general. Exceptions to this rule are the more complex objects whose descriptions are broken into more sub-objects. These sub-objects can be recognized when they are completely visible, even when the object itself is partly occluded.

4 Conclusions

The goal of this assignment was to design and implement a system that could recognize the objects depicted in a Nijntje scene, and to identify relations between those objects. It was to be constructed in such a way that occlusion and composed objects could be handled in a satisfying way. Also, an ontology had to be constructed that described the objects from the Nijntje world in such a way that it could still be used when only parts of an object were visible.

I have approached this assignment by dividing it into two separate parts: an image analysis part and a reasoning part. These are discussed in the following sections.

4.1 Image analysis

The image analysis part has been implemented in the ParleVision framework. The analysis has been broken apart in several sub-processes that have been implemented by processors that can be run in the framework.

With the current set of images, the implementation proved to be satisfactory. All the information that was needed to run the interpretation process could be extracted from the images. There are, still some issues left:

- Most HSV color space boundaries that are used to differentiate different colors have been hard-coded in the color quantization process (only some boundaries are adjustable by hand). This means that in different lighting conditions, the quantization might be entirely off.
- The shape finding process has been implemented in a very expensive way by finding all shapes per color. This doesn't scale when using for other kinds of comics, where more than 8 colors can be used.

The image analysis isn't a very fast process. Due to the fact that all processing has been offline this has not posed a large problem. And because the scanning of books is a relatively controlled process in terms of lighting (in contrast to using a camera), the color quantizer could perform adequately for the purposes of this assignment.

4.1.1 Suggestions for improvements

The image analysis wasn't the main scope of the assignment, the reasoning parts were far more important. Some shortcuts were taken to quickly implement the image analysis, and therefore there are some points for improvements.

Image color quantizer

As mentioned earlier, the image color quantizer uses fixed boundaries to determine for each pixel what is color from the know set it corresponds to. Because of image noise some pixels in a shape can cross that boundary, leading to stray pixels that of the wrong color.

An improvement for the image color quantizer would be to take the context of a pixel into account. A single white pixel in a sea of blue pixels could then be determined to be

noise. Or, when the quantizer notices that an image is particularly dark, it could dynamically lower the threshold for black, so that dark colors (like dark green in the Nijntje images) would be quantized correctly.

Currently, the quantizer has a fixed color list (the known Nijntje colors). By analyzing the pixels are present in an image; the quantizer could instead make a custom list of colors (by first detecting patches of roughly the same color and then trying to merge those patches when their colors are roughly similar). This process might also be able to separate shapes of a different shade of the same color. This would be useful when using the quantizer to analyze other comics; although it will probably pose a problem to use the dynamically generated color list in object descriptions.

Shape finder

The shape finding processor currently uses the OpenCV functions to identify all the shapes in an image. It does this by repeatedly cycling to the colors of the Nijntje universe, and finding all shapes of that particular color. When small shapes are found (so small that they are probably quantizing artifacts) these are recolored in the next color in the cycle. Multiple cycles can be necessary to get rid of all artifacts, which is computationally intensive.

If the image quantizer is updated to get rid of the artifacts, these multiple cycles can be removed. Also, a custom process can be built that can find all shapes of all different colors in one invocation; the current process doesn't scale when the number of colors is large.

Shape classification

The current shape classification process takes 16 sample points from the outline of a shape, uses a representation function to transform the points into a vector, which is then transformed with the Fourier transformation.

A weakness in this approach is the sampling of the shape. If a shape is large and complex enough, 16 samples may not be enough to get a complete view of the outline. The result can be that the resulting descriptor differs depending on the placement of the samples on the outline. In such complex shapes, it might be necessary to increase the number of samples taken, although the resulting descriptor must not have too many numbers because that decreases the resemblance between descriptors of different shapes of the same type. An after-processing step should be done to keep only the most significant part of the descriptor.

Also, with relatively small images some small, pixel-size deformations could have a relatively great impact on the final descriptor. If this is the case, the threshold for matches should be increased somewhat, to compensate for the greater variation in the descriptor.

4.2 Reasoning

In the course of this assignment, a system has been built to interpret the contents of images, using the information provided by the image analysis system. Currently, this system can recognize the contents of a Nijntje image, provided that it does not contain many shapes, and the objects within the image are known, each contain a relatively high

number of shapes and not occluded. This means that one major goal, the handling of occluded objects, has not been achieved.

The reason that this goal has not been reached is that, in hindsight, the wrong approach to object recognition has been chosen. This approach was to use a purely symbolic system to represent the shapes. As it turned out, the expressive power of the description system was not good enough to effectively judge whether hypotheses containing a small number of shapes were correct, or to select the most promising hypotheses from a set of hypotheses.

This led to the following problems:

- The complexity of the generate-and-test algorithm gets out of hand when the number of shapes is sufficiently great. This algorithm has an inherent exponential complexity, but when an effective test can be done on hypotheses (one that can catch most of the incorrect interpretations) this complexity can be controlled. The current description system is not good enough to judge hypotheses effectively.
- The system cannot determine what the correct interpretation is when a list of alternative hypotheses is available, it cannot discriminate between them.
- The system must rely on positional constraints in an absolute coordinate system to describe objects, and these hamper recognition when objects are tilted.

The net end result is that relatively complex objects can be recognized quite well, under the conditions that they are known, not many other shapes are present, they are not occluded, and they are not tilted.

No effort has been put in making occlusion work. The approach that was taken to build the reasoning system was the iterative approach; gradually extending the system with features that were missing. The problems with the extreme computational complexity surfaced before occlusion could be implemented. A lot of energy has been put in ways to keep the complexity under control (most of them work by invalidating hypotheses as soon as possible). However, this didn't address the root cause of the problem.

To be able to judge small hypotheses, and to correctly handle occlusion, a new, more elaborate, description system is necessary. Features that are needed will include pose-estimation (explicitly knowing the place and, more importantly, the direction of a form, and being able to use that information), occlusion-detection (there are certain hints that a shape is partly hidden behind another one).

My estimation is that these features are not compatible with the currently implemented system. Much of the measures taken to control the complexity rely on the assumption that a missing mandatory shape will invalidate an object; that assumption is not valid when occlusion is taken into account. Also, the entire description system will need to be replaced; the work it will take modifying all those parts of the existing system will probably exceed the work it will takes to begin from scratch.

4.2.1 Recommendations for the current system

The current implemented system is only suited for recognizing unoccluded, relatively complex objects (5 or more shapes), in images with few other shapes. Modifying it to handle occlusion would probably be impractical.

However, there are a few small enhancements that could be made to the current system that would make it more effective, and recognize more types of objects. These can somewhat mitigate the complexity problems the current system is facing. However, these there is no guarantee that implementing recommendations will make the system generally usable (measured in time taken to complete an interpretation).

Adding more complex types of constraints

Currently, the system can only handle constraints that directly compare attributes or an attribute and a fixed value with each other. This prevents adding of constraints comparing the size of individual shapes with each other. Extending the attribute system in such a way that size can be handled can bring the number of generated hypotheses down, because hypotheses are invalidated earlier.

Identification of the background shape(s)

The background shape of an image is currently treated as any other shape. This means that objects that are separated from each other by the background aren't seen as different objects by the reasoning stage. Moreover, by treating the background as any other shape enables the creation of a lot of hypotheses in which the background shape is used as a normal shape (which are probably all incorrect interpretations).

This is a big problem; a lot of hypotheses are created in this way. Segmenting the shapes into groups which all contain just one or two objects will probably decrease the number of generated hypotheses by a great amount.

Grouping of similar shapes

When there are a lot of shapes adjacent to each other that share the same value for a specific property such as color and/or form, it is likely they belong to the same object (if not, it would be almost impossible to separate them from each other).

It makes sense to group these shapes together into one group that can be matched as a whole, instead of one shape at a time (and only to ungroup them when it is determined that they don't form a group after all).

An example is the shapes forming the beads of a necklace. These beads all have the same color, and roughly the same shape. Combining them into a single group would eliminate a lot of hypotheses that use the bead-shapes as normal shapes that could be part of, for example, a train.

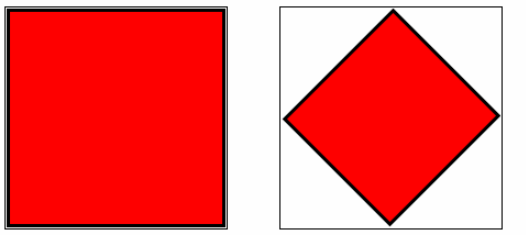
4.3 Recommendations for a another approach

In the current system, all that is known about a single shape is:

- The coordinates of its bounding box (and the middle of that box)
- The color of the shape

- The form of the shape (when it can be recognized, using the outline of the shape)

Nothing is known within the system about the orientation of the shape. As an example, the system cannot distinguish between these two red shapes: the sizes of their respective bounding box are the same, as are their color and their form.



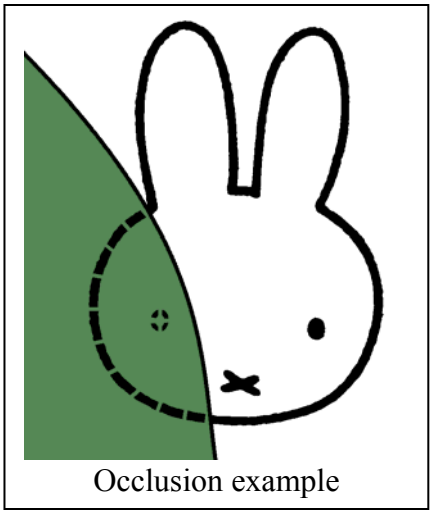
Shapes that generate identical information

This information can be quite useful however. For example, when shape is found of a rabbit's head, a prediction can be made for the expected position of the shapes within that head (the both eyes and the mouth).

Then, when shapes are matched to those template fields, a comparison can be made with the expected position, and the actual position of the shapes. Then, the hypothesis can be rated using the outcome of this comparison (a good match gives the hypothesis a good rating, but several bad matches will lower the rating of the hypothesis significantly).

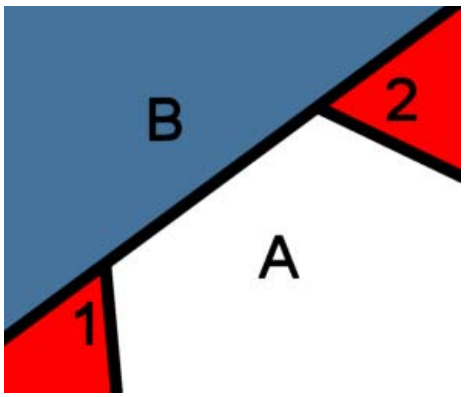
A rating calculated in this way will be much better then any rating the current system can generate, which will decrease the number of hypotheses that will have to be tested before a complete hypothesis is found (because the 'good' hypotheses can be extended first).

This mechanism can also be used to support interpretation in the face of occlusion. Then a partial shape is detected, a prediction should be made about the actual form of the (unoccluded) shape. The unoccluded shape can then be used to predict the positions of related shapes, and whether they themselves might be occluded.



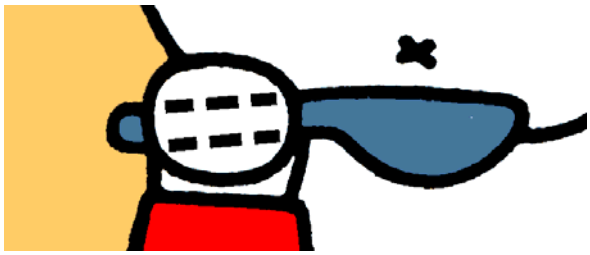
Occlusion example

In this example, the head of Nijntje has been partly occluded by a green shape. The actual shape of the head is show by the dotted line. Using the entire outline of the head shape, a prediction can be made about the position of the left eye. It can then be seen that the left eye lies at a position where it is occluded; therefore it is not a problem if no shape can be found to match the left eye.



Another occlusion example

Hints can probably be extracted from an image which parts of a shape are occluded. In this example, the outline of shape A has two T-crossings with the outline of shape B, where the outline of B remains straight. In this case, it could very well be a possibility that shape A is occluded by shape B (it isn't sure, but it is a strong hint).



Tracing an occluded spoon

This mechanism can also be used to support merging of shapes that have been split into to multiple parts by other shapes, like the spoon in this example is split into two shapes by the hand shape.

The combining mechanism can be augmented by a form of tracing through the occluding shape. There are strong hints that a shape is occluding another. By following the outline of an occluded shape from the point where it is occluded, an attempt can be made to connect it to compatible shapes. In the example above, the spoon shape could be traced through the hand, and be reconnected. In turn, this reconnection could enable the recognition of the spoon shape, because its entire outline (or at least, a prediction thereof) is now known.

By itself, these measures are probably not enough to support full blown occlusion-handling. The current form analysis uses the entire outline to determine which form a shape has. If the shape is partly occluded, the outline will be deformed, preventing accurate recognition. A possible approach to solve this problem could be the decomposition of an outline into parts (straight pieces, bends, etc.), and/or the approximation of the shape form into primitive forms (somewhat like the geon approach [BIED87], adapted for 2D-forms).

A system that implements all these techniques will probably perform much better when occlusion is present. The better rating (and evaluation) capabilities will probably reduce the needed computing power needed to recognize an image by directing the search into promising directions. However, the described techniques aren't without costs themselves. It might be very worth while to investigate their effectiveness.

4.4 Ideas for future work

A fun idea is to use this system to input images into the interpretation system, and use the interpreted contents as an input for a system like the Virtual Storyteller [THEU02]. The storyteller could generate a story in which the characters in the images play a role; and (with some more effort) the scenes depicted in the images are included.

Furthermore, this system could be used to interpret episodes of animated TV-series. There are series that are drawn in such a way that this system (with occlusion handling) could interpret, maybe with some specific enhancements like color gradient handling. An example for such a TV-series is Pucca [KIML06].

Using the image interpretation system, multiple images of an episode could be interpreted, after which another system can use the interpretation data to see how objects (and characters) move from image to image. This data can be used to automatically interpret the situations in the episode, as well as the behavior of the characters. This information can then subsequently be used to annotate the episode.

5 Literature

[BAAD91] F. Baader and B. Hollunder. *KRIS: Knowledge representation and inference system*. SIGART Bulletin, vol. 2, no. 3, p. 8-14, 1991. ISSN 0163-5719.

[BIED87] I. Biederman, *Recognition-by-components: A theory of human image understanding*, Psychological Review, vol. 94, no. 2 p. 115-117, Apr. 1987. ISSN 0033-295X.

[DORI99] D. Dori and W. Liu, *Sparse pixel vectorization: an algorithm and its performance evaluation*, IEEE Transaction on Pattern Analysis and Machine Intelligence, vol. 21, no. 3, p. 202-215, 1999. ISSN 0162-8828.

[FORS02] D. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*, Prentice-Hall, 2002. ISBN: 0-13-085198-1.

[GRIM90] W. Grimson, *Object recognition by Computer*, Massachusetts Institute of Technology, Cambridge, Massachusetts, United States of America, 1990. ISBN 0-262-07130-4.

[HAAR01] V. Haarslev, R. Moller. *RACER system description*. Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001), Lecture Notes in Artificial Intelligence, vol. 2083, p. 701–705. Springer, 2001. ISBN 3-540-42254-4

[HAN94] C.-C. Han, K.-C. Fan, *Skeleton generation of engineering drawings via contour matching*, Pattern Recognition, vol. 27, no. 2, p. 261-275, 1994. ISSN: 0031-3203.

[HE00] Ji He, Man Lan et al., *Initialization of Cluster Refinement Algorithms: A Review and Comparative Study*, Proceedings 2004 IEEE Int. Joint Conf. on Neural Networks, vol. 1, p. 25-29, July 2004. ISBN: 0-7803-8360-5.

[HECK82] Paul Heckbert, *Color image quantization for frame buffer display*, Computer Graphics (Proc. Siggraph), vol. 16, no. 3, pp. 297-307, July 1982. ISBN: 0-89791-076-1.

[HOPP93] T.Hoppe et al., *BACK Tutorial and Manual*, Technische Universität Berlin, Project KIT-BACK, March 1993.

[HOUG59] P.V.C. Hough. *Machine analysis of bubble chamber pictures*. Int. Conf. on High Energy Accelerators and Instrumentation, p. 554-556. September 1959.

[INTE06] Intel, *Open Source Computer Vision Library*, August 2006. http://www.intel.com/technology/computing/opencv/index.htm

[KATS94] Ioannis Katsavoundis, C.-C Jay Kuo and Zhen Zhang, *A new initialization technique for generalized Lloyd iteration*, IEEE Signal Processing Letters, vol. 1, no. 10, p. 144-146, October 1994. ISSN: 1070-9908.

[KIML06] Kim's licensing Co., Ltd., *Pucca introduction webpage*, July 2006. http://www.kimslicensing.com/english/pucca.htm

[MONO93] G. Monogan and M. Roosli, *Appropriate base representation using a run graph*, Proc. ICDAR 93, p. 623-626, Tsakuba, Japan, 1993. ISBN: 0-8186-4960-7.

[NEUM04] B. Neumann, R. Möller, *On Scene Interpretation with Description Logics*, FBI-B-257/04, Fachbereich Informatik, Universität Hamburg, 2004.

[OTTE89] P.J. van Otterloo, *A Contour-Oriented Approach to Digital Shape Analysis*, Prentice Hall, Hertfordshire, UK, 1991. ISBN: 0-13-173840-2.

[POPE94] Arthur R. Pope, *Model-Based Object Recognition: A Survey of Recent Research*. Technical Report 94-04, The University of British Columbia - Department of Computer Science, January 1994.

[QUAN96] J. Quantz, G. Dunker, F. Bergmann, and I. Kellner, *The Flex System*, KIT-Report 124, Technische Universität, Berlin, Germany, 1996.

[SAFA00] Maytham Safar, Cyrus Shahabi and Xiaoming Sun, *Image Retrieval by Shape: A Comparative Study*, IEEE Int. Conference on Multimedia and Expo (I), p. 141-144, 2000. ISBN: 0-7695-1198-8.

[SHEH04] Shehroz S. Kham Amir Ahmad, *Cluster center initialization algorithm for K-means clustering*, Pattern Recognition Letters 25, 2004, p. 1293-1302. ISSN: 0167-8655.

[SONG00] J. Song, F. Su, J. Chen, and S. Cai, *A Knowledge-Aided Line Network Oriented Vectorization Method For Engineering Drawings*, Pattern Analysis and Application, vol. 3, no. 2, p. 142-152, 2000. ISSN 1433-7541.

[STEF71] R. Stefanelli and A. Rosenfeld, *Some Parallel Thinning Algorithms for Digital Pictures*, Journal of the ACM (JACM), vol. 18, issue 2, April 1971, p. 255-264, 1971. ISSN: 0004-5411.

[SUET92] P. Suetens et al., *Computational strategies for object recognition*, ACM Computing Surveys, vol. 24, Issue 1, p. 5-62, March 1992. ISSN: 0360-0300.

[THEU02] Theune, M., Faas, S., Nijholt, A. and Heylen, D., *The Virtual Storyteller*, ACM SIGGROUP Bulletin, vol. 23, Issue 2, pages 20-21, 2002.

[USC06] University of Southern California, *PowerLoom™ Knowledge Representation & Reasoning System*, August 2006. http://www.isi.edu/isd/LOOM/PowerLoom/

[UT06] University of Twente, *ParleVision*, August 2006. http://hmi.ewi.utwente.nl/showcases/parlevision

5.1 Nijntje books

[BRUN96a] Dick Bruna, *Nijntje in de dierentuin*, Mercis Publishing, 1996. ISBN: 90-73991-83-8.

[BRUN96b] Dick Bruna, *Nijntje aan zee*, Mercis Publishing, 1996. ISBN: 907399134X.

[BRUN97] Dick Bruna, *Het huis van Nijntje*, Mercis Publishing, 1997. ISBN: 90-5647-211-9.

[BRUN99] Dick Bruna, *Nijntje en Nina*, Mercis Publishing, 1999. ISBN: 90-5647-362-X.

[BRUN02] Dick Bruna, *Nijntje danst*, Mercis Publishing, 2002. ISBN: 90-5647-164-3.

A Appendix A: Description system

The code in this appendix follows the following conventions:

- Words in uppercase (like CLASS) are keywords.
- Words in lower-case are names (of fields or constants).
- Parts between square brackets (like this: [, shape-form]) are optional
- When 'list of item' is used, a comma-separated list of 'item' can be used.
- When a slash ('/') is used, one of two options must be chosen

This description system predates the first use of the template-based terminology used in the rest of this thesis. Instead, an object-oriented naming scheme was used, of which traces can be found back in the names of the rules.

A.1 Template declaration

A new template is declared by the following line:

```
// Declare a new template
CLASS template-name [ ( IS-A parent-template-name ) ]
```

The inheritance system is used to allow multiple templates in a complex field. The complex field can designate a base template, and then all templates deriving from that base template can be used to match that complex field.

An advantage of this problem is the easy enforcement of a common set of attributes and named field lists in the templates that can match a complex field.

A.2 Field declaration rules

The following rules declare shape fields, complex fields, shape groups and holes. The first set declares mandatory fields, while the second set declares optional fields.

```
// Declares a shape field
REQUIRED-SHAPE(name [ , shape-form ]);

// Declares a mandatory complex field, that can be matched by the
// descendants of template 'template–type'
REQUIRED-OBJECT(name, template-type);

// Declares a shape group field, that can be matched by a group of
// similar shapes (not implemented in the current system)
REQUIRED-SHAPE-GROUP(name [ , shape-form ]);

// Declares a hole within a shape (not implemented in the current
// system)
REQUIRED-HOLE(name, shape, [ , shape-form ]);

// Optional counterparts of declaration rules
OPTIONAL-SHAPE(name [ , shape-form ]);
OPTIONAL-OBJECT(name , object-type);
OPTIONAL-SHAPE-GROUP(name [ , shape-form ]);
OPTIONAL-HOLE(name, [ shape-form ] );

// Declare a named field list (and optionally adding fields)
NAMED-MEMBER(name [ , list of field [ .named-field-list ] ] );
```

A.2.1 Shape classes

The following names are the names of the different classes of shape forms.

```
CIRCLE
RECTANGLE
RABBIT-HEAD
HALF-CIRCLE-OUTLINE
CHILD-MOUTH
GROWN-UP-MOUTH
SQUARE
HOUSE-SHAPE
BOOMERANG-SHAPE
HANGING-JACKET
TRAIN-LOCOMOTIVE
TRAIN-TRAILER
BED
TRIANGLE
WATERING-CAN
CAR-COACH-WORK
CAR-MUD-GUARD
SHOVEL-IRON
SNOUT
PIGLET-MOUTH
SHEEP-HEAD
PARROT-HEAD
PARROT-BODY
ZEBRA-SHAPE
KANGAROO-HEAD
ELEFANT
GIRAFFE-HEAD
HAND-WITH-THUMB
```

A.3 Form rules

The following rules deal with the form of shapes.

```
// Indicates that a shape group is a connected cluster of shapes
IS-CLUSTER(shape-group);

// Indicates that the outline of a combined shape group should have
// a specific form
HAS-COMBINED-SHAPE(shape-group, shape-form);
```

A.4 Structuring rules

These rules describe miscellaneous structuring constraints that haven't been implemented.

```
// Indicates that only one instance of this template may be present
// in an image (not implemented)
IS-UNIQUE(template-name);

// Indicates that this template is always a component of another
// template (not implemented)
IS-ALWAYS-COMPONENT(template-name);
```

```
// Indicates that this template is an abstract base template
// (not used in the current implementation)
IS-ABSTRACT-BASE(template-name);
```

A.5 Positioning rules

The following rules describe how shapes and objects are positioned.

```
// Indicates that the matches to two fields should be adjacent
IS-ADJACENT(field [ .named-field-list ] , field [ .named-field-list);

// Indicates that match to one field should lie within the match
// to another field
IS-WITHIN(field, field);

// Indicates that matches to two fields should have a specific spatial
// relation
HAS-X-POSITION(field1, field2, x-relation);
HAS-Y-POSITION(field1, field2, y-relation);
HAS-POSITION(field1, field2, x-relation, y-relation);

HAVE-EQUAL-Y-POSITIONS(shape-group);
HAVE-EQUAL-X-POSITIONS(shape-group);
```

A.5.1 Relations

X-coordinate-based relations:

```
LEFT              // totally left
MIDDLE            // 1 overlaps middle of 2
UNMIDDLE          // 1 does not overlap middle of 2
RIGHT             // totally right
LEFTOV            // overlap left of the middle larger than o. right
RIGHTOV           // overlap right of the middle larger dan o. left
UNIMPORTANT       // x-relation is unimportant
```

Y-coordinate-based relations:

```
ABOVE             // totally above
MIDDLE            // 1 overlaps middle of 2
UNMIDDLE          // 1 does not overlap middle of 2
UNDER             // totally under
ABOVEOV           // overlap above of the middle larger than o. under
UNDEROV           // overlap under of the middle larger than o. above
UNIMPORTANT       // y-relation is unimportant
```

A.6 Attribute rules

```
// Indicates that a shape matched to the field should have a specific
// color
HAS-COLOR(shape-field, color);

// Indicates that shapes (or shape-groups) should have the same color
HAVE-SAME-COLOR(list of shape/shape-group);

// Indicates that a shape group can contain only shapes with the
// two specified colors
HAS-TWO-COLORS(shape-group, color1, color2);
```

```
// Declares an attribute (not implemented in the current system)
ATTRIBUTE attribute-name = attribute-expression
```

A.6.1 Colors

```
WHITE
BLACK
BLUE
GREEN
RED
YELLOW
BROWN
```

B Example descriptions

B.1 Eyes

Base template for eyes:

```
CLASS Eye
// Describes an eye, is a base template with deriving classes Open-eye
// and Closed-eye

// Attribute all eye-templates must have
ATTRIBUTE is-open;

IS-ABSTRACT-BASE(Eye);
IS-ALWAYS-COMPONENT(Eye);
```

The open eye template:

```
// Template describing an open eye
CLASS Open-eye (is-a Eye)

REQUIRED-SHAPE(eye, CIRCLE);
HAS-COLOR(eye, BLACK);

ATTRIBUTE is-open = TRUE;


// Template describing a closed eye
CLASS Closed-eye (is-a Eye)

REQUIRED-SHAPE(eye, HALF-CIRCLE-OUTLINE);
HAS-COLOR(eye, BLACK);

ATTRIBUTE is-open = FALSE;
```

B.2 Rabbit-head

Base template for rabbits' heads:

```
// Abstract base, we have multiple types of rabbit-head
CLASS Rabbithead

// Attributes all head-templates must have
ATTRIBUTE is-child-head;
ATTRIBUTE has-open-eye;
ATTRIBUTE is-looking;
ATTRIBUTE color;

// Declare the named member list
NAMED-MEMBER(neck-joint);

// Rabbithead is an abstract base
IS-ABSTRACT-BASE(Rabbithead);
IS-ALWAYS-COMPONENT(RabbitHead);
```

The normal rabbit's head template:

```
// Normal rabbit head template
CLASS Normal-rabbithead (is-a Rabbithead)

// Declare all fields
REQUIRED-SHAPE(head-outline, RABBIT-HEAD);
REQUIRED-OBJECT(left-eye, Eye);
REQUIRED-OBJECT(right-eye, Eye);
REQUIRED-OBJECT(mouth, Mouth);

// The eyes and mouth must both be within the head-shape
IS-WITHIN([left-eye, right-eye, mouth], head-outline);

// Spatial relations
HAS-Y-POSITION([left-eye, right-eye], mouth, ABOVE);
HAS-X-POSITION([left-eye, mouth], right-eye, LEFT);
HAS-X-POSITION([mouth, right-eye], left-eye, RIGHT);

// Attributes
ATTRIBUTE color = head-outline.color;
ATTRIBUTE is-child-head = mouth.is-child-mouth;
ATTRIBUTE has-open-eye = left-eye.is-open OR right-eye.is-open;
ATTRIBUTE is-looking = EXISTS(left-eye) OR EXISTS(right-eye);

NAMED-MEMBER(neck-joint, head-outline);
```

B.3 Body and rabbit

The rabbit-body and rabbit templates illustrate how the named-member facility is used.

```
// Describes a body of a person (rabbit) wearing clothes.
CLASS Body

ATTRIBUTE color;

NAMED-MEMBER(neck-joint);
```

The rabbit template:

```
// A rabbit
CLASS Rabbit

// Field declarations
REQUIRED-OBJECT(head, Rabbithead);
REQUIRED-OBJECT(body, Body);

// Spatial relations
IS-ADJACENT(body.neck-joint, head.neck-joint);
HAS-Y-POSITION(body, head, UNDERMDL);
HAS-POSITION(head, body, MIDDLE, ABOVEOV);

// Other attributes
HAVE-SAME-COLOR([body, head]);
```

The spatial relation that makes sure that the head must be adjacent to the body uses the neck-joint member lists from both the Rabbithead and the Body templates.

C Test images

The following images were used as the main test set.