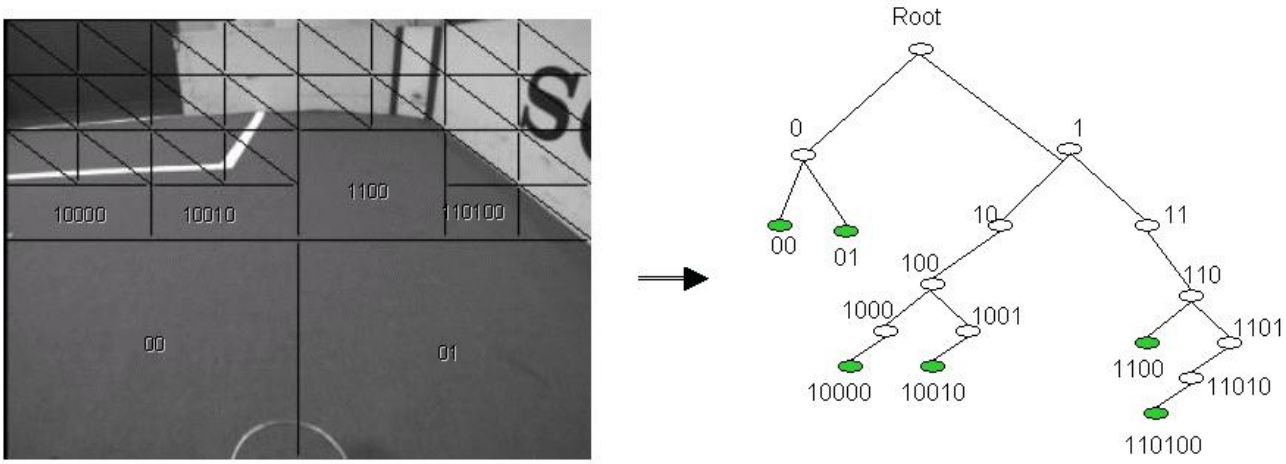
A tree-based appearance model for robot localization



Susanne M. Maas
Artificial Intelligence - Autonomous Systems
Informatics Institute, Faculty of Science (FNWI)
University of Amsterdam(UvA), The Netherlands
email: smmaas@science.uva.nl

Supervisor: Nikos Vlassis

Keywords: self-localization, appearance model, image segmentation, tree-based modelling, database indexing, trie

August 8, 2003

Contents

Acknowledgments

First of all, I would like to thank Nikos Vlassis, my supervisor during my graduation research, for his guidance, advice and patience. Especially during the writing of my thesis he has been a great help. Further, I also want to thank Bas Terwijn for his help, especially on the implementation of the software. I am also thankful for the help of Marius Zaharia and Mathijs Spaan during my research.

Furthermore, I am also grateful for the help of other people during my writing-process: Floris for commenting my thesis; Casper for his advice; Rizgar for his help on processing some important images. Also Yanming, Tesla, Stephen, Zohra, Thanimporn, Nicky, Shirley and Yeunying, I thank them for their support and belief in me that I can make it. Also I would like to thank them all for making my time as a student memorable and fun.

Last but certainly not least, I want to mention my family and especially my parents for their love and support, and being patient with me all the time. Thank you.

List of Figures

Chapter 1

Introduction

Intelligent mobile robots are becoming more and more common these days. Independent moving and interacting robots do not just belong to science fiction stories anymore. There are different situations imaginable in which it is convenient to have autonomous mobile robots operating without human interference, like tasks under dangerous conditions (for instance, working in a mine field) or at hard—if not, impossible—to reach places for humans (like missions to other planets [9]). To have them moving around autonomously they need to be aware of their environment and their position in it to be able to make the right decisions. When humans walk through a room they—consciously or not—observe the area to know how the environment is structured. They need to form an idea of the environment in order to reason what their position is. This is necessary to plan a path that is efficient and bump-free. This is also the case for mobile robots. For them it is preferable that they move around efficiently. This means that they need to move around without useless time-consuming operations, for instance collision with other robots or obstacles. To build up its idea of the world a robot does not need to know every detail but a model of it containing essential information would be sufficient—the so called appearance model. To create this model it needs to look around in its environment. There are different kinds of sensors available for this task. The one that works almost like the human eyes are cameras.

To build up a model of its world a robot first needs to do some exploration. It will first observe its environment by taking prototype images from different locations with varying orientations and store these in a database. When it needs to know its position it can consult this database to find the prototype that matches the current image that has been acquired at the time self-localization was requested. There are several ways to find the right match. A common method is a correlation-based comparison of images. Pixel-groups in the current image are compared to the pixel-groups of the images in the database. The problem is that images taken from digital cam-

eras can contain a huge amount of pixels. The images used in this thesis are 320 x 240. This means that one image contains more than 76.000 pixels and has to be compared with other images of the same size. Consequently, this way it can take a considerable amount of time to find the right match. In a real-time system we want to know our position as quickly as possible.

The images need to be described in a more compact way. As mentioned before, we do not need details of the image but just a model with essential information for the robot to localize itself. To achieve this a proper feature set needs to be selected from the image. This feature set has to represent the image in a compact but discriminative way such that fast lookup is possible. Extracting features from an image can be regarded as a segmentation process. The dimensionality will be reduced because parts of the image will be grouped, and clusters that belong to the feature set we are looking for will be filtered. This will contribute to the reduce of the dimension. In [4] several segmentation methods to achieve this are discussed. One of the methods that is similar to the one used in this thesis is called a divisive clustering. The initial state is the complete image that represents one big cluster which will be divided into smaller clusters in order to separate the feature from the non-features. To decide whether a cluster is good enough—and dividing is not necessary anymore—some criteria are needed. An example could be color or texture.

1.1 Objectives

The research described in this thesis is a part of a larger project: the robot soccer project of the University of Amsterdam (UvA). This concerns robots of the Dutch team named Clockwork Orange. This project demonstrates AI techniques in soccer playing robots with the help of annual soccer tournaments between different teams [1]. To play soccer, this the robots need to do several tasks autonomously among which self-localization [10]. Currently several geometric vision-based methods are applied, for instance the Hough transform which computes the (straight) lines in an image. These methods take too much time to compute the robot's position. What we need is a method by which the robots localize themselves in a more simple yet quick way. To get to this we can split this problem up into three subproblems:

- Decrease the dimensionality of the input (image) data. We need to find an appropriate feature set that describes what we see in an image such that it helps to localize the robot. Of course we also need a method to extract this from the raw data and as a result will have compact descriptions of each image (images in the database as well as the ones taken real-time)

- Store the compacted images in a database such that fast lookup is pos-

sible. This actually corresponds to building up the appearance model of the world. Database construction has to be done in an efficient way which makes fast lookup possible.

- A lookup method to find the image in the database corresponding to the current position of the robot.

Initially the field that is going to be explored is empty but during the game the view may be (partly) occluded by other robots or the ball. During our research we also have to keep occlusion in mind. In short we need to find a feature extraction method such that a fast lookup is possible. The method has to be robust enough against occlusion and other effects.

Finally a few remarks on handling colors in this project. It is known that we are dealing with robots operating in a soccer field and which colors occur in the observations. For instance we know that the field is green and the lines are white, the ball is orange and the goals are blue and yellow. We want the robots to operate at best in this soccer environment we used these colors. Knowing what colors occur in an environment makes the search for the features easier but still it will be hard to define the colors we are looking for. Another problem to be considered when using color is that its appearance is dependent on the lighting of the room or area. During the soccer-game this problem is being suppressed by constant lighting conditions.

1.2 Layout of this thesis

This thesis is organized as follows. The second chapter explains the method that is currently used in the Dutch team and its downsides. It will also motivate why we have chosen for the method which is described in this thesis. Initially different approaches were developed and tested, and led to the approach which is described here. In chapter 3 we will explain the feature extraction method: which feature is selected and why; how it is extracted from the data; how it is stored and used for lookup. Chapter 4 describes the set of experiments performed and the produced results. Conclusions and future work will be discussed in chapter 5.

Chapter 2

Robot Localization Methods

In real-time systems it is essential to get up-to-date information. Namely, for mobile robots it is important to know their accurate position in the environment in order to make the right decisions. The current localization method used for the soccer team has its disadvantages. This chapter describes shortly the localization method currently used and its downside. This also will be the motivation of the research in this thesis. A more detailed description of this method can be found in [2] and [3]. At the end of this chapter the new approach will be discussed shortly.

2.1 A geometric approach

The localization method in [2] and [3] adopts a geometric two-tiered approach. This means it has a global search and a local search. The global search detects the lines and transforms the found lines into robot coordinates. With the use of a field model it generates candidate positions. The local search tracks the robot's position at real-time speed.

The localization method focuses on the field lines. In order to find them two tools are used to detect them: a reliable line/field edge detector and a line detector. In a earlier process a labelled image called image L is the result of object detection. Based on its color the pixels in the image are labelled according to the object(s) they belong to. This labelled image is used to detect the lines by checking whether the pixels are classified as floor or lines. The result is an edge image (E). A second edge detector will define edges more specifically. In order to find the nearest edge it fits a second-order polynomial to the first derivative's distribution in a 5x3 neighborhood (see Figure 2.1). Assuming that the extremes of the polynomial function represent the white-green and green-white borders the edges can be found.

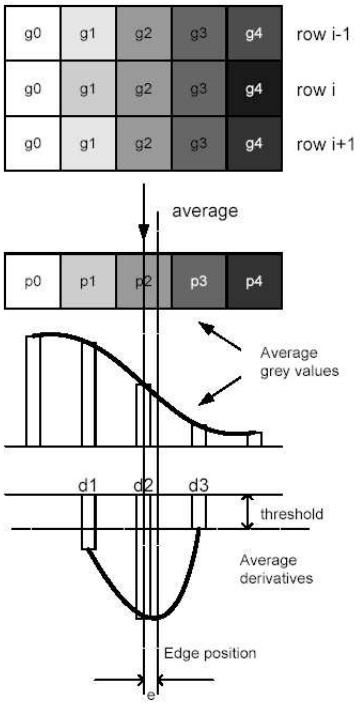


Figure 2.1: *The edge detector [2].*

2.1.1 Global search

The goal of the global search is to determine the robot's position without any prior knowledge. It transforms measured lines in the image to robot coordinates. To achieve this it first needs to detect the field lines. This is done in two steps. First it will find line segments then next their positions are determined.

The Hough transform is used but not on the total image because of lens distortions and the presence of a center circle. To this end, the image is first divided into sub-images which are small enough to assume that the lines are mostly straight. Next the Hough transform is used to locate the positions of the lines. A number of sub-pixel edge positions are found with a sub-pixel edge detector. Now these edge positions are converted to calibrated camera frame and then to robot coordinates.

2.1.2 Local search

The local search matches the position of a given candidate with an image in real-time. Four local methods have been implemented. They all take an estimate of the robot's position and calculate the corresponding field line pattern using a CAD model to match the result with the measured lines. These methods are called:

1. The Distance Transform Method
   The bottom-line of this method is that it creates a template of the lines for a certain robot position and matches this template with the image real-time. This is done iteratively on the robots estimated position parameters including a small deviation. The position with the smallest distance (error) will be considered the best match. The new position is found.

2. Delta method
   This method minimizes the distance between the lines generated from the assumed position and lines found in the edge image (E). To calculate the minimized distance a least squares approach has been adopted.

3. Delta2 method
   Detlta2 method works the same as the delta method but without creating and using an edge image just to speed up the method. To find the edges the horizontal and vertical edge detectors are operating on the Y of the YUV images. Once the edges are found the distance is—like the delta method—calculated by a least squares approach.

4. Average distance method
   The average distance was implemented in [2] to see if it was possible to give updates on the position purely based on global method data. It uses the edge detectors which are also used by the delta2 method to calculate the average distance between the measured and the model lines. But differs from the delta2 because it is calculated in pixel space instead.

2.1.3 Downsides

Of all the implemented local methods the delta2 method was preferred in [2] because of its speed and accuracy. Yet this method has several limitations:

- The prediction error that grows rapidly with the error in the a priori position estimate.
- Accuracy of the method
- When lines were partly or completely occluded the method would return a low matching score.

2.2 Appearance model

The localization method discussed in the previous section uses a geometric approach to determine the position of the robot. Based on the estimated position and measured robot position the positions of the lines are calculated. The match is ranged by means of their differences. In this thesis an appearance model will be used instead. The appearance based approach uses a feature set as a measurement in its matching function. This approach has already been studied in several applications, e.g. object recognition [8], robot navigation and localization ([6] and [11]).

In [6] a probabilistic appearance-based approach was used to model the environment. Eventually this model was used to localize the mobile robot. Localization was done by the Markov localization method. This means that we represent the position of the robot by a probability density $p(x)$. The Markov localization needs two probabilistic models to get a good position estimate, namely a motion model and an observation model. The motion model describes the effect a motion command has on the position of the robot and can be represented by a conditional probability density

$$p(x_t|u, x_{t-1})$$

which determines the distribution of x_t, the position of the robot after the motion command u at time t. The observation model describes how the observation, robot's location and parameters of the environment are related to each other. The conditional distribution of the observation model can be described as

$$p(z|x;\theta)$$

where θ describes the distribution and represents the 'underlying' environment and z is the observation. Using the Bayes' rule the location of the robot after observation z can be estimated by the posterior distribution

$$p(x_t|z) = \frac{p(z|x_t;\theta)p(x_t)}{\int p(z|x_t;\theta)p(x_t)dx_t}$$

2.2.1 Principal Component Analysis

Additionally, in [6] the dimensionality of the images (d) needs to be reduced. This is done by means of Principle Component Analysis (PCA). The eigenvectors of an image set are calculated and used as a orthogonal basis for representing individual images. The first q eigenvectors are only retained, where q is smaller than d. These eigenvectors are vectors in d-dimensional space and can be displayed as images which are known as eigenimages. Now from the projection of the d-dimensional images z on the q-dimensional eigenspace we get a feature vector y. We use this feature vector y for localization, as above:

$$p(x|y) \propto p(y|x;\theta)p(x)$$

The next step in [6] was to estimate the observation model $p(y|x;\theta)$ from a dataset $\{x_n, y_n\}$, $n = 1, .., N$. Here a kernel density estimation or a so called 'Parzen' estimator is used. The density function is estimated by taking the sum of kernel functions around the N data points of the training set. The width of the kernel and the training points themselves are considered the parameters θ of the environment. Once the observation model is known the estimation of the robot's position is possible. Instead of a Parzen method, a nearest-neighbor method can also be used, which is faster to compute [11].

The experiments in [6] showed that localization with this method works well. The experiments were done in an office environment but could also take place somewhere else. However this method has not been tested on situations with occlusion. It is known that PCA under occluded circumstances does not work well [7]. PCA is a global feature extraction method which will cause an error that will spread over the whole (reconstructed) image. Therefore PCA is quite sensitive to occlusion.

Because of this in this thesis another feature extraction method will be applied instead of PCA. Here in our proposed method occlusion will be taken into account during the whole process, also in the extraction process.

2.3 Motivation of our approach

The last limitation of the geometrical method mentioned in section 2.1.3 will be the focus of this thesis. We started a new localization method from scratch and developed a new localization method with the aim to handle occlusion. Occlusion seems to be an annoying but in practice unavoidable occurring aspect. The key is to find a method which is simple and robust enough.

In this thesis we will use an appearance based method. This localization method depends on the selection of the right feature. Several features are more logical to use than others. In the robot soccer field, the first feature that comes to mind are the white lines. That was the feature which we used in our first attempt.

The first idea was to divide the image in smaller patches of size, for instance, 60x40 and count the white pixels per patch. The idea was that we describe an image in the form of a series of numbers that represents the distribution of the white pixels in the image. However, detecting the white lines did not seem as easy as it appeared at the beginning. Because different shades of white exists it was hard to define what we could consider as white.

Another problem that arises is due to the camera and how it registers the lines. Lines nearby do not become a problem but lines that lie further away become too small and too thin to detect. Their pixel values are tending to be reddish or greenish. This results in a wrong detection of the white lines.

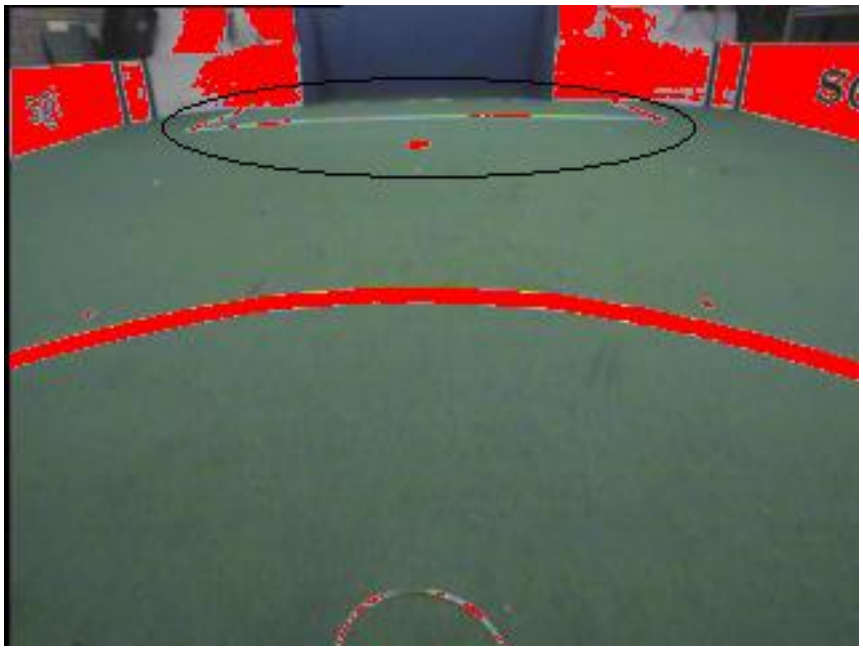


Figure 2.2: *Simple white line detector.* In a simple white line detector the pixels that meet the conditions for being white (thresholds on the red, green and blue values) are painted red to see if it detects the white (lines) the right way. As can be seen in this figure white lines that lie further away are too small and not defined as white lines. What is supposed to be seen as a solid line will be detected as a broken line.

A solid line is seen as a broken line (Figure 2.2).

Even if these problems did not occur it still would be the question if the lines were the best choice or whether there would be another feature more suitable for the localization task. When we look at an image of the soccer-field it is obvious that relatively more green pixels than white pixels occur. As can be seen in Figure 2.3 it is obvious that searching for something like the green of the field can be easier found than the white lines. Because the input data are complete images it is easier to start at image level and work the way down to smaller subimages. Considering all this it seemed better to choose as our feature the complement of the white lines, namely the green of the field.

It is important to find the green patches in a fast way. The question if color is necessary was raised. To detect the green in a color image three histograms are needed (the red, green and blue histogram). It seemed that it was possible to recognize green patches based on certain characteristics of its intensity histogram of the grey level images. In this case just one histogram is needed which is much simpler.

In the next chapter we describe in detail our feature extraction method.

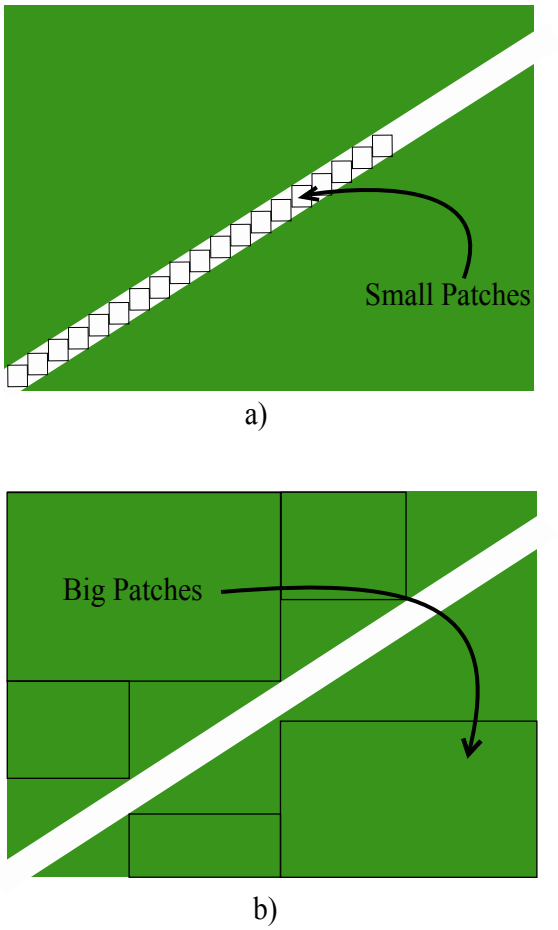


Figure 2.3: *Size of patches.* When selecting white lines as features, smaller patches *(a)* are needed to find them. Patches that can identify green chunks of the field *(b)* can be much larger. This means that green patches are easier and faster to detect than white patches.

Chapter 3

The Proposed Model

In this chapter the developed localization method will be explained. We will describe how the feature extraction operates, how its result will be stored and the image matching process.

For the robots in order to localize themselves they need to build some reference. This will be the appearance model of the world. After an exploration stage—which will result in an amount of pictures taken from different location in the empty soccer-field—the dimensionality of the images have to be reduced. This will be done by our feature extraction method. They will take out the important parts of the image that characterizes the image. This will result in a compact description of each images in a tree-form.

After this step these trees will be collected in a *trie*, which is a structure that supports fast indexing. This trie will be our final appearance model that will be used during the localization. It gives us the possibility to find images quickly. These two steps are both done off-line. This has to be done before a soccer-game begins.

In real-time, during the game the robot will use the trie as a lookup database. The robot first takes a snapshot from which the features will be extracted the same way as the images which were taken off-line. The result will also be a tree like the images in the database. Once they have the same format, it is easier to compare them with each other in order to find the best match. The location where this images has been taken must be the same location of the robot at that time.

For an overview of this method is showed in Figure 3.1. It will be described in more detail in this chapter.

3.1 Feature extraction

The green of the field is selected as the feature used for segmentation. It seemed to be possible to detect the green parts of the field by just using the intensity histogram of a greylevel image. When using color images the

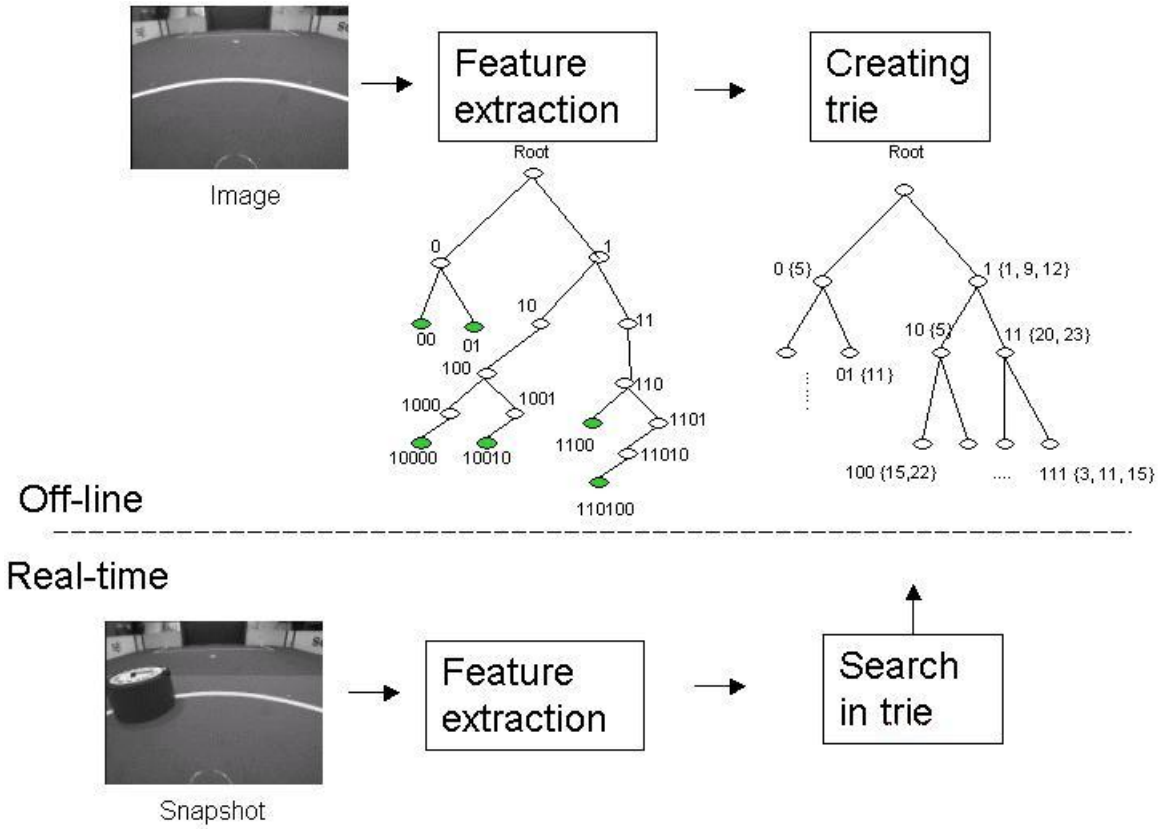


Figure 3.1: *Diagram of our proposed method.*

colors are defined in terms of three values: the contribution of red, green and blue. This means that histograms of these basic-colors would have to be used, so three histograms in total. To keep it simple (which mostly means 'fast') a greylevel image with only its intensity histogram will be used. The characteristics of a typical histogram of a green patch are determined in advance and used as a measurement to detect if a patch just contains green of the field or not. These characteristics are: the mean and the variance of the intensity value, which values have to lie in a range of a minimum and maximum. Once we know these characteristics it is possible to detect the green patches.

We start by dividing the acquired image—which is transformed to a grey image—into smaller patches dynamically. This means that we do not use a static lattice but create smaller patches of the initial image by splitting it up repeatedly until we got blocks that contain the part of the image that we were searching for, namely the green patches. The division process will start by dividing the image horizontally into two subimages (Figure 3.2(a)), the next division will be done vertically on the two subimages which will result

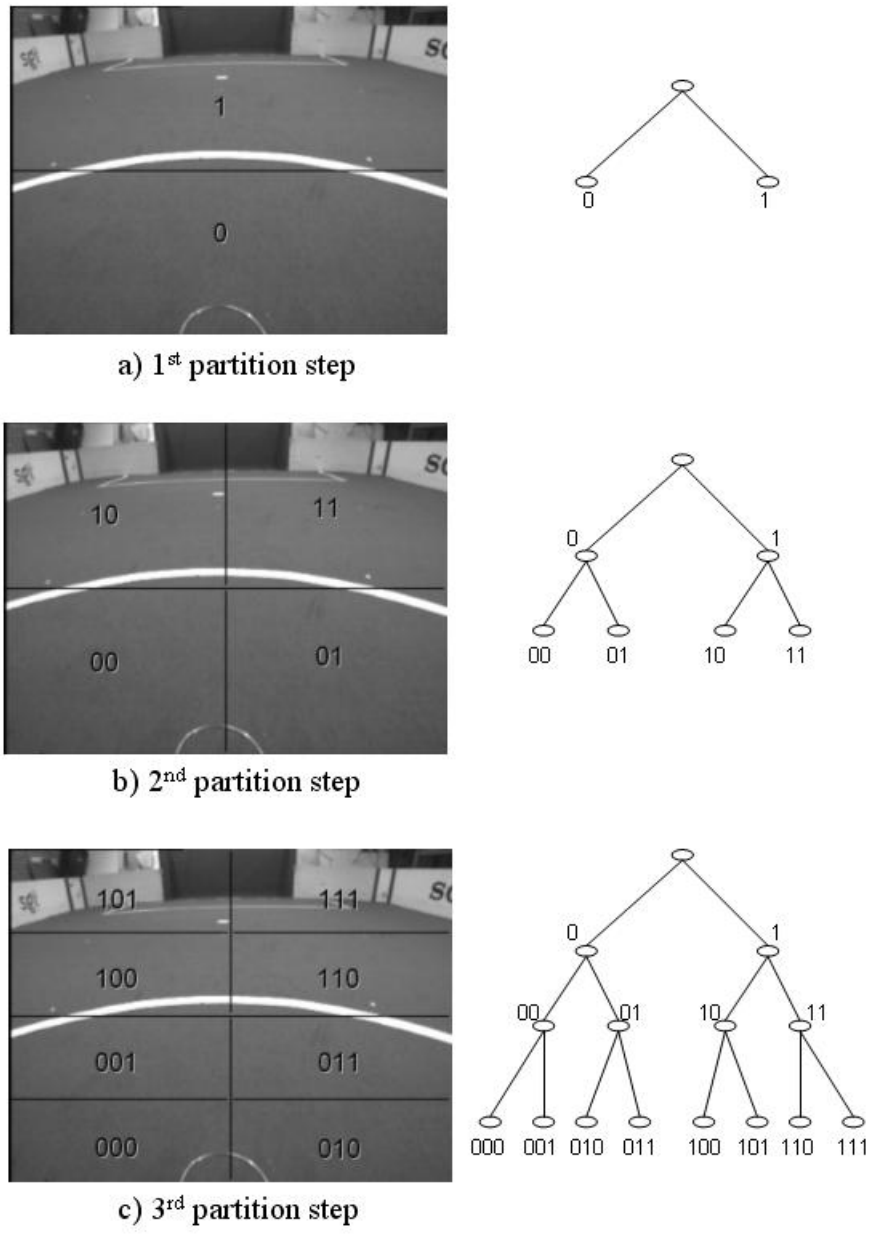


Figure 3.2: *Partitioning process and the resulting tree.*

in four subsubimages (Figure 3.2(b)), after that it will be horizontally again on the previous resulted subsubimages (Figure 3.2(c)) and so on. These sub(sub..)images will be referred to as patches in the rest of the thesis. This process of dividing the patches horizontally and vertically will result in a set of rectangularly shaped patches.

After each partitioning, the resulting patches will be labelled and checked

whether we are dealing with a green patch or not. A green patch is a patch which only contains the green of the field. If this is the case for the current patch, it does not need to be divided any further. If a line runs through it or any other object other than the green 'grass', the patch will not be detected as a green patch. Instead, it needs to be divided further to find the green and separate it from the non-green. The intensity histogram of the patch should tell us if the patch is totally green or not by means of its mean and variance, as we explained earlier in this section.

The labelling of the patches that result from the division process will be as follows. After the first division of the initial image the resulting patches will represent the upper part and the lower part of the image. These will be labelled as *1* and *0* respectively. The next division will be done vertically like stated before. The patches will result in two smaller patches. The parts will be labelled like their parent-patch but with a *0* or *1* concatenated at the end, for the left or right part respectively. For instance, the patch labelled as *0* will have children called *00* (the left part of the patch) and *01* (the right part of the patch). This kind of labelling provides us the ability to create a binary tree of the resulting partitioning, as showed in Figure 3.2. At the same time we are also labelling a 'path' in the tree that describes an image. This way the image can be represented as a binary tree which will be the result of the partition process. This process will be repeated until a maximum of divisions have occurred. In our experiments we use maximum depth of six. This maximum is applied to avoid generating patches that are too small to contain useful information.

To summarize, every image is represented by a set of green patches, forming a labelled binary tree, as showed in Figure 3.3. The label of a node (patch) is given by the path from the root of the tree to this node. All leaf nodes' labels constitute the features of a given image. This modelling provides us the ability to store the images in a compact way that supports fast and robust lookup, as it will be described next.

3.2 Storing the features using a trie

Once the features are extracted from the images they need to be stored such that a fast and robust lookup can be achieved. A lot of methods have been developed to store data efficiently. Some examples are hash-tables and inverse indices in relational databases.

The labels of the green patches that are detected by our method allow us to store them in a tree-structure. Different kinds of trees have been developed. A convenient one for our problem is the trie which is discussed by Knuth in [5]. First we will explain what a trie is and how it works. Then we will explain how we applied the idea of a trie in our method.

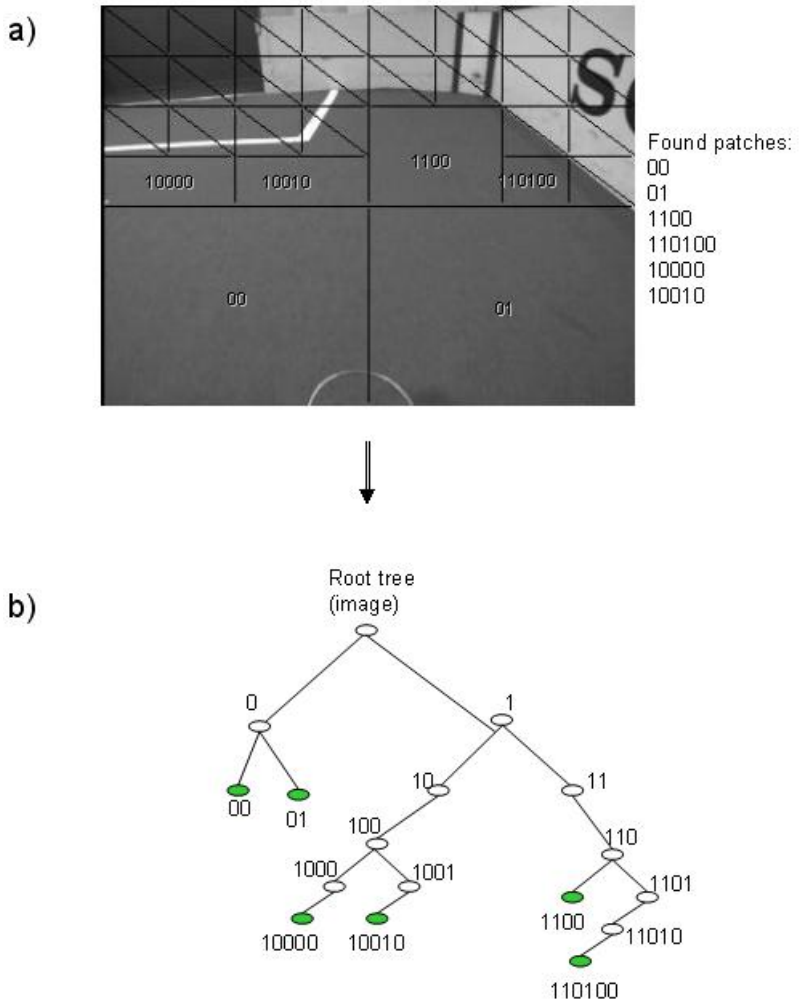


Figure 3.3: *Tree of image as a result of feature extraction.* From the discovered green patches in the image (a) a tree can be build which represents the image in a compact way (b).

3.2.1 Trie

As described in [5] a trie is a way to efficiently store data based on the representation of the data. A trie can compared to the thumb index on dictionaries. The thumb index creates the possibility to find the pages where words are grouped that starts with the same letter.

A trie is an M-ary tree consisting of nodes that represent M-place vectors. These vectors contain components that corresponds to digits or characters. Each node on a level l represents the set of all keys that begin with a certain sequence which is l characters long. This is called the prefix. The nodes in its turn represent an M-way branch, depending on the $(l+1)$st

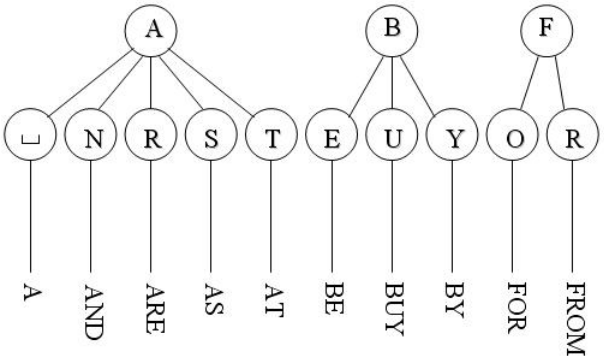


Figure 3.4: *Trie [5]*

character/digit. An example of a trie which stores words is showed in Figure 3.4. These words are stored in a M-ary trie. The trie will be searched starting from the top node downwards.

A special case is the binary trie. In this case $M = 2$. For this case two kinds of methods exist. The first one is called the digital tree search. It stores full keys in the nodes. But searching in the tree will be done with pieces of the argument. Depending on the value of this piece of the argument it is decided whether we have to take the left or the right branch. This tree also contains KEY, LLINK and RLINK fields which indicates whether we are dealing with a key, where the left and right link refers to respectively.

The other method is called "Patricia" (Practical Algorithm To Retrieve Information Coded In Alphanumeric). This is constructed such that it can handle long keys like for instance phrases. Also the Patricia method needs KEY, LLINK and RLINK, LTAG and RTAG, and SKIP fields. A specific explanation about the working of these methods can be found in [5].

3.2.2 How we used a trie in our method

In our case we used a binary trie. Because we have chosen to represent our feature set with binary digits that indicate its position in the image, we did not need that many fields stored in the trie nodes.

Our trie is like in Figure 3.5. No additional fields are needed because direction instructions are included in the label of the green patches. To build up the trie the set of trees, which represents the images in terms of the found green patches, are inserted in the trie. In Figure 3.3 an example of a tree as a result from the feature extraction of the image is showed. In the trie we will collect all trees like these which we got from the former stage which was explained in the previous section.

In the off-line phase, each (image)tree will be inserted in the trie one by one. Each node in the binary trie represents a possible green patch. If a

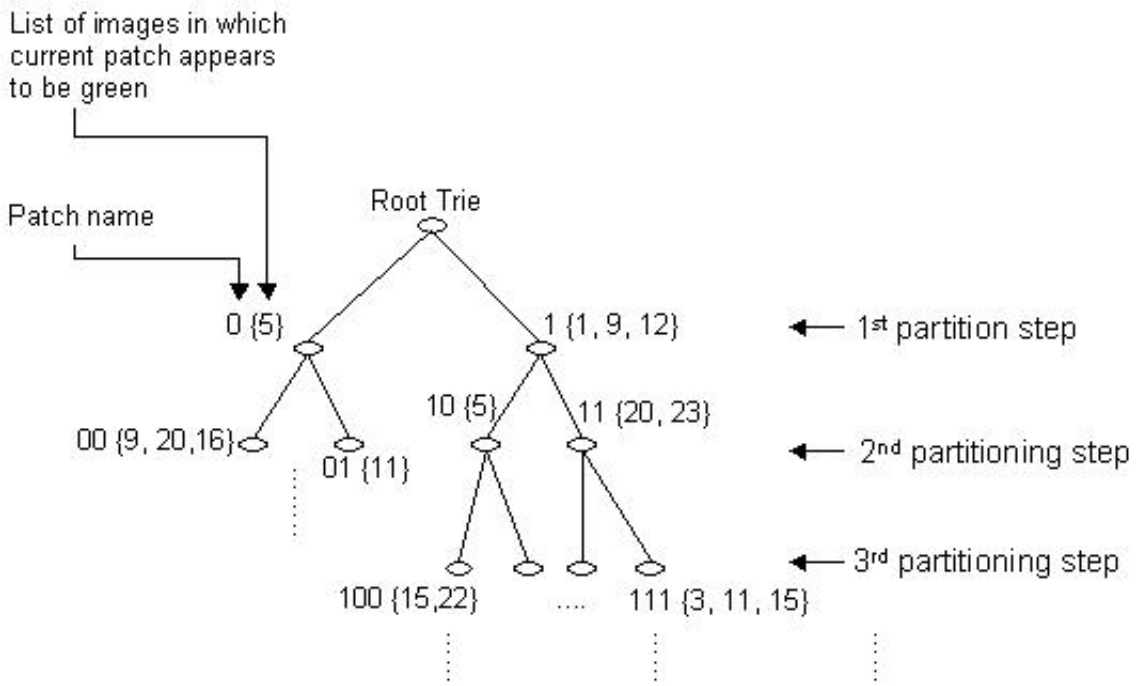


Figure 3.5: *The trie used in our method.* The indices of the images that contain the corresponding feature/patch are listed in brackets after the patch label.

green patch is detected in an image an identification (like an index or pointer to that particular image) will be added to a list in that node. Once the trie has been build up, each node of it will contain a list with all the indices of the images in which that particular patch was found. This resulting trie will be like the one showed in Figure 3.5.

A trie facilitates searching with prefix-keys which makes matching of images by lookup in the tree quickly. This matching process will be discussed in the next section.

3.3 Fast lookup

In principle a trie allows for fast lookup as follows. We will look up each patch from the root of the trie to that particular node. In each node the images that contain that patch are stored. At each search these images are retrieved.

In our case we need to find best matches for each patch label. The result of one such green patch-search is a set of collected candidate images. One of these images is probably the one we are looking for but to be sure we need to lookup the other green patches the same way. This will result in

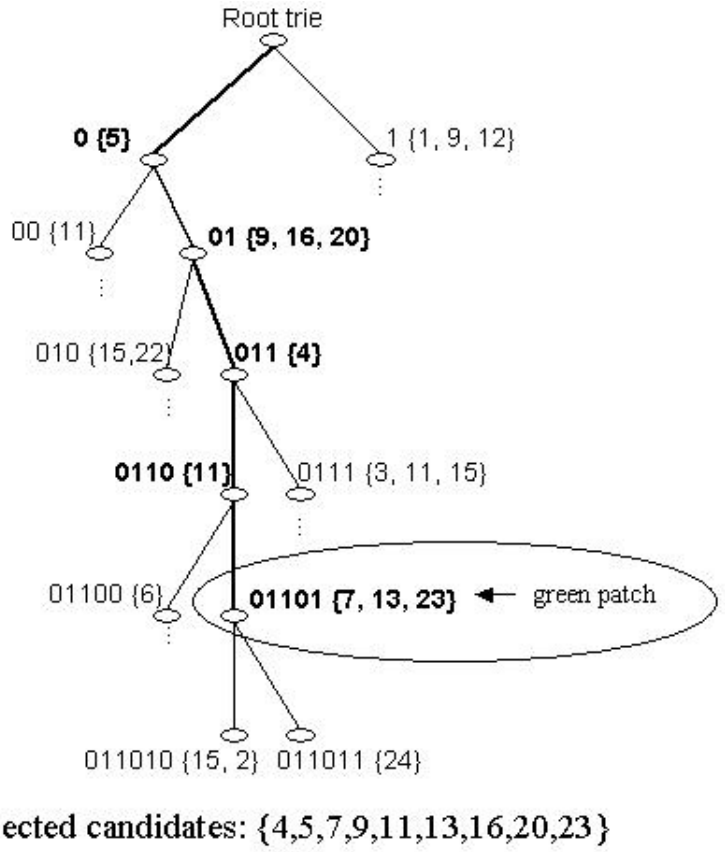


Figure 3.6: *Fast lookup with the trie.* The green patch labelled *01101* was found in the snapshot and will be looked up in the trie. To find the images in which this green patch might occur, we work our way up towards the root and collect all the candidates.

different sets of candidate images. Taking the intersection of these sets will result in a set of best matching images. In case of a perfect match, the matching image will show up in all the sets. It is not realistic to aim for perfect matches. Therefore, our method counts the times a candidate shows up in the retrieved sets. The one which occurs mostly during the lookup is likely to be the one we are looking for. Alternatively, the first k-highest matches give us the first k neighbors in the appearance space.

Because the depth of the trie is small (6) for simplicity in our experiments we just stored all $O(2^6)$ patches in an array, together with the images they came from. This way the leaves of the trees (green patches) can be found directly without going through the path from the root of the trie toward the node. However for larger tries this would be inefficient.

3.4 Handling occlusion

Our method can handle occlusion because of the way the features of the images are represented in a tree-form. As explained above, for each green patch we search for its corresponding node in the trie. That node holds a list of all the images that contain such a patch. Then we go one level up in the trie to the parent node. Note that this parent is labelled like its child, for example *01101* but without the last digit. So the parents label is *0110*. We also gather the images that can be found in its list. This way we proceed upward towards the root and gather all the candidate images we find on our way (Figure 3.6). The result of one such green patch-search is a set of collected candidate images as above.

Note that we are not only retrieving the specific node but all its ancestors. When an image is occluded, a green patch is blocked by some object which can be a robot or the ball. These objects are not green and will not pass the green-test. If the patch is partly occluded then as a result the patch which normally would be considered as green will be further partitioned, while it has not reached its minimum size yet. Now a green patch will be detected at a later stage as shown in Figure 3.7. The patches *10011* and *0011* are occluded and will be divided into smaller patches in the snapshot, namely *100111* and, *00111* and *00110* respectively. We note that these patches are the children of the ones which we want to retrieve. The nodes which will be retrieved are from a lower level in the trie. Because we are not just collecting the images listed in that node but also the ones listed in its ancestors the right green patch will also be collected like in Figure 3.8. This way the method is robust to handle occlusion.

3.5 Localization

The observations of the robots are raw images which are high dimensional. Our proposed method which is described in this thesis reduces this to more compact descriptions of the images. At the end they only contain the core information which is needed for localization. In this case it is reduced to how the green is structured in the images.

Now to localize the robots, the reduction will be such that a fast lookup is possible and quickly a set of candidates are selected. The final result of the method will be a ranked list of images from the database and their matching scores according to the method.

This ranking of these images will be passed to the particle filter. The particle filter will use this information to filter out the right location. This will be done by reasoning from the initial state of the robot through a sequence of observations. In combination with the scores given by our model it should find the right location throughout the soccer-game. For more

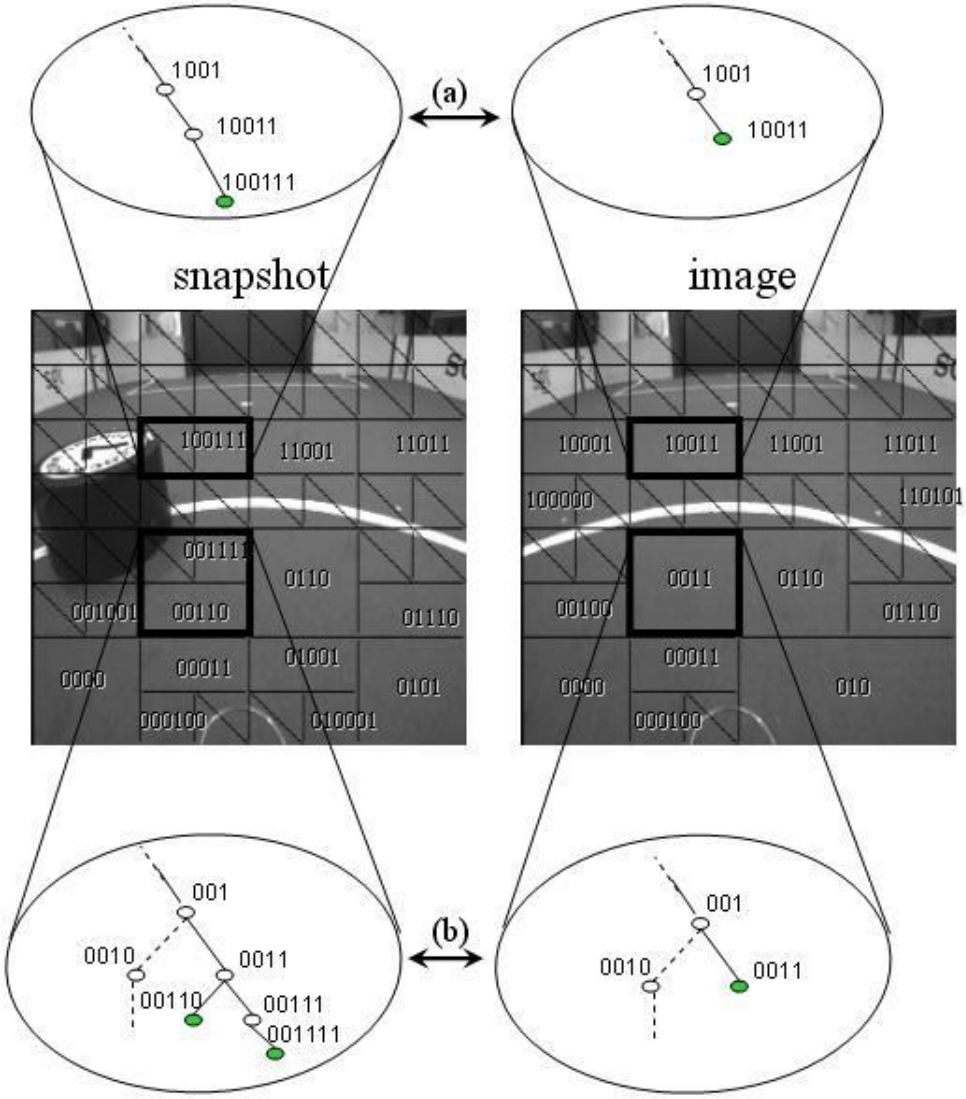
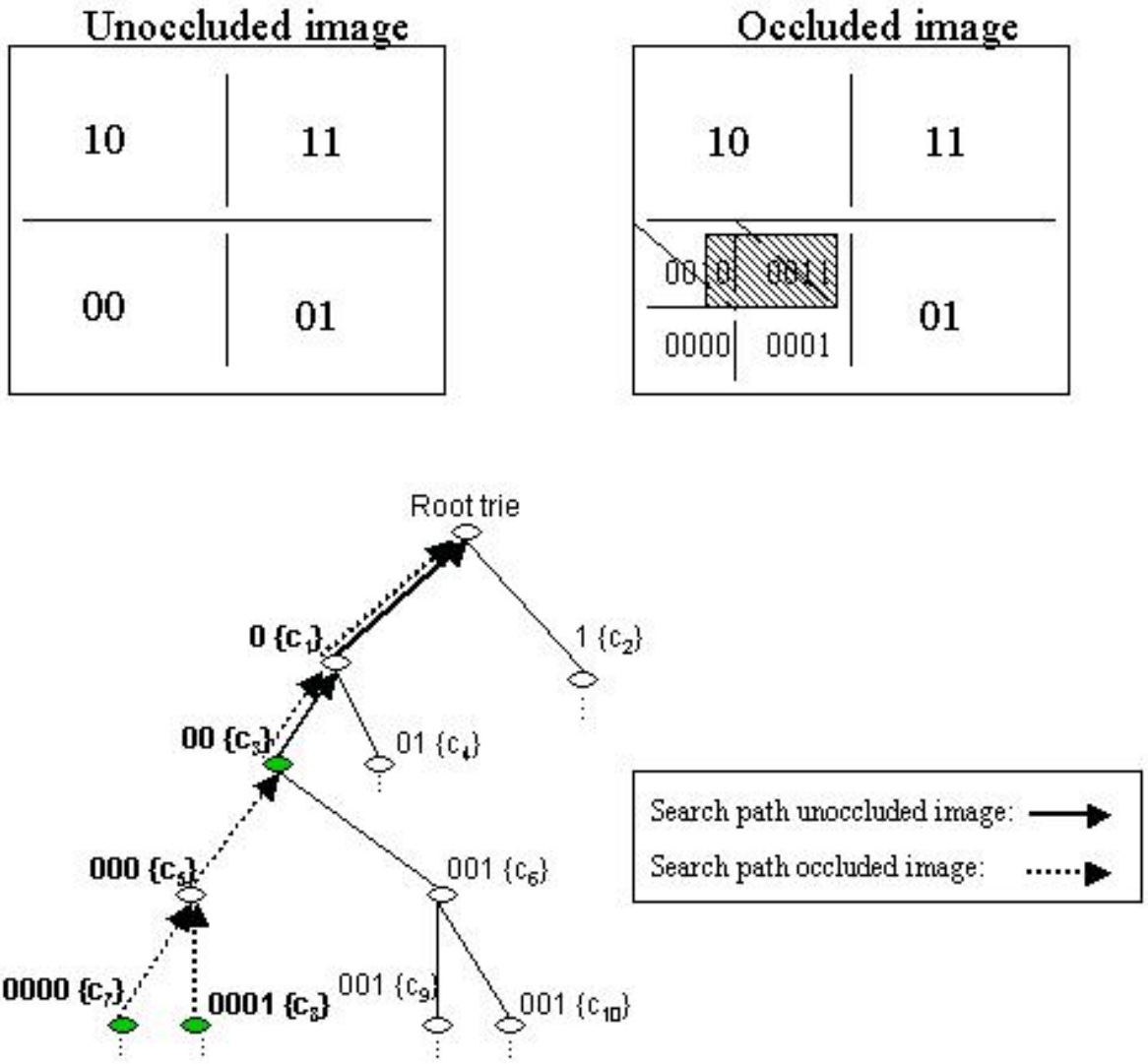


Figure 3.7: *Occlusion handling.* An occluded snapshot and the same image without occlusion. The branches of the trees are also shown. The two images are alike but the snapshot contains an occlusion. This has consequences for several patches and the structure of its corresponding tree. Case (a) and (b) show the difference in the branches that are caused by occlusion.

information on particle filters we refer to [11].



After searching	Collected candidates
patch 00 of unoccluded image	$\{c_1, c_3\}$
patch 0000 of occluded image	$\{c_1, c_3, c_5, c_7\}$
Patch 0001 of occluded image	$\{c_1, c_3, c_5, c_8\}$

Figure 3.8: *Occlusion handled by the trie approach.* An unoccluded image, the same image with occlusion and their path in the trie.

Chapter 4

Experiments

4.1 Test setup

The experiments are done on a database of 25 images. This test database is created by taking pictures at a 5 x 5 grid of the soccerfield as showed in Figure 4.1. These test images were taken with the same orientation. They all are facing the blue goal. These images were taken while the soccerfield was empty because they have to have a model of the world without any occlusion.

Next to those 25 images, also some test snapshots were taken. These are typical images that can occur during the game. To simulate occlusion of robots we took black plastic caps instead. Twelve of these snapshots are taken from different positions. Some were taken at an exact position as one of the 25 images taken beforehand, with and without occlusion to see whether there will be a difference in performance and how it differs. Others were taken from off-grid locations and with (slightly) different orientations.

4.2 Test Results

The first test was done with a snapshot taken from the center of the field facing the blue goal. This result in a match with the image that was also taken from that same place and with same orientation. Figure 4.2 shows the snapshot and its matching results. Because the snapshot was taken at the same position and orientation as image 2 it should be a perfect match with that image. Figure 4.2 shows that this is indeed the case.

The next one was a snapshot that was taken from the same location as the previous one but this time with occlusion. Here the occlusion does not really derange the matching process and results in a best match with the image taken from the same location but also selects another image from a different position as shown in Figure 4.3.

Also snapshots with differing orientations and off-the-grid locations were

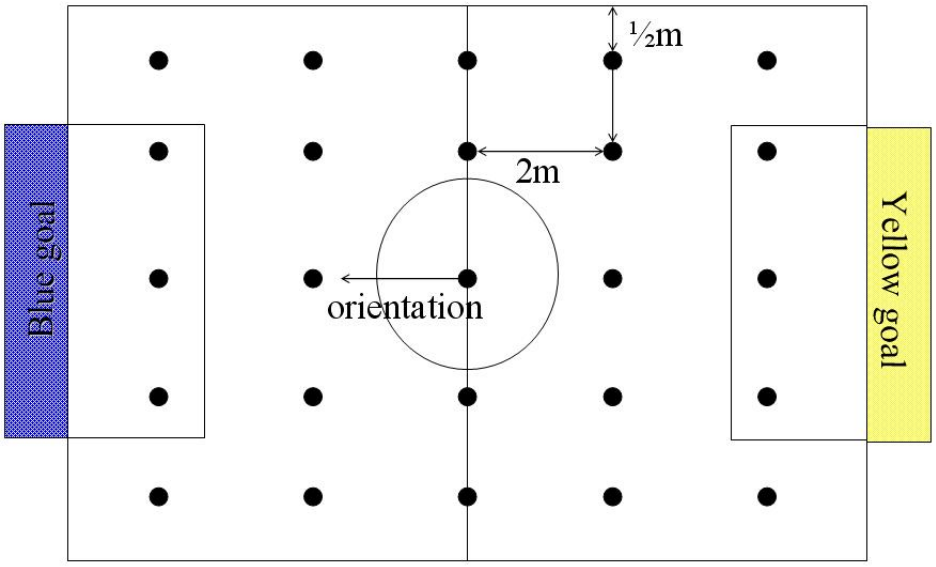


Figure 4.1: *Exploring the field.* How the test images are taken. They are taken on a 5x5 grid, 2 meters apart from each other and about half a meter from the sideboards. The orientation is towards the blue goal.

taken. However these sometimes had more trouble finding the right match. In Figure 4.4 a snapshot was taken with the same orientation near the location of image 16. The scoring, however, says that it could be more likely image 22, 18, 8 or 13 which are not located near the snapshot-location. However, if we include the first 5 (or more) hits, then image 21 is also included (with score 12) which is very near the correct location. Similarly, Figure 4.5 shows an example of an off-grid snapshot where the best match (image 22) is near the correct position.

From these preliminary experiments we conclude that our method seems to work well under occlusion. On the other hand, the method gives sometimes suboptimal results with off-grid locations and/or varying orientations. This might be due to the fact that we used a very simple and restricted environment—only 25 images on a field of 9x5m with fixed orientation.

Imagenr	Score
0	7
1	10
2	15
3	9
4	2
5	10
6	10
7	9
8	11
9	0
10	12
11	10
12	12
13	12
14	4
15	10
16	10
17	11
18	11
19	0
20	11
21	10
22	13
23	10
24	5

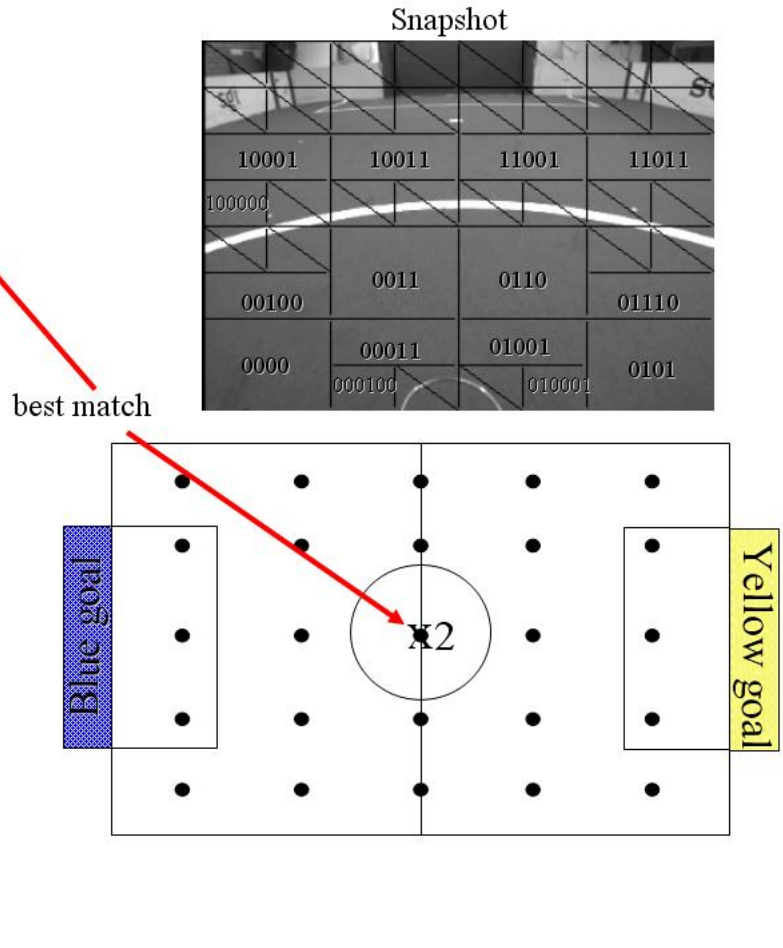


Figure 4.2: *Scores for snapshot without occlusion and taken from the middle of the field.* Position marked with x is the location where the snapshot was taken.

Imagenr	Score
0	6
1	11
2	14
3	9
4	2
5	10
6	11
7	8
8	11
9	0
10	11
11	11
12	12
13	12
14	4
15	9
16	10
17	9
18	11
19	0
20	13
21	11
22	14
23	11
24	5

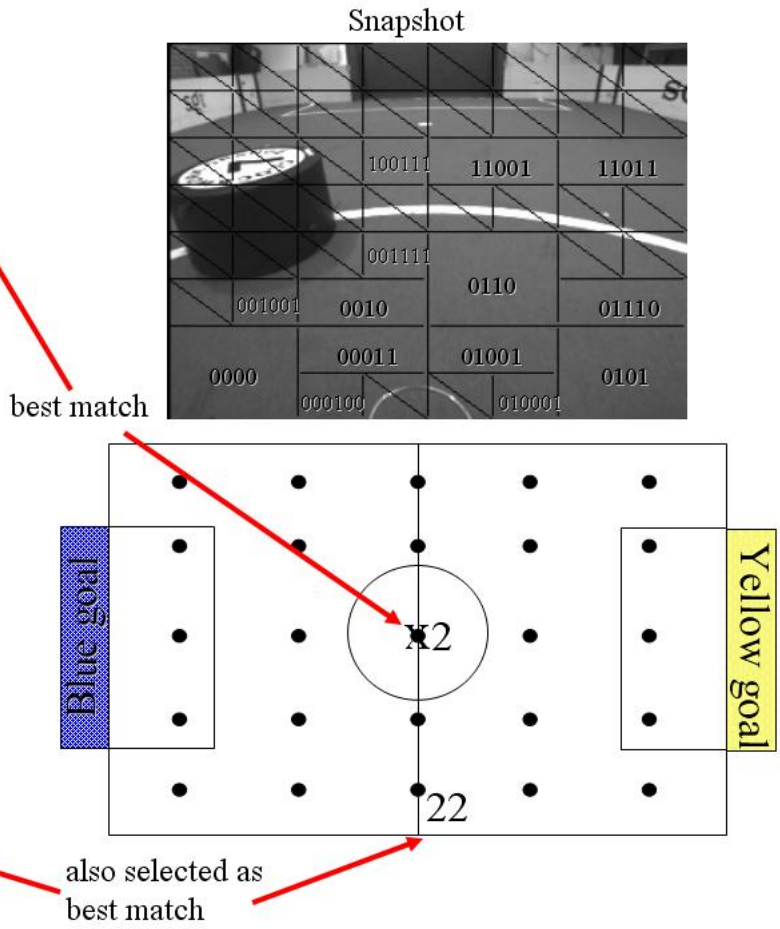


Figure 4.3: *Scores of snapshot with occlusion from the middle of the field.* Position marked with x is the location where the snapshot was taken.

Image	Score
0	7
1	10
2	8
3	11
4	2
5	10
6	10
7	8
8	13
9	0
10	12
11	11
12	12
13	13
14	5
15	10
16	9
17	8
18	13
19	0
20	11
21	12
22	13
23	12
24	3

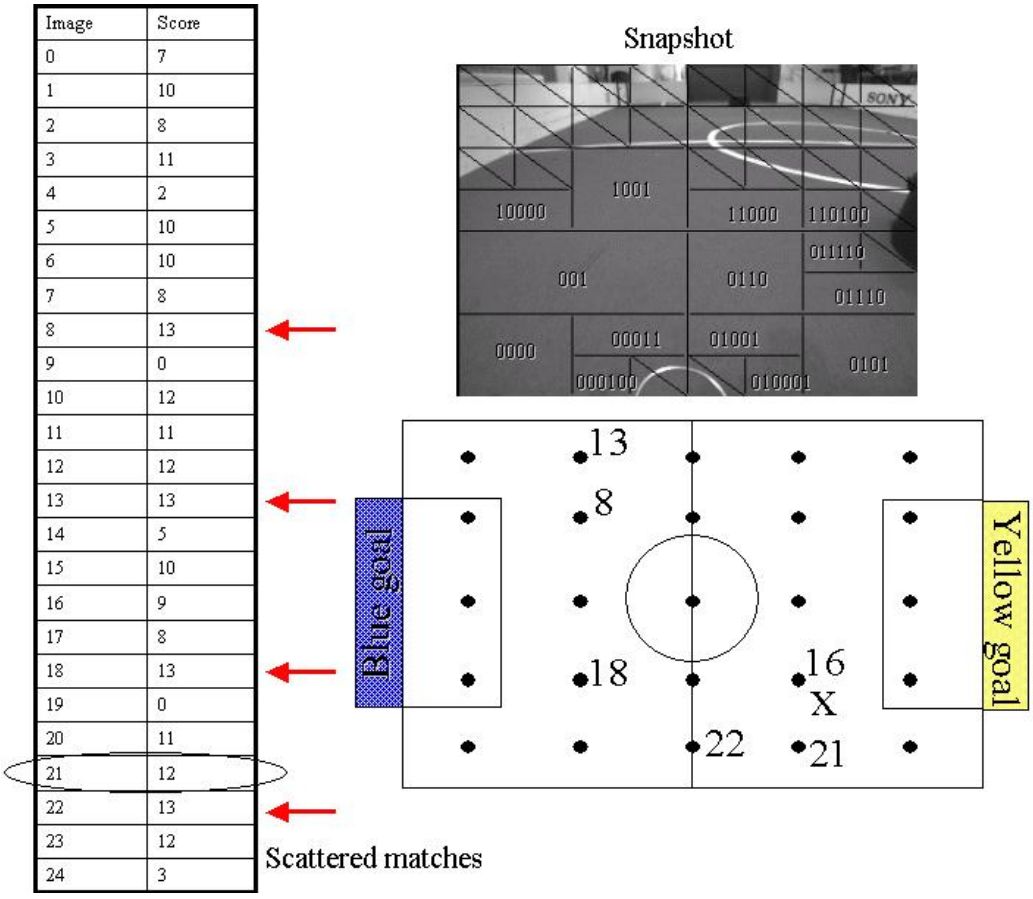


Figure 4.4: *Matching of off-grid taken snapshot.* Position marked with x is the location where the snapshot was taken.

Image	Score
0	8
1	11
2	11
3	10
4	3
5	11
6	10
7	10
8	12
9	0
10	13
11	10
12	12
13	13
14	6
15	11
16	10
17	11
18	12
19	0
20	12
21	11
22	14
23	11
24	3

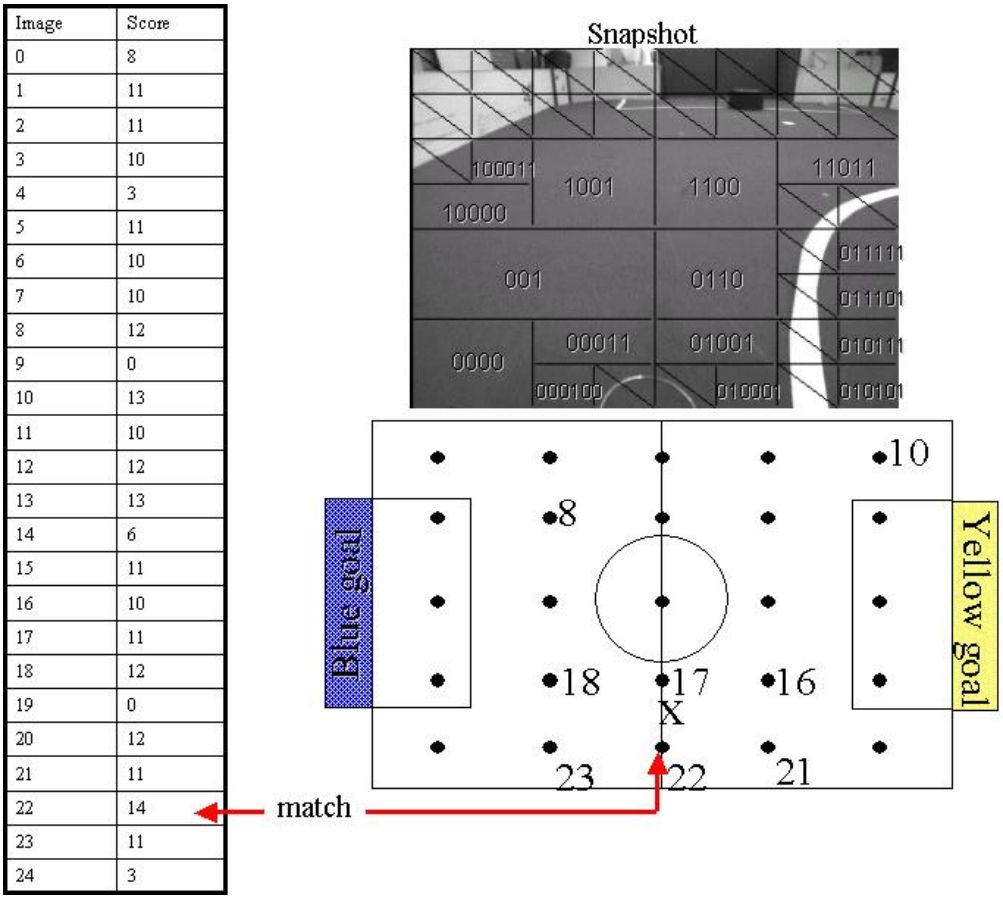


Figure 4.5: *Matching scores of the images after comparison with off-grid snapshot.* Position marked with x is the location where the snapshot was taken.

Chapter 5

Conclusions and Future Work

In this chapter the results of the experiment will be discussed. The strength and weakness of our method will be pointed out and a final conclusion will be drawn from them. Finally future work will be discussed, like how the proposed model can be improved to fix the minor points of it.

The aim was to develop a localization method that was quick and also robust enough against occlusion. To achieve this we used an appearance model as our reference. This appearance model was created by observations taken by the robot. This was done by taking pictures of its environment. Because these pictures contain high dimensional information which is hard to cope with, we needed to extract essential information that can be useful for the localization of the robots. The feature extraction method converts the high dimensional pictures to simple binary trees which represent the green in the field as it is structured in the picture. These trees are our reference of the world. These all are gathered in a trie which is basically a database structured as a binary tree.

When a robot during the game wants to know its position it takes a snapshot and searches the trie for an image that matches the best. The positions from where the first best matching images were taken will form the set of candidate locations of the robot. This set can be further pruned by using, e.g. a particle filter technique [11].

During the experiments it seemed that the method works quite well. The conditions were simplified but once there was an occlusion it was handled well. On the other hand, because of the simplified conditions (small amount of pictures and single orientation) suboptimal results were obtained when pictures were taken off-grid and/or had a deviating orientation. The off-grid problem could be solved by taking a larger grid, for example, a grid of 20x20. Finding the location could then be done more detailed.

As future work, it would be interesting to see whether our method can be

applied in other robot localization settings, for instance, for office robots. We believe that the proposed tree-based appearance model is general enough, and can be used in several other settings where ample texture is available (e.g., brick walls, tiled floors, etc.). Moreover, the tree-based matching operation by using a trie [5] allows for robustness against occlusion and other effects. More detailed experiments are however needed for measuring the sensitivity of the method against large occlusion and orientation mismatch.

Bibliography

[1] www.robocup.org.

[2] R. Bartelds. *Real-time vision-based self-localization.* PhD thesis, University of Amsterdam, April 2002.

[3] Frank de Jong, Jurjen Caarls, Robert Bartelds, and Pieter Jonker. A two-tiered approach to self-localization. In *Proceedings International Symposium (Seattle, USA, Aug.2-10)*, volume 2377, page 405410, 2002.

[4] David Forsyth and Jean Ponce. *Computer vision - a modern approach.* http://www.cs.berkeley.edu/ daf/book3chaps.html.

[5] D. E. Knuth. *The Art of Computer Programming*, volume 3, Sorting and Searching. Addison-Wesley, 2nd edition, 1998.

[6] B.J.A. Kröse, N. Vlassis, R. Bunschoten, and Y. Motomura. A probabilistic model for appearance-based robot localization. *Image and Vision Computing*, 19(6):381–391, April 2001.

[7] Ales Leonardis and Horst Bischof. Robust recognition using eigenimages. *Computer Vision and Image Understanding: CVIU*, 78(1):99–118, 2000.

[8] H. Murase and S. K. Nayar. Visual learning and recognition of 3-d objects from appearance. *Int. Journal of Computer Vision*, 14:5–24, 1995.

[9] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian C. Williams. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2):5–47, 1998.

[10] M. Spaan. Team play among soccer robots. Master's thesis, Informatics Institute, University of Amsterdam, 2002. www.science.uva.nl/research/ias.

[11] N. Vlassis, B. Terwijn, and B. Kröse. Auxiliary particle filter robot localization from high-dimensional sensor observations. In *Proc. IEEE*

Int. Conf. on Robotics and Automation, pages 7–12, Washington D.C., May 2002.