Interaction between Face Detection and Learning Tracking Systems for Autonomous Robots

Herke van Hoof

June 2011

Master Thesis

Artificial Intelligence
University of Groningen
the Netherlands

Supervisors

Dr. Marco Wiering
Artificial Intelligence, University of Groningen

Dr. Tijn van der Zant
Artificial Intelligence, University of Groningen;
Stem-cell and Brain Research Institute, INSERM, Lyon, France

Dr. Peter Ford Dominey
Stem-cell and Brain Research Institute, INSERM, Lyon, France

university of groningen

faculty of mathematics and natural sciences

Abstract

Domestic service robots are becoming increasingly available and affordable. However, interaction with the real world is still a challenge. Fast and reliable visual perception is one of these challenges. Systems that track the target over time are fast and can yield good results because they exploit temporal coherence, but they often need manual initialization.

In this study, such a tracking system is combined with a detector system that initializes and corrects it automatically. Different combinations of features are explored and evaluated on a dataset. The combined detection and tracking system is used to guide a mobile, autonomous robot in the real world.

Real-world data is used to continuously train the tracker system, to make it more robust to changes in illumination conditions and other environmental conditions. This robustness is necessary for the tracking system to handle the changing conditions in the real world with only limited feedback from an external system.

The combined detection and tracker system is able to run faster or perform better than the detector system in isolation, depending on the combined system's settings. Combination rules that adapt to the data helped to choose the best features depending on the conditions. The trained tracker performed better than the non-learning tracker on data gathered in the real world.

Keywords: Robot Vision, Machine Learning, Face Detection, Tracking, Multi-Classifier Systems, Adaptive Vision, On-line Learning, RoboCup@Home.

Contents

Chapter 1

Introduction

Domestic service robots are no longer just a topic for science fiction authors, as robots are becoming increasingly available and affordable. In 2006, Bill Gates predicted that the robotics industry could develop like the computer industry did thirty years ago, possibly leading to a robot in every home [1]. South Korea has even stated its ambition to give every household access to a robot by 2020 [2]. According to the International Federation of Robotics, already 5.6 million service robots for domestic use had been sold by the end of 2009 [3]. This all means that there is a growing application domain for domestic robotics, a research topic that is still relatively young.

Also within the scientific community interest in domestic robotics is increasing. A number of initiatives encourages development of domestic robots, such as the RoboCup@Home competition [4], the largest international competition for domestic service robots. This competition has a fast-growing community and on average a large increase in performance each year. Still, there are many challenges and open problems to be addressed.

One of the main difficulties of domestic robots is interaction with the real world [4]. In contrast to industrial robots, which often work on a standardized task in a standardized environment, domestic service robots have to autonomously perform a wide range of non-standardized tasks in a household, a non-standardized dynamic environment. As domestic robots operate in a non-standardized dynamic environment, reliably perceiving the world around them is a crucial ability. Perception has to be both real-time and robust for the robot to operate safely.

Robots can contain many types of sensory modalities, but one popular option is visual perception of the environment. Compared to the task of off-line computer vision, the problem of robot vision is harder in some aspects as the system has to process images real-time using limited on-board computing power. Furthermore, the camera of (humanoid) robots often has a limited resolution and quality. However, in some aspects the problem is also easier as subsequent images are related to each other and the robot can use the information from the past to make better decisions in the present.

1.1 Robot and computer vision

Computer vision for (domestic) robots should recognize people, objects, and landmarks in the robot's environment. One of the problems is that the target to be recognized can appear anywhere within the robot's camera image. To solve this problem, points in the image that are somehow of interest for targets to be recognized can be extracted, which might correspond to similar points extracted from a database of known objects. Another approach is to simply consider every possible window (over a range of locations and scales) in the image a candidate location for the target object. A third possibility is object tracking: narrowing down the possible locations of an object by its location in the past. These three approaches are described in more detail in the following sections.

1.1.1 Interest points description

One successful approach to visual processing is based on local descriptions of keypoints, such as used in the SIFT method described in [5, 6]. In this approach, the keypoints are the scale-space extrema in the difference-of-Gaussian function. These points correspond to places with contrast, such as edges or corners.

The local region around the keypoint is used to generate a distinctive representation. The size of this region depends on the scale at which the keypoint was found to provide scale invariance. This region is divided into 4x4 grid cells, and each of these grid cell is summarized by a 8-bin weighted gradient direction histogram, where pixels with a larger gradient magnitude or nearer to the keypoint have a larger weight. The gradient orientations are taken relative to the dominant gradient orientation of the local region, so the descriptor is orientation invariant. Furthermore, this descriptor is quite robust against contrast and intensity change, affine and 3D viewpoint changes, pixel noise and stretching.

Once these keypoints are extracted, their descriptors can be matched to those of keypoints extracted from images of known objects. If enough of these keypoints match close enough, and agree on the scale, rotation, and location of a target object, the object can be concluded to be present in the image.

A successful application of this method is described in [7], where SIFT keypoints are used to recognize cars. There are also other approaches that use interest points description, but use a different way to localize the keypoints and/or a different description of the keypoints. An overview and comparison of different interest points detection and description algorithms is given in [8, 9]. An application is described in [10], where keypoints were detected at places with circular configurations of edge points. Then, the SIFT descriptor was used to match the keypoints to a database of object parts. The detected object parts could be used successfully to detect the presence of complex objects.

The advantage of this class of computer vision algorithms is that they generally have a good performance, even when the size of the training set is small. Even on the basis of one example of the object, it can be recognized quite reliably in other

images. However, these SIFT-like approaches often find very many keypoints. The description of each found keypoint has to be compared to the descriptions of all keypoints of every object in the database. This is a relatively slow process as the descriptor has 128 dimensions. If the number of target objects in the database is large, this makes it impossible to match all keypoints in real-time.

1.1.2 Sliding window approach

In another type of approach, a window slides across the image and at each possible location features are extracted. Due to the many possible positions and scales of the window, this can only be done in real time if fast-to-compute features, such as Haar-like features, are used. However, these features are weak and many of them have to be combined to make a reliable detection. This approach was applied successfully using sets of Haar-like features in [11] to detect faces, people, and cars. In [12–14], Haar-like features were used as well to detect faces, but the set of features that were used was divided over multiple stages so that non-ambiguous windows could be rejected very early in the process, limiting the required computation time.

The authors of [15] used wavelet features instead of Haar features. They were able to successfully detect faces and cars. The authors of [16] used a slightly different approach, where the different components of the target object (a human body) were detected separately using Haar-like features under geometric constraints within the sliding window. In that last study, false positives could be rejected more easily if the required number of components was not found or they were not in the correct geometric relation to each other.

The sliding window approach is very popular because it has proven to be successful and it is relatively fast. Haar features can be used to recognize any type of objects as long as relative positions of dark and light regions in the appearance of the object stay the same. This means that Haar features have no rotational invariance and are not suitable to detect amorphous objects. However, a lot of objects in our daily lives have a fixed structure and are normally encountered in the same orientation.

The disadvantage of this approach is that it needs a large number of training examples. One publication mentions using 14460 original images [12] while another mentions using 4000 original images [17]. From these images, which contain positive and negative examples, a far larger dataset was constructed by transforming the images. Obtaining and labeling such numbers of images is still a job that has to be performed by a human, which is a labor-intensive process and limits the capacity of the system to learn autonomously. Furthermore, needing access to such a large database of images means this method is not suitable for real-time on-line learning.

1.1.3 Tracking

Another approach to visual processing is object tracking. In a tracking approach, the (approximate) location of the target in the past is known, so processing can be constrained to a small region of the image. This also allows weaker features to be used, speeding up the process even more. This means that in real-time conditions, tracking approaches are quite popular. Successful real-time tracking applications are mentioned in [18], where the ball of a RoboCup soccer competition was tracked and [19] where obstacles were tracked without identifying them.

Tracking is often done using particle filters. These approximate the probability distribution of the location of the target using a set of particles. To keep track of the target over time, the particles from the previous step that best approximate the target features are selected and random alterations of these particles are added to the set. One possible kind of particle filter that is used often for visual tracking is the condensation filter, which was introduced in [20] to track curves in clutter.

A problem in this approach was that the tracker still needs to be initialized by hand, which is of course a problem for application on an autonomous robot. A solution was proposed in [21], where the condensation filter was extended to allow automatic initialization using an object detection algorithm or by placing particles at locations where the target is likely to appear. Furthermore, the filter was changed to work on the basis of color features. Color features are simple to use, and are computationally inexpensive to calculate, allowing real-time tracking [22]. In contrast to curves, that have to be initialized very precisely to allow the tracker to follow the object, color features are more robust to an imprecise initialization. A tracker based on color features works better if it is initialized precisely, but if some of the background colors are included in the object model the tracker will try to find the correct mix of foreground and background colors, and so stays quite close to the target.

Color features however also have disadvantages: the apparent color depends on the illumination and the camera parameters. This problem can be solved by making the tracking adaptive to slow color changes [21, 22]. Color features might not always provide enough information for reliable tracking and detection. To improve this, color information can be combined with edge and texture information as was done in [23–25].

1.2 Combining multiple strategies

Summarizing, it is indeed possible to detect objects with a high accuracy. One of the possibilities is by using strong features such as used in the interest point based methods. However, these methods are not able to perform in real time. Another possibility is using an ensemble of weak features such as in the sliding window approach. This however has the disadvantages of still being on the slow side, and needing large amounts of labeled training data.

On the other hand, objects might be tracked. Trackers can function in real time, and do not need prior training. However, they have to be initialized and might slowly drift away from the correct target as they are usually somewhat less powerful than the approaches mentioned before.

A solution to this problem might be found by combining different kinds of classifiers. Slow, but powerful, detector systems can be used to detect the presence of different kinds of objects in the image. These detector systems can initialize trackers, and also provide feedback to those trackers at regular intervals. The trackers themselves are faster than the detector systems, and so are able to perform in real-time on modest hardware, but because they exploit the temporal coherence of subsequent video frames they are still able to maintain a high performance. This approach creates two visual processing routes running at different frequencies, that together might outperform systems based on just one of the approaches.

1.3 Research questions and thesis outline

The research question of the current study will be: What is the best way to create a combined system consisting of a powerful detector system and a fast tracker system, and how does such a system perform? This question will be split up in several subquestions. Each of these subquestions will require theoretical and empirical research, which will be described in a separate chapter for each subquestion.

First of all, is a basic version of such a combined system able to outperform the detector system used in isolation? The answer to this question will show whether the main research question is worth pursuing. So, it will be the focus of the next chapter of this thesis.

In the third chapter, the tracker subsystem will be investigated more closely: Which features optimize the performance of the tracker subsystem, and does combining features yield an improvement in performance? If so, what is the optimal method to combine these features?

Rather than just using the feedback from the detector to (re-)initialize the tracker, the tracker could try to learn from the feedback it receives so it does not make the same mistakes again. Learning from feedback is especially important if the combined tracking system is used to control an autonomous, mobile robot. In the fourth chapter, such a learning tracker will be developed and used to analyze sensor data of an autonomous robot in a non-standardized real-world setting. The main question of this chapter will be whether the learning tracker is able to outperform the non-learning tracker in this setting.

In the concluding chapter, after these three subquestions have been addressed, all results will be summarized and discussed.

Chapter 2

Temporal Coherence

To research how to improve a visual detection system by exploiting temporal coherence, in this chapter a powerful detector system will be connected to a real-time tracker. Both of these systems should have their own role in the combined system: somehow, the detector system should initialize and correct the tracker; and the tracker should find the target in frames where the detector cannot find it. This might be possible even though the tracker is less powerful than the detector, because the tracker is able to exploit the temporal coherence present in the world. For instance, objects in the world cannot suddenly disappear, so a detected target should be close to the current location in the next frame. Furthermore, a functioning tracker system might eliminate the necessity to run the detector system every frame. This might make the combined system faster.

This chapter will first discuss the advantages and disadvantages of tracking, to arrive at the research questions for this chapter. Then, in the methods section the implementation of the combined detector system will be described, as well as two alternative systems that are included for comparison. The methods section will also describe the dataset and the way the measurements are analyzed. The outcome of the experiment will be given in the results section, and finally a discussion of the results will be given.

2.1 Introduction to combined detection and tracking

Computer vision is a difficult problem, and many complex solutions have been proposed. However, the state of the art is not perfect. For instance, in face detection not all faces can be recognized, especially under difficult circumstances. Furthermore, many of the methods are so computationally intensive that they can not be used in a real-time setting.

2.1.1 Structure and tracking

The fact that the real world is structured can fortunately be exploited to alleviate those problems. The structure of the world allows the generation of predictions, which can be used to speed up perceptual processes and make them more reliable. Specifically, the temporal coherence can be exploited. This is what makes tracking in general easier than detection: with each new frame, the tracker already has a prediction about where to look. Confirming or slightly correcting the prediction is often easier and quicker than starting the detection process from scratch in every frame.

To track the visual targets, a particle filter can be used. This kind of filter represents the probability distribution of the location of the target using a limited set of discrete particles. Particle filters have been used successfully in visual tracking tasks: they have been used to track object-edges in clutter [20] and to track soccer balls [18] or faces [26] using wavelet features. Other studies successfully used color distribution [21,23] features or a combination of different features [24,25] to track both objects and people with a particle filter.

2.1.2 Disadvantages of tracking

There are also disadvantages to tracking approaches. The first problem is that trackers need to be initialized. This is sometimes done by hand in the first frame of a sequence, for instance in [22]. This solution is far from ideal, as it needs a lot of manual labor and does not enable the system to function autonomously. Autonomy is essential if the system is to be used as the vision system of an autonomous (service) robot. A better solution is to let a detection system initialize the tracker. This was the approach used in [26], where a tracker was periodically initialized by a more time-consuming detection step. However, that study did not focus on real-time performance (the detection step took 80 seconds, whereas the tracking step took 26 seconds).

Another problem of trackers is that if the conditions (such as illumination or pose) of the target change, the tracker might lose the target object. One solution to this problem is by making the tracker adapt itself to the slowly changing conditions. An example of this approach is the work described in [21]. However, this solution has a catch. If the tracker slowly drifts away from the target, the tracker might adapt itself to elements of the background. As the system receives no feedback, we cannot be sure whether the changes in the visual features are due to changing conditions or due to the tracker drifting away from the target.

2.1.3 Combined system

Another approach to handling the changing conditions is to run a detection system periodically to re-initialize the tracker. This makes sure the tracker does not drift away from the target, and secondly makes sure that the model of the target can be adapted according to the feedback provided by the detector system. This means

that two subsystems are used that run at different timescales. The powerful detection system runs at a low temporal frequency, but is able to initialize the tracker and provide it with feedback. The tracker is suitable for real-time processing, but needs the feedback to adapt to changing conditions and also because it might lose the target as it has a weaker representation of the target. Such a multi-system solution has another advantage: it is pretty straightforward to make each system run on a different machine. This means that a robot with limited processing power could just run the tracker, with initialization and corrections being provided by the detection subsystem running off-board on a different machine.

2.1.4 Target choice

The methods described up to now could be used to detect and track all different kinds of visual targets. In the current study, the targets to be localized will be faces. Face detection has already been studied, so prior research is available. Furthermore, some researchers have made their algorithms or software available, providing some components for a recognition system. Thirdly, there are some databases available to test face localization software, meaning new techniques can be tested on an accessible benchmark.

2.1.5 Research questions

The main question that will be addressed in this chapter is whether the accuracy and speed of the detection system can be increased by exploiting the temporal coherence present in video sequences. Furthermore, this study will address whether the proposed particle filter for visual tracking performs better than a simple memory system that only remembers the last detection. It is expected that both the memory system and the particle filter will increase the detection accuracy because they exploit the temporal structure present in the video sequences. Furthermore, both of them will enable the detection system to run at a lower frequency, thereby speeding up the processing of the video data. The particle filter will, especially if the detector subsystem runs at a low frequency, expectedly outperform the memory system as it uses the visual features present in the video sequences to locate the target.

2.2 Methods for tracking and hybrid systems

2.2.1 Detector system

Boosted cascade of Haar-like features The basic detector system is a face detector that processes every frame of the video independently. This detector employs a boosted cascade of Haar-like features, as first described in [12]. This approach uses quite simple features: the difference of the summed intensity in two rectangles or two pairs of rectangles. The integral image representation allows these features to be calculated very quickly. This calculation is described in detail in [12]. A

combination of weak classifiers, each based only on one selected feature from an over-complete set, is used to classify the images as either containing a face or no face. Multiple of such combined classifiers were used to construct an 'attentional cascade' of increasingly complex classifiers, where small, efficient classifiers are used to reject many possible sub-windows, so that only the more ambiguous or difficult to classify sub-windows are processed by more complex, time-consuming classifiers.

In [14], the authors describe an improvement of this system. First of all, the authors extended the feature set with rotated features and center-surround features. Furthermore, fast contrast stretching was added to compensate for illumination changes. Lastly, rather than tuning the combined classifier of a certain stage for a certain recall score by simply changing its overall threshold, the classifier was actually post-optimized for that specific recall score by changing the thresholds of its constituent weak classifiers. This achieves the same recall score, but keeps the amount of false positives lower.

Implementation of the detector system This improved detector is implemented in the OpenCV library [27], which also contains cascades pre-trained on several datasets. In this research, the 'default frontal face' cascade is used. The cascade usually detects the face in multiple overlapping candidate locations. The detector combines all of these neighbors into a single detection. It is possible to make the system more reliable by requiring a minimum number of neighbors for a successful detection. This minimum is one of the parameters of the OpenCV implementation. Another is the scale factor, which decides how close the different scales are to each other in scale space. Furthermore, the 'do Canny pruning' flag can be set which rejects possible windows if they do not contain enough, or contain too many edges. The parameters that are used in the current research are OpenCV's suggested parameters for still images (scale_factor = 1.1, min_neighbors = 3), and the suggested settings for images from a video stream (less robust, but faster: scale_factor = 1.2, min_neighbors = 2, with the 'do Canny pruning' flag set to speed up the detector).

This detector is applied to every frame of the video. The frame is first down sampled by a factor 1.4 to speed up the processing. If the detector finds a face, its location is compared to the ground truth of the video. If in the same frame more than one face is detected, the detection which has the most neighbors is used by the detector system.

2.2.2 Combined system

The combined system consists of the face detector system, as described above, augmented by a tracker to exploit the temporal coherence of the videos.

Particle filters A tracker tries to estimate the probability distribution over the state at time t, $p(\vec{x_t})$, based on the history of observations $Z_t = \{z_1, \ldots, z_t\}$. Using

Bayes' theorem, this can be written as in equation 2.1. Because $p(z_t|Z_{t-1})^{-1}$ is independent of x_t, this can be written as a normalization factor k_t. Furthermore, if the Markov assumption holds z_t is independent of Z_{t-1} given x_t. So the equation can be simplified to equation 2.2.

$$p(\vec{x_t}|Z_t) = p(\vec{x_t}|\vec{z_t}, Z_{t-1}) = \frac{p(\vec{z_t}|\vec{x_t}, Z_{t-1})p(\vec{x_t}|Z_{t-1})}{p(z_t|Z_{t-1})} \tag{2.1}$$

$$= k_t p(\vec{z_t}|\vec{x_t})p(\vec{x_t}|Z_{t-1}) \tag{2.2}$$

By marginalization over x_{t-1}, an expression for $p(\vec{x_t}|Z_{t-1})$ can be derived, as shown in equation 2.3. Using the Markov assumption that $\vec{x_t}$ is independent of Z_{t-1} given $\vec{x_{t-1}}$, equation 2.5 is derived.

$$p(\vec{x_t}|Z_{t-1}) = \int_{x_{t-1}} p(\vec{x_t}, \vec{x_{t-1}}|Z_{t-1}) \tag{2.3}$$

$$= \int_{x_{t-1}} p(\vec{x_t}|\vec{x_{t-1}}, Z_{t-1})p(\vec{x_{t-1}}|Z_{t-1}) \tag{2.4}$$

$$= \int_{x_{t-1}} p(\vec{x_t}|\vec{x_{t-1}})p(\vec{x_{t-1}}|Z_{t-1}) \tag{2.5}$$

Because the observation density $p(\vec{z_t}|\vec{x_t})$ is generally non-Gaussian, the evolving state density $p(\vec{x_t}|Z_t)$ is also generally non-Gaussian [20]. This means the state-density has to be approximated. One possible way to do this is via factored sampling. This means the state-density at time $t-1$ is represented by a sample set (2.6) of particles s and their associated weights π.

$$p(\vec{x_{t-1}}|Z_{t-1}) \approx \left\{ \left(\vec{s_{t-1}^{(n)}}, \pi_{t-1}^{(n)} \right), n = 1, \ldots, N \right\} \tag{2.6}$$

To keep the sample set at fixed size N, first a sampling has to be performed where N samples are chosen from $\{s_{t-1}^{(n)}\}$ with probability $\pi_{t-1}^{(n)}$. Each of these samples is then propagated according to the motion model $p(\vec{x_t}|\vec{x_{t-1}})$. Now, a sample set is obtained that approximates $p(\vec{x_t}|Z_{t-1})$ from equations 2.3-2.5. Then, the observation step can be applied. Similar to equation 2.2, every particle is weighted using the observation probability $p(\vec{z_t}|\vec{x_t})$, after which the weights are normalized to make sure the sum of all weights is 1. This final set is an approximation of the state-density at time t: $p(\vec{x_t}|Z_t)$.

Each of these three steps (selection, propagation, and observation) can be applied differently in different trackers. The next paragraphs describe the implementation of these steps in the current research.

The sample set and the selection step The number of particles N in the filter is an important parameter for the tracker. Using a too low number of particles means there might be no particle near the actual location of the face, and so the face is lost to the filter. However, using too many particles bogs down the system and makes the tracker too slow. 100 particles were chosen as performance hardly improved further if more particles were added, while the evaluation of 100 particles is still fast enough.

The particles represent the $(x, y, width, height)$ parameters of a rectangle. This four-dimensional representation is simpler than the seven-dimensional representation in [21], where also the speed of parameter change was modeled. However, if the dimensionality becomes larger, more particles are needed to cover the search space which makes the entire system slower. This argument also applies to the choice to represent the particles by rectangles: more complex shapes, such as the ellipses in [21], require more computations to be evaluated.

In the selection step, N samples are drawn with replacement, by choosing a particular sample with a probability proportional to its weight. It is certainly possible that a sample with a high weight is chosen multiple times, or that a sample with a low weight is not chosen at all. Finally, the state of the object is estimated by the weighted mean of all particles, as shown in equation 2.7.

$$E[S] = \sum_{n=1}^{N} \pi^{(n)} s^{(n)} \tag{2.7}$$

Propagation step The second step is propagation of every particle. As discussed above, the speed of particles is not represented: a simple 0^{th} order model is used, to keep the dimensionality of the search space low. This means each particle is updated with a Gaussian component, so that $\vec{s}_t = \vec{s}_{t-1} + \vec{w}$, where $\vec{w}$ is normally distributed with a mean of $\vec{0}$, as there is no expected scaling or translation. The variance of the distribution should be large enough to account for movement of the face, yet too large values would make the probability of finding particles that correspond to the actual face location small. For the x and y components the standard deviation was set to 5 pixels, while it was set to 2 pixels for the height and width, as the apparent scale of the face (related to the distance to the camera) does not change as fast as the location of the face in the image in the used datasets.

Observation step Color features are quick to evaluate and can be used successfully to track people or objects, as was done in [21]. For this reason, like in that study, the weight of the particles will be based on the Bhattacharyya distance between the $8 \times 8 \times 8$ dimensional RGB histograms of the initial face model and the window described by the particle.

To start with, the color distribution of each particle n, $(p_{s_t^{(n)}})$ is observed. This distribution is compared to the initial color distribution q using the Bhattacharyya

distance shown in equation 2.8, where $q^{(u)}$ indicates the value of bin u of distribution q. Using these distances the weight of the particles is calculated as shown in equation 2.9. This means particles with a small distance to the original model are given a large weight. With high σ, there is not much difference in the weight of particles with a small and particles with a large distance to the model. However, with a very low σ, all the weight centers around a couple of particles with a small distance, which means the face might be lost if it appears in a slightly different color. In the experiments a value in the middle of these extremes of $\sigma = 0.1$ was used.

$$d(p_{s_t^{(n)}}, q) = \sqrt{1 - \sum_{u=1}^{m} \sqrt{p_{s_t^{(n)}}^{(u)} q^{(u)}}} \tag{2.8}$$

$$\pi_t^{(n)} \propto \frac{1}{\sqrt{2\pi}\sigma} \exp - \frac{d^2(p_{s_t^{(n)}}, q)}{2\sigma^2} = \frac{1}{\sqrt{2\pi}\sigma} \exp - \frac{1 - \sum_{u=1}^{m} \sqrt{p_{s_t^{(n)}}^{(u)} q^{(u)}}}{2\sigma^2} \tag{2.9}$$

Combining the systems In the combined system, the detector subsystem is run once every n frames. This detector subsystem is the detector system using either the 'video' or the 'still image' settings. In case the detector subsystems finds a face, the tracker model histogram of the tracker is set to the histogram of the detected region, and all particles are initialized at the location of the detection with uniform weight. If the tracker was already tracking, this still happens, so all particles are moved to the detected location and the model histogram is reset. This detected location is also compared to the ground truth. In all other frames, or if the detector does not successfully find a face, the face location as estimated by the particle filter is compared to the ground truth to measure the performance of the combined system.

If $n = 1$, the detector is run in every frame like in the detector system, but if there is no detection the tracker tries to keep track of the target since the last time it was seen. In this case, the detection quality is expected to be at least as high as when running the detector in isolation, as the tracker only runs when the detector system detects nothing. If $n > 1$, the detector is not run at every time step. In this case, it is expected that the detection time goes down, as the tracker (which evaluates 100 particles) is faster than the detector (which evaluates many more possible sub-windows). The larger n gets, the larger the probability that the tracker starts to lose its target, and so the detection quality is expected to go slightly down.

In the current experiment, for n the values of 1, 5, 10, 50, and 100 were considered. $n = 1$ is expected to give the highest detection score, $n = 5$ is the number of frames used in [26] to update the tracker with new face information,

and $n = 10$, $n = 50$ and $n = 100$ are used to show how the system performs when it is pushed to its limit. Furthermore, the larger values of n are expected to yield the largest speed increase.

2.2.3 Memory system

In order to measure the relative performance of the combined detection and tracking system, it is compared to a much simpler system that also tries to keep track of the location of the target over time. Rather than refining the distribution of the distribution step using visual features of the potential face location, this system skips the observation step and weighs all particles evenly. This means the system is retaining the information from the detection step, but between detection steps the system is not using the visual information from the videos. This means the particles will stay centered around the location where the target was last detected. Because of this, it will be called the memory system.

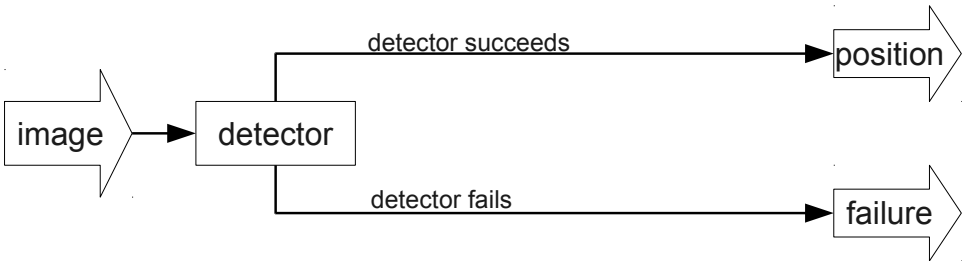
2.2.4 Differences between the systems

The combined system will be compared to the detector system, which does not exploit temporal coherence, and the memory system which does exploit the coherence between the frames by storing the last detection, but does not update the location of the target in the past frame using visual information. The combined system exploits temporal coherence using visual information, which makes it the most complex system. All three systems are presented schematically in figure 2.1, which clearly shows the differences between the systems.

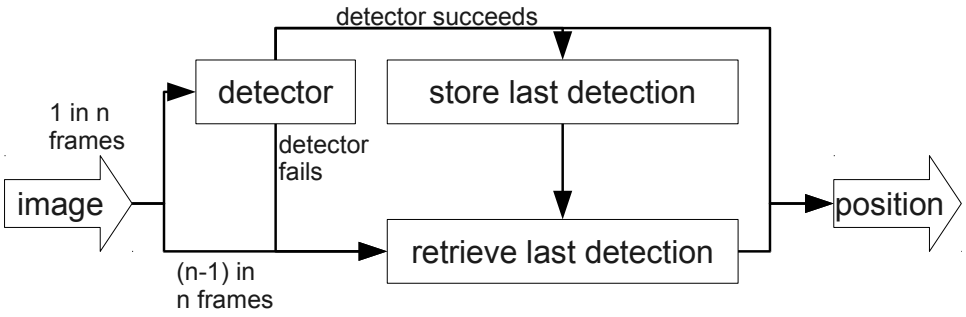
2.2.5 Dataset

Original dataset The systems were tested on the UO Face Video Database [28]. This database contains 21 video sequences, out of which 13 are manually annotated with the true location of faces in each frame. These 13 sequences are used to test the performance of the systems. The sequences are shot using different cameras under different, realistic lighting conditions such as walking indoors under different kinds of lamps, or driving a car through light and shadowy spots. The changing conditions make the dataset a hard challenge for detector systems and trackers. Sample frames from three sequences illustrating these conditions are shown in figure 2.2. In these sequences, the different subjects are looking in the direction of the camera.

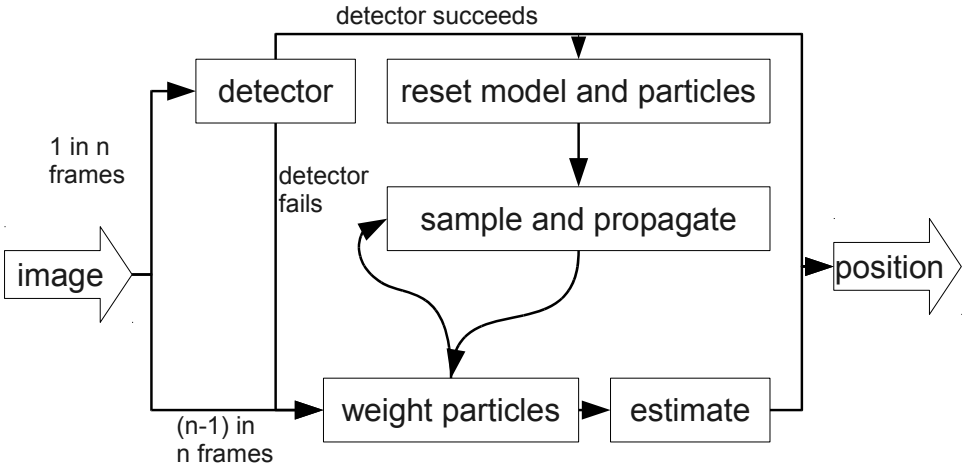
Increasing the challenge A possible problem with the dataset is that in many of the videos, the face only moves slightly and so it stays very close to its original position. This could make the task for the tracker almost trivial. A possible remedy for this problem is to add artificial motion to the dataset, by translating every image a certain distance with respect to the previous image in the sequence. An operation



(a) The detector system



(b) The memory system



(c) The combined system

Figure 2.1: The combined system and the systems it will be compared to.

(a) frame 1 (b) frame 242 (c) frame 436

(d) frame 1 (e) frame 60 (f) frame 222

(g) frame 1 (h) frame 60 (i) frame 300

Figure 2.2: Stills from the UO Face Video Database sequences 'almov3' (a)-(c), 'almov11' (d)-(f) and 'nomov3' (g)-(i). Sequences are shot with multiple cameras and can contain drastic variation in pose, illumination, or background.

that has the same effect, but is easier to implement and requires no computationally expensive image processing operations is to instead translate the tracker window in the opposite direction over the same distance. If this solution is applied, the particles are shifted a distance of three pixels between every two frames to the right or to the left. The direction of this translation is decided by chance at the beginning of each video, and stays the same during that entire video. No further vertical translation or scale change is applied to the video or tracker window.

2.2.6 Measurement and analysis

Performance measure In each of the frames in the video, the location of the face as found by the system is compared to the ground truth of the face location. The performance measure suggested in [28] is used, which is shown in equation 2.10, where $A_{ground\ truth}$ is the area of the ground truth box, $A_{detection}$ is the area of the box found by the tracking system, and $A_{overlap}$ is the area of the overlap between those two boxes. If the face is not found at all, the score for the current frame is 0.

$$Score = \frac{A_{overlap}}{\sqrt{A_{ground\ truth} \times A_{detection}}} \tag{2.10}$$

In addition to this measure of the system performance, the speed of the system is measured. This is done by summing the total time that is actually needed by the

vision system, excluding the time needed to load the frames and to calculate the system performance.

Data analysis The videos are processed using the different procedures and parameter settings described above. The required time and the cumulative scores for the videos are calculated using the described performance measures. By dividing these measures by the number of frames in the video, the average time-per-frame and score-per-frame are calculated.

The number of sequences in which the average performance was higher for the combined system, or the number of sequences in which the combined system was faster, can be compared to the expected value under the null hypothesis that the systems have equal performance or are equally fast. Under this hypothesis, the expected value is half of the 13 sequences: 6.5. The binomial hypothesis test [29] can be used to find the two-sided probability (P) values of finding a more extreme number of sequences than the observed number under the null hypothesis. If this probability is smaller than $\alpha = 0.05$, there is enough evidence to reject the null hypothesis.

2.3 Comparative results of the combined system

Initial results The results that were obtained on the original dataset are shown in figure 2.3 for the three different systems. From the figure, it is visible that both combined tracker systems obtained higher performances than the two detector systems when n is low enough. However, even the memory system, that does not use any information from the actual videos, did quite well. The difference between the tracker system and the memory system was quite small. The memory system was able to get a high performance because of the rareness of significant movement in the dataset videos. This made the task very easy.

The more challenging dataset The dataset augmented with artificial movement should be a more significant challenge for the tracker system. Performance statistics of the detector and tracker systems are given in tables 2.1 and 2.2. These tables show the average score-per-frame and time-per-frame of all three systems.

The detector system with video settings is very fast, but attains a low detection score. Because of this, the detector system with still image settings is a more useful standard of comparison. So the tables show the number of sequences in which the combined system with 'video' ($\mathrm{Comb_V}$) or 'still image' ($\mathrm{Comb_S}$) settings outperformed the memory system ($\mathrm{Mem_V}$, $\mathrm{Mem_S}$) or the detector system with still image settings ($\mathrm{Det_S}$). The tables also show P values for a two-sided binomial test with an expected proportion of 0.5 and a sample size of 13. Values of 'n.s.' in these columns mean the combined system did not outperform the system it is compared with in a number of sequences significantly higher than the expected number of sequences (at the $\alpha = 0.05$ significance level).

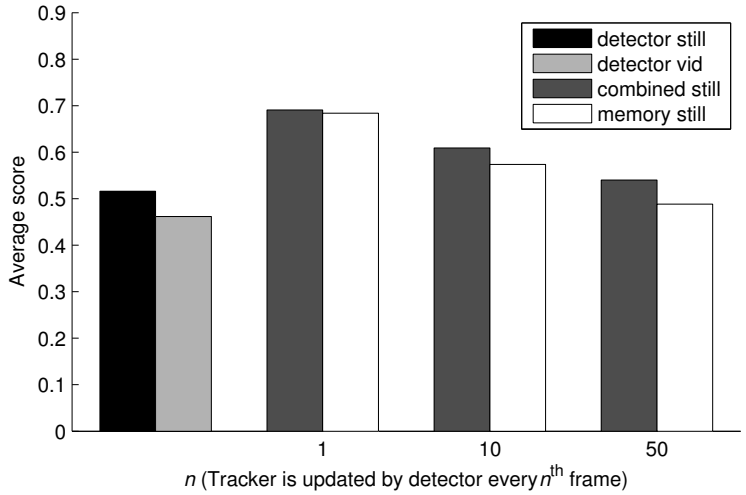


Figure 2.3: Performance score on the original dataset. Data is shown for the memory systems as well as the combined system. These systems are compared to the face detector from the OpenCV library using recommended video ('vid') or still image ('still') settings. The memory system and the combined system were updated every n^{th} frame. The reported performance score is a measure of overlap between the ground truth and the detection, as explained in section 2.2.6.

	Scores					
Det_S	0.52					
Det_V	0.46					
n	Scores		$\text{Comb}_\text{S} > \text{Mem}_\text{S}$		$\text{Comb}_\text{S} > \text{Det}_\text{S}$	
	Comb_S	Mem_S	no.	P	no.	P
1	0.67	0.64	8	n.s.	12	< 0.005
5	0.62	0.52	12	< 0.005	9	n.s.
10	0.61	0.48	12	< 0.005	10	n.s.
50	0.53	0.23	13	< 0.0005	6	n.s.
100	0.45	0.24	10	n.s.	5	n.s.
n	Scores		$\text{Comb}_\text{V} > \text{Mem}_\text{V}$		$\text{Comb}_\text{V} > \text{Det}_\text{S}$	
	Comb_V	Mem_V	no.	P	no.	P
1	0.65	0.60	6	n.s.	11	< 0.05
5	0.58	0.58	7	n.s.	8	n.s.
10	0.58	0.52	9	n.s.	8	n.s.
50	0.48	0.25	12	< 0.005	4	n.s.
100	0.39	0.22	10	n.s.	4	n.s.

Table 2.1: Average scores per frame for the systems under comparison, as well as the number of sequences in which the combined system worked better and P values for a two-sided binomial test. The reported performance score is a measure of overlap between the ground truth and the detection, as explained in section 2.2.6.

	Time (ms)		
Detector_S	14.7		
Detector_V	8.1		
	Combined_S	$\text{Combined}_\text{S} < \text{Detector}_\text{S}$	
		no.	P
$n = 1$	18.4	1	n.s.
$n = 5$	13.2	9	n.s.
$n = 10$	12.4	10	n.s.
$n = 50$	11.3	11	< 0.05
$n = 100$	10.2	12	< 0.005
	Combined_V	$\text{Combined}_\text{V} < \text{Detector}_\text{S}$	
		no.	P
$n = 1$	12.6	10	n.s.
$n = 5$	11.5	12	< 0.005
$n = 10$	11.2	12	< 0.005
$n = 50$	10.2	12	< 0.005
$n = 100$	8.5	12	< 0.005

Table 2.2: Average processing times per frame for the systems under comparison, as well as well as the number of sequences in which the combined system worked better and P values for a two-sided binomial test.

Figure 2.4 compares a subset of the scores attained by the different systems. On this dataset, the memory system performed significantly worse than the tracker system in a significant number of sequences when the systems are updated by the detection system every 5, 10, or 50 frames for the recommended still image settings, and when the systems are updated every 50 frames for the video settings. Furthermore, the combined system using either of the recommended settings outperforms the detector system when it is corrected every frame.

Figure 2.5 shows for a selection of systems how much processing time they needed, plotted against their score. The detector using the video settings was very fast, but did not attain a very high score. The detector using the still image settings is a more useful standard of comparison. This detector attained a reasonable score, but the combined system with still image settings was significantly faster when it was updated by the detector system every 50 frames or less often.

2.4 Discussion of the combined system's performance

A real-time particle filter was connected to a face detection system to create a combined system. This system was tested on an accessible dataset of 13 videos with a ground truth. Using this dataset, the combined system was compared to two other systems, the face detector from the OpenCV library (based on the Viola and Jones face detector) and the memory system (a simplified version of the combined

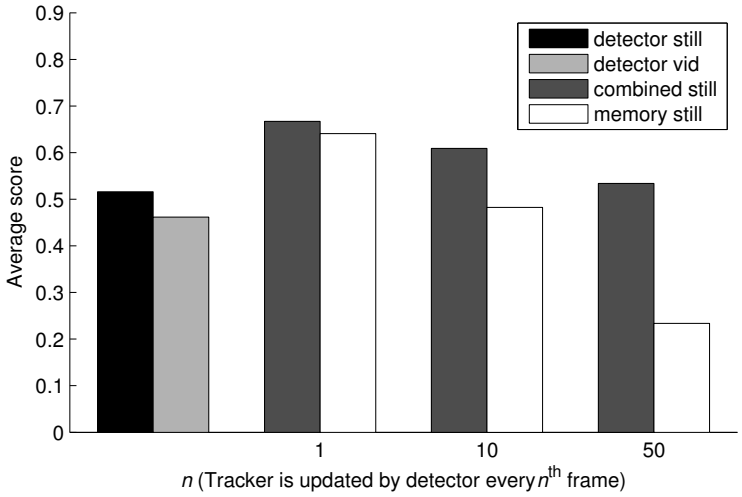


Figure 2.4: Performance score on the dataset with artificially added movement. Data is shown for the memory systems as well as the combined system with still image settings at selected values of n (these systems were updated every n^{th} frame). These systems are compared to the face detector from the OpenCV library using recommended video ('vid') or still image ('still') settings. The reported performance score is a measure of overlap between the ground truth and the detection, as explained in section 2.2.6.

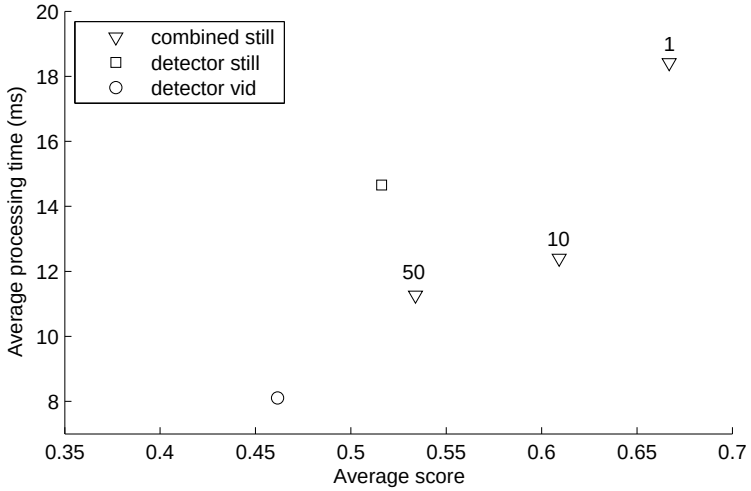


Figure 2.5: Required processing time plotted against system performance score. Data is shown for the combined system using still image settings at selected values of n (the combined system was updated every n^{th} frame). Data points for the combined systems are labeled with n. The reported performance score is a measure of overlap between the ground truth and the detection, as explained in section 2.2.6.

system). With certain parameter settings, the face location estimated by the tracker system was closer to the ground truth than the face location estimated by the detector-only system.

In contrast to the memory system, the combined system was even able to handle an increased challenge, when artificial movement was added to the dataset. By changing n, the number of frames between two runs of the detector subsystem, it could be tuned for a higher speed or higher precision. For instance, the combined system could be tuned to run faster than the detector system if still image settings were used, while retaining a similar (numerically slightly higher) score. Regardless of parameters, the combined system was able to process images in real time on modest hardware (processing between 50 and 100 frames per second).

Harder problems

The speed improvement offered by the tracker was quite modest, but that is related to the straightforwardness of the current task. In a more open-ended setting, where the system would have to be able to detect not only faces, but also landmarks and objects in the environment, the detector subsystem would have to be more complex, considering models of all possible objects to be recognized rather than just the face model. In contrast, the tracker subsystem would not consider all possible objects, but just the objects that are present according to the detector subsystem. This means that whereas the detector subsystem will expectedly be a lot slower in such a complicated task, the tracker subsystem is expected to be almost as fast as in the task described in this chapter.

For targets other than faces, a different detection system might be necessary. The face detector from the OpenCV library is not only good, but also fast compared to other detectors. Better quality object detectors exist, but they are usually much slower. So when one of those systems would be used, the difference in speed between the combined system and the detector system would be larger.

Parameter optimization

The described tracker used just one set of features, the pixel values in RGB color space. Other features, or combinations of features, could be used. Also, the settings of a number of tracker parameters, like the number of particles in the filter, were not optimized extensively. Optimizing the features used by the tracker and its parameters are likely to improve the performance of the tracker even more.

Dataset

Between some conditions, promising numerical results could not be shown to be statistically significant. This might be caused by the size of the dataset: with only 13 trials differences between the conditions have to be quite big to be significant. Another cause is the wide variation between the trials, as circumstances in the

videos varied wildly. However, varying circumstances are a given in the real world, so in that respect this variation of conditions in the dataset is good, as this makes it more likely that a system that can handle the dataset can handle the real world as well.

A slight problem was the relative rareness of movement of the face in the dataset. Practically, this could in part be solved by adding artificial movement. For further research, it would be preferable to have a dataset with more prevalent natural movement as a benchmark for visual tracking algorithms.

Chapter 3

Multi-Feature System

In the previous chapter, a color-based description of the input image to the computer vision system was used to track visual targets in contiguous frames of a video stream. However, there are more kinds of visual features possible to describe such an input image. These might be color-based descriptions in different color spaces, or they might focus on other properties of the image such as intensity, edges, gradients, or spatial lay-out. These features are based on different representations of the visual input, and might offer complementary information about the visual scene. If that is indeed the case, combining these features might improve the performance of the visual system. This chapter will address how such information sources can be combined, and whether combined features outperform the best single feature.

3.1 Introduction to multi-feature systems

In any realistic setting, sooner or later every pattern recognition system will make mistakes. However, systems based on different representations of the data might fail on different patterns. If different representations offer complementary information about the pattern to be classified, these can be combined to improve the performance of the system.

One way to approach this is to construct one large classifier which uses all available representations to make a decision about the input data. However, this means the dimensionality of the problem representation will become very large, meaning a lot of training data is needed to train the classifier reliably. This of course also increases training time.

Another approach is to train separate smaller classifiers, each operating on a single representation, and combining their output in a second stage of the classification process. In such a multiple-feature set-up a higher robustness and performance can be obtained [30]. Other advantages of such a set-up are that the independent small classifiers can be trained and evaluated in parallel, which makes real-time evaluation easier. Furthermore, this approach is extensible. New classifiers can

be trained and added to the system without having to retrain the entire system. These classifiers can of course be created by different people, making it easy to collaborate to improve the system and taking full advantage of the different methodologies used to train and design these subsystems. Last of all, new classifiers could even be generated automatically, for instance by combining algorithms, features, and functions in new ways. Such a generative approach could uncover combinations a human designer would not have thought of, but that might turn out to be very successful. A good multi-classifier system should be able to appreciate the contribution of the different constituent classifiers, and combine their output in such a way as to perform at least as good as the best single classifier.

In the literature on multiple-classifier systems there are two basic types of solutions: static combination rules and learning multiple classifier systems. Static classification rules are not based on a set of training data, but rather they are heuristics that have proven to yield acceptable results in a lot of cases. Sometimes such rules are simplifications of the joint probability density function. These rules make statistically optimal decisions if certain assumptions hold. These kinds of rules will be discussed in the next section.

If nothing is known about a dataset, it cannot be guaranteed that the assumptions of the commonly-used heuristics will hold. It might be safer to use a combination rule that actually adapts itself to the training data. If the assumptions do not hold, trained combiners often yield superior results [31]. Adaptive systems will be discussed in section 3.1.2. After that, multi-classifiers applied to the field of computer vision specifically will be addressed in section 3.1.3. Finally, section 3.1.4 will discuss the research questions to be answered in the rest of this chapter.

3.1.1 Static combination rules

The tracker needs to estimate the probability of observing the data given the state of the particle n to calculate the weight of the particle $\pi_t^{(n)}$. If multiple features $x_i, i = 1, \ldots, k$ are available, this probability depends on all of these features: $\pi_t^{(n)} = p(x_1, \ldots, x_k | s_t^n)$. However, with a limited amount of data it is often not possible to make a good estimation of this joint probability density function, especially if the dimensionality of the training data is high.

In [32], the authors derive several simplifications of these rules. Under the assumption that the feature vectors are statistically independent, the naive Bayes rule or product rule shown in equation 3.1 can be derived.

$$p(x_1, \ldots, x_k | s_t^n) = \prod_{i=1}^{k} p(x_i | s_t^n) \tag{3.1}$$

A sum rule can be derived from the product rule under the assumption that the a posteriori probabilities will not deviate too much from the prior probabilities [32, 33]. However, the sum rule might also be used for different reasons, under different

assumptions. Note that the sum rule is equivalent to the mean rule, as the outputs of the rules will be normalized and so the scalar division to obtain the mean can be left out without consequences. If the prior probabilities are equal, the sum rule can be viewed as computing the average a posteriori probability over all the classifier outputs [32]. If features are identical ($x_1 = x_2 = \ldots = x_k$), different classifiers provide different approximations of the same probability, and the average estimator is a well-known estimator for the true probability [30]. Even if features are not identical, averaging the probabilities might show very good results [30]. A robust estimator of the mean is the median, so a median rule might also be appropriate for classifier combination [32].

The average and the product rules can be approximated using their upper or lower bounds [32]. These are shown in equation 3.2. Both the minimum and the maximum could be used to approximate the true probability $p(x_1, \ldots, x_k|s_t^n)$

$$\prod_{i=1}^{k} p(x_i|s_t^n) \leq \min_{i=1}^{k} p(x_i|s_t^n) \leq \frac{1}{k} \sum_{i=1}^{k} p(x_i|s_t^n) \leq \max_{i=1}^{k} p(x_i|s_t^n) \tag{3.2}$$

The minimum rule and the maximum rule are very sensitive to outliers caused by constituent features estimating the posterior probability respectively (much) too low or too high. The product rule is very sensitive to features estimating the posterior probability to be (close to) zero: if just one constituent classifier estimates the posterior as 0, the outcome of the product rule will be 0 as well. The mean is less sensitive to outliers, but can still be strongly influenced by just one estimate far from all the others. The median is a robust estimator of the mean, which is insensitive to outliers far away from most of the estimates.

In a practical comparison, the authors of [30] found that the mean rule outperforms the product rule when the constituent classifiers are not able to make a good estimate of the posterior probability, or when they output values close to one or zero. This is probably because the error sensitivity of the product rule is much higher than that of the sum rule, according to the mathematical analysis performed in [32, 33]. An experimental comparison [32] found that the sum and median rules performed best on a multi-modal identity verification task and a handwritten character recognition task. However, the authors of [30] found that if the number of classes to assign the pattern to is larger than two, the product rule outperformed the sum rule on a digit recognition task.

3.1.2 Learning systems

Just as there are many different static rules, there are a lot of different methods incorporating learning or adaptation in one form or another. A straightforward extension of the static mean rule is a trained weighted mean rule, where the weights for each of the classifiers are learned from the dataset. The product rule can be extended by raising each of the factors to the power of some reliability factor [34].

The reliability factors or weights are parameters that have to be learned: which features are most useful depends on the dataset.

In [35, 36], the authors discuss several ways of learning optimal parameters, in this case for optimizing the parameters for a walking quadrupedal robot. Out of all methods discussed in those publications, the policy gradient learning algorithm had a good exploration speed and achieved the highest performance in the presence of noise. This technique works by starting from a reasonable starting policy. Then, it tries to estimate the gradient in weight space by generating and evaluating a number of policies around the starting policy. The starting policy is adjusted in the direction of the gradient, and this process is repeated until a local optimum is reached.

In case the classifier output is a discrete decision, weights may indicate confidence in the classifier, or the probability that the classifier makes the right decision. For classifiers with a binary output, the Weighted Majority Algorithm [37] could be used to calculate these weights. For the current application however, not a classification, but an estimation of the likelihood of the current observation is needed. A way to calculate these weights adaptively with respect to the current observation is discussed in [38]. In that publication, the weight is a combination of three components: the classifier commitment coefficient (that quantifies the confidence of the classifier in the chosen class, compared to the other possible classes), the cross-classifier coherence coefficient (that quantifies how much the current classifier agrees with other classifiers on class-specific activation), and the classifier accuracy coefficient (prior knowledge of the classifier's accuracy). This method is adaptive in the sense that not only the prior training data is considered, but also the class activation for the current sensory input. However, it is tuned specifically to classification problems with more than two classes, and aims mainly at classifying patterns over time.

In addition, [10] proposes using the maximum entropy principle for combining features, resulting in a posterior probability of the form $Z^{-1}e^{\sum_i w_i f_i}$, with Z a normalization term and w_i the weight of feature f_i. This principle does not assume independence, and yields decision boundaries equal to those obtained by taking the (weighted) average of features. The authors also discuss a closed form for calculating incremental updates of the weights, for which the target values have to be known.

Another direct way of estimating the likelihood given a set of classifier outputs is by training a combining classifier on the output values of the primitive classifiers. This is discussed in [31] for example. However, in the current problem the target value for the feature combiner is not known.

Two alternative methods discussed by [31] do not combine classifiers, but rather select them. First of all, it could be tried to select an optimal subset (ensemble) of classifiers from the total set of classifiers. This enables the system to discard irrelevant, poor-performing, or duplicate classifiers, hopefully improving recognition performance and decreasing computational cost (as less classifiers have to be evaluated). The second option is to partition the feature space in regions and

identifying the best base classifier for each region.

The current problem is not a classification problem, but a problem of assigning the good weights (likelihood estimates) to each of the particles. As the target value of the weights to be assigned to each particle is not known, a lot of the methods above cannot be used. Also, the number of classifiers is limited, so discarding features will have a limited effect.

However, it is possible to assess the performance of a certain set of parameters. So to solve the current problem of weight optimization, a generative approach will be taken. Different combinations of parameters will be generated after which they can be evaluated. The policy gradient learning algorithm (discussed above) uses the evaluations to estimate the gradient around the current solution. The gradient can be used to update the weights. By iterating this procedure, incrementally a better solution can be found.

3.1.3 Different kinds of visual features

Visual features

A lot of different features can be used in computer vision methods to recognize objects. Color features are very popular for tracking objects, being used in for instance [21–23]. This can be because they are very simple: no calculations are needed as the RGB (red, green, blue) values for each pixel are provided by the camera. Also, color features are easy to interpret: colors that look similar to the human eye will in most cases also be similar to the vision algorithm.

However, the big disadvantage of using color features is that they are not invariant to illumination conditions. This can lead to problems in a room which has different illumination conditions than the room that was trained in, or if illumination changes with the weather conditions outside. In those conditions, RGB values of trained objects might change substantially, even though colors may look unchanged to the human eye (which adapts to these situations). A partial remedy for this is to use another color space, for instance HSV-space (hue, saturation, value). By disregarding the value component, the color feature becomes much more robust to changes in the brightness of the illumination. However, if the color of the illumination changes this will still introduce problems.

Of course, colors alone do not describe an image. Another property to look at might be the intensity of the image. However, here the problem of being dependent on the illumination is still stronger: in a brighter illumination, all pixels will have a higher intensity. If this increase is the same for all pixels, the spatial derivatives of the intensity will stay the same. So, the gradient of the gray scale image in every point might be used. This feature is quite robust against illumination variance and is quite a powerful feature. Weighted histograms of gradient directions in a 4x4 grid surrounding a point is used as the descriptor of SIFT key-points [5, 6]. These descriptors are widely used because they are very specific and quite robust.

Combining visual features

Different features can provide complementary information about the objects in the visual scene. Especially color features are combined often with other features, to obtain more specific or more robust representations. It can for instance be combined with texture and optionally edge features for tracking objects in video sequences, like in [24, 25]. In both of these publications, color and texture are assumed to be independent, such that the overall likelihood function is a product of the likelihoods of the separate cues. These color and texture features were both represented by histograms, which ignore all spatial information. Optionally, some spatial information can be retained by giving greater weight to pixels in the center of the image region [25]. It is also possible to create separate histograms for each region of a spatial grid, like in the SIFT descriptor [5, 6].

Shape can also be represented explicitly, for instance by using the curve features discussed previously. In [34, 39], the authors combined such curves with color features. Like in the work discussed in the previous paragraph, in [34] color and shape features are assumed to be independent. However, rather than simply multiplying the cues, they are first raised to the power α. This α is called the reliability factor and can be used to weigh cues with respect to each other.

A different approach for cue integration is described in [39]. In that work, not a fixed color and shape model are used, but the color and shape model can be adapted each time step based on the sample weights generated by the other feature. So, if the color distribution of an object changes, but the shape stays roughly similar, the color model will adapt to the samples with a high weight according to the shape model, thereby adapting to the new illumination conditions. The authors call this process co-inference learning.

In this work, both features that describe color (in RGB or HSV color space) and shape (using intensity gradients) will be used. These features will be used both in isolation and in combination.

3.1.4 How to combine different kinds of visual features?

This chapter will focus on the question what representation to use of the input images, and in what way the different representations can be combined. Important issues will be what combination of features maximizes the score of the tracking system, and whether the best combination of representations can outperform the best single representation. Are adaptive combination rules better than static rules? And what is the consequence of having non-informative features? These issues will also be addressed in this chapter.

Because the dataset contains quite a lot of noise and the features used will be partially redundant, it is expected that the product rule will not do very well as it is sensitive to noise and assumes representations to be independent. The minimum and the maximum rules are rather coarse approximations of the product rule and the mean rule, which for that reason might not work too well. The mean rule has

in previous work often shown to perform well in different circumstances, mainly because it is quite robust to noise. For this reason the mean rule, and the median rule as robust estimator of the mean, are expected to do well.

A good combiner should be able to handle redundant and non-informative features. Learning combiners are therefore expected to be able to outperform the combiners mentioned above, as it will learn which features to use. Color and gradient features are popular and are therefore expected to be useful. Also, they offer complementary information that could be exploited by a combination scheme.

3.2 Methods for feature combination

3.2.1 Basic features

The multiple-feature system works by combining the results of a set of basic features. In the current research, those features represent different properties of the input data. The representations that are considered in this research will be outlined in the next paragraphs. These features are used to construct normalized histograms. These histograms are compared using the Bhattacharyya coefficient, from which the weights are calculated using equations 2.8 and 2.9 on page 13.

The RGB features are the same kind of features that were used in the first experiment. This is simply a three-dimensional histogram of the (R, G, B) values of all pixels belonging to the rectangular sub-window described by the particle. As one histogram describes the entire sub-window, this means spatial information is lost. Like in the first experiment, histograms with 8 bins in each dimension (R, G, and B) are considered. However, in this chapter also histograms with 4 bins in each direction will be considered as they might be more suitable and are faster to compare thanks to their smaller dimensionality.

The HSV features are very similar to the RGB features, however the image is first transformed from the RGB color-space to the HSV color-space. Like with the RGB features, three-dimensional histograms with 8 bins in every dimension and with 4 bins in every direction are considered. [21] suggests to use a 8 bins for the H (hue) and S (saturation) components but only 4 bins for the V, or value, component. This is because the value component is mainly dependent on the amount of illumination and not on the object itself, and using less bins for this component might make the tracker more robust. Therefore, this histogram size is considered as well.

The Spatialized Gradient Features. The spatialized gradient features are inspired by the descriptors used in the SIFT algorithm, described in [6]. First, the gradient direction is calculated. For this, the numerical derivative in the x- and

y-directions are calculated with the Sobel-operator using a 3×3 sized aperture. This is simply convolving the image using each of the following two kernels

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

After that, the direction of the gradient is calculated by converting the Cartesian representation of the gradient to a polar representation. Secondly, the subregion of the image described by the particle is divided by a $n_{columns} \times n_{rows}$-sized grid. For each grid cell, a one-dimensional histogram with $n_{gradient}$ bins is made.

This means that this feature takes some spatial information into account by making a separate histogram for each grid cell. This is different from the HSV and RGB features, that pooled information from the entire sub-window into one histogram.

8-dimensional histograms on a 4×4 grids have proven successful as the SIFT descriptor, so this feature size ($4 \times 4 \times 8$) was considered. Like the RGB and HSV feature, also feature sizes $4 \times 4 \times 4$ (four-bin histograms on a four by four grid) and $8 \times 8 \times 8$ (eight-bin histograms on an eight by eight grid) were considered.

The entire grid of histograms are compared to the model (which is in this case also a grid of histograms). This is done by treating the bins for all small histograms as if they were bins of one big histogram. So, the distance between two features p and q consisting both of l small histograms ($p_1 \dots p_l$ and $q_1 \dots q_l$) of m bins will be calculated by a variation of the Bhattacharyya distance shown in equation 3.3.

$$d(p,q) = \sqrt{1 - \sum_{v=1}^{l} \sum_{u=1}^{m} \sqrt{p_v^{(u)} q_v^{(u)}}} \tag{3.3}$$

The spatialized intensity features are computed by first converting the image to gray scale, and then making a one-dimensional histogram of the intensities in each cell of a grid overlaying the image region. So like the spatialized gradient feature, this feature does take spatial lay-out of the sub-window into account. Like for the spatialized gradient feature, 8-bin histograms on a 8×8 and 4×4 grid were evaluated, as well as 4-bin histograms on a 4×4 grid.

The spatialized hue features are again spatial features. This time, the image is converted to HSV color-space. Then, one-dimensional histograms of the hue values for each grid cell are made. Like for the spatialized gradient feature, 8-bin histograms on a 8×8 and 4×4 grid were evaluated, as well as 4-bin histograms on a 4×4 grid.

3.2.2 Combiners

There are multiple ways to combine the results of the basic features. This can be done using a static rule, or using an adaptive rule.

Static rules

Static rules are heuristics that operate on the outputs of the basic features to estimate the likelihood of making the current observation given the state of the particle. The outputs of the basic features form a set $X = \{x_1, x_2, \ldots, x_k\}$. The heuristics to estimate the likelihood are the maximum rule ($f(X) = \max_i(x_i)$), the minimum rule ($f(X) = \min_i(x_i)$), the mean rule ($f(X) = k^{-1} \sum_{i=1}^{k} x_i$), the median rule ($f(X) = \mu_{1/2}(X)$) and the product rule ($f(X) = \prod_{i=1}^{k} x_i$).

Adaptive rules

Weighted versions of the average and product rules were used as adaptive methods for feature combination. Weights $w_1, w_2, \ldots, w_k$ were used as confidence factors for features $x_1, x_2, \ldots, x_k$ to obtain a weighted average rule ($f(X) = k^{-1} \sum_{i=1}^{k} x_i \times w_i$) and a weighted product rule ($f(X) = \prod_{i=1}^{k} x_i^{w_i}$)

Gradient estimation The optimal weights are found by starting a local gradient ascent from a reasonable starting policy. This policy has k parameters $\theta_1, \ldots, \theta_k$, which correspond to the k weights of the adaptive rules. Then, a collection of policies $R_1 \ldots R_t$ in the vicinity of the starting point is generated. These policies have parameters $\theta_1 + \Delta_1, \ldots, \theta_k + \Delta_k$. Each Δ_i is chosen randomly to be $+\epsilon$, 0, or $-\epsilon$. ϵ is a fixed, relatively small value. The generated policies are subsequently evaluated on the optimization (training) fold.

This is similar to policy gradient learning [35,36]. Different from that research however, from the t policies exactly $t/3$ policies will have a Δ_i of $+\epsilon_i$, exactly $t/3$ will have a Δ_i of 0, and exactly $t/3$ policies will have a Δ_i of $-\epsilon_i$ for all i.

These evaluations are used to estimate the partial derivative in each direction of parameter space. This is done by first dividing the evaluations over three groups for every dimension n [35,36]:

$$R_i \in \begin{cases} S_{+\epsilon,n} & \text{if the } n\text{th parameter of } R_i \text{ is } \theta_n + \epsilon_n \\ S_{+0,n} & \text{if the } n\text{th parameter of } R_i \text{ is } \theta_n + 0 \\ S_{-\epsilon,n} & \text{if the } n\text{th parameter of } R_i \text{ is } \theta_n - \epsilon_n \end{cases} \tag{3.4}$$

Then, the partial derivative is estimated to be 0 if the average of $S_{+0,n}$ is larger than the average of both other groups, or to be the term $r^{-1} * (Avg_{+\epsilon,n} - Avg_{-\epsilon,n})$ otherwise (where $Avg_{+\epsilon,n}$ and $Avg_{-\epsilon,n}$ are the averages of $S_{+\epsilon,n}$ and $S_{-\epsilon,n}$, respectively). The final gradient will be the normalized value of all these terms: r is the norm of the vector consisting of all of these terms.

Updating the policy In [35, 36], once an estimate of the gradient was made, in that direction a step of size η was made. However, the Rprop method, which is based only on the sign of the partial derivatives rather than the magnitude, is significantly more robust and requires less manual tuning [40]. In this method, the size of parameter change is determined by the update value δ. δ depends on an estimate of the gradient of the expected return $\nabla_\theta J(\theta)$ [40]:

$$\theta_i^{(t+1)} = \begin{cases} \theta_i^{(t)} + \delta_i^{(t)} & \text{, if}(\nabla_\theta J(\theta))_i^{(t)} > 0 \\ \theta_i^{(t)} - \delta_i^{(t)} & \text{, if}(\nabla_\theta J(\theta))_i^{(t)} < 0 \\ 0 & \text{, else} \end{cases} \tag{3.5}$$

Then, the update value for the following time step is calculated according to equation 3.6. In this equation, the sign of the partial derivative at time t is compared to the sign of the partial derivative at time t^*. In [40], the sign at time t was compared to the sign at time $t-1$ ($t^* = t-1$). However, in the current problem the estimate of the partial derivative was often 0 due to effects of noise. This means the δ's often were not adapted, making convergence slow. So for this problem, it worked better to take the time of the latest non-zero value of the estimation of the partial derivative as t^*.

$$\delta_i^{(t+1)} = \begin{cases} \eta^+ \times \delta_i^{(t)} & \text{, if}(\nabla_\theta J(\theta))_i^{(t)} \times (\nabla_\theta J(\theta))_i^{(t^*)} > 0 \\ \eta^- \times \delta_i^{(t)} & \text{, if}(\nabla_\theta J(\theta))_i^{(t)} \times (\nabla_\theta J(\theta))_i^{(t^*)} < 0 \\ \delta_i^{(t)} & \text{, else} \end{cases} \tag{3.6}$$

If the algorithm has made a too big update step, overshooting a local minimum, the sign of the parameter update changes. Because $\eta^- < 1$, this means the size of future updates for that parameter becomes smaller. $\eta^+ > 1$, so if the update retains its sign, it slightly increases to accelerate learning in flat regions of the parameter space. The increase and decrease factors are set to fixed values: $\eta^- = 0.5$ and $\eta^+ = 1.2$. The generation and evaluation of policies is repeated until changes to the policy become smaller than some small value.

The initial value of the δ's was 0.1, while ϵ (the amount added or subtracted from the parameters to estimate the gradient around the current policy) was set to 0.05. 9 policies were generated at each iteration of the algorithm. These values were found to yield a satisfying performance of the learning algorithm during experimentation.

3.2.3 Complete system

The complete system, consisting of a detector and a tracker, is very similar to the complete system described in section 2.2 on page 9. However, rather than using RGB-histograms to calculate the weights of the particles, the tracker can use various features or combination strategies. In a pilot study, the tracker will use one

of the single features discussed in section 3.2.1. In the final experiment the tracker will use one of the possible combination strategies described in section 3.2.2 to combine the weights of the features. The difference between the structure of the (single-feature) combined system described in the previous chapter and the final multi-feature system described in the current chapter is shown schematically in figure 3.1.

3.2.4 Dataset

13 sequences from the UO Face Video Database [28] with added artificial movement were used to test the performance of the tracking system. This dataset is described in more detail in section 2.2.5 on page 14.

3.2.5 Performance measure

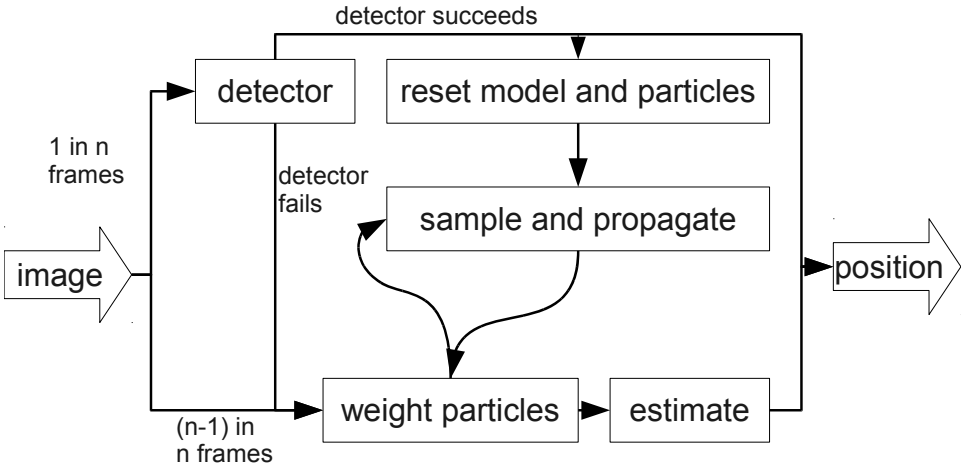
To measure the performance of the multi-feature system and the baseline system, the performance measure described in section 2.2.6 on page 16 was used. This is a measure of overlap between the ground truth and the detection.

3.2.6 Parameter optimization

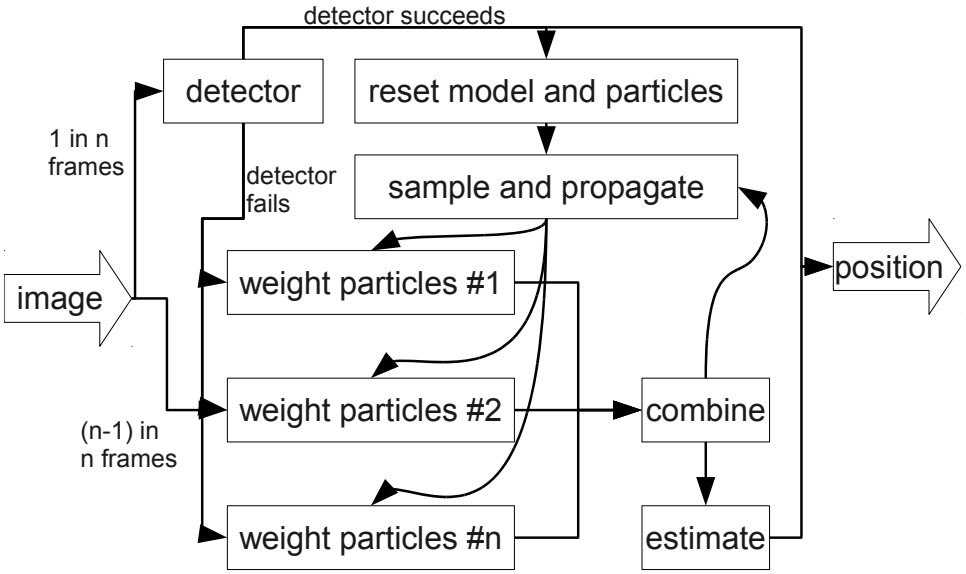
The system is very dependent on the value of the σ parameter. To get a fair comparison between different features, this parameter has to be tuned for each specific feature. If the features are to be combined, it is also important that the output of features are scaled similarly across different features, so in this case it is not about having the best value of σ for each feature, but to have a set of values $\sigma_1 \ldots \sigma_k$ that optimize the performance of the multi-feature system composed of k individual features.

Parameter sweep In case there is only one parameter, so when only one feature is used, it is straightforward to try all relevant values for this parameter. Values of σ smaller than 0.04 sometimes give each frame a weight of 0, so no information about the target is retained. In trials where $\sigma > 0.14$, performance was always low as the weight given to 'good' particles and 'bad' particles was very similar. Very small differences do not have much of an influence on the final outcome, so all multiples of 0.02 between these values are tried to find a reasonable parameter setting. All of those values are evaluated on a training set to select the value performing best on that set. Subsequently, the optimal setting is evaluated on a separate test set to prevent bias from evaluation on the training set.

Policy gradient reinforcement learning The static systems need 5 scale parameters (σ). This means that to try 10 values for each of the parameters, a complete sweep will have to try out between 10^5 parameter combinations, which is not realistic. However, if it is possible to identify a reasonable starting point, gradient



(a) The one-feature combined system



(b) The multi-feature combined system

Figure 3.1: The multi-feature system compared to the single-feature system described in the previous chapter.

ascent can be used to find a useful local optimum. This is done using the policy gradient learning algorithm, explained in section 3.2.2 on page 31.

For the static rules, the scale parameters (σ values) of the individual features can be used as a starting point for optimizing parameters on the multi-feature system. The algorithm will then optimize these values.

For the adaptive rules, the weights or confidence factors on the adaptive systems will be optimized using this algorithm. The adaptive rules will use the optimized σ parameters from the static versions of the same rules. The starting policy gives equal weights to all of the features. For the weighted average rule, the starting weight for each of the features was k^{-1} (with k the number of features).

The weighted product rule is very sensitive to noise, especially single features estimating the probability to be (close to) 0. If all features start out at a weight of some positive value, the non-informative features reduce performance to a point where small changes to the weights do not matter, and so learning by gradient descent is pointless. For this reason, the starting weights for the product rule were 0 for all features. This means all features are totally ignored, unless a weight of $0 + \delta$ achieves better results.

Both static and adaptive rules will be optimized on the training fold, and then evaluated on a separate test fold.

3.2.7 Pilot

To find out what the optimal histogram sizes are for each of the features, a pilot study was performed where each of the histogram sizes suggested in section 3.2 on page 29 was evaluated on the dataset. In each evaluation, the dataset was split in two folds. Then, the σ parameter was optimized using a parameter sweep as discussed in the previous section for each of the folds, with the optimal parameter value being tested on the other fold. In the evaluation, the tracker was updated by the detector every 100 frames. This low update frequency should make differences between the features and histogram sizes more noticeable.

The results of this pilot are shown in table 3.1. For the HSV feature, two histogram sizes performed equally well: $8 \times 8 \times 4$ and $8 \times 8 \times 8$. Out of these two options, the $8 \times 8 \times 4$ histogram will be used in the evaluation of combination rules as it requires only half the amount of memory to store the histogram and half the amount of operations to compare two such histograms. From the other features, the best-performing histogram sizes will be used. The results of combinations of feature and histogram size that will be used in the evaluation of the combination rules are printed in bold.

The creation of the histograms was one of the most time-consuming steps. For each of the combinations of feature and histogram size, an equal amount of pixels had to be distributed among the histogram bins. Because of this, there were no very consistent differences in the requirements for processing time for different settings. All settings performed in real time, with frame rates between 56 and 81 frames per second. The difference was not consistent for certain parameter settings, therefore

Feature	$4 \times 4 \times 4$		$8 \times 8 \times 8$		$4 \times 4 \times 8$		$8 \times 8 \times 4$	
Fold	1	2	1	2	1	2	1	2
RGB	0.38	0.39	**0.41**	**0.44**	-	-	-	-
HSV	0.37	0.37	0.39	0.45	-	-	**0.39**	**0.45**
Spat. gradient	0.53	0.53	0.57	0.58	**0.59**	**0.59**	-	-
Spat. intensity	**0.46**	**0.41**	0.40	0.42	0.44	0.36	-	-
Spat. hue	0.39	0.43	**0.47**	**0.48**	0.46	0.48	-	-

Table 3.1: Results of the pilot measuring the performance of histograms of different sizes containing single features. Results shown are the evaluations of the test fold after σ was optimized on the training fold. Performance scores are shown for both folds. The tracker was updated once every 100 frames by the detector system. The reported performance score is a measure of overlap between the ground truth and the detection, as explained section 2.2.6 on page 16. Feature / histogram-size combinations that will be used in the experiment are printed in bold.

it is plausible that the difference was caused by other factors such as system load.

3.2.8 Experimental setup

The output of the five features from the pilot will be combined using the various combination rules. The parameters for the non-adaptive rules are the σ values for each of the features. The adaptive versions of the average and the product rule, will retain the optimized σ values from the non-adaptive versions of these rules, but will optimize the weights or confidence values for each of the features.

Each of the parameters is optimized and then evaluated using 2-fold cross validation. This means that half of the dataset is used for optimization and the other half is used for evaluation of the optimized rule, after which the two halves are swapped and the process is repeated. The optimization procedure is described in section 3.2.6. During optimization and evaluation, the tracker system was again updated by the detector system every 100 frames.

Even the RGB histogram feature, that achieved one of the lowest performance scores, was able to outperform the benchmark 'memory' system in the previous chapter. So, all of these features can be assumed to attain a 'reasonable' performance. However, a more general multi-feature system might depend on large numbers of (possibly automatically generated) features, most of which might not provide 'reasonable' scores. A good combination rule should be able to handle such features.

To test this property, in addition to the five informative visual features, five non-informative values (random values) are generated in the same range as the five visual features. These non-informative features are drawn from five Gaussian distributions with means and variances equal to the means and variances of the output of the five visual features when processing the faces from the training set.

In this test, the combination rules received an input set consisting of the five visual features and the five non-informative values. All adaptive rules and the static versions of these rules will be evaluated on this input set, again using 2-fold cross validation.

3.3 Results of the multi-feature system

Figure 3.2 shows the results of the different combination rules on the dataset with the combiners getting input from the five informative features. Shown performance scores are the average of the scores achieved on the two folds.

Differences between the rules are not very large, but from the non-adaptive rules, the product rule performs best. There is no significant difference between the product rule and its adaptive version, or between the average rule and its adaptive version. Furthermore, the minimum and median rule perform a little better than the average rule and its adaptive version, and the maximum rule performs the worst.

All systems required on average between 44 and 60 ms to process each frame. This means a frame rate of 17-23 frames per second. The creation and comparison of histograms consumed most time, so the differences between the different combination rules were negligible. The differences in processing time are therefore likely to be caused by other factors, such as system load.

In addition to this experiment, a pilot was performed where one or more fea-

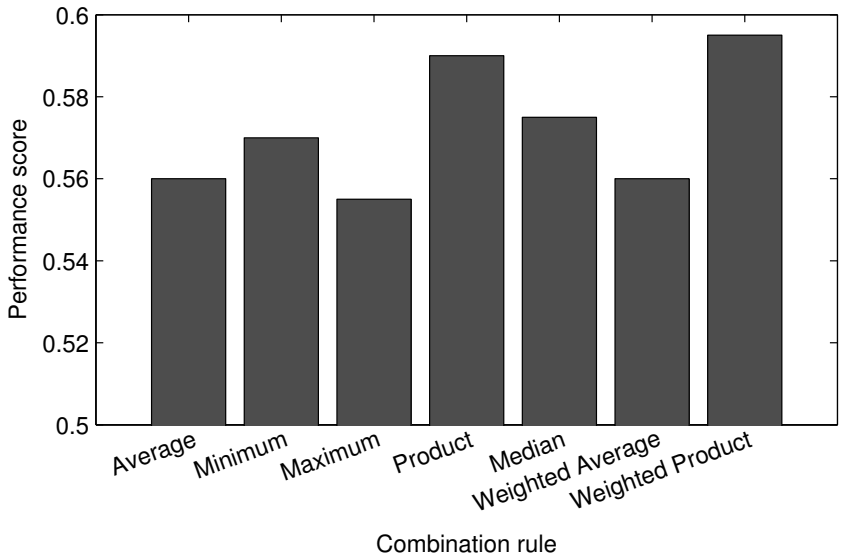


Figure 3.2: Performance score of the various combination rules processing the output of five features. Data points are the averages between the evaluation on the two test folds. The reported performance score is a measure of overlap between the ground truth and the detection, as explained section 2.2.6 on page 16.

tures were dropped from the input feature set. This was done by either removing the features with the worst performance, or the features with the least independence of other features (The RGB and HSV feature are strongly dependent on each other, while RGB and spatialized gradient are not). It was expected that some combiners (like the product rule) would do better if there were no features that were strongly dependent on each other, while for others just taking the features with the best performance would be better. However, preliminary results did not indicate interactions between the feature input set and the performance of classifiers. Also, the learning classifiers should already be able to discard features that decrease performance. For those reasons, these experiments were not continued.

In a second experiment, a selection of the combination rules were tested in a set-up where they were receiving ten input values. These ten values were made up of the five visual features discussed before and five non-informative values. The result of this experiment are shown in figure 3.3. When comparing these results to the results of the five feature experiment, it is apparent that the product and average rule perform much worse than before (notice that the scale on the vertical axis is different in this figure). The product rule suffered slightly more than the average rule, as the performance of both of these rules is now similar.

Confronted with the non-informative input values, the adaptive version of the product rule performs much better than the non-adaptive version, achieving similar results as on the set consisting of only the informative visual features. The

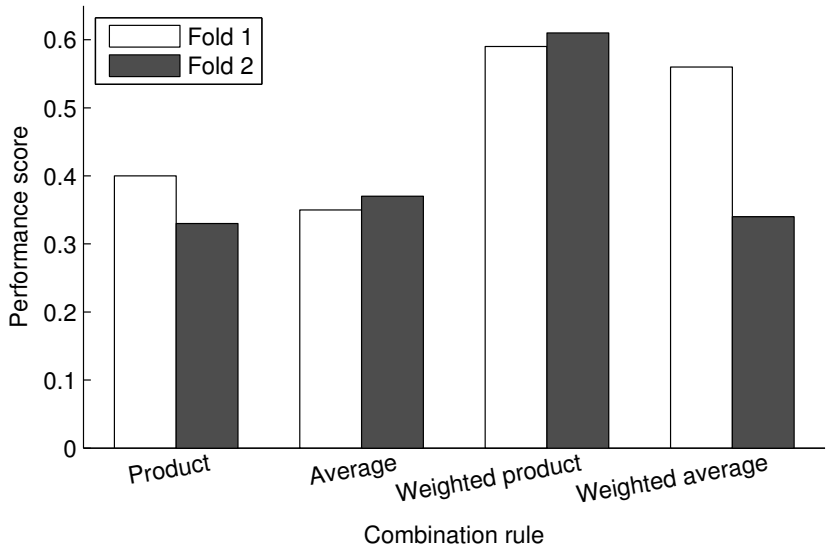


Figure 3.3: Performance scores of selected combination rules processing five visual features and five non-informative values. Data points for both test folds are shown separately. The reported performance score is a measure of overlap between the ground truth and the detection, as explained section 2.2.6 on page 16.

weighted average rule achieved a high performance similar to the performance on the five feature set on one fold. However, on the other fold it performed just as bad as the unweighted average rule.

3.4 Discussion of the multi-feature system

Five different features were implemented for use in the particle filter. A number of different combination rules were implemented to combine these features to improve performance of the particle filter.

Combination performance on the five-feature dataset

The combination rules were given the results of the five features as input and were evaluated on the dataset. Under these conditions, the product rule (and the adaptive version of this rule) achieved the highest performance. Furthermore, the median and minimum rule did slightly better than the maximum rule, the average rule, and the adaptive version of the average rule. This does not confirm the expectations in the introduction of this chapter, which stated that the average rule would probably perform better because the conditions are noisy. In this case the fact that the product rule is more sensitive to single low values could actually have helped rather than hindered performance, quickly selecting away bad particles.

Contrary to the expectations expressed in the introduction, the static and adaptive versions of the product rules achieved similar performance. The same holds for the average rule. This could be because all features perform reasonably and so differences in weights between the features did not improve the results. However, another cause could be optimization of the σ values. With a higher σ, the output of the feature becomes less dependent on the actual overlap between the histogram. So, during the optimization badly performing features might already get a lower impact on the final combination rule output (and indirectly the tracker's performance).

None of the combination rules were able to convincingly outperform the best single feature, the spatialized gradient feature. This could be because this feature performed so much better than the second-best feature. This means that the second-best feature might not make much of a contribution to information already present in this spatialized gradient feature. So, the combination rules might for this reason not be able to achieve a higher performance than the gradient feature.

Performance on a set containing non-informative values

In systems provided with many features (possibly automatically generated features), there might be many features that do not generate a reasonable output and so should not have a significant impact on the final output. Such a situation was simulated by adding five random non-informative values to the input of the combination rules. In this situation, as expected, the adaptive rules performed a

lot better than the non-adaptive versions of the same rules. This shows that the learning system successfully learned the five visual feature were more reliable than the five non-informative values.

Also, when provided with non-informative values, performance scores of the (non-adaptive) product and average rules are similar to each other, while on the 5-feature set the product rule performed better. This might indicate that the product rule was indeed more sensitive to the noisy features.

Improving the learning method

The adaptive average rule on one fold performed as good as on the 5-feature set, so in this case the system discovered successfully which rules to give a high weight. However, on the other fold this rule performed as bad as the unweighted product rule. So in this case learning was not successful. This could be because the algorithm stopped too early (so the stopping conditions are not right). It could also mean that random effects were stronger than the actual differences between the files evaluated by the learning algorithm, meaning it should evaluate more possibilities at each iteration. Or it could simply indicate that the used learning method is not optimal for this type of problem.

Chapter 4

Continuous Learning for Autonomous Robots

In the previous chapters a hybrid system was developed consisting of a tracker subsystem receiving feedback from a detector subsystem. The way the tracker processed the feedback was rather simple: its internal model of the target was simply reset to the part of the image that contains the target according to the detector system.

This made the combined system adaptive to changing circumstances, such as changing pose or changing illumination conditions. However, because the previous model was entirely replaced, the tracker does not remember the appearance of the target in different circumstances. This means that if conditions revert to how they were previously, again the tracker will have to rely on feedback from the detector system. However, feedback might not always be available. A learning tracker would become less dependent on feedback over time.

The experiments described previously have been performed on a dataset of video frames. Even though the video sequences were taken under realistic environmental conditions, good performance on the dataset is no guarantee for good performance in the real world. Ideally, the work described here and in the previous chapters can be used to provide vision to mobile, autonomous robots operating outside the lab in dynamic conditions. To test this, the combined system will be used to control an autonomous robot.

4.1 Introduction to continuous learning

Most detection and tracking systems have a model of the appearance of the target. However, the appearance of an object depends on both internal properties (shape, reflectance properties and pose) and external properties (illumination conditions, camera parameters, and occlusion) [41, 42]. When tracking, the appearance of the target model might change. Handling those changes is sometimes considered the main challenge of visual tracking [42].

External properties of the target can be fixed to some point to reduce appearance changes, for instance by evaluating the system under standardized illumination conditions. However, this is not an option when analyzing sensor information on a mobile, autonomous robot operating in a non-standardized real-world environment. As the robot moves around in its environment, the camera parameters and occlusion are constantly changing. And as the robot operates in a realistic and dynamic environment, it also has to handle wildly different backgrounds and illumination conditions.

4.1.1 Feedback availability

The combined system described in chapters 2 and 3 relied on feedback from a detector system to reset the model of the tracker if the appearance of the target object changed. However, feedback might not be available in all circumstances. In the previous chapters the target was a face, but for other target types, there might be no detector system available. If the tracking system is to be used on a mobile robot, the computational resources might be so limited that some detector systems cannot run on the robot. Also, the tracking system could be presented with completely new objects while running. As there is no prior knowledge on what objects the system might encounter, it is not possible to build in a detector system for those objects.

In those cases, different sources of feedback might be used. If a shortage of computational resources is the problem, a detector system could run off-board on a more powerful machine, or it could even be distributed over a network with multiple machines. However, the connection to the off-board system could be unreliable, in which case continuous feedback is not guaranteed.

In case the system is presented with an unexpected object, or a target for which there is no detector available, feedback could be provided by interaction with humans or with the environment. Interactions with humans might mean verbally asking people near the robot to provide the names of unknown objects; storing images of objects the robot is uncertain about to be labeled at a later time; or even asking for labels of ambiguous percepts through a web interface. Bringing a human in the loop was successfully done in [43] to bootstrap a system with information that was otherwise unavailable, in that case in the domain of handwritten word recognition.

Interaction with the environment means active vision: using the robot's actuators to improve its percepts. Possible examples are trying to pick the ambiguous object up to examine it in a different environment; getting feedback from physical interaction with the object (like in [44]) or driving around an ambiguous object to disambiguate it as more information is gathered from different view points (done in [45], for example).

These types of feedback are however not always available. A service robot is not valuable to a human if a lot of hours have to be spent to teach the robot. So the time that needs to be invested by the human to teach the robot has to be limited.

Interaction with the environment requires the robot to invest time that might be better spent. So, alternative feedback might be available from an off-board computer, human-robot interaction, or direct interaction with the environment. Even so, it is preferable to rely less on feedback so less feedback is needed, and so the system might continue functioning if feedback is not immediately available.

4.1.2 Reliance of the non-adaptive tracker on feedback

The tracker model used in the previous chapters was a static model. Adaptability of the combined system was achieved by resetting the tracker model to the new appearance of the target as feedback from the detector system came in. This meant that if the tracking system would be tracking a target in a brightly lit environment and the light would be turned off, it had to wait for feedback to reset the model to the appearance of the target in a dark environment.

However, no information about the target in a brightly lit environment would be retained. So, when the light would be turned back on, the system would again have to wait for feedback. This means the system is not making efficient use of the feedback, and will need to be corrected regularly to prevent the tracker from drifting away from the target or from losing the target entirely if the appearance of the target changes. As the system is not robust to appearance changes, limited feedback will hurt the system's performance to a large extent (this is apparent in table 2.1 on page 18: with a lower feedback frequency, the combined system's performance decreases).

4.1.3 Adaptation without feedback

In the literature on visual tracking, separate subsystems for feedback are not commonly used. To be able to handle changing circumstances, multiple other techniques are used. Firstly, it is possible to try to make the tracker model invariant to appearance changes. This was the approach used in [41]. However, this approach needed prior training on a large set of images under different pose and illumination conditions, so it is not suitable for learning new objects as they are encountered in the world.

A second approach is to adapt the model to the appearance of the target in detections in previous frames. In contrast to the approach described in the previous paragraph, this approach does not need a separate training phase but learns on the task continuously[1], making it possible to track novel objects for which no training database is available. This was the approach used by [42,46] to incrementally learn a low dimensional eigenspace representation to track faces or objects.

The danger of this approach is that the target might disappear or the tracker might slowly drift away from the target, with the model adapting more and more to

[1] Continuous on-the-task learning is also commonly referred to as on-line learning. This term will be avoided here as the term on-line is also used to refer to algorithms that process input data points as they become available, rather than waiting for the total set of input data points to become available.

the background rather than to the target. Some visual tracking models take specific measures to prevent this. In [47], the color model was adapted only to those pixels in the current frame that the current model could 'confidently' classify as belonging to the target. In [21,48] the color model was only adapted to the current detection if the current detection is sufficiently similar to the previous model. In [22] the color model was only adapted to slow appearance changes to prevent learning when the target disappears or is occluded, when appearance changes sharply. Both a long-term and a short-term appearance model were used in [49], with the short-term appearance model allowing rapid adaptation, but with the long-term model providing greater persistence of the model to cope with outliers. So-called co-inference learning was used by [39], where the color features were adapted based on the best-matching shape features and vice-versa to prevent the tracker from drifting away.

4.1.4 Using limited or unreliable feedback

Updating the model to something that is actually not the target is a problem for systems without feedback. The previous section described different solutions to this problem that have been proposed. However, even with those solutions it is not guaranteed that the image segment used to update the model is actually the target. For instance, even if sharp changes are prevented from impacting the model as they could indicate occlusion or disappearance of the target, the tracker might still be drifting slowly away, adapting more and more to the background rather than the target. Also, the appearance of the target might actually be truly changing very fast, for instance if lights are turned on or the target goes from indoor conditions to outdoor conditions. In these cases, limiting the adaptation to slow changes means the tracker will lose its target. To be sure the tracker is indeed updated in the right way, some amount of feedback is still needed.

If there is some feedback (even if it is limited or unreliable), this can be used to construct a stronger model over time. As feedback comes in, the tracker can learn the appearance of its target in different conditions. This will make the tracker more robust, requiring less feedback as it is slowly able to handle more situations by itself.

4.1.5 Research questions

This chapter will address two main questions. Firstly, how can the tracker use the feedback it receives to construct a representation of its target that is robust to appearance changes? Secondly, is this learning tracker better at tracking faces in sensor data from an autonomous, mobile robot than the non-learning tracker from the previous chapters?

4.2 Methods for visual learning on autonomous robots

This section will first describe an improvement to the tracking system, as it is changed to make the tracker model more robust over time, rather than resetting it every time feedback is received. Then, the robot which will be used in the experiment will be described, as well as its control architecture. The last parts of this section will describe how the tracker will be tested, and how the tracker's performance will be measured and analyzed.

4.2.1 Visual tracking module

The basis for the visual tracking module will be the combined tracking and detection system from the previous chapter. However, as combining features did not improve performance over just using the best single feature, here just single features will be used. Also, as the frames of the camera are of a higher resolution than in the dataset of the previous chapters, the image is down-sampled with a factor 3 rather than 1.4 before being passed on to the detector (to speed up processing). The standard deviation of the Gaussian component used to update the particle filter was made four times larger, from 5 to 20 pixels per frame for particle location and from 2 to 8 pixels per frame for particle size. This is to make sure the update component matches the expected movement in the higher resolution images.

Most importantly, the tracker will be changed to handle feedback differently. Previously, if feedback was received the tracker model was reset to a completely new model of the target in the present conditions. This made the tracker adaptive to changing circumstances, but the model did not become more robust over time. If circumstances reverted, the tracker had to wait for feedback again. Of course, it would be preferable if the tracker remembered how the target looked in different circumstances.

Learning system

If no feedback is available, the learning tracker system will choose the target model corresponding most closely to the current conditions, rather than simply the most recent target model. Whenever feedback is available, the appearance of the target under current conditions is learned for later use. These steps will be explained in the next paragraphs.

Feedback available. Every time feedback is received, two histograms are constructed. One of these models the target as it appears in the current image. The other histogram is a model of the complete image. This represents the environment conditions associated with the target model, and will be called the 'background model'. These two histograms are stored together in a database with earlier pairs of target model and background model.

The target histogram and the background histogram do not have to use the same features. Actually, it might even be better if they do not. The background histogram should represent the environmental conditions, especially illumination conditions. The target histogram, on the other hand, should be invariant to illumination conditions as much as possible. Also, the target model might benefit from a spatialized description of the target, while that might not be useful for representing environmental conditions.

No feedback available. In every frame where no feedback is received for whatever reason, the best target model will have to be found out of the memory database. For this, first a histogram is made of the entire image. This histogram again represents the environmental conditions in the current frame. The background histogram is matched against the database of earlier background histograms. The nearest neighbor is the image with conditions most similar to the current frame. The distance measure used for this nearest neighbor search is the Bhattacharyya distance (shown in equation 4.1). This measure was successful in tracking tasks in [21] and in previous chapters.

$$d(p_{s_t^{(n)}}, q) = \sqrt{1 - \sum_{u=1}^{m} \sqrt{p_{s_t^{(n)}}^{(u)} q^{(u)}}} \tag{4.1}$$

It is assumed that the target model associated with the background histogram best matching the current background histogram is the most appropriate target model (out of all target models in the database) for the current image. That model is used to compute the weight of the particles in the particle filter.

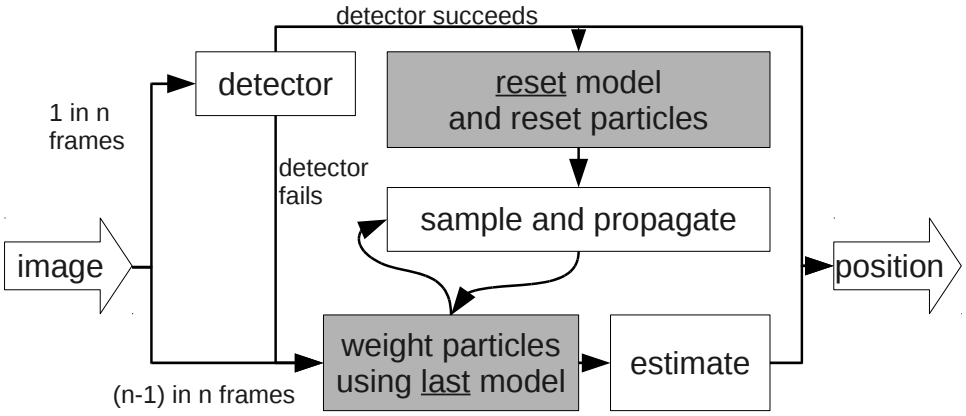
System Comparison

Figure 4.1 compares the combined system with a non-learning tracker to the combined system with a learning tracker. The difference is that rather than keeping and using only the most recent target model, all target models are stored together with a model of the associated background. When a new image comes in, the model associated with the background most similar to the current model is used to compute the weight of the particles in the particle filter.

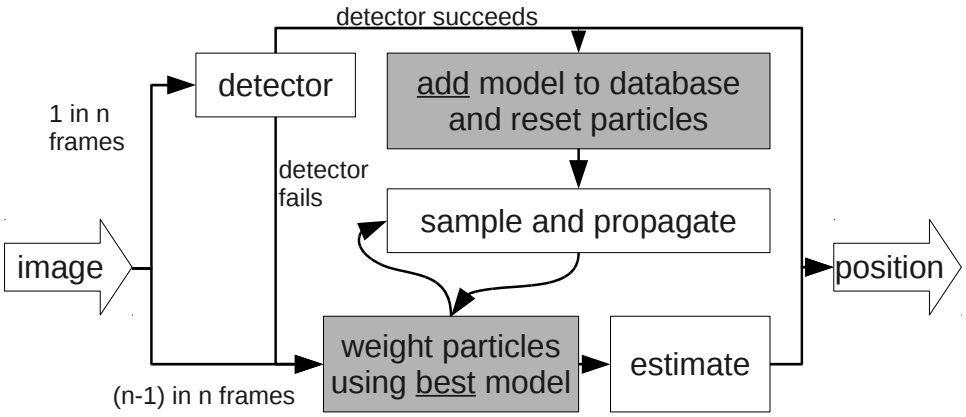
4.2.2 Autonomous robot

Hardware

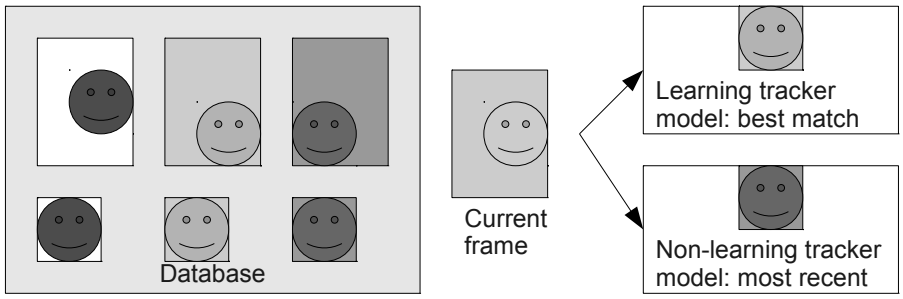
The hardware platform that will be used in the experiment is a Pioneer 2 mobile platform. This is a platform with two wheels that can be independently controlled, and an uncontrolled caster wheel. The platform carries five dual-core 1833 MHz laptop computers to provide additional computational resources, a Nao humanoid



(a) The system with a non-learning tracker



(b) The system with a learning tracker



(c) Models used by both systems

Figure 4.1: The combined system with the learning tracker compared to the combined system with a non-learning tracker. Sub-systems that were adapted are shown in highlighted boxes. Rather than using the last target model, the learning system uses the target model associated with the best matching background model from a database as illustrated in figure 4.1(c).

robot for human-robot interaction, and several additional sensors. These sensors include HD video cameras and IR depth finding cameras. The HD video cameras are used to provide images for the tracker system. A more detailed description of the platform can be found in [50]. The hardware platform is shown in figure 4.2(a) on page 49.

Robot control

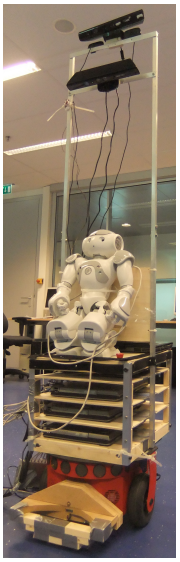
The complete learning tracker is used as a vision module, providing data to the main robot control program. The robot control program selects the behavior most suitable for the current task. In this case, that behavior is a 'follow' behavior, that turns the robot to the face and approaches it to a distance of one meter.

The goal of the follow behavior is to keep the face in the middle of the visual field and at an apparent size that corresponds to a distance of one meter. If the apparent size of the detected face is smaller (so, the face is farther away) or the detected face is not centered in the middle of the field of view, a control signal is sent to the robot to decrease the error (the difference between the actual value and the target value). The control signal is generated by adding the output of two proportional controllers. These proportional controllers are PID controllers with the proportionality constants for integrative and derivative terms (K_i and K_d) set to 0. PID controllers are widely used and described, for instance in [51]. In a proportional controller the magnitude of the controller output is proportional to the size of the error, with a proportionality constant of K_p.

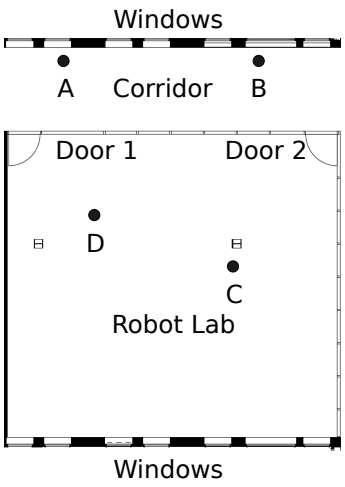
The first proportional controller decreases the difference between the middle of the detected face to the middle of the horizontal field of view. If this difference is positive, it means the face is detected in the right half of the visual field. The proportionality constant for the left wheel is positive, and for the right wheel negative. So, in case of a positive difference the robot turns to its right, and vice versa, centering the object in the middle of the horizontal field of view.

The second proportional controller decreases the difference between the desired apparent size of the face (corresponding to a distance of approximately one meter) and the actual apparent size of the face. A positive value means the apparent size is too small, indicating the robot has to approach. So, the proportionality constant is positive for both wheels. If the difference is negative (indicating the apparent size is too big), output is suppressed as the user might want to approach the robot. Also, driving backwards might be dangerous.

The proportionality constants should not be too big, to avoid overshoot, but big enough to provide an adequate response to differences between actual values and target values. Finding the right value is a process of trial and error. For the turning controller, a proportionality constant $K_{p,turn}$ of 100 mm / s is used (with the error expressed in rad). For the forward controller, a proportionality constant $K_{p,forward}$ of 3 mm / s is used, with the error expressed in pixels of apparent size. If this resulted in values higher than 200 mm / s, 200 mm / s is used to ensure safe operation of the robot. The output of the turning controller and the forward



(a) The robot



(b) Robot lab map

#	Description
1	D - door 1 - A - turn
2	A - door 1 - D - turn
3	C - door 2 - B - turn
4	B - door 2 - C - turn
5	C - door 2 - B - A - door 1 - D - C
6	C - D - door 1 - A - B - door 2 - C
t	D - C - door 2 - B - door 2 - D - door 1 - A - D - door 2 - B - A - door 1 - D - door 1 - A - B - door 1 - C

(c) Trajectories

Figure 4.2: (a) The robot used in the experiment. (b) A map of the robot laboratory and adjacent corridor, with labeled start points and end points of recorded sequences. (c) Table describing the six trajectories that were used to test the robot. Every trajectory was repeated three times to obtain a reliable dataset. It also describes the trajectory used to train the robot ('t').

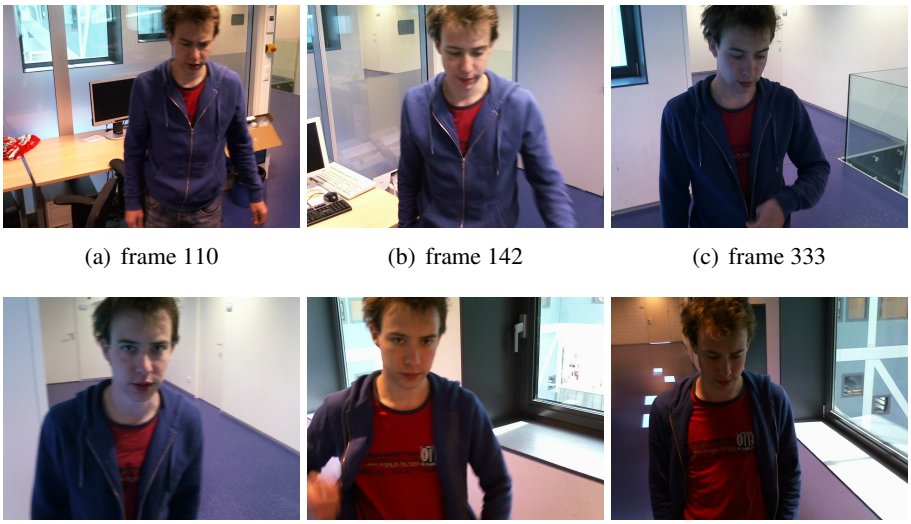
controller for each wheel are added. The sum is sent as a command to the Pioneer platform.

4.2.3 Experimental setup

Data collection

In the data collection phase, the robot is guided by a human walking backwards, facing the robot. The robot follows the face using the combined system. The tracker gets feedback from the detector subsystem whenever this is able to make a detection. The output of the combined system is used by the follow behavior to guide the robot. The detections of the detection subsystem and all images are saved to disk to be used later.

The paths along which the robot was guided are shown in figure 4.2. The robot was guided along one long path to allow it to learn. It was also guided along six testing paths to collect evaluation data. Each of the paths contains at least a section in the robot lab and a section in the corridor. The illumination in these areas is very different: the robot laboratory has more artificial light, and the sections in the corridor get very close to the windows, causing varying light intensity as the windows are passed. Figure 4.3 illustrates the kind of images in the dataset and the



(a) frame 110 (b) frame 142 (c) frame 333

(d) frame 374 (e) frame 407 (f) frame 434

Figure 4.3: Six frames from a sequence recorded at trajectory 3 illustrating the different illumination conditions along a trajectory. Best viewed in color.

different types of illumination encountered along a path.

The robot was guided along each of the testing paths three times to increase the size of the dataset. This makes test results more reliable. Each of these three repetitions results in a different dataset, as the exact trajectory in the three repetitions is not the same, and light conditions might change over the repetitions as the weather changes outside.

The training sequence contains 1263 images with 1021 frames with successful detections by the detector system. The test sequences contain between 231 and 577 images. In those sequences, the minimum number of successful detections was 134, while the maximum number of successful detections was 366.

Training phase

In the training phase, a database of models is made from the data in the training sequence. All images in which the detector subsystem was able to make a detection are used for this. Different kinds of features can be used to make these histograms. A model of the entire image is stored (this 'background model' is expected to provide some information about illumination conditions) together with a model of the target found by the detector. These models are (possibly spatialized) histograms like the features used in chapter 3.

The background (illumination) conditions are represented either by a 8-bin gray scale value histogram, or a $8 \times 8 \times 8$ RGB histogram. The target itself is represented either by a spatialized gradient histogram, that performed best in the previous chapter, or an $8 \times 8 \times 8$ RGB histogram, the simplest feature of the

previous chapter. These feature histograms are explained in detail in section 3.2.1 on page 29.

Testing phase

In the testing phase, there is no feedback. The detector subsystem is only used to initialize the tracker. After that, the tracker is not re-initialized or corrected anymore. This simulates a situation where feedback is unavailable, such as a breakdown of the feedback system, or a situation in which an off-board feedback system is unavailable. In these conditions, two tracking systems are compared. These systems have to track the target over a set of recorded images from the data collection phase.

The learning system tries to apply the knowledge from the training sequence on the current testing sequence. The knowledge was gathered with help of the detection subsystem in the training phase. The learning system takes a database of pairs of background model and target model. Then, it attempts to select the best target model for the current situation. The baseline non-learning system uses a target model created from the frame in which the tracker was initialized. This model is not updated during the processing of the sequence.

4.2.4 Procedure

For every combination of background and target model feature, the system is trained using the training sequence and then evaluated on each of 18 testing sequences (3 different sequences for each of the six paths in figure 4.2).

4.2.5 Data analysis

To measure the performance of the learning tracker (that was trained on the training sequence) and the non-learning tracker, the target location estimates of these trackers are compared to the detections by the detector subsystem. So, in the testing phase the detector subsystem is not used to provide feedback to the trackers, but it is used as 'ground truth' because there are no manual labels to compare the trackers' estimates with.

The performance score is a measure of overlap between the detector system's output (regarded as ground truth) and the tracker's estimate. The same performance measure as in section 2.2.6 on page 16 (suggested in [28]) is used. It is repeated in equation 4.2 with $A_{ground\ truth}$ the area of the ground truth box (in this case detector system output), $A_{detection}$ the area of the box found by the tracking system, and $A_{overlap}$ the area of the overlap between those two boxes. In frames where the detector system is not able to make a detection, no performance score is calculated.

$$Score = \frac{A_{overlap}}{\sqrt{A_{ground\ truth} \times A_{detection}}} \tag{4.2}$$

For every sequence, the performance score is calculated for all frames in which a detection from the detector system is available. Then, the average of all these performance scores is taken. Then the number of sequences in which average performance of the learning tracker was higher than the average performance of the non-learning tracker is counted.

Under the null hypothesis, the learning and non-learning trackers perform equally well on average so the expected number of sequences in which the learning tracker performs better is 9 (half of 18 sequences). The two-sided binomial test [29] is used to test whether the number of sequences where the learning tracker performed better is significantly higher than expected under the null hypothesis.

4.2.6 Effect of number of detections in the training file

It is interesting to see the effect of the number of detections in the training file on the performance of the learning tracker. This is inspected by using each of the 18 testing sequences as training data, and then evaluating the trained tracker on the other 17 files. The correlation coefficient between the number of detections in the training sequence and the performance of the tracker can be calculated. The correlation coefficient can be transformed to a statistic with a distribution that is approximately normal using the Fisher transformation [52]. The resulting value can be compared to an expected value of 0 with a Z test to check whether the null hypothesis that the number of detections in the training set and the performance of the tracker are uncorrelated can be rejected.

4.3 Comparative results of the learning system

4.3.1 Comparing the learning tracker to the non-learning tracker

Results are shown in figure 4.4. Some important statistics are summarized in table 4.1. Using the RGB feature for both background and target models, the trained learning tracker outperformed the baseline non-learning system in 16 out of 18 sequences.

Under the null hypothesis that both trackers perform equally well, the expected number of sequences in which the learning system performs better than the non-learning tracker is half of 18 sequences: 9. The difference between the observed number of sequences and the expected number of sequences is 16-9: 7. Under a binomial distribution, the two-sided probability that the difference between the observed number of sequences is equal to or greater than 7 is < 0.005. The number of sequences in which the learning system outperforms the non-learning system is significantly higher than expected under the null hypothesis in a two-sided binomial test.

When the value feature is used to create the background model and the RGB feature is used for the target model, performance for the learning tracker drops. When the spatialized gradient feature is used as target feature, performance is

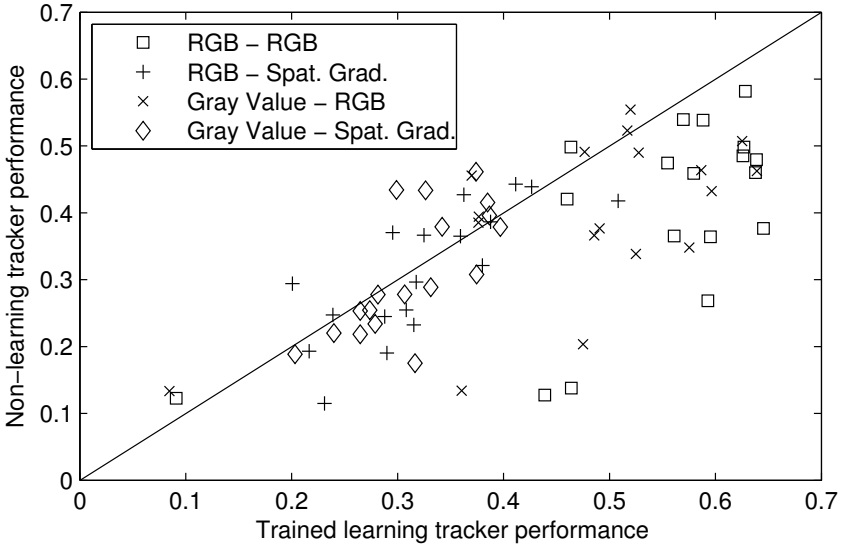


Figure 4.4: Performance for different 'background-target' feature pairs on all test sequences for the trained tracker and the non-learning tracker. Points below the $x = y$ line have a higher performance for the trained tracker. Performance is a measure of overlap between detection and tracker estimate, as explained in section 4.2.5.

low regardless of which background feature is used for both trackers. With these feature combinations, the learning tracker did better than the non-learning tracker in more sequences than expected under the null hypothesis, but this difference is not significant.

The performance of the trained tracker and the non-learning tracker over time is shown in figure 4.5. Data is shown for one sequence recorded on path 6 in figure 4.2. Within the lab, both trackers degrade in performance over time but the non-learning tracker suffers more. The trained tracker has different models for different illumination conditions. This makes it more robust to the slowly changing conditions. When the corridor is entered around frame 180, illumination conditions

Background	Target	Learn	Base	# (Learn > Base)
RGB	Spat. Gradient	0.33	0.31	10
RGB	RGB	0.54	0.40	16
Gray Value	Spat. Gradient	0.31	0.31	12
Gray Value	RGB	0.48	0.39	11

Table 4.1: For every feature combination, mean performance of the trained and non-learning trackers over all test sequences are shown. Furthermore, the number of sequences where the trained system outperformed the non-learning system is given. The performance measure is explained in section 4.2.5.

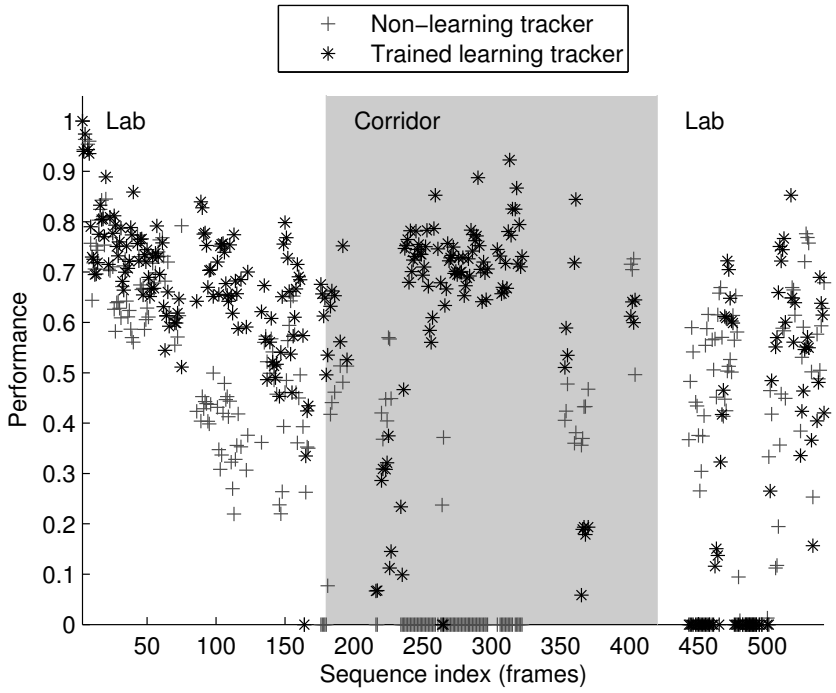


Figure 4.5: Performance of both tracker types over time evaluated on a sequence from path 6. The non-learning tracker suffers as conditions change. Performance score measures overlap between detection and tracker estimate, as explained in section 4.2.5. RGB histograms were used as target and background models.

change more abruptly and the non-learning tracker performance is 0 for a long time, while the trained tracker can switch to a different model and still has a good performance. When the lab is entered again, the initial model of the non-learning tracker fits the illumination again and it can recover, although both trackers also fail in some frames.

4.3.2 Comparing different sizes of the training set

Learning tracker performances on training sets after training on different sequences are shown in figure 4.6. The correlation coefficient r between the number of detections in the training set and the performance score of the learning tracker is 0.536. This means $z(r) = 0.602$ according to the Fisher transformation:

$$z(r) = 0.5 \ln \frac{1+r}{1-r} \tag{4.3}$$

There are 18 data points, $Z = 2.33$. This means the null hypothesis that

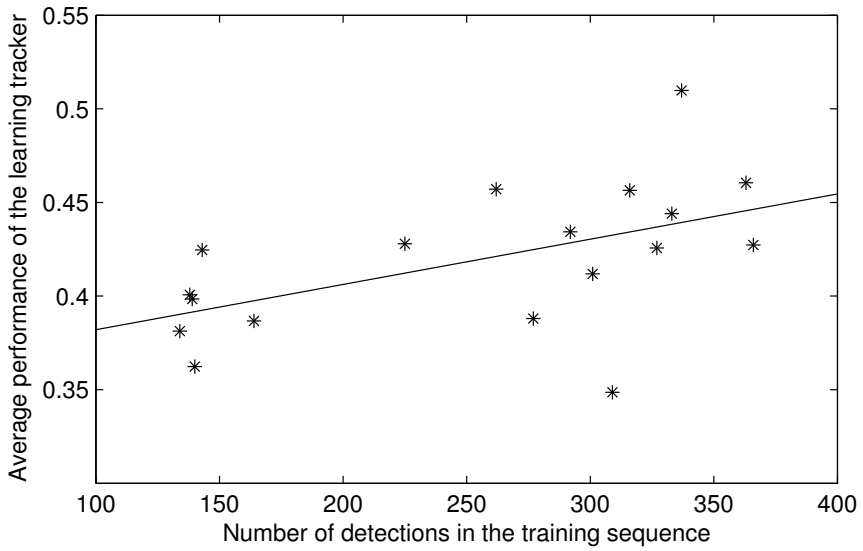


Figure 4.6: Trained learning tracker performance score depends on the amount of training data. Performance score measures overlap between detection and tracker estimate, as explained in section 4.2.5. In the figure, RGB histograms were used as target and background models. The line shows the best least-squares fit of a first-order polynomial to the data.

performance is independent of the number of detections in the training set can be rejected at the $\alpha = 0.05$ significance level.

4.4 Discussion of the learning system

The tracker developed in the earlier chapters was expanded with a learning algorithm that attempts to use the best target model given the color distribution of the current frame. Learning was added to make target models for different illumination conditions available, providing robustness against illumination changes.

Performance of the learning tracker

Different kinds of feature histograms were used to represent the target model and the background model. RGB and value histograms were used to represent the background, and RGB and spatialized gradient histograms were used to represent the target.

Using RGB features for both background and target model, the trained learning tracker did better than the non-learning tracker in a significant amount of test sequences. For both kinds of target model, using RGB histograms as background model achieved a higher average performance than using gray scale value his-

tograms as target model. Apparently, the RGB histograms were better at predicting which target model to use. The RGB histograms contain more information than value histograms, and it could be that this is why the RGB histograms are better at making distinctions between different illumination conditions.

Surprisingly, the spatialized gradient histogram as target model did not do well, regardless of background model feature. This is interesting, as this was the feature with the best performance in the previous chapter. A possible cause for this is that the dataset obtained from the autonomous robot contained much more (apparent) movement of the face to be detected than the dataset used in the previous chapters. Apparent movement is caused by movement of the person and movement of the robot itself.

The spatialized gradient features consist of one histogram for each of 16 cells in a four-by-four grid overlaid on the target. This is illustrated in figure 4.7. It is easy for cells to completely lose track of the face area they are supposed to track, while the large area of the non-spatialized features will almost always retain some overlap with the actual face, making it easier to center the tracker on the face again.

The tracker that used RGB histograms to represent both the target model and the background model did better in predicting the face location in 16 out of 18 test sequences. This is significantly different from the expected value under the null hypothesis of doing better in half of the test cases.

In a different test, performance of the learning tracker was correlated with the number of detections in the training sequence. This correlation was significantly higher than 0. So, if more pairs of background model and target model are known, the tracker performs better on average. This indicates that the learning process is indeed improving the tracker over time. Of course, not only the number of training instances are important, but also how varied those instances are. Ideally, the robot has experienced all possible illumination conditions before feedback is interrupted.

The learning algorithm

The learning tracker described in this chapter was rather simple: it used instance-based learning to associate pairs of target and background models. However, instance-based learning methods do not scale up well, as the memory requirement goes up linearly with the amount of training data. In naive implementations the running time is also directly proportional to the size of the training set. To scale this method up, unsupervised learning (clustering) could be used to group similar training instances together. This could also eliminate or at least diminish the effect of faulty training data: if the detector makes a mistake, wrong models get stored, but if these models are somehow combined with correct models made in the same conditions harmful effects could be minimized. This topic could be the focus of future work.

The described learning method is able to work continuously: it stores pairs of target and background model as feedback comes in over time, even while the robot is performing its task. Despite this, the experiment was split up in a train

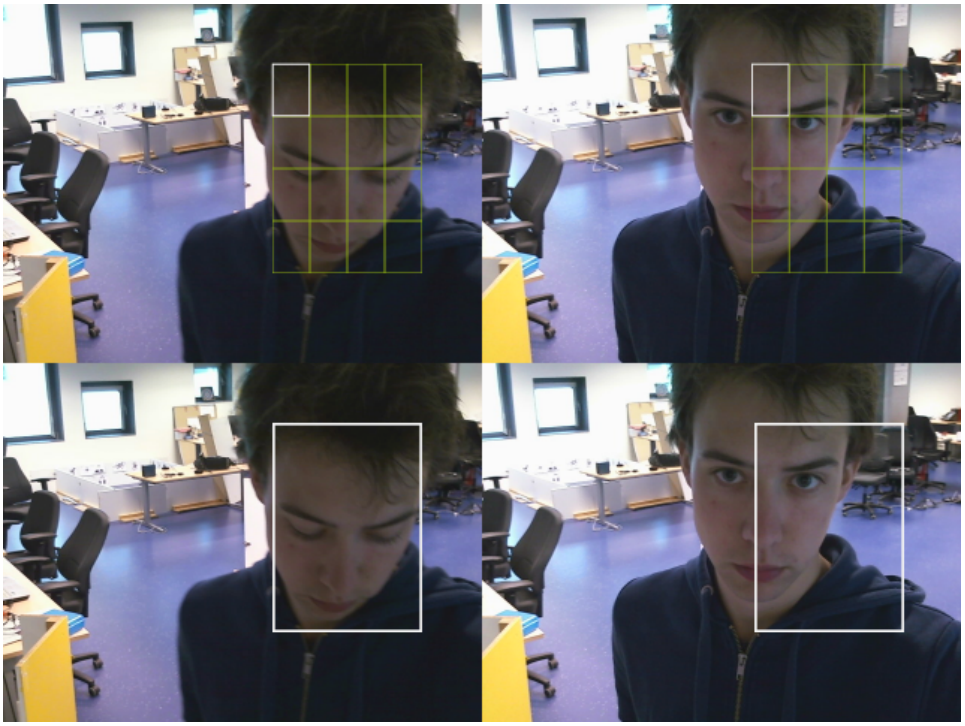


Figure 4.7: Comparing spatialized (top row) to non-spatialized features (bottom row), in a sequence with a lot of movement. The left column shows a possible detection in the previous frame, which is used in this example as prediction in the current frame (right column). For a non-spatialized feature (bottom), the prediction overlaps the true face even if there is a lot of movement between two frames. However, the upper left cell of the spatialized feature (top row) does not overlap the upper-left corner of the actual face. If none of the predictions overlaps with the corresponding region of the target, the tracker has lost the target. Images are frame 438 and 439 from one of the sequences taken at trajectory 6. 'Detections' shown are for illustration only, not the actual output of the detector system.

and a test phase, and evaluation was done on recorded sequences rather than on the working robot. This was done just to be able to evaluate the different features on the same data to get a more reliable measure of the performance of different features. However, during the data collection phase, the robot was learning while tracking the target, demonstrating continuous learning while executing the task.

Using prediction to improve tracking

In the current system, the particle filter propagation step is very simple: selected particles are updated with a Gaussian component with a mean of $\vec{0}$. This is explained in section 2.2.2. This can be refined: based on the actions executed by the robot, a prediction can be made in which direction the target will move. For example, if the robot turns left and the target does not move, the target is expected to move to the right in the visual field.

This knowledge can be integrated quite easily in the particle filter by setting the mean of the Gaussian component to the expected translation and scaling. This extension to the tracker system would improve performance as more particles would be situated near the actual location of the face, as some of the apparent movement was anticipated. There was no time to extend the system in this way, but it would be an interesting topic for future work.

Chapter 5

Discussion

This thesis has outlined that detection and tracking approaches both have their own shortcomings, and describes the creation of a hybrid system that is able to exploit the strong points of both of these methods.

5.1 Overview

The application domain for domestic robots is growing fast. Perception in a dynamic, non-standardized setting is however still problematic. Detection systems are often slow and computationally expensive, and do not work perfectly in all situations. On the other hand, tracker systems are very fast and can exploit temporal coherence, but require (manual) initialization and can lose their target over time. A detection system running at a low temporal frequency to initialize a tracker system and provide it with feedback can solve these problems.

In chapter 2, such a combined system was compared to the face detector from the OpenCV library. On an accessible dataset, the tracker could be tuned to either be significantly faster, or obtain a significantly better performance than the OpenCV face detector.

Chapter 3 explored different features and feature combination strategies. It found that a spatialized gradient feature (inspired by the SIFT descriptor) outperformed all other features under consideration on a dataset. The considered combination rules were not able to significantly outperform this feature. However, on different datasets (or in different environments) different features might perform best. For that reason, adaptive rules can be used to select the best (combination of) features for every environment. When there were non-informative features present, adaptive combination rules performed better than static rules.

Rather than evaluating the system on a dataset, the combined system was successfully used to guide a mobile autonomous robot to gather data in chapter 4. The combined system was also enhanced to be continually learning as long as it is receiving feedback. Learning should enable the system to remember the appearance of the target model in different (illumination) conditions. This enabled

the tracker part of the system to perform better than the baseline non-learning tracker when feedback was interrupted in a significant number of test sequences.

5.2 Target choice

In the described experiments, faces were used as the target of the combined system. This choice was made because for faces a suitable and accessible dataset was found. However, nothing in the tracker is made specifically for faces. The only component that is necessary is a detector system for that kind of objects. This could be based on a boosted cascade of Haar-like features, like the detector from the OpenCV library used in the experiments, but it could also be a totally different system.

Different targets might work best with different kinds of tracker features and different types of detector systems. For instance, the boosted cascade of Haar-like features is not rotationally invariant while for instance the SIFT detector is. Also, the RGB feature can be used for any target type with a (more or less) constant color, while for the spatialized gradient feature the orientation and morphology of the target should stay constant.

5.3 Dataset

In chapters 2 and 3 the systems were tested and compared on the UO Face Video Database [28]. In general, a good dataset with realistic conditions helps develop systems by providing a ground truth and a common problem set to compare different approaches to the same task. The UO Database provides these advantages, although no similar work on the same dataset was found.

However, the disadvantage of this dataset was that there was not a lot of (apparent) movement of the face over most of the sequences. Also, the number of available sequences was rather small. These features constrain the possible uses of this dataset.

5.4 Learning tracker for autonomous robots

The combined detection and tracking system enables the robot to continuously learn, and so become less dependent on the feedback system. The combined system is fast enough to run real-time on a robot, and has been used to successfully guide a robot in chapter 4. Besides learning about different kinds of environmental conditions, the continuous learning system should enable the tracker to learn about new object types on the fly if the right kind of feedback is provided. This feedback could be provided by a detector system, but also by a user. This functionality has not yet been implemented, but could be a topic for future work.

5.5 Feature type

Chapter 3 concluded that although no combination rule outperformed the spatialized gradient feature on the dataset, an adaptive ('learning') rule could select the best feature (or feature combination) for every environment. The spatialized gradient feature does not need to achieve the best performance in every environment. Indeed, in chapter 4 the RGB feature performed much better than the spatialized gradient feature, possibly due to the amount of movement in this dataset. This supports the need for adaptive combination rules, which could be the topic of more detailed future work.

5.6 Conclusion

The developed combined detection and tracking system was shown to successfully exploit temporal coherence to improve face detection. The tracker was enhanced to learn from detections over time. The learning version achieved a better performance, likely because it is more robust to changes in (illumination) conditions. These results can be applied to enhance the quality of detection systems for different kinds of targets for perceptual systems in robotics and other applications.

Exploiting temporal coherence and using learning to adapt a system to its environment are strategies that are not limited to visual tracking. Using learning to adapt to the system's environment is useful whenever the environment of the system is not known beforehand. For robots operating in non-standardized conditions, this is always the case. But when a pattern recognition or machine learning system is applied on a new dataset, the properties of the dataset are also unknown, so learning is important to adapt to these properties. Temporal coherence is present in almost any kind of temporal data, including day-to-day weather prediction, stock market data, and audio data. If this coherence is present, exploiting it will likely improve performance.

Acknowledgments

I would like to thank dr. Marco Wiering and dr. Tijn van der Zant for their ideas, advise, support, and many hours of proofreading throughout the entire duration of my project.

I want to thank the members of the BORG team for working together on the RoboCup project. Without their work on robot body design and building, design and implementation of the behavior- and vision architectures, and all other issues, it would have been much harder to perform an experiment on a real robot. And, of course, thanks for making the lab a nice and inspiring place to work! The RoboCup project was partially made possible by the Artificial Intelligence department of the University of Groningen, which provided hardware and financial support, for which I am grateful.

Preliminary work on this research was performed at the Robot Cognition Laboratory, INSERM, Lyon, France, and I thank dr. Peter Ford Dominey for support in that work. Also, I would like to thank all members of the Radical Dudes RoboCup team for working together and making my time in France a positive experience!

Finally, I would like to thank my friends and family for their support throughout this project!

Bibliography

[1] B. Gates, "A robot in every home," *Scientific American Magazine*, vol. 296, no. 1, pp. 58–65, 2007.

[2] S. Lovgren, "A robot in every home by 2020, South Korea Says," *National Geographic News*, May 2006.

[3] International Federation of Robotics, "Service robots statistics," 2010. [Online]. Available: http://www.ifr.org/service-robots/statistics/

[4] T. Wisspeintner, T. van der Zant, L. Iocchi, and S. Schiffer, "RoboCup@ Home: Results in Benchmarking Domestic Service Robots," *RoboCup 2009: Robot Soccer World Cup XIII*, pp. 390–401, 2010.

[5] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh International Conference on Computer Vision*, vol. 2, 1999, pp. 1150–1157.

[6] ——, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[7] G. Dorkó and C. Schmid, "Selection of scale-invariant parts for object class recognition," in *Proceedings of the Ninth IEEE International Conference on Computer Vision*, 2003, pp. 634–639.

[8] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1615–1630, 2005.

[9] J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid, "Local features and kernels for classification of texture and object categories: A comprehensive study," *International Journal of Computer Vision*, vol. 73, no. 2, pp. 213–238, 2007.

[10] S. Lazebnik, C. Schmid, and J. Ponce, "A maximum entropy framework for part-based texture and object recognition," in *Proceedings of the Tenth IEEE International Conference on Computer Vision*, vol. 1, 2005, pp. 832–838.

[11] C. Papageorgiou and T. Poggio, "A trainable system for object detection," *International Journal of Computer Vision*, vol. 38, no. 1, pp. 15–33, 2000.

[12] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2001, pp. 511–518.

[13] ——, "Robust real-time object detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2002.

[14] R. Lienhart and J. Maydt, "An extended set of Haar-like features for rapid object detection," in *Proceedings of the IEEE International Conference on Image Processing*, vol. 1, 2002, pp. 900–903.

[15] H. Schneiderman and T. Kanade, "A statistical method for 3D object detection applied to faces and cars," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2000, pp. 900–903.

[16] A. Mohan, C. Papageorgiou, and T. Poggio, "Example-based object detection in images by components," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 4, pp. 349–361, 2001.

[17] R. Lienhart, A. Kuranov, and V. Pisarevsky, "Empirical analysis of detection cascades of boosted classifiers for rapid object detection," *Pattern Recognition*, pp. 297–304, 2003.

[18] A. Treptow and A. Zell, "Real-time object tracking for soccer-robots without color information," *Robotics and Autonomous Systems*, vol. 48, no. 1, pp. 41–48, 2004.

[19] B. Heisele and W. Ritter, "Obstacle detection based on color blob flow," in *IEEE Intelligent Vehicles Symposium*, 1995, pp. 282–286.

[20] M. Isard and A. Blake, "Condensation-conditional density propagation for visual tracking," *International Journal of Computer Vision*, vol. 29, no. 1, pp. 5–28, 1998.

[21] K. Nummiaro, E. Koller-Meier, and L. Van Gool, "An adaptive color-based particle filter," *Image and Vision Computing*, vol. 21, no. 1, pp. 99–110, 2003.

[22] S. McKenna, Y. Raja, and S. Gong, "Tracking colour objects using adaptive mixture models," *Image and Vision Computing*, vol. 17, pp. 225–231, 1999.

[23] P. Pérez, C. Hue, J. Vermaak, and M. Gangnet, "Color-based probabilistic tracking," *Lecture Notes in Computer Science*, pp. 661–675, 2002.

[24] P. Brasnett, L. Mihaylova, N. Canagarajah, and D. Bull, "Particle filtering with multiple cues for object tracking in video sequences," in *Society*

of Photo-Optical Instrumentation Engineers Conference Series, vol. 5685, 2005, pp. 430–441.

[25] P. Brasnett, L. Mihaylova, D. Bull, and N. Canagarajah, "Sequential Monte Carlo tracking by fusing multiple cues in video sequences," *Image and Vision Computing*, vol. 25, no. 8, pp. 1217–1227, 2007.

[26] R. Verma, C. Schmid, and K. Mikolajczyk, "Face detection and tracking in a video by propagating detection probabilities," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 10, pp. 1215–1228, 2003.

[27] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, 2008.

[28] B. Martinkauppi, M. Soriano, S. Huovinen, and M. Laaksonen, "Face video database," in *Proceedings of the First European Conference on Colour in Graphics, Imaging and Vision*, 2002, pp. 380–383.

[29] R. Larsen and M. Marx, *An introduction to mathematical statistics and its applications*, 4th ed. Pearson/Prentice Hall, 2006, pp. 440–446.

[30] D. Tax, M. Van Breukelen, R. Duin, and J. Kittler, "Combining multiple classifiers by averaging or by multiplying?" *Pattern recognition*, vol. 33, no. 9, pp. 1475–1485, 2000.

[31] R. Duin, "The combining classifier: To train or not to train?" in *International Conference on Pattern Recognition*, vol. 16, 2002, pp. 765–770.

[32] J. Kittler, M. Hatef, R. Duin, and J. Matas, "On combining classifiers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 3, pp. 226–239, 1998.

[33] J. Kittler, "Combining classifiers: A theoretical framework," *Pattern Analysis & Applications*, vol. 1, no. 1, pp. 18–27, 1998.

[34] C. Shen, A. Van den Hengel, and A. Dick, "Probabilistic multiple cue integration for particle filter based tracking," in *International Conference on Digital Image Computing-Techniques and Applications*, 2003, pp. 309–408.

[35] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *IEEE International Conference on Robotics and Automation*, vol. 3, 2004, pp. 2619–2624.

[36] ——, "Machine learning for fast quadrupedal locomotion," in *Proceedings of the 19th National Conference on Artifical intelligence*, 2004, pp. 611–616.

[37] N. Littlestone and M. Warmuth, "The weighted majority algorithm," in *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, 1989, pp. 256–261.

[38] C. Chibelushi, F. Deravi, and J. Mason, "Adaptive classifier integration for robust pattern recognition," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 29, no. 6, pp. 902–907, 1999.

[39] Y. Wu and T. Huang, "Robust visual tracking by integrating multiple cues based on co-inference learning," *International Journal of Computer Vision*, vol. 58, no. 1, pp. 55–71, 2004.

[40] M. Riedmiller, J. Peters, and S. Schaal, "Evaluation of policy gradient methods and variants on the cart-pole benchmark," in *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 2007, pp. 254–261.

[41] H. Murase and S. Nayar, "Visual learning and recognition of 3-D objects from appearance," *International Journal of Computer Vision*, vol. 14, no. 1, pp. 5–24, 1995.

[42] D. Ross, J. Lim, R. Lin, and M. Yang, "Incremental learning for robust visual tracking," *International Journal of Computer Vision*, vol. 77, no. 1, pp. 125–141, 2008.

[43] T. van der Zant, L. Schomaker, and K. Haak, "Handwritten-word spotting using biologically inspired features," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1945–1957, 2008.

[44] R. Pfeifer and C. Scheier, "Sensory–motor coordination: The metaphor and beyond," *Robotics and autonomous systems*, vol. 20, no. 2-4, pp. 157–178, 1997.

[45] G. Kootstra, J. Ypma, and B. de Boer, "Active exploration and keypoint clustering for object recognition," in *IEEE International Conference on Robotics and Automation*, 2008, pp. 1005–1010.

[46] J. Lim, D. Ross, R. Lin, and M. Yang, "Incremental learning for visual tracking," *Advances in Neural Information Processing Systems*, vol. 17, pp. 793–800, 2005.

[47] Y. Wu and T. Huang, "Color tracking by transductive learning," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, 2000, pp. 133–138.

[48] T. Nakamura and T. Ogasawara, "Online visual learning method for color image segmentation and object tracking," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, 1999, pp. 222–228.

[49] A. Jepson, D. Fleet, and T. El-Maraghi, "Robust online appearance models for visual tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1296–1311, 2003.

[50] J. Dávila Chácon, T. van Elteren, B. Hickendorff, H. van Hoof, E. Jansen, S. Knuiver, C. Lier, P. Neculoiu, A. Nolte, C. Oost, V. Richthammer, F. Schimbinschi, M. Schutten, A. Shantia, R. Snijders, E. van der Wal, and T. van der Zant, "BORG - The RoboCup@Home team of the University of Groningen: Team Description Paper," 2011. [Online]. Available: http://www.ai.rug.nl/crl/uploads/Site/BORG_TDP_2011.pdf

[51] K. Ang, G. Chong, and Y. Li, "PID control system analysis, design, and technology," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 4, pp. 559–576, 2005.

[52] P. Cohen, *Empirical Methods for Artificial Intelligence*. The MIT Press, 1995, pp. 130–132.