Master Thesis: Discriminative Learning for Particle Filters in Micro-Biological Applications

Friedemann Pochert
jointly supervised by Alexander Krull, Carsten Rother, and Ivo F. Sbalzarini

December 22, 2015

Declaration of Authorship

Hereby I declare that his thesis is composed by me and based on my own work except where otherwise indicated. All direct or indirect sources used are acknowledged as references.

Friedemann Pochert

Dresden, 22th December 2015

Contents

List of Figures

List of Algorithms

List of Tables

1 Introduction

Observing the microbiological processes in the cardiovascular system of zebrafishes help to achieve greater understanding the development processes of the cardiovascular system in a living organism and the function of the single cells in this process. Large amount of microscopy recordings are gathers and need analyzing. Software tool can be designed to guide such analysis by, for instance, locating the heart muscle cells in the recordings and separating them from the background.

Biological Context The Max Planck Institute of Molecular Cell Biology and Genetics (MPI-CBG) in Dresden investigates on the smallest structures of live to find how they form complex structures and to explain their effect on living organisms. The Huisken Lab investigates on development principles in living organisms by observing them under a microscope in, for instance, zebrafishes. They aim to answer questions like: "What initializes the heartbeat?" and "How do the heart cells collaborate in this process?" Answers to these questions also help to understand the human heart [Nemtsas et al., 2010].

The zebrafish allows for investigating the cardiovascular processes with selective plane illumination microscopy (SPIM) [Huisken et al., 2004] in the intact organisms. This technique records image planes of the 3D space in which the sample is in for a time period. Therefore, this capturing process returns 4D image sequences - one 3D stack per time frame. Figure 1.1 depicts one slice out of a single 3D image stack at a certain point in time. It shows marked heart muscle cells on the tissue of the heart chambers.

Tracking Problem An important issue for the investigations on the cardiovascular system is to reconstruct the heart structure from the visible cells in the microscopy recordings to establish a connection between the cell internal process with the dynamics of the whole heart during the initialization of the heartbeat. Reconstructing the full structure needs tracking each cell in a video sequence. Tracking all cells in the recording manually is, however, a tedious task and, therefore, a software tracking tool shall be deployed to automatically locate the cells in space. A common approach to visual object tracking is to apply particle filters [Gordon et al., 1993]. Given defined models for motion and appearance of the targeted object, they analyze a video recording frame by frame and estimate the object's location based on its previous location. There are usually many

Figure 1.1: A slice out of a 3D image stack of a microscopy recording. The heart muscle cells in the upper left corner form the contour of one heart chamber. On the bottom right one can see the cells on the upper surface of the second heart chamber. The whole given recording shows such cells during a full cardiac cycle, the cells on the heart tissue move during the heartbeat.

reasonable locations where the object could have moved and, therefore, the particle filtering framework does not estimate only one location, but models a distribution of possible location based on the previous positions. Object trajectories are derived from these distribution such that the likelihood of a estimated trajectory is maximized.

This approach works quite well, if there is just one object or if different object have different appearance. When working with multiple objects, one can either run a particle filter for each targeted object and track them on their own, or one can track all objects jointly by a single tracker. In the case where multiple objects with similar appearance are tracked by independent trackers, the trackers are not able to distinguish their target from other targets. When ever two objects interact, the trackers may switch to the appearance of a neighbouring object and the original target is lost. If all objects are tracked by a single tracker, it would be possible to distinguish the cells from another and, therefore, prevent losing a target. But this approach does not scale with the amount of target objects and quickly becomes infeasible. These problems are discussed in [Khan et al., 2004]. Both approaches can not be applied to track more than 200, similar looking heart cells in the given recordings.

A solution might be to apply pseudo-independent trackers that evaluate more expressive models - models which are able to handle multiple occurrences -to solve this problem. The discriminative particle filter framework, which have been presented in [Hess and Fern, 2009], extends the common particle filtering framework such that they are able to express any arbitrary feature of the targeted object in specifically designed feature functions. Such features can define different motion models and they can describe different aspects of the object's appearance, but they may also describe interactions or relations between different objects. An important part is, that these trackers are run in parallel and all their information are shared between each other. Furthermore, the presented framework is able to automatically learn the importance of the features that are necessary to describe a specific tracking problem.

An alternative to filtering is tracking by assignment, the Feature Point Tracker [Sbalzarini

and Koumoutsakos, 2005] provides one such implementation. The trajectories of multiple similar looking objects are found by, first, detecting cell appearances independently in each frame and, afterwards, linking the detected appearances through time. The Feature Point Tracker has been successfully applied to many biological problems.

Response This thesis examines the discriminative particle filtering framework of [Hess and Fern, 2009]. It establishes the connection between the generative models of the common particle filters and explains specifically designed features that explain the heart muscle cells. An implementation as Fiji[1] plug-in is provided and tested on simulated data. This approach is evaluated on the heart cell tracking problem, the results are compared to the Feature Point Tracker - a Fiji plugin is provided by [Sbalzarini and Koumoutsakos, 2005] in the MosaicSuite[2].

The experiments show that the Feature Point Tracker detects cells more precisely and the estimated locations are closer to the true appearances in the image than the locations which are output by the implemented discriminative particle filter. However, the provided collection of feature functions help the implemented discriminative particle filter to find better trajectories for the whole video sequence, where the linking algorithm of Feature Point Tracker is distracted by appearances of the different cells and jumps between them.

Structure The basic foundations are explained in Chapter 2. This chapter covers the theory on filtering, introduces the particle filter framework, and explains the difficulties that occur when tracking multiple objects. Chapter 3 illustrates how the particle filters can be extended by discriminative feature functions as it is done in [Hess and Fern, 2009]. Chapter 4 defines specifically designed feature functions, each of them models a single characteristic of the targeted cells. After defining these feature functions they are applied on to the heart tracking problem and compared to the Feature Point Tracker in Chapter 5. At last, this thesis concludes with a summary in Chapter 6.

[1] `http://fiji.sc/Fiji`
[2] `http://mosaic.mpi-cbg.de/?q=downloads/imageJ`

2 Preliminaries

This chapter explains the basic principles and how they relate to the task of tracking heart muscle cells in a video. It covers the Hidden Markov Model that is used to explain the target objects, the Bayes filter that explains how to estimate hidden states in Hidden Markov Models, the particle filter as a sampling-based approximation to the Bayes filter, and the problem of applying particle filters to multi-object tracking.

2.1 Hidden Markov Model

The underlying model that is used in this thesis to track the heart muscle cells is the Hidden Markov Model. It describes a time-dependent process that has an inherent system state $\mathbf{x}_t$, which is not directly observable and therefore also called hidden state. It can only be captured by an indirect and noisy observation $\mathbf{y}_t$. Tracking moving objects in videos aims to estimate a sequence of object locations frame by frame. The object's time-dependent 3D locations are modeled by the hidden states $\mathbf{x}_t$. But it also needs adding more parameters to the system state, if also the appearance of each object changes over time. The heart muscle cells can be approximately pictured by a Gaussian blob with a fixed point spread function; however, the illumination intensity of two cells differs and it may even change from frame to frame. Therefore, the hidden state $\mathbf{x}_t$ becomes a 4-dimensional vector that represent the 3D location and the illumination intensity of a heart muscle cell. The observation $\mathbf{y}_t$ is the image of the video at time t.

To compute a state estimate from the observations, the Hidden Markov Model requires that the relation between a system state $\mathbf{x}_t$ and an observation $\mathbf{y}_t$ is specified. This means a probabilistic observation model $p(\mathbf{y}_t|\mathbf{x}_t)$ must be defined. This model describes how an observation looks like, given the system state is known. Also a motion model $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ of the system state must be specified in advance. It describes how the system changes over time, that is, it explains how the targeted object moves between two frames.

Figure 2.1 graphically depicts modeled state sequence. Starting at time $t = 0$ with an a-priori given state estimate, the motion model connects the hidden states in time. The initial location at time $t = 0$ is either given by the user or can also be determined by an upstream blob detector. The observation likelihood is computed by comparing

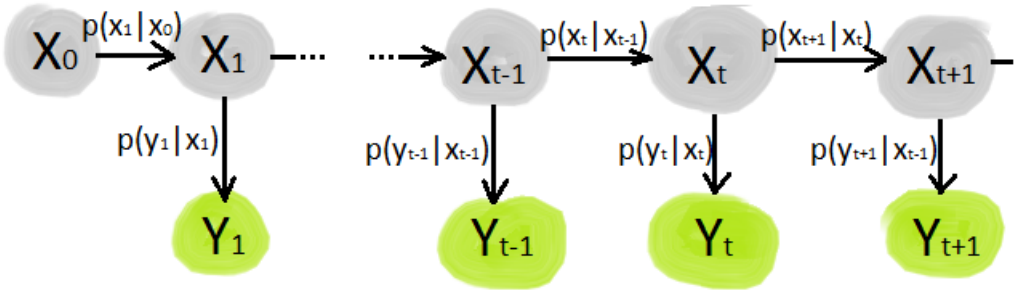


Figure 2.1: Hidden Markov Model

the observed image $\mathbf{y}_t$ with an artificial Gaussian blob which is placed at the estimated position $\hat{\mathbf{x}}_t$. Therefore it determines how well the estimate fits the image.

2.2 Recursive Bayes Filter

The recursive Bayes filter is a framework that is to sequentially determine state estimates $\hat{\mathbf{x}}_t$ by computing their probability distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t})$. Starting with the given initial location and illumination intensity the Bayes filter iterates through the video, it estimates the next state by integrating the motion $p(\mathbf{x}_t|\mathbf{x}_{t-1})$, and it evaluates the probability of the next observation $p(\mathbf{y}_t|\mathbf{x}_t)$ for each frame t. A target location and its intensity value can be estimated from the posterior distribution.

At time $t = 1$ the filter takes the probability of the initial state $p(\mathbf{x}_0)$ as prior knowledge. The posterior distribution $p(\mathbf{x}_1|\mathbf{y}_1)$ can be calculated using Bayes theorem $p(\mathbf{x}_1|\mathbf{y}_1) = \frac{p(\mathbf{y}_1|\mathbf{x}_1)p(\mathbf{x}_1)}{p(\mathbf{y}_1)}$. The probability $p(\mathbf{x}_1)$ can be calculated by marginalizing over all previous states and move them according to the motion model, i.e., $p(\mathbf{x}_1) = \int_{\mathbf{x_0}} p(\mathbf{x}_1|\mathbf{x}_0)p(\mathbf{x}_0)d\mathbf{x}_0$. The denominator $p(\mathbf{y}_1)$ depicts the probability of the observation itself. It is constant for each point in time, because the observation image is recorded and will not change. Therefore the posterior distribution of the first state estimate can be determined by $p(\mathbf{x}_1|\mathbf{y}_1) \propto p(\mathbf{y}_1|\mathbf{x}_1) \int_{\mathbf{x_0}} p(\mathbf{x}_1|\mathbf{x}_0)p(\mathbf{x}_0)d\mathbf{x}_0$.

For the following time points t the posterior distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ is still based on that initial state estimate, but also on each other observation that has been seen until t. To incorporate the previous observations the posterior likelihood at time t is given by:

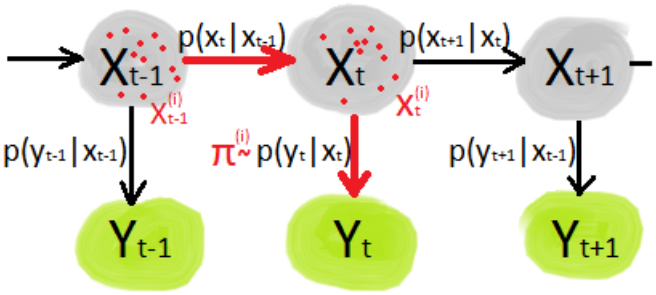


Figure 2.2: Particles in the Hidden Markov Model

$$
\begin{aligned}
p(\mathbf{x}_t|\mathbf{y}_{1:t}) &= \frac{p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{1:t-1})}{p(\mathbf{y}_{1:t})} \\
&\propto p(\mathbf{y}_t|\mathbf{x}_t)\int_{\mathbf{x}_{t-1}} p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})d\mathbf{x}_{t-1}.
\end{aligned} \tag{2.1}
$$

Since the posterior likelihood at time t is based on the previous posterior likelihood, the Bayes filter can calculate a posterior likelihood of any state recursively starting at $t = 1$.

2.3 Particle Filter

The Bayes filter requires integrating over all previous states. But when the object makes non-linear moves or has a non-linear appearance, then the posterior distribution becomes complex and therefore analytically intractable. Instead of computing the exact posterior distribution for each time t the particle filters approximate this distribution by sampling a set of particles that are to represent it. Each particle is a sample $\mathbf{x}_t^{(i)}$ of the system state and each particle has a weight $\pi_t^{(i)}$ attached. The particle weight represents the likelihood of the sampled system state according to the posterior distribution, hence it holds that $\pi_t^{(i)} \propto p(\mathbf{x}_t^{(i)}|\mathbf{y}_{1:t})$. Therefore, the whole set of weighted particles $\{(\mathbf{x}_t^{(i)}, \pi_t^{(i)})\}$ approximate the posterior distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t})$.

Just like the Bayes filter, the particle filter approximates the posterior distribution sequentially. Starting with the given initial state at the first frame, the particles are moved from frame to frame by sampling new states according to the motion model. In each frame the weights of the newly sampled particles are updated according to the current observation. However, this might lead to a situation where only few highly probable particles and many improbable particles are drawn. To avoid these situations the particles can be resampled after updating the weights. Therefore a new set of equally weighted

particles are drawn with replacement from a set of weighted particles - the particles are drawn with a probability that is proportional to their weights.

The Algorithm 1 is an implementation of this approach, which is called Sampling Importance Resampling (SIR) algorithm and which has been reviewed in [Doucet et al., 2000].

- Starting at $t = 0$ with a set of equally weighted particles $\{(\mathbf{x}_0^{(i)}, \frac{1}{N})\}_{i=1}^N$.
  1. Set $t := t + 1$.
  2. Move particles according to motion model $\hat{\mathbf{x}}_t^{(i)} \sim p(\mathbf{x}_t|\mathbf{x}_{t-1}^{(i)})$ and retrieve intermediate particles $\hat{\mathbf{x}}_t^{(i)}$.
     - Note: $\hat{\mathbf{x}}_t^{(i)}$ is intermediate, because the observation has not been considered yet.
  3. Re-weight the particles according to the observation model $\hat{\pi}_t^{(i)} \propto p(\mathbf{y}_t|\hat{\mathbf{x}}_t^{(i)})$.
  4. If necessary: Resample a set of equally weighted particles $\{(\mathbf{x}_t^{(i)}, \frac{1}{N})\}_{i=1}^N$ from the set $\{(\hat{\mathbf{x}}_t^{(i)}, \hat{\pi}_t^{(i)})\}_i^N$, where each particle is drawn with a probability that is proportional to its weight $\hat{\pi}_t^{(i)}$. Otherwise continue with the weighted particles $\{(\hat{\mathbf{x}}_t^{(i)}, \hat{\pi}_t^{(i)})\}_i^N$.

Algorithm 1: Sampling Importance Resampling (SIR)

2.4 Sampling from a Proposal Distribution

When new particles are sampled according to the motion model $p(\mathbf{x}_t|\mathbf{x}_{t-1})$, then these particles might easily be drawn from areas of low probability. To create a set of meaningful particles it is advised, according to [Rui and Chen, 2001], to draw them from a more focused proposal distribution $q(\mathbf{x}_t|\mathbf{x}_{t-1}^{(i)}, \mathbf{y}_t)$. Such distribution can take the current observation to into account and generate particle that will have higher particle weight. Therefore, the posterior distribution can be better approximated by fewer particles and the overall number of necessary particles can be reduced.

Using the proposal distribution in the Sampling Importance Resampling algorithm needs replacing Step 2 by

$$\hat{\mathbf{x}}_t^{(i)} \sim q(\mathbf{x}_t|\mathbf{x}_{t-1}^{(i)}, \mathbf{y}_t). \tag{2.2}$$

To stick to the Bayes filter framework the influence of the proposal distribution must be canceled out and the original motion must be taken into account in the particle re-

weighting step. Thus, the equation for particle re-weighting in Step 3 of Algorithm 1 must be changed to

$$\pi_t^{(i)} \propto p(\mathbf{y}_t|\mathbf{x}_t^{(i)}) \frac{p(\mathbf{x}_t^{(i)}|\mathbf{x}_{t-1}^{(i)})}{q(\mathbf{x}_t|\mathbf{x}_{t-1}^{(i)}, \mathbf{y}_t)}. \tag{2.3}$$

2.5 Multi-Object Tracking

Section 2.1 describes the underlying model and specifies the state space of a single target cell. However, the given microscopic recording of the heart contains more than 200 individual cells, which are of a similar nature. They look alike, they move alike, and they affect each other when they approach each other. In the given recording all cells are always present; no cell appears or disappears neither at the image border nor due to occlusion or any interaction.

A naive idea to track a fixed-sized set of objects is to expand the system state by the number of target cells. The system state would describe the location and intensity of each cell in separate dimensions and a single estimate of the system state would estimate the locations and intensities of all cells jointly. However, a single sample from a high-dimensional space can easily be drawn from a low-probability-area of the posterior distribution and, therefore, exponentially many particles are needed to sufficiently cover the whole distribution.

Keeping the state space small is desirable to keep the number of needed particles low. Running an independent tracker for each heart cell would keep the state space to four dimensions, as described in Section 2.1. However, any interaction between cells can not be considered by independent trackers. The estimate of an independent tracker could easily slip away when two neighbouring cells come close.

To address this issue of drifting, [Rasmussen and Hager, 2001] present a Joint Likelihood Filter (JLF) in which single targets are tracked by single target tracker, but the probabilities of their estimates are evaluated in a joint measurement process. Therefore, particles can be drawn independently from a small state space, but the interaction between cells can be considered by a joint image likelihood. This prevents that the estimates of different trackers overlap.

[Khan et al., 2004] argue that this kind of joint measurement focus only on the visual observation and does not capture information about joint behaviour of the targets, because the individual particles are drawn independently. They introduced a multi-object particle filter that has an additional interaction term in the motion model. This term is inspired by the clique potentials of Markov Random Fields (MRF); the interaction between cells can be evaluated using a penalty function $\psi(\hat{\mathbf{x}}_{i,t}, \hat{\mathbf{x}}_{j,t}) \propto \exp(-g(\hat{\mathbf{x}}_{i,t}, \hat{\mathbf{x}}_{j,t}))$,

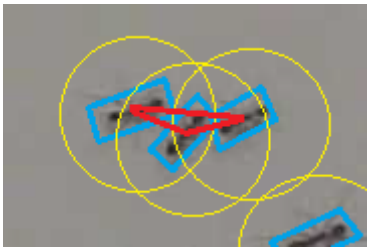


Figure 2.3: Ant tracking example from [Khan et al., 2004] for multi-object tracking.

where $\hat{\mathbf{x}}_{i,t}$ and $\hat{\mathbf{x}}_{j,t}$ are state estimates of two different targets. This function does not need to be evaluated for all targets, but only for neighbouring objects. Figure 2.3 shows an example from [Khan et al., 2004], it shows how the neighbourhood of multiple targets can be determined by circular influence area. The joint motion model need to be evaluated only for this neighbouring ants. Furthermore, it is stated that samples from the temporary joint space can be efficiently obtained using MCMC sampling techniques.

Comparing the problem of ant tracking from [Khan et al., 2004] with the problem of tracking heart muscle cells, it strikes out that an ant can move "freely," and therefore be tracked independently, until it encounters another ant. Heart muscle cells move jointly, because they are part of a steadier cellular tissue. The influence area must be large enough to capture the joint motion of cells, but once a neighbourhood of cells is defined it will not separate over time. Therefore, particles must be drawn from the joint neighbourhood during the whole time. In addition, [Hess and Fern, 2009] argue that sampling from a joint motion model always increases the computational effort, even for small groups of neighbouring objects. Their idea is to extend the independent object tracker by sharing their state estimate between all trackers. The advantage is that particles are always independently drawn, but the interaction of objects can be considered in the particle re-weighting process using the shared information. They note that their kind of pseudo-independent trackers are more restricted in the types of dependencies which can be represented.

This thesis investigates how the approach of [Hess and Fern, 2009], the pseudo-independent discriminative particle filter, can be applied to the given problem of cell tracking. The following chapter focuses on how to integrate the shared information into the re-weighting process of the pseudo-independent particle filter.

3 Discriminative Particle Filter

In in contrast to the particle filter framework, which describes an observed process by specifying the process dynamics and defining a measurement for its observation as probabilistic models, the discriminative particle filter aims to abstract from this representation by using discriminative feature functions, which are real-value functions that can describe any arbitrary characteristics of the observed process. Generalizing the models by specifying universal feature functions makes it easier to adapt the discriminative particle filter to other problems. Feature functions can be adjusted to a specific problem by weighting them. Each feature function gets a weight assigned that represents its relative importance. These feature weights are then automatically adjusted to a given problem by learning them from example data.

This chapter introduces the discriminative particle filter framework as it has been proposed in [Hess and Fern, 2009]. This approach evaluates feature functions in the particle weighting step after the new particles have been drawn. Other than that, it sticks to the particle filtering framework and calculates the posterior distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ for a given observation sequence. This approach is covered in Section 3.1.

An alternative approach to discriminative filters whose parameters can be learned are the CRF-filter, which have been presented in [Limketkai et al., 2007]. The CRF-filter generalizes from the Bayes model by replacing the underlying directed model by an undirected conditional random field (CRF). In the undirected model, the conditional likelihood $p(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})$ of the entire state sequence can be computed at once. The feature functions represent clique potentials in that CRF model. Since this filter is still based on the original Bayes model, the feature functions are divided into two kinds: On the one hand it has features that represent the transition between the process states $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$. They are called prediction potentials and replace the motion model in the Bayes filter. On the other hand it uses features to evaluate the transitions between the process states $\mathbf{x}_t$ and the observations $\mathbf{y}_t$. These are the measurement potentials and replace the observation model of the Bayes filter. The CRF-filters are not further observed in this thesis.

Both approaches weighted the features by their importance. Proper importance weights for features are found by an automatic learning process. This saves the user from the complex and error-prone manual search for optimal parameters for the selected models. To automate this learning process it needs, on the one hand, an objective that defines

what optimal parameters are and, on the other hand, a method that adjusts the parameters such that they meet this objective. Both these aspects are considered in Section 3.2 of this chapter.

At last, this chapter concludes with the composition of the filtering framework in the provided tracking plug-in in Section 3.3. This chapter only introduces the concepts of discriminative particle filters. The design of feature functions that can be applied to tracking heart muscle cells is covered in the next chapter.

3.1 Discriminative Particle Filter

Applying and learning a discriminative function that calculates the particle weights in the particle filtering framework was introduced in [Hess and Fern, 2009]. In the particle filter framework a particle's weight $\pi_t^{(i)}$ was determined by the observation likelihood $p(\mathbf{y}_t|\mathbf{x}_t^{(i)})$ as defined in Equation 2.2 and Equation 2.3. Thus, the particle weight presents the likelihood that the state estimate $\mathbf{x}_t^{(i)}$ of the given particle describes the observed process $\mathbf{y}_t$. Since the discriminative framework abstracts from that model, the particle weights do not represent the observation likelihood of this particle anymore. As denoted in this chapter's introduction, each part of the observed process can be modeled by a specific feature function. A feature function takes a particle estimate and the current observation to calculate a feature value.

Definition 1. Discriminative Feature Function [Hess and Fern, 2009]. Let $\mathbf{x}_t^{(i)}$ be a state estimation of the process state $\mathbf{x}_t$ by particle i and $\mathbf{y}_t$ an observation of this process. A feature function is an user-defined function $f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t) \in \mathbb{R}$ that compares the particle's estimate $\mathbf{x}_t^{(i)}$ with the observation $\mathbf{y}_t$ and returns a real value, which is the feature value.

Given two feature values $f_j(\mathbf{x}_t^{(k)}, \mathbf{y}_t)$ and $f_j(\mathbf{x}_t^{(l)}, \mathbf{y}_t)$, the particle with the higher feature value is a better representation of $\mathbf{y}_t$ with respect to this feature function $f_j(\cdot)$.

Denote that the observation is written as a vector which encapsulates all information that is necessary to compute the function and is not limited to the observation image at time t anymore. It might also, for instance, contain the state estimates of other objects as well as the previous particle history $\mathbf{x}_{1:t-1}^{(i)}$. This is, because there is no strong independence assumption for the features.

Given a set of feature functions it can be used to evaluate the particles weights. However, feature values of different feature functions might not be comparable and, in addition, a feature function's model might have different impact in the description of the observed

process. Therefore, each feature function is weighted according to its importance by an individual feature weight. These feature functions and their weight are combined in the discriminative particle weighting function.

Definition 2. Discriminative Particle Weighting Function [Hess and Fern, 2009]. Let $\mathcal{F}$ be a set of feature functions, $\mathbf{y}_t$ be an observation, and $\mathbf{x}_t^{(i)}$ be a particle estimate. The discriminative weight $\pi_t^{(i)}$ of the given particle is defined by:

$$\pi_t^{(i)} \quad \propto \quad \exp\left(\sum_j w_j \cdot f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t)\right). \tag{3.1}$$

Here, w_j is the weight of the feature $f_j(\cdot)$.

The relation between the common particle filters and the discriminative filters can still be established by choosing the logarithm of the observation likelihood as the only feature function and setting its feature weight to 1. Therefore, the particles are again weighted according to the observation likelihood $p(\mathbf{y}_t|\mathbf{x}_t^{(i)})$. Even when the particles were drawn from the proposal distribution this model can be expressed in terms of discriminative features. Using the logarithms of the observation model, the motion model, and the proposal distribution as discriminative features. Again, all feature functions are weighted by 1. Then particle weight is calculated according to $\pi_t^{(i)} = \exp\Big(\log p(\mathbf{y}_t|\mathbf{x}_t^{(i)}) + \log p(\mathbf{x}_t^{(i)}|\mathbf{x}_{t-1}^{(i)}) - \log q(\mathbf{x}_t^{(i)}|\mathbf{x}_{t-1}^{(i)})\Big)$, which equals Equation 2.3.

Computing the Particle Weighting Function The computation of the discriminative particle weighting function combines various kinds of feature functions, each of them can return any real value. But very large or very small feature values can easily introduce numerical problems. However, the ultimate purpose of a particle's weight is to determine the quality of this particle's estimate in relation to all other particle estimates; thus, it is sufficient to normalize the particle weights. Normalized weights are still consistent with Definition (2), because they are defined as being proportional to the particle weighting function.

The weights are normalized using Algorithm (2). First, it determines the maximal sum of weighted feature values. Second, it shifts all these values into the interval $(-\infty; 0]$ in log-space by subtracting this maximal sum. Third, the normalized weights can be computed on the the intermediate values $\tilde{\pi}_t^{(i)}$.

This algorithm shifts the intermediate values $\tilde{\pi}_t^{(i)}$ into $(-\infty; 0]$, because the floating point data types are limited to an finite range. The returned value of the exponential function

$$
\begin{aligned}
C_{max} &\leftarrow \arg\max_i \left[\sum_j w_j \cdot f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t) \right] \\
\text{for all } i: \tilde{\pi}_t^{(i)} &\leftarrow \sum_j w_j \cdot f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t) - C_{max} \\
\text{for all } i: \pi_t^{(i)} &\leftarrow \frac{\exp\left(\tilde{\pi}_t^{(i)}\right)}{\sum_k \exp\left(\tilde{\pi}_t^{(k)}\right)}
\end{aligned}
$$

Algorithm 2: Normalization of the feature weights.

might just become infinity, if the intermediate values $\tilde{\pi}_t^{(i)}$ is sufficient large. Furthermore, by the rules of the IEEE 754 arithmetic, dividing an infinite value by another infinite value returns NaN. In addition, when dealing with only very small sums of feature values, then step two of this procedure saves us from $\exp\left(\tilde{\pi}_t^{(i)}\right)$ becoming zero for all particles. This case is problematic, because then the denominator in step three evaluates to zero, and the division of zero by zero also return NaN.

Claim 1. The particle weights that are computed with Algorithm (2) are consistent with the weights that are defined in Definition (2).

A proof for this claim can be found in the appendix.

3.2 Learning Process

The learning process is to adapt the feature weights of the discriminative function automatically to the observed data. It takes annotated ground truth data and runs a discriminative particle filter. The particle filter's performance is continuously observed. Therefore, the error between the filter prediction and the ground truth data is measured. After each pass through the training data, a gradient descent step is performed on the feature weights for which they are updated such that the measured error is reduced.

This learning process requires, first, an objective that measures the filter's performance concerning the prediction of the ground truth location. Any adjustment of the parameters shall increases the particle filter's performance with respect to this objective. For example, an objective can define an error between estimate and ground truth which shall be minimized. Second, the process needs the gradients regarding the feature weight on the objective. The gradient describes the direction of the strongest increase of the objec-

tive function for a specific location. Section 3.2.1 presents three different objectives and the gradients on them. Third, a concrete update step for the feature weights is needed. These updates are calculated on the gradients such that the overall objective is better satisfied with the new parameters than before. Section 3.2.2 depicts how the update steps look like.

3.2.1 Learning Objectives and their Gradients

Four definitions follow: First, an objective that is to maximize the posterior likelihood of the ground truth on the observed data, second, an approximation of this maximum likelihood, third, an objective that is to minimize the squared error between the filter's estimation and the ground truth, and, at last, an objective that is to minimize the mean squared error between the estimations of the filter's particles and the ground truth.

Maximizing the Likelihood of the Training Data The prediction of a state sequence $\mathbf{x}_{1:T}$ of the discriminative particle filter is based on the inherent posterior likelihood $p(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})$. Given a specific observation sequence $\mathbf{y}_{1:T}$ and the related ground truth trajectory $\mathbf{x}_{1:T}^{(GT)}$, the value of the posterior likelihood $p(\mathbf{x}_{1:T}^{(GT)}|\mathbf{y}_{1:T})$ only depends on the set of unknown feature weights $\theta = (w_1, ..., w_J)$. This objective is to tune the parameters such that this posterior likelihood becomes maximal for the training data.

The Definition 3, which is given in [Sutton and McCallum, 2006], provides a density function of the posterior likelihood using discriminative feature functions. Furthermore, a detailed explanation of the relations between the naive Bayes classifier, the Hidden Markov Model, and a linear-chain Conditional Random Field and how all this leads to the given posterior likelihood can be found in the mentioned article.

Definition 3. Linear-Chain Conditional Random Field [Sutton and McCallum, 2006]. Let $\mathbf{x}_{1:T}, \mathbf{y}_{1:T}$ be random vectors, $\theta = \{w_j\} \in \mathbb{R}^J$ be a parameter vector, and $\{f_j(\mathbf{x}_t, \mathbf{y}_t)\}_{j=1}^J$ be a set of real-valued feature function. Then a linear-chain conditional random field is a distribution $p(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})$ that takes the form

$$p(\mathbf{x}_{1:T}|\mathbf{y}_{1:T}) = \frac{1}{Z(\mathbf{y}_{1:T})} \exp\left\{\sum_t \sum_j w_j f_j(\mathbf{x}_t, \mathbf{y}_t)\right\}, \tag{3.2}$$

where $Z(\mathbf{y}_{1:T})$ is an instance-specific normalization function

$$Z(\mathbf{y}_{1:T}) = \sum_{\mathbf{x}'_{1:T}} \exp\left\{ \sum_t \sum_j w_j f_j(\mathbf{x}'_t, \mathbf{y}_t) \right\}. \tag{3.3}$$

Denote again that the observation is is not only an image anymore, but also contains the particle history $\mathbf{x}^{(i)}_{1:t-1}$. Therefore, pairwise features as the particle's motion can be expressed by these feature functions $f_j(\mathbf{x}_t, \mathbf{y}_t)$, even though the they are defined by $f_j(\mathbf{x}^{(i)}_t, \mathbf{x}^{(i)}_{t-1}, \mathbf{y}_t)$ in [Sutton and McCallum, 2006]. This chosen representation allows for features which depend on many more arguments that are not listed here and, also, it is consistent with the one in [Hess and Fern, 2009].

It is mentioned in this article that optimizing the likelihood directly is a complex task. One can simplify the calculations by optimizing the conditional log-likelihood $\log p(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})$ instead, as logarithm has its maximum where also the likelihood is maximal. Applying the given observation sequence $z_{1:T}$ and the ground truth trajectory $\mathbf{x}^{(GT)}_{1:T}$ to the conditional log-likelihood, it reads as

$$\log p(\mathbf{x}^{(GT)}_{1:T}|\mathbf{y}_{1:T}) \quad = \quad \sum_t \sum_j w_j f_j(\mathbf{x}^{(GT)}_t, \mathbf{y}_t) - \sum_i \log Z(\mathbf{y}_{1:T}). \tag{3.4}$$

The gradient with respect to the feature weight w_j reveals as

$$\begin{aligned} \frac{\delta \log p(\mathbf{x}^{(GT)}_{1:T}|\mathbf{y}_{1:T})}{\delta w_j} &= \sum_t f_j(\mathbf{x}^{(GT)}_t, \mathbf{y}_t) - \sum_{\mathbf{x}'_{1:T}} \sum_t f_j(\mathbf{x}'_t, \mathbf{y}_t) p(\mathbf{x}'_t|\mathbf{y}_t) \\ &= \mathbb{E}_{GT-data}\left[f_j(\mathbf{x}_{1:T}, \mathbf{y}_{1:T})\right] - \mathbb{E}_{model}\left[f_j(\mathbf{x}_{1:T}, \mathbf{y}_{1:T})|\mathbf{y}_{1:T}\right]. \end{aligned} \tag{3.5}$$

This gradient can be interpreted as the balance between the expected value of the feature function on the modeled distribution and the expected value for the ground truth data, which is the empirical distribution. When the difference becomes zero, then the parameter is optimally adjusted.

Approximated Maximum Likelihood The posterior likelihood $p(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})$ defines the joint probability for an entire observation sequence and all process states in time. Particle filter, as a realization of the Bayes filter, calculate the posterior distribution step-by-step for each point t in time; starting at $t = 0$ it runs until $t = T$. Moreover, in [Hess and Fern, 2009] the posterior likelihood of a specific target i at the time t is given as

$$
\begin{aligned}
p(\mathbf{x}_t^{(i)}|\mathbf{y}_t) &= \frac{1}{Z(\mathbf{y}_t)} \exp\left\{\sum_j w_j f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t)\right\}, \\
\text{where } Z(\mathbf{y}_t) &= \sum_{\mathbf{x}'_t} \exp\left\{\sum_j w_j f_j(\mathbf{x}'_t, \mathbf{y}_t)\right\}.
\end{aligned} \tag{3.6}
$$

That is, the likelihood of a particle is solely determined by its estimate $\mathbf{x}_t^{(i)}$ and the single observation $\mathbf{y}_t$.

As mentioned in this section's introduction, the parameters are to be tuned on-the-fly while running a particle filter. Therefore, the parameters of posterior likelihood is optimized step-wise, that is, the gradient is computed on only one training sample at a time. This is "a drastic simplification" [Bottou, 2012] of the Gradient 3.5, which is used for stochastic gradient optimization. The stochastic gradient with respect to the parameter w_j is, just as the gradient on the joint posterior likelihood, formulated on the log-likelihood of the training data:

$$
\begin{aligned}
\frac{d \log p(\mathbf{x}_t^{(GT)}|\mathbf{y}_t)}{dw_j} &= f_j(\mathbf{x}_t^{(GT)}, \mathbf{y}_t) - \sum_{x'_t} f_j(\mathbf{x}'_t, \mathbf{y}_t) p(\mathbf{x}'_t|\mathbf{y}_t) \\
&= f_j(\mathbf{x}_t^{(GT)}, \mathbf{y}_t) - \mathbb{E}_{model}\left[f_j(\mathbf{x}'_t, \mathbf{y}_t)|\mathbf{y}_t\right].
\end{aligned} \tag{3.7}
$$

The Perceptron update in [Hess and Fern, 2009] further simplifies this gradient. It is proposed to approximate the estimation of the feature value in Equation 3.7 by the feature value of the state's MAP estimate $\mathbf{x}_t^{(MAP)}$. This idea is based on the Perceptron update in [Collins, 2002], which they use to optimize the parameters of a maximum-entropy tagger. Also in [Limketkai et al., 2007] this Perceptron update is used to approximate the gradient of the conditional log-likelihood. The characterized gradient approximation is

$$
\frac{d \log p(\mathbf{x}_{1:T}^{(GT)}|\mathbf{y}_{1:T})}{dw_j} \approx f_j(\mathbf{x}_t^{(GT)}, \mathbf{y}_t) - f_j(\mathbf{x}_t^{(MAP)}, \mathbf{y}_t). \tag{3.8}
$$

Minimizing the Squared Residual Error of the Mean The second objective that is presented in [Hess and Fern, 2009] focuses on tuning the performance of the particle filter directly. The particles of the particle filter estimate a cell's location and its illumination

over time. The filter's estimate $\mathbb{E}[\mathbf{x}_t]$ can be computed by averaging the on the particles' estimated locations and illumination values. The ground truth trajectory contains the true location and illumination intensities of the targeted cell. Intuitively, a particle filter shall be configured such that it predicts the true location and the true illumination in each frame.

The distance between these two, the predicted state $\mathbb{E}[\mathbf{x}_t]$ and the ground truth $\mathbf{x}_t^{(GT)}$, defines the residual error $\epsilon_r = ||\mathbb{E}[\mathbf{x}_t] - \mathbf{x}_t^{(GT)}||$. The objective is to adjust the parameters such that squared residual error becomes minimal:

$$\mathbf{w}_{opt} = \arg\min_{\mathbf{w}} ||\mathbb{E}[\mathbf{x}_t] - \mathbf{x}_t^{(GT)}||^2. \tag{3.9}$$

The parameter vector $\mathbf{w}_{opt} = (w_1, ..., w_J)$ contains all feature weights, but each weight is optimized on its own. The optimum is found by searching in the negative direction of the gradient using gradient descent. The gradient with respect to the parameter w_j is

$$\begin{aligned} \frac{d||\mathbb{E}[\mathbf{x}_t] - \mathbf{x}_t^{(GT)}||^2}{dw_j} &= \left(\mathbb{E}[\mathbf{x}_t] - \mathbf{x}_t^{(GT)}\right)^\top \\ &\quad \cdot \left(\mathbb{E}[\mathbf{x}_t \cdot f_j(\mathbf{x}_t, \mathbf{y}_t)] - \mathbb{E}[\mathbf{x}_t] \cdot \mathbb{E}[f_j(\mathbf{x}_t, \mathbf{y}_t)]\right). \end{aligned} \tag{3.10}$$

Minimizing Mean Squared Error In [Hess and Fern, 2009] it is argued that the residual error is based only on the estimation which is computed on all particles, but the performance of a single particle on its own is not considered by this error. The residual error $\epsilon_{res} = ||\mathbb{E}[\mathbf{x}_t] - \mathbf{x}_{GT,t}||^2$ is equal to $\mathbb{E}[||\mathbf{x}_t - \mathbf{x}_{GT,t}||^2] - \widehat{var}(\mathbf{x}_t)$, which is again the mean squared error of the particles minus the variance of the particles. The minimum of the squared error can be found where the mean squared error and the variance are in balance and cancel each other out. This allows the particles of the discriminative filter to be distributed far away from the ground truth as long as their center of mass is close to the ground truth.

Since each of the particles shall represent a valid estimation of the object's location, they propose to optimize the mean squared error of all particles instead. This requires that each particle is close to the ground truth to meet this goal. The optimal parameters are then found by computing

$$\mathbf{w}_{opt} = \arg\min_{\mathbf{w}} \mathbb{E}[||\mathbf{x}_t - \mathbf{x}_{GT,t}||^2]. \tag{3.11}$$

The derivation with respect to each weight w_j yields the gradient[1]

[1] The derivation can be found in the Appendix 6.

$$\frac{d\mathbb{E}[||\mathbf{x}_t - \mathbf{x}_{GT,t}||^2]}{dw_j} = \mathbb{E}[||\mathbf{x}_t - \mathbf{x}_{GT,t}||^2 \cdot f_j(\mathbf{x}_t, \mathbf{y}_t)] - \mathbb{E}[||\mathbf{x}_t - \mathbf{x}_{GT,t}||^2] \cdot \mathbb{E}[f_j(\mathbf{x}_t, \mathbf{y}_t)] \quad (3.12)$$

3.2.2 Update Step

Once the objective is defined and the gradients are derived, the feature weight update can be computed. Minimizing an error needs adjusting the weights in the negative direction of the gradient; that is, the new weight is determined by $w_j \leftarrow w_j - \gamma \cdot \nabla_{w_j} \epsilon$. Maximizing a probability needs adjusting the parameters in the direction of the positive gradient, which is setting $w_j \leftarrow w_j + \gamma \cdot \nabla_{w_j} p(...)$. In both cases the factor γ is a learning rate that scales an update step to a reasonable size.

- Starting at $t = 0$ with a set of equally weighted particles.
- For each frame t:
  1. Move particles according to motion model/proposal distribution.
  2. Re-weight the particles according to particle re-weighting function.
  3. Compute estimate from set of particles.
  4. If $\epsilon = ||\mathbb{E}[\mathbf{x}_t] - \mathbf{x}_t^{(GT)}|| > \tau_{update}$:
     - Update parameters according to either $w_j \leftarrow w_j - \gamma_t \cdot \nabla_{w_j} \epsilon$ or $w_j \leftarrow w_j + \gamma_t \cdot \nabla_{w_j} p(...)$.
  5. If $\epsilon = ||\mathbb{E}[\mathbf{x}_t] - \mathbf{x}_t^{(GT)}|| > \tau_{reset}$:
     - Re-initialize particles with ground truth.

Algorithm 3: Sampling Importance Resampling (SIR) with parameter tuning as in [Hess and Fern, 2009].

In the training algorithm, which is given in [Hess and Fern, 2009], this kind of update may be applied to the parameters in every frame after the particle filter estimated a new state, but only if the error between this estimate and the ground truth is larger than a specified threshold τ_{update} (see modification to the SIR algorithm in Algorithm 3). To prevent that the tracker adjusts the parameters to estimates which are too far away from the original target, the tracker is reset to the latest ground truth location.

These kind of frame-wise updates are motivated by the Perceptron update and are similar to stochastic gradient updates. In [Bottou, 2012] it is argued that stochastic gradient

methods can only be applied successfully when updates are computed on randomly selected training examples. However, the sequential filters do not allow for picking random frames and updating the parameters only according to them. Furthermore, since the given video sequence is not exceedingly long, it is reasonable to accumulate the frame-wise gradients and delay the update until the whole training video has been processed. A single run through the whole training video is considered as one training epoch. The resulting epoch-wise update is given by

$$w_{j,k+1} = w_{j,k} - \gamma_k \cdot \frac{1}{N} \sum_{t}^{T} \sum_{i}^{N} \nabla_{w_j} \epsilon, \tag{3.13}$$

where k indicates the number of training epochs. This follows the gradient update step that is given in [Bottou, 2012]. Denote that this given update is to minimize an error function ϵ; to maximize a probability the sign must be changed. The learning rate γ_k is adjusted to control the convergence speed such that the parameters can smoothly approach to their optimal value. It is determined by $\gamma_k = \frac{\gamma_0}{1+k\gamma_0\lambda_{decay}}$ and therefore decrease with each training epoch.

3.3 Tracking Plug-In

Algorithm 4 summarizes this chapter and depicts the program flow that has been implemented as ImageJ plug-in. It is based on Janick Cardinale's PFTracking3D plugin that can be found in [Cardinale, 2008]. The training process is repeated for a fixed number of epochs. For each epoch the training video is processed and the gradients for the updates are computed, afterwards all feature weights w_j are updated using these gradients, and at last the new feature weights are evaluated on a set of validation videos.

The object trackers for all target objects are processed independently frame by frame, but previous estimates are shared can be accessed by the feature functions $f_j(\cdot)$ as it has been explained after Definition 1. Once all particles have been moved and evaluated, the trackers estimate can be computed on them and its distance to the ground truth can be measured. As in the Algorithm 3, the gradients are calculated only if the error is above a certain threshold, which motivated by the Perceptron updates that focus only on correcting errors. However, unlike Algorithm 3 the gradients are stored for the single update step as explained in Section 3.2. The second if-clause is to reset a tracker in case it drifted completely off.

After adjusting all feature weights $w_j \in \mathbf{w}$ the validation videos are processed using the updated weights.

for $k = 1, \ldots,$ *max. epochs* **do**
  $t \leftarrow 1$
  for *each target object obj* **do**
    for *each particle i* **do**
      $\mathbf{x}_{t,obj}^{(i)} \leftarrow \mathbf{x}_{t,obj}^{(GT)}$
    end
  end
  for *each frame t in training video* **do**
    for *each target object obj* **do**
      for *each particle i* **do**
        $\mathbf{x}_{t,obj}^{(i)} \sim q(\mathbf{x}_{t,obj}|\mathbf{x}_{t-1,obj}^{(i)}, \mathbf{y}_t)$
        $\pi_{t,obj}^{(i)} \propto \exp\left(\sum_j w_j \cdot \ f_j(\mathbf{x}_{t,obj}^{(i)}, \mathbf{y}_t)\right)$
      end
      $\epsilon = |\mathbb{E}[\mathbf{x}_{t,obj}|\mathbf{y}_t] - \mathbf{x}_t^{(GT)}|$
      if $\epsilon > \lambda_{update}$ **then**
        $\mathbf{g}_{t,obj} \leftarrow \nabla_{\mathbf{w}}\epsilon$
      end
      if $\epsilon > \lambda_{reset}$ **then**
        for *each particle i* **do**
          $\mathbf{x}_{t,obj}^{(i)} \leftarrow \mathbf{x}_{t,obj}^{(GT)}$
        end
      end
    end
  end
  $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \gamma_k \cdot \frac{1}{N} \sum_t^T \sum_{obj}^N \mathbf{g}_{t,obj}$
  ValidateAndMeasureError($\mathbf{w}$, validation videos)
end

Algorithm 4: Implementation of the discriminative particle filter.

4 Discriminative Features that Model the Heart Muscle Cells

The problem at hand is to track heart muscle cells in 3D microscopic images. Figure 4.1 shows 4 example z-slice out of the given image stack. The two heart chambers are outlined by their muscle cells. These cells appear as blobs in 3D space and move in 3D space. The heart chambers expand quickly, then rest for a brief moment before they contract. At the end they are back in their initial position. By that, the heart muscle cells change their motion during the different stages of the cardiac cycle. The cells shall be tracked through the whole cycle and shall be separated from their neighbours during the whole time.

This chapter describes the discriminative feature functions that model the problem. It will cover features that explain the observation of a single cell and observations of a neighbourhood, features that describe interaction between cells which therefore create a neighbourhood, and features that express the motion of single cells and neighbourhoods.

It is important to remember that the feature functions, which are described in this chapter, adhere to Definition 1. That is, each feature is a function of the form $f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t)$ that takes a particle estimate $\mathbf{x}_t^{(i)}$, which is to be evaluated, and any observation $\mathbf{y}_t$ to return a real feature value. Denote that the observation $\mathbf{y}_t$ contains not only the current observation image, but also the current object's particle history and the previous estimates of all other objects that are shown in the video.

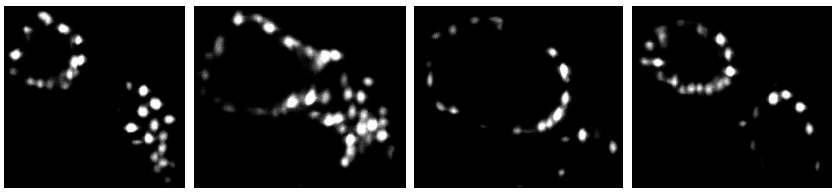


Figure 4.1: Example data: These images shows one z-slice out of the image stack different times during the cardiac cycle.

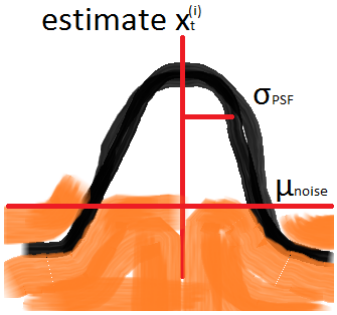


Figure 4.2: Sketch of the artificial image of a single cell.

4.1 Observation Features

A generative observation model compares an artificially generated image of the targeted cell to the appearance in the given image. The targeted heart cell is approximated by a Gauss blob with a known point spread function (PSF), its the center and the illumination of this blob is determined by the state estimate. The variance of the Gauss blob is fixed by the point spread function. The observation image contains additional noise, which is not drawn into the generated images. To compensate this noise in the artificial image, a constant background value is added. This value is set to the mean of background noise in the observation image. Figure 4.2 depicts the generation of an artificial observation for a given state estimate.

The observation likelihood $p(\mathbf{y}_t|\mathbf{x}_t^{(i)})$ determines how well the original observation is represented by the particle estimate. Therefore, it compares the original observation $\mathbf{y}_t$ with the artificially generated image $\mathbf{g}(\mathbf{x}_t^{(i)})$, which is expressed by the normal distribution

$$p(\mathbf{y}_t|\mathbf{x}_t^{(i)};\sigma_{observation}) = \frac{1}{\sqrt{2\pi\sigma_{observation}^2}} \exp\left(-\frac{1}{2\sigma^2}\sum_k (y_{t,k} - g_k(\mathbf{x}_t^{(i)}))^2\right). \quad (4.1)$$

Here, $g_k(\mathbf{x}_t^{(i)})$ is the k^{th} pixel of the generated image $\mathbf{g}(\mathbf{x}_t^{(i)})$ and $y_{t,k}$ is the k^{th} pixel of the observation image $\mathbf{y}_t$. Each pixels of an image is considered to be conditionally independent from the other pixels of that image given the state $\mathbf{x}_t$.

Single-Object Observation Feature To express this observation likelihood in terms of a feature, first, it is translated into the log-likelihood:

$$\ln p(\mathbf{y}_t|\mathbf{x}_t^{(i)};\sigma) = -\frac{1}{2\sigma^2}\sum_k (y_{t,k} - g_k(\mathbf{x}_t^{(i)}))^2 - \ln C(\sigma). \quad (4.2)$$

That is, because feature functions are evaluated in the exponent of the particle weighting function, see Definition 2, and therefore a log-likelihood feature reflects the role of the original likelihood.

Second, to simplify the feature function the term $-\ln C(\sigma)$ is omitted in the feature function. In the likelihood function, this term normalizes the whole function such that the area beneath sums up to 1. However, the feature value can be any real value and it is only necessary that it maintains the relation of original likelihood between two particle estimates.

At last, the remaining term can be expressed by the feature function

$$f_{single\,observation}(\mathbf{x}_t^{(i)}, \mathbf{y}_t) = -\frac{1}{2}\sum_k (y_{t,k} - g_k(\mathbf{x}_t^{(i)}))^2. \tag{4.3}$$

The parameter σ is pulled out and, therefore, it is represented by the feature weight $w_{single\,obsv} = \frac{1}{\sigma^2_{obsv}}$, which is part of the particle weighting function. Since the parameters w_j in the particle weighting function are to be adjusted by parameter learning, the optimal σ of the observation log-likelihood can now be obtained from data.

Clipped Observation Feature The generated image of a cell $\mathbf{g}(\mathbf{x}_t^{(i)})$ shows a single cell in an image that is of the same size as the original observation. However, as the Gaussian blob is only apparent within a close range around the estimate $\mathbf{x}_t^{(i)}$ and the remaining image is almost constantly filled with the value of the background noise μ_{noise}. This introduces two problems for the implementation, which can be solved using a clipped version of the observation feature.

The first problem is that the comparison of the constant noise model μ_{noise} to the real noise in the observation $\mathbf{y}_t$ adds up many differences. This does not express how well the estimated blob fits to the observed cell, but only how bad the simple noise model explains the observed noise and other unexplained objects. Therefore, the summed difference can easily become very large. Even though the feature values can be any real value, working with very large or very small values lead to numerical problems in the implementation.

The second problem is that calculating the sum of differences needs looping over all image pixels. Therefore, the computation of the observation feature for the whole 3D image takes very long time.

The following solution computes the observation image only for a small section around the particle estimate. It does not punish any particle for lying too close to another unexplained observation or for different noise in this image section. The size of such image section can be fixed such that it covers, for instance, $4 \cdot \sigma_{PSF}$ of the Gauss blob

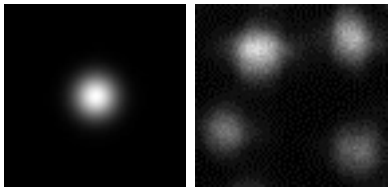


Figure 4.3: Image of a single Gaussian blob and excerpt from an observation image.

in any direction from its center. Outside of this section, the influence of the blob is negligible small.

The image section around the particle estimate is determined by the bounding box $\mathbf{B}(\mathbf{x}_t^{(i)})$. If a pixel is inside this bounding box, the function $g_k(\mathbf{x}_t^{(i)})$ creates the Gaussian blob image as described before. If the pixel is outside this box and the function returns only the constant background noise μ_{noise}. When applying this bounding box to the single observation feature, then it translates to

$$f(\mathbf{x}_t^{(i)}, \mathbf{y}_t) = -\frac{1}{2} \sum_{k \in \mathbf{B}(\mathbf{x}_t^{(i)})} (y_{t,k} - g_k(\mathbf{x}_t^{(i)}))^2 + \sum_{k \notin \mathbf{B}(\mathbf{x}_t^{(i)})} (y_{t,k} - \mu_{noise})^2. \tag{4.4}$$

The next step is to make the second term independent from a particle's estimate. It can be computed once for the whole image and afterwards subtracted again only for the pixels in the bounding box $\mathbf{B}(\mathbf{x}_t^{(i)})$ as follows

$$f(\mathbf{x}_t^{(i)}, \mathbf{y}_t) = -\frac{1}{2} \sum_{k \in \mathbf{B}(\mathbf{x}_t^{(i)})} \left[(y_{t,k} - g_k(\mathbf{x}_t^{(i)}))^2 - (y_{t,k} - \mu_{noise})^2 \right] + \sum_{y_{t,k} \in \mathbf{y}_t^{(i)}} (y_{t,k} - \mu_{noise})^2. \tag{4.5}$$

The last term $c(\mathbf{y}_t) = \sum_k (y_{t,k} - \mu_{noise})^2$ is a constant term in the log-likelihood feature and a factor in the observation likelihood. It is shared by all particles. To speed-up the computations and to decrease the feature values, this term is removed from the clipped observation feature:

$$f_{clipped\,single\,observation}(\mathbf{x}_t^{(i)}, \mathbf{y}_t) = -\frac{1}{2} \sum_{k \in \mathbf{B}(\mathbf{x}_t^{(i)})} \left[(y_{t,k} - g_k(\mathbf{x}_t^{(i)}))^2 - (y_{t,k} - \mu_{noise})^2 \right]. \tag{4.6}$$

Multi-Object Observation Feature The observation feature for single cells compares just a single blob image with an observation that contains multiple objects. Thus, the particles may try to explain other appearances by trying to cover multiple cells (e.g., lying in between them) or just slip away to the next best appearance. To avoid such errors it is necessary to consider other appearances in the artificial images.

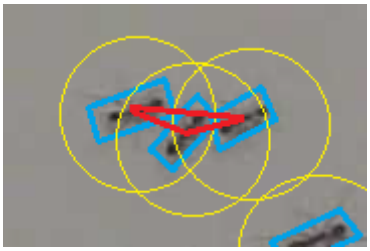


Figure 4.4: Ant tracking using interaction potentials, from [Khan et al., 2004]

Let assume the estimates of all the other object's particle filters are known at time t. They might be either approximated by their previous estimates at time $t-1$ or estimated by taking their previous motion into account. Details on such estimations are postponed to Section 4.4. The given estimated locations of the other objects can be considered in a multi-object observation feature just as the background noise is taken into account in the single observation feature. That means that foreign appearances are removed from the observation and each particle must only explain the "cleared" observation:

$$f_{multi-object\,observation}(\mathbf{x}_t^{(i)}, \mathbf{y}_t) = \tag{4.7}$$
$$-\frac{1}{2} \sum_{k \in \mathbf{B}(\mathbf{x}_t^{(i)})} \left[(y_{t,k} - g'_k(\mathbf{x}_t^{(i)}))^2 - (y_{t,k} - \mu_{noise} - \sum_{l \in \mathcal{N}(\mathbf{x}_t^{(i)})} h_k(\mathbb{E}[\mathbf{x}_{l,t}]))^2 \right].$$

In this equation, $\mathbf{x}_{l,t}$ is understood as the state of object l at time t and $\mathbb{E}[\mathbf{x}_{l,t}]$ is its estimate, which is computed by the particle filter that is associated to object l. The function $h_k(\mathbf{x}_t)$ generates the pixel value that explains the Gaussian blob for the input state $\mathbf{x}_t$. In contrast to function $g_k(\mathbf{x}_t^{(i)})$, it does not model the background noise. The observation function $g_k(\mathbf{x}_t)$ of a single object is extended to $g'_k(\mathbf{x}_t^{(i)})$ such that it explains other objects by their estimates, i.e., $g'_k(\mathbf{x}_t^{(i)}) = h_k(\mathbf{x}_t^{(i)}) + \sum_{l \in \mathcal{N}(\mathbf{x}_t^{(i)})} h_k(\mathbb{E}[\mathbf{x}_{l,t}]) + \mu_{noise}$. The set $\mathcal{N}(\mathbf{x}_t^{(i)})$ determines all objects in the neighbourhood of the respective particle estimate $\mathbf{x}_t^{(i)}$. Since the feature function is clipped by the bounding box $\mathbf{B}(\mathbf{x}_t^{(i)})$, it is sufficient to consider only objects whose bounding boxes overlap.

4.2 Interaction Features

The multi-object observation feature aims to explain object appearances, which can not be explained with a particle estimate, by taking external knowledge into account. It removes the unexplained appearances from the observation images so that the particles can be evaluated only on the actual target object. However, it needs precise estimates of the neighbouring cells to get an accurately cleared observation image. An alternative

to the multi-object observation feature is to model the interactions between cells. This approach is more robust to small deviations as interaction features are not evaluated on the image, but only on distances between a particle's estimate and the estimates of neighbouring objects.

Using interaction potentials has been proposed in [Khan et al., 2004] to enhance the joint motion model of interacting ants. It considers the single motions of each ants, but additionally aims to distinguish the ants from each other. Therefore, the joint motion likelihood is given as

$$p(x_t|x_{t-1}) \propto \prod_{n \in Objects} p(x_{n,t}|x_{n,t-1}) \prod_{n,m \in Objects} \varphi_{interact}(x_{n,t}, x_{m,t}). \tag{4.8}$$

The interaction term $\prod_{n,m \in Object} \varphi_{interact}(x_{n,t}, x_{m,t})$ is motivated by the edge-potentials of a Markov Random Field (MRF) and they are defined such that overlapping motions are punished. However, it is not necessary to generate an MRF that connects all visible objects, but only neighbouring, i.e., interacting objects. Figure 4.4 depicts an example from [Khan et al., 2004]. The yellow circles define an influence area around each ant; the neighbourhood is defined by ants that enter the areas of other ants. The red lines sketch the connected edges of the MRF that is created for this neighbourhood. There is no edge connecting the bottom-right ant, as it is too far from the thee ants in the center.

Edge Potential as Pseudo-Independent Interaction Feature The edge potentials are defined by

$$\varphi_{interact} \propto \exp(-g(x_{n,t}, x_{m,t})), \tag{4.9}$$

where $g(x_{n,t}, x_{m,t})$ is a penalty function. The interaction feature is based on this interaction potential. However, it does not create the MRF to sample from a joint motion, since the discriminative particle filter is defined as pseudo-independent filter. This feature is used to evaluate single particles $\mathbf{x}_t^{(i)}$ by their distance to the neighbouring object estimates $\mathbb{E}[\mathbf{x}_{l,t}]$ and it is defined as follows:

$$f_{interaction}(\mathbf{x}_t^{(i)}, \mathbf{y}_t) = - \sum_{l \in \mathcal{N}(\mathbf{x}_t^{(i)})} g(\mathbf{x}_t^{(i)}, \mathbb{E}[\mathbf{x}_{l,t}]). \tag{4.10}$$

Similar to the influence region in the ant example, the neighbourhood $\mathcal{N}(\mathbf{x}_t^{(i)})$ of a particle can be defined a radius around this particle in 3D space. The feature functions are evaluated in the exponent of the particle re-weighting function and, therefore, the interaction feature constitutes the interaction term of the joint motion model in Equation 4.8.

Penalty Functions The penalty function $g(\mathbf{x}_t^{(i)}, \mathbf{x}_{l,t})$ can be any function that assesses the particle location $\mathbf{x}_t^{(i)}$ and punishes it when it comes close to another object's location

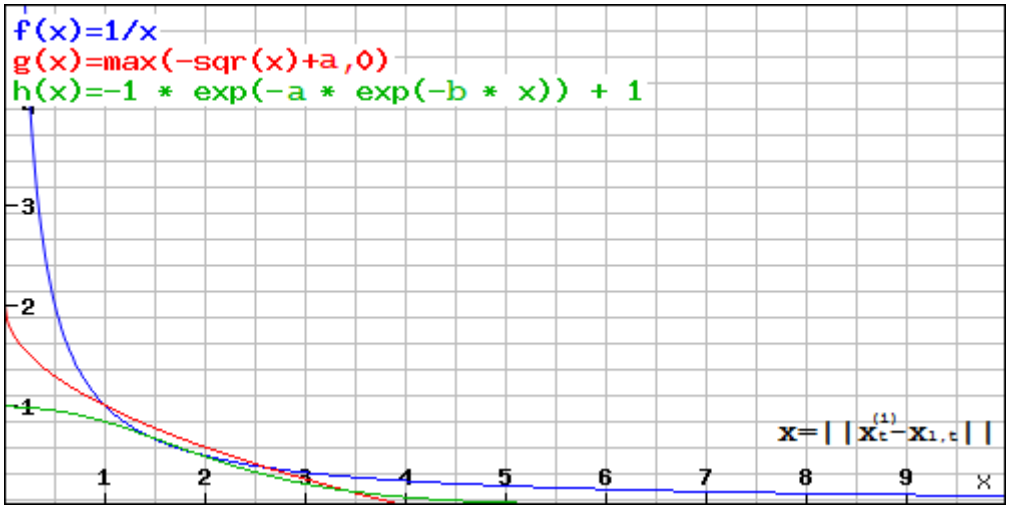


Figure 4.5: Three penalty functions.

$\mathbf{x}_{l,t}$. It shall not punish any cells that is outside of the influence area. The following functions have been implemented in the heart cell tracker and are depicted in Figure 4.5:

- $g_1(\mathbf{x}_t^{(i)}, \mathbf{x}_{l,t}) = \frac{1}{||\mathbf{x}_t^{(i)} - \mathbf{x}_{l,t}||}$,
- $g_2(\mathbf{x}_t^{(i)}, \mathbf{x}_{l,t}) = \max(-\sqrt{||\mathbf{x}_t^{(i)} - \mathbf{x}_{l,t}||} + a, 0)$,
- $g_3(\mathbf{x}_t^{(i)}, \mathbf{x}_{l,t}) = -1 \cdot \exp(-a \cdot \exp(-b \cdot ||\mathbf{x}_t^{(i)} - \mathbf{x}_{l,t}||)) + 1$ (Gompertz sigmoid function).

For all these functions, the feature weight scales their amplitude.

4.3 Motion Features and Proposal Distribution

Next to its appearance in the observation and its interaction with neighbouring objects, a particle can be evaluated by its motion between two frames. A simple approach is to look at the difference between the current particle state $\mathbf{x}_t^{(i)}$ and its state $\mathbf{x}_{t-1}^{(i)}$ at time $t-1$. However, it is also possible to predict the current object's location and then compare the particles estimate $\mathbf{x}_t^{(i)}$ with this prediction $\mathbf{x}_t^{(guess)}$. In this section presents two different motion features.

Random Motion Feature The random motion model is based on the assumption that a particle can move from its position previous $\mathbf{x}_{t-1}^{(i)}$ in any direction to its new location $\mathbf{x}_t^{(i)}$ in the current frame. The probability of the particle's location decreases the farther away it is from the location $\mathbf{x}_{t-1}^{(i)}$.

This feature evaluates the Gaussian log-likelihood of the motion:

$$f_{random\,motion}(\mathbf{x}_t^{(i)}, \mathbf{y}_t) = -\frac{1}{2}\sum_{k=1}^{3}(x_{k,t}^{(i)} - x_{k,t-1}^{(i)})^2, \tag{4.11}$$

where the feature weight constitutes the factor $\frac{1}{\sigma_{motion}^2}$ of the likelihood.

Intensity Feature Also the intensity of a particle is sampled around its previous value, just as the new locations. However, location and intensity are not comparable and so they are modeled by two separated features. Therefore, their feature weight is trained independently.

$$f_{intensity}(\mathbf{x}_t^{(i)}, \mathbf{y}_t) = -\frac{1}{2}(x_{intensity,t}^{(i)} - x_{intensity,t-1}^{(i)})^2 \tag{4.12}$$

This approach can also be extended to each of the 3 dimension of the location such that each dimension gets its own feature function and its own feature weight.

Linear Motion Feature The linear motion feature assumes that any particle continuously moves as it moved before. A motion vector $\mathbf{m}_t$ can be computed from two consecutive states by $\mathbf{m}_{t-1} = \mathbf{x}_{t-1} - \mathbf{x}_{t-2}$. The linear motion feature compares the current particle estimate $\mathbf{x}_t^{(i)}$ with the linearly moved estimates $\mathbf{x}_t^{(guess)}$ from the previous frame, which is $\mathbf{x}_t^{(guess)} = \mathbf{x}_{t-1}^{(i)} + \mathbf{m}_{t-1}$.

$$f_{linear\,motion}(\mathbf{x}_t^{(i)}, \mathbf{y}_t) = -\frac{1}{2}\sum_{k=1}^{3}(x_{k,t}^{(i)} - x_{k,t}^{(guess)})^2 \tag{4.13}$$

There are different ways how the motion vector $\mathbf{m}_{t-1}$ can be computed. One approach is to use the particle history and see how the particle itself has moved in the past. Such motion vector can be computed by $\mathbf{m}_{p,t-1} = \mathbf{x}_{t-1}^{(i)} - \mathbf{x}_{t-2}^{(i)}$. These vectors may, however, steer in different directions than the overall motion of the object, as it is shown in Figure 4.6. Instead of using the particle's history, one can use the history of the object's estimates to obtain more stable guesses. That is, the motion vector is determined by $\mathbf{m}_{e,t-1} = \mathbb{E}[\mathbf{x}_{t-1}] - \mathbb{E}[\mathbf{x}_{t-2}]$.

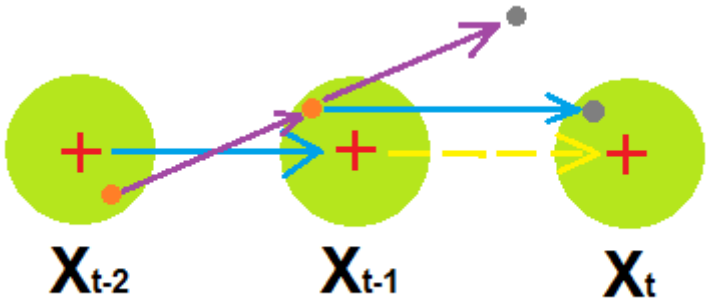


Figure 4.6: Linear motion vectors based on the particle history (purple) and motion vector based on the object's estimate history (blue). The orange dots show example particles - one at time $t-2$ and one at time $t-1$. The gray dots are the predicted locations $\mathbf{x}_t^{(guess)}$ - once based on the particle history (gray dot at the end of the purple arrow) and once based on the estimate's history (gray dot at the end of the blue arrow).

As neighbouring heart muscle cells are bound to the same surface and as they all move alike, it seems to be advantageous to compute a collective linear motion vector of the neighbourhood. Such feature can exploit the same definition of neighbourhood $\mathcal{N}(\mathbf{x}_t^{(i)})$ that has been used by the interaction features. The collective motion vector is computed by averaging over all motion vectors of a neighbourhood $\mathbf{m}_{n,t-1} = \frac{1}{|\mathcal{N}(\mathbf{x}_t^{(i)})|} \sum_{l \in \mathcal{N}(\mathbf{x}_t^{(i)})} \mathbb{E}[\mathbf{x}_{l,t-1}] - \mathbb{E}[\mathbf{x}_{l,t-2}]$.

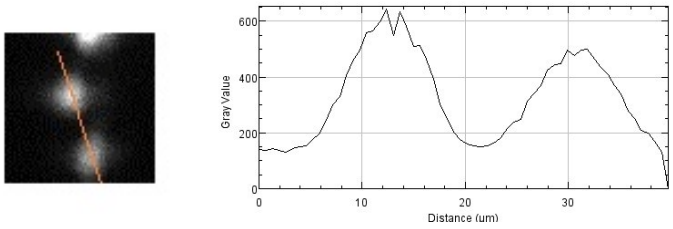
4.4 Enhancing the Features

This last section covers ideas, which have been found while running some experiments and which are to enhance the performance of specific features.

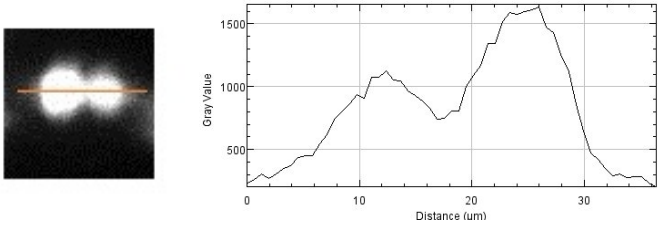
Predicting Location of other Objects The multi-object observation feature and as well as the interaction feature are computed on the estimates of the other objects' trackers. However, they become available only after a frame has been processed so that these estimates are always delayed. It shows that it is possible to counter such delay by assuming that all objects in a neighbourhood collectively moved into the same direction. That is, the estimates $\mathbb{E}[\mathbf{x}_{l,t}]$ of the multi-object observation and the interaction features can be computed by $\mathbb{E}[\mathbf{x}_{l,t}] = \mathbb{E}[\mathbf{x}_{l,t-1}] + \mathbf{m}_{n,t-1}$, where $\mathbb{E}[\mathbf{x}_{l,t-1}]$ is the output of the respective object tracker at time $t-1$.

Influence Factor for Illumination Intensity Even though all cells look alike and act alike, they are not the same. Heart cells that show high intensity values are much easier represented by the observation likelihood as cells that have lower intensity values. Especially when cells come very close and they are pressed together, then they show these high intensity values. At the same time it needs a stronger focus on the interaction features, because they are that close.

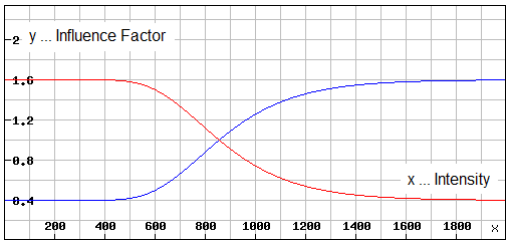
To consider this behaviour in these two feature functions, they are multiplied by an additional illumination-dependent factor, which is applied next to the feature weight. This factor strengthens or reduces the feature values of the original value and it can be designed as shown in Figure 4.7c.



(a)



(b)



(c)

Figure 4.7: The Figures 4.7a and 4.7b show two examples of neighbouring cells. The cells in the first figure are less brighten compared to the ones in the second. Their maximum intensity value is about 600, their distance about 20µm. The second figure shows two cells which have intensity values > 1000 and their distance is about 12µm. Such different cells can be expressed by the same features, however, the tracker performs better when they are differently weighted. Figure 4.7c roughly depicts how an additional influence factor that depends on the intensity values can look like.

5 Experimental Evaluation

The discriminative particle filters that have been explained in Chapter 3 and the necessary features from Chapter 4 have been implemented in Java as a Fiji plugin. This chapter is all about running experiments so that, in the end, the problem of tracking heart muscle cells can be solved by the discriminative particle filters.

The first experiments concentrate on evaluating the plugin on a artificial and known problem. They demonstrate that the discriminative particle filter works as expected and that the feature weights are successfully learned from training sequences. These experiments are covered in Section 5.1.

The second part of this chapter is about solving the original problem. That includes selecting appropriate feature functions which model the heart cells and comparing the results with the Fiji built-in Feature Point Tracker [Sbalzarini and Koumoutsakos, 2005]. It shows that the implemented discriminative particle filter is able to track the heart muscle cells. On the one hand, the tracking plug-in reports fewer failures than the Feature Point Tracker, i.e., the estimated trajectories do not jump between different image appearances. On the other hand, the Feature Point Tracker is able to detect cells more precisely.

5.1 Evaluating the Tracking Plug-In on Artificial Sequences

The particle filters, which are introduced in Section 2.3, model target objects by an appearance model and their motion between two observations. In Section 3.1, which presents the discriminative particle filters, it was stated that the relation between these two filters can be established by using the log-likelihoods of the generative motion and observation model as features. The first experiments, which are presented in this section, build the bridge between the generative and the discriminative models. The artificial sequences are generated according to simple generative models whose parameters are known, the log-likelihood features of the discriminative particle filter are then adjusted on these sequences. It is shown that the implemented learning procedure works as expected and the parameters of the generative model can be recovered.

Another important issue is to find how the multi-object features work. The implemented

plug-in is tested in a second experiment which focuses on tracking multiple interacting blobs. However, this kind of interactions are not generated according to some generative model and, therefore, the true feature weights are unknown. This experiment demonstrates the capabilities of the interaction features.

Experimental Setup In the generated artificial sequences a Gaussian blobs is randomly moving in a noisy environment. Therefore, the Gauss blob randomly moves according to the likelihood $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ which explains a normal distributed motion of the blob around its previous location. The generative observation $p(\mathbf{y}_t|\mathbf{x}_t)$ explains a fixed-size Gauss blob to which normal distributed background noise is added. The discriminative particle filter evaluates the particles by the the random motion feature $f_{random\,motion}(\mathbf{x}_t^{(i)}, \mathbf{y}_t)$ (Equation 4.11) and by the the single observation feature $f_{single\,observation}(\mathbf{x}_t^{(i)}, \mathbf{y}_t)$ (Equation 4.3). Both are presented in Chapter 4. To further simplify the model, the intensity of the generated blobs is fixed. The unknown parameters which shall be learned from data are standard derivations σ_{motion} and $\sigma_{observation}$ - the feature weights are inversely proportional to the respective standard derivation, i.e., $w_{random\,motion} = \frac{1}{\sigma^2_{motion}}$ and $w_{single\,observation} = \frac{1}{\sigma^2_{observation}}$.

The generated sequences are 100 frames long and display the Gauss blob which is drawn with $\sigma_{PSF} = 5$ pixel and a fixed maximal intensity of 15 (in the center of the blob). The background noise is drawn with $\sigma_{observation} = 15$ (intensity value) around a fixed mean of $\mu_{observation} = 300$ (intensity value). The blob moves around its previous location with $\sigma_{motion} = 2$ pixel. Six of such sequences are generated, one to learn the feature weights and five to validate them.

The evaluation of the interaction features needs generating sequences with multiple blobs. The video length and parameters for the observation and motion model are the same as before, but they show six moving blobs. Since the cells in the original problem do not overlap, the generated blobs must not overlap either. Therefore, if any of the randomly moved blobs was moved closer than $\tau_{redraw} = 10$ pixels to another blob, then all blobs were redrawn in this frame. However, because of rejecting overlapping motions the true σ_{motion} is unknown. Also the true value of the feature weight $w_{interaction}$ is not determined by the setup.

The generated ground truth locations of the blobs are stored next to the video for the evaluation.

Evaluating the Learning Process As explained in the setup, the artificial problem that shows one artificial blob can be tracked with a tracker that uses only two features: the clipped single observation feature of Section 4.1 and the random motion feature of Section 4.3. The true parameters are known from the generative model, these are $\sigma_{motion} = 2$

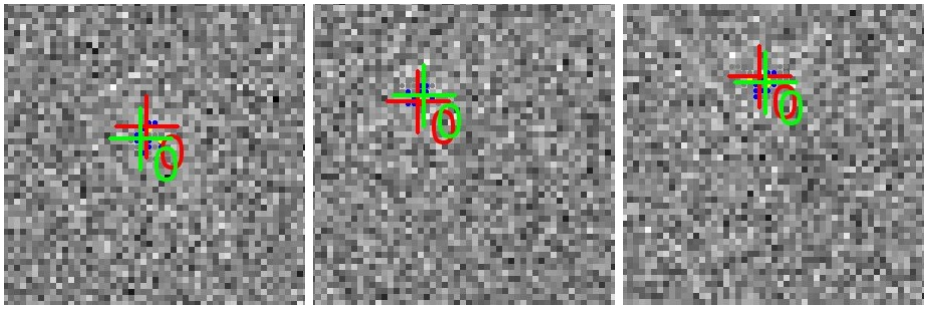


Figure 5.1: Noisy sequence with moving Gaussian blob. The images show the ground truth (green cross), the trackers estimate (red cross), and the particles with the highest weight. The particle likelihood is encoded in the color - the color order from highest to lowest probability is: red, orange, yellow, cyan, blue, gray.

pixel and $\sigma_{observation} = 15$ (intensity value). The first experiments is about finding these true values by the automated learning process. Therefore, two independent training runs are evaluated, both starting with two different initial parameters. The feature weights are then learned from five training sequences by minimizing the mean squared error of the particles' location and the ground truth location of the blob. The standard deviations of these models can be computed from these feature weights, since for both features it holds that $w = \frac{1}{\sigma^2}$. In the following figures only these standard deviations are reported and compared against the true values. The reported errors have been measured on and averaged for another five validation sequences. The error is defined as the distance (in pixel) between a particle's location and the ground truth location in space.

Figure 5.1 illustrates three frames of the sequence in which a Gaussian blob moves around. The targeted blob is marked by the green cross that is the ground truth. The red cross shows the estimate of the particle filter, which is surrounded by the particles (see the figure's description).

At first, Figure 5.2 shows a 3D plot of the error surface. The x-axis shows the deviation of the motion model σ_{motion}, which indicates the valid spread of the particles from their previous position. The y-axis shows the deviation of the observation model $\sigma_{observation}$, which specifies the spread of the noise in the observation. The lowest error is where $\sigma_{motion} = 2$ and $\sigma_{observation} = 15$ - the ground truth parameters which are indicated by the green arrow. To the left and to the right of this point one can see two corridors, in which the error is slightly increasing. When σ_{motion} increases, the particles of the tracker are allowed to mover farther from its last position. If σ_{motion} is sufficient large, i.e., if it exceeds the borders of the image, then the feature allows any motion in the image and the particles are weighted solely by the observation. If, however, σ_{motion} decreases and approaches to zero, then the particles are not allowed to move at all. This case is removed from the figure otherwise one could not see the corridors in the first place.

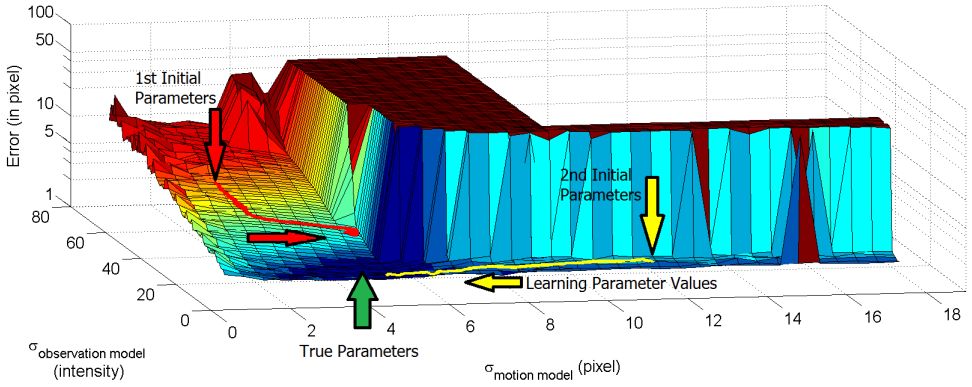
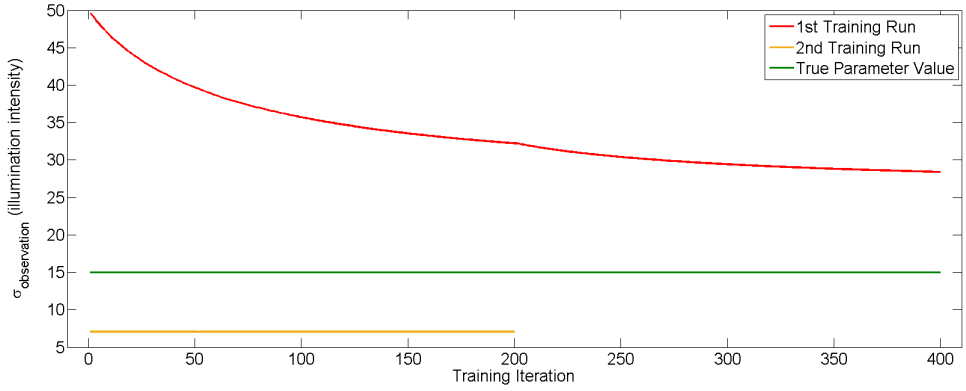


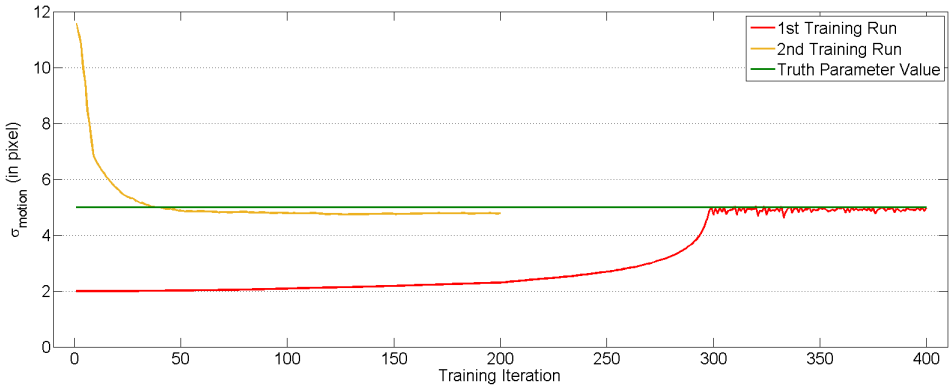
Figure 5.2: Plot of the error (in pixel) against σ_{motion} of the motion feature and $\sigma_{observation}$ of the observation feature. The red and the yellow graphs depict two different learning processes, which start with different initial parameters. The parameters in both runs are found by minimizing the mean squared error. The green arrow points to the true values with which the videos were generated.

Looking only at the observation feature, one can observe a similar behaviour for $\sigma_{observation}$. With $\sigma_{observation} \to \infty$ the feature weight $w_{single_observation}$ tends to zero and the parameters are weighted solely by the motion model. However, the motion feature is not sufficient to find the blob in the noisy image. Therefore, with increasing an $\sigma_{observation}$ the error increases too. For this problem the observation is a stronger feature than the motion and therefore the error goes up as $\sigma_{observation}$ increases. But decreasing the parameter $\sigma_{observation}$ too much is equivalent to assuming that there is no noise in the image. In the extreme case, the posterior distribution is represented by only one particle, which is the one with the best fitting appearance. This also comes with a higher error.

The next plot in Figure 5.2 shows two graphs that represent the same learning curves that are depicted in Figure 5.2. Figure 5.3a shows the standard deviation of the observation during the learning iterations, Figure 5.3b shows the learned σ_{motion} during the learning process. The two graphs (red and yellow) depict the two different learning runs, which start with different initial values. It shows that σ_{motion} approaches to the true value in both tries, but the true standard deviation of the observation model was not found. Only the first attempt goes into the right direction. Assuring better convergence needs further tweaking the learning parameters, i.e., the learning rate and the number of iterations. Both parameters have not been properly set for this problem because of limited time. However, the plots indicate that the learning process works in general. The other learning procedures (Perceptron learning, maximizing the posterior likelihood, and minimizing the residual error of the mean) have been evaluated in the same way and show similar results.



(a)



(b)

Figure 5.3: These figures depict the learning process in greater detail, which is roughly sketched in Figure 5.2. Both figures show the process of two different learning runs (red and yellow). The upper figure depicts the process for $\sigma_{observation}$. In the first attempt (red), the parameter $\sigma_{observation}$ slowly converges to the true parameter (green). However, the learning rate is too small and true value is not reached in 400 iterations. In the second attempt (yellow), the parameter does not change much from the initial values. That is, because the learned parameter graph just leaves the corridor before it the training run stops (see again Figure 5.2). The second figure depicts an exemplary learning processes of the parameter σ_{motion} for both runs (red and yellow).

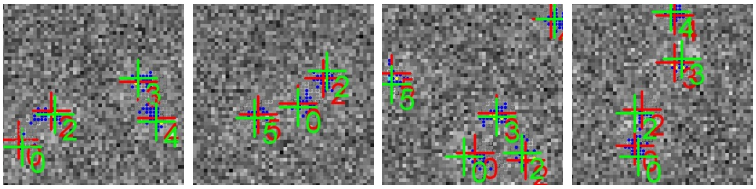


Figure 5.4: Few selected frames from a generated sequence showing multiple interacting blobs. The color encoding is as in Figure 5.1, that is, the green cross shows the ground truth, the red cross shows the trackers' estimates, and the surrounding dots are particle locations.

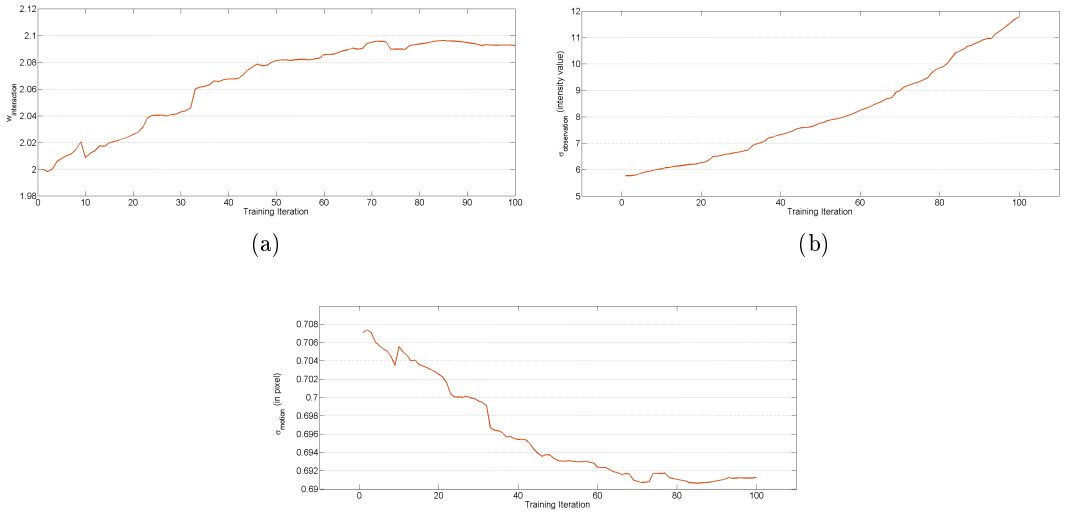
Evaluating the Joint Features Inspecting the interaction features needs evaluating the sequences that show multiple artificial cells. These sequences are generated as explained in the setup paragraph. The tracker uses the single observation feature, the random motion feature, and the joint interaction features to evaluates previous locations of the other blobs. As an example, Figure 5.4 shows four frames out of a tracked training sequence.

The reported parameters in Figure 5.5 are again learned by minimizing the mean squared error. The feature weight of the interaction feature as well as the standard deviation of the motion converge after about 100 learning epochs. The final value for $\sigma_{observation}$ was not found in the same time. The errors on the training sequence and on two validation sequences are reported in Figure 5.6.

Concluding the Experiments with the Artificial Sequences The generated sequences with artificial cells are used to debug the learning process and examine the interaction features. The results indicate that the tracking plug-in is working as expected. These experiments were not designed to show strengths and weaknesses of certain learning approaches or feature functions. The reported results make clear that the automatic adjustment of the feature parameters works. In addition, these experiment show that hyper-parameters like the learning rates and, even more, the initial values have huge impact on the success of the learning process. It is not important further adjust these parameters on the artificial sequences, but it is necessary to keep this in mind in the following experiments.

5.2 Tracking Heart Muscle Cells

The second task is to locate all heart muscle cells in the given microscopy data. This needs setting up the tracking plugin, labeling the data, choosing proper features, learning the parameters from data, and evaluating the configured tracker on the problem. The



(a) (b)

(c)

Figure 5.5: Learning the features' parameters: (a) depicts the learned weight of the interaction feature, (b) shows the process of $\sigma_{observation}$, and (c) shows the learned values for σ_{motion}.

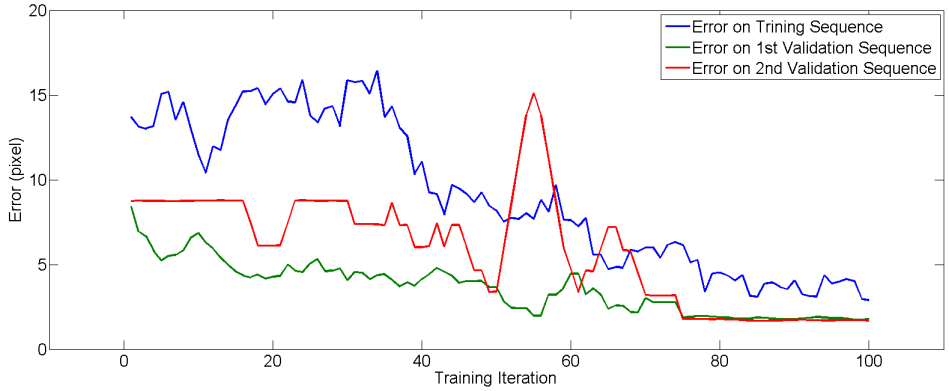


Figure 5.6: Error (in pixel) on the training sequence (blue) and error of two validation sequences (green and red).

tracking plugin is working and it has been shown that the learning procedure works as expected. This section covers the experimental setup, the selection of proper features, and the evaluation of the problem.

Experimental Setup The given data was provided without any annotations and, therefore, all cells need manual labeling. The recording contains more than 200 cells in 3D space that are visible in all 87 frames, but it is not always trivial to distinguish multiple cells when they lie very close together. The data is illustrated in Figure 5.7.

To simplify the manual labeling task, the experiments are not evaluated on the whole image stack, but only on three manually selected sections. In each of them the cells can be distinguished in the z-projection by a non-expert. These sections are cut out of the original 3D-image sequence by defining a fixed 3D bounding box, whose location does not change over time. They show the front (Figure 5.8a) and the back (Figure 5.8b) of the right heart chamber, as well as the front of the left heart chamber (Figure 5.8c). Each section from the original video forms again a video sequence that is 87 frames long. Figure 5.8 shows a single z-projected and annotated frame of each of the sequences. There are 48 annotated cells in the first sequence (front of the right heart chamber), 22 annotated cells in the second sequence (back of right chamber), and 37 annotated cells in the third sequence (front of left chamber). These annotations are manually added using the implemented plug-in.

This, of course, does not solve the original problem, but it is sufficient to report the performance of the discriminative particle filters. This comes with the additional advantage that model parameters can be trained and validated on different sequences. In the presented experiments, the parameters are always learned on the image sequence that shows the front of the right heart chamber, which is therefore called training sequence. The other sequences are used to validate the learned parameters and therefore referred to 1st (back of right chamber) and 2nd (front of left chamber) validation sequence.

Each of object is tracked by an individual tracker with 200 particles. The parameters in the reported experiments are found by minimizing the mean squared error, the training algorithm run for 15 iterations. The thresholds of the training algorithm were set to $\tau_{update} = 2500nm$, which is about 3.8 pixels, and $\tau_{reset} = 7800nm$, which is equal to 12 pixels in the 3D stack. The learning rate for the k^{th} iteration is determined by $\gamma_k = \frac{\gamma_0}{1+k\gamma_0\lambda_{decay}}$, where the parameter λ_{decay} is set to 0.05 and therefore the learning rate decreases slowly. The initial learning rates γ_0 depend on the different feature functions and the initial feature weights.

Interpreting the Results Each of the following experiments reports on two different measurements. The first is the average error of the objects' estimates to their respective

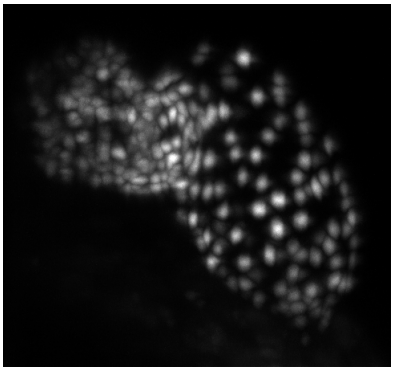
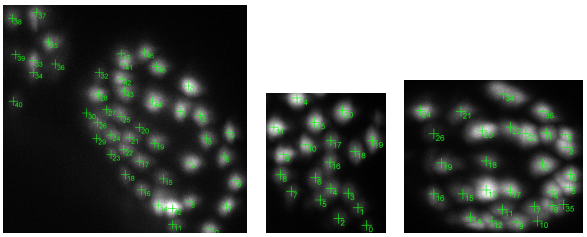


Figure 5.7: View on the projected data (4th frame, z-projection using maximum values)



(a) (b) (c)

Figure 5.8: Three different sections from the original video. They show projections of the front (a) and the back (b) of the right heart chamber, as well as a projection of the front of the left heart chamber (c).

ground truth location. This error is defined by the Euclidean distance in 3D space. After evaluating a sequence, the errors of each cell and each frame is averaged over the number of cells and the number frames for which a cell is visible; that is, the average is computed over the number of all accumulated values. To measure the error, the trackers are evaluated after a parameter update with the newly learned parameters without resetting any tracker even if the reset threshold τ_{reset}.

The second measurement is the failure rate of the objects' trackers. A failure is reported once a tracker's error exceeds the reset threshold τ_{reset}. The failure rates average over the number of tracked cells.

In the original data no cell appears or disappears at some point in time. But since the three chosen sections of the original data lie inside the heart, cells may cross the defined bounding boxes and, therefore, appear or disappear inside these sections. In the presented experiments a cell is tracked until it disappears. Recurring cells are treated as if they are different cells, that is, no linkage between the disappearing and the recurring cell is drawn.

Evaluating a Simple Model The artificial problem in Section 5.1 could be solved using a simple model and for this reason the first idea is to apply a simple model to the heart cell problem as well. The selected features that are used for this case are: The linear motion model for which each particle is assumed to move linearly from its previous position, the intensity features that new values are normally distributed around their previous value, the clipped observation features that models single cells as Gaussian blobs, and the interaction feature that evaluates the function $g_1(\mathbf{x}_t^{(i)}, \mathbf{x}_{l,t}) = \frac{1}{||\mathbf{x}_t^{(i)} - \mathbf{x}_{l,t}||}$ for each particle and each estimate at time $t-1$ of neighbouring cells.

The results of the trackers that use these features are visualized in Figure 5.9. One can see that the error strongly decreases on all sequences only in the first iterations. When considering also the failure rate, it can be observed that the tracker's performance increases only on the training sequence. The results of each sequence after 15 training iterations are reported in Table 5.1.

In addition to these results, Figure 5.10 depicts two frames out of the tracked sequence and it illustrates two types of errors that occur. The first one can be observed in Figure 5.10a; this shows that simple motion model does not cope with the motion of the cells, because some estimates lie behind the corresponding target cells after they start moving. After losing their target, these trackers either find some false appearance or keep floating in space. The second error can be observed in Figure 5.10b, where multiple estimates collapse on one appearance. The single observation feature and the interaction feature, which is computed on the previous location of the other estimates, are not able to keep the different trackers apart.

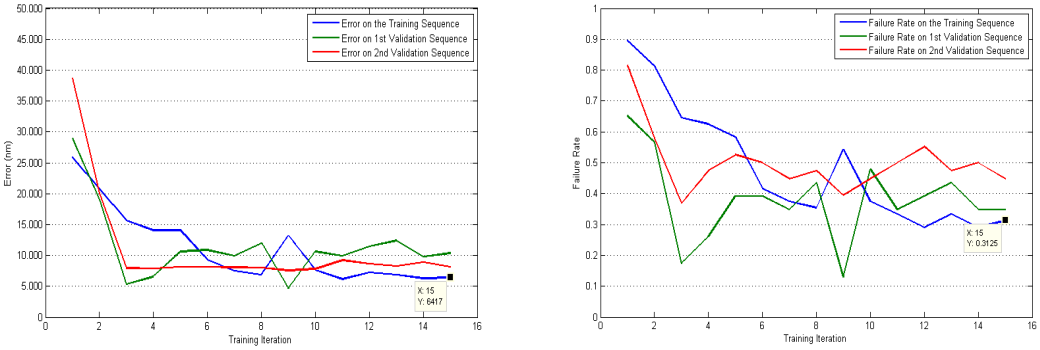
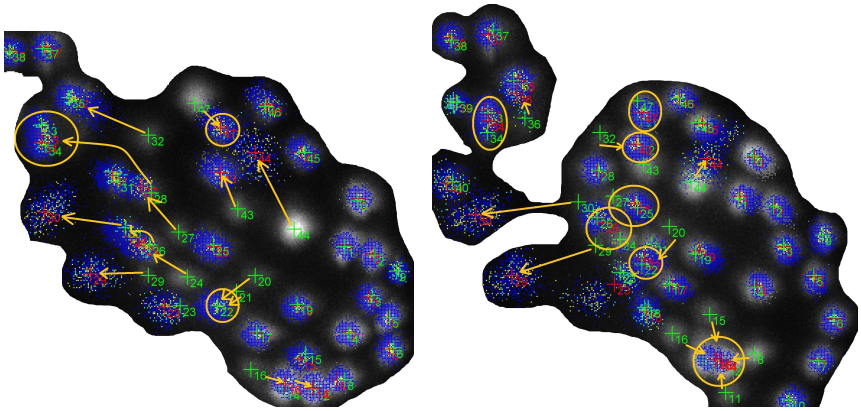


Figure 5.9: The left chart depicts the average error in 3D space; the right chart shows the failure rates per iteration for the simple model. The final values of the training sequence are marked for better recognizability.

	Training Sequence	1st Validation Sequence	2nd Validation Sequence
Average Error (in nm)	6417	10529	8635
Failure Rate	0.3125	0.3478	0.4474

Table 5.1: Results after 15 training iterations of the trackers that uses the simple model



(a) (b)

Figure 5.10: Selected frames that show the different types of errors. The displacements or drifts are caused by strong neighbouring appearances (e.g. for cell 16, 15 or 21) or by fast motion (as for cells 27, 30, or 44). The images are the frames 27 (left) and 35 (right) of the training sequence. Parts of the black background was removed to save ink.

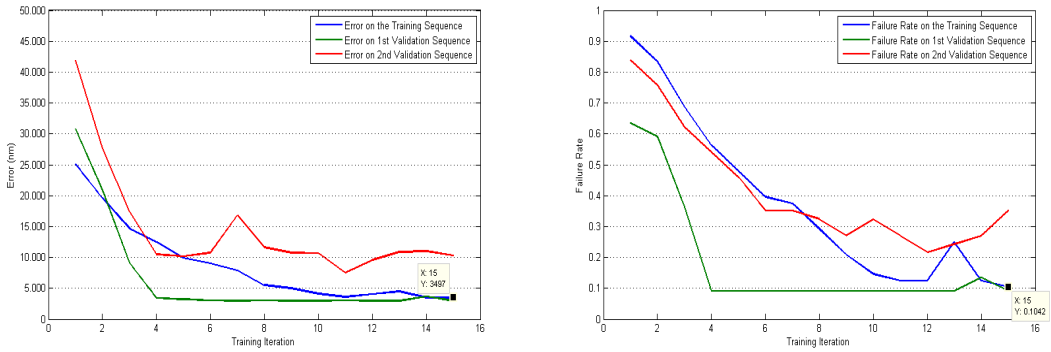


Figure 5.11: Average error in 3D space (left) and failure rates (right) per iteration for the "collectively-moved" model

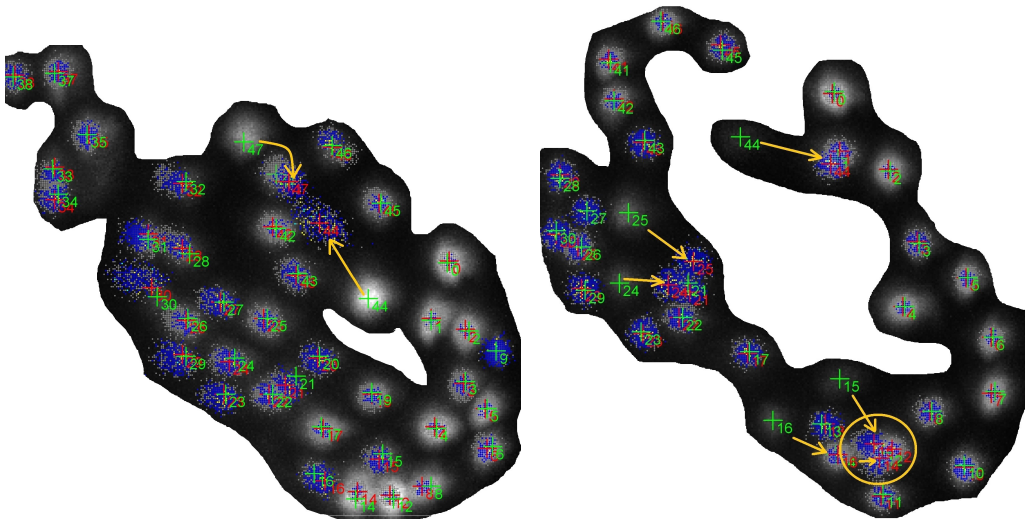
	Training Sequence	1st Validation Sequence	2nd Validation Sequence
Average Error (in nm)	3497	2994	10301
Failure Rate	0.1042	0.0909	0.3514

Table 5.2: Results after 15 training iterations of the trackers that utilizes the "collectively-moved" features

Evaluating a Set of "Collectively-Moved" Features The features which are based on a collective motion are designed to increase the performance, as the heart cells lie on a common surface that forms the heart chambers and as neighbouring cells on this surface move alike. To confirm this, a second experiment is run using a set of collective motion features: The linear motion feature which is computed on the collective motion vector, the normal-distributed intensity feature (as before), as well as the joint observation feature and the interaction feature, which are both computed on the other cells' estimates that have been moved from the previous frame according to the collective motion of their neighbourhood. For all these features the collective motion vector always refers to the neighbourhood-average motion as explained in Section 4.3 and Section 4.4.

Comparing the results to the ones of the previous experiment, the errors drastically decreased for all but the 2nd validation sequence and the failure rates go down by more than 10% (see Figure 5.11 and Table 5.2).

Figure 5.12 shows two exemplary frames out of the tracked training sequence. As particles are evaluated by the collective linear motion feature, they are either carried along or slowed down by motion of the neighbourhood. In general, this reduces the number of failures, but once a tracker has lost its target it might be easily lead into a false direction. This figure also shows a similar problem as before, where three trackers are grouping around a very bright appearance of a single cell. However, the trackers do not collapse on such cell, but they are trying to explain this appearance together.



(a) (b)

Figure 5.12: Two selected frames - (a) Frame 27 and (b) Frame 81 - from the tracked training sequence. The errors are not caused by fast motion anymore, as the motion is explained by the features. However, drifts to brighter neighbouring cells still occur (e.g. cells 14, 15, and 16 in in Frame 81). Also the collective motion features introduce errors where a tracker's particles are carried away due to their neighbours motion (e.g. cells 24 and 25 in Frame 81).

	Training Sequence	1st Validation Sequence	2nd Validation Sequence
Average Error (in nm)	2993	2926	11194
Failure Rate	0.0833	0.0909	0.2973

Table 5.3: Results after 15 training iterations of the tracker that utilizes 8 feature functions

Evaluating all Features Together The features that are computed on the collective motions explain the problem quite well, but there is till room for improvement. An open question is whether a tracker that exploits all feature functions can yield better results. Another interesting question is when learning the feature weights of all features, which features turn out to be significant and which are not. Is it thereby possible to avoid the overhead of selecting and testing the proper feature set?

In this third experiment the trackers evaluate the particles on the following features: The intensity feature (as before), the random motion feature that assumes that a particle makes normal distributed moves around its previous location, the linear motion feature that assumes that single particles continue moving with their previous velocity, the collective linear motion that consider the motion of the neighbourhood when evaluating the new particle's position, the single observation feature, the joint observation feature which is computed on the collectively moved estimates of the neighbouring cells, the interaction features that are evaluated on the previous estimates of the neighbouring cells, and the interaction features which are also evaluated on the collectively moved estimates of the neighbourhood.

Error and failure rate during the learning process is shown in Figure 5.13. The results after the latest training step are listed in Table 5.3. It shows that these results are almost similar to the results for the "collectively-moved" feature set. The tracks look similar, but it shows that there are fewer estimates that drifted away from their targeted cell. Compare Figure 5.12 to Figure 5.14. This is also reflected by the failure rate, which is slightly reduced.

The course of the feature weights which have been computed by the learning process are displayed in Figure 5.15. It shows that weights of the collective linear motion feature and the joint observation of the collectively moved neighbourhood increased the most. But also the weights of the single observation and the particle's linear motion feature increased compared to their initial position. It can be seen that the set of "collectively-moved" features from the previous experiments can be improved by these two features. On the other hand, the intensity feature, the random motion feature, and the interaction feature that evaluates the previous neighbourhood are not significant for the description of the heart muscle cells.

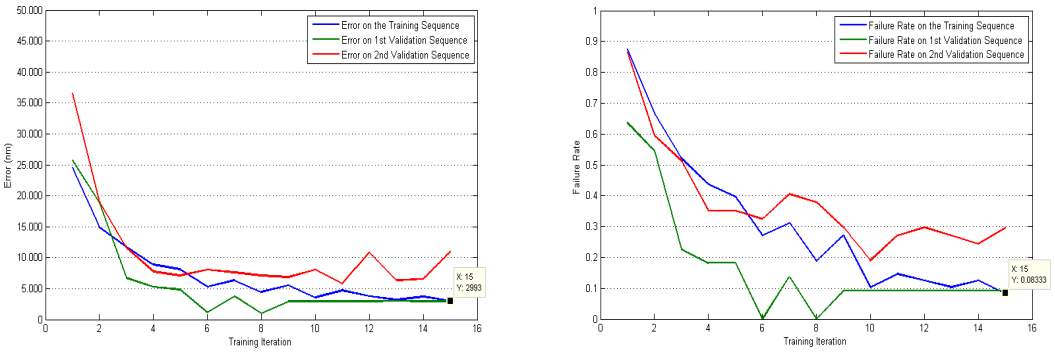
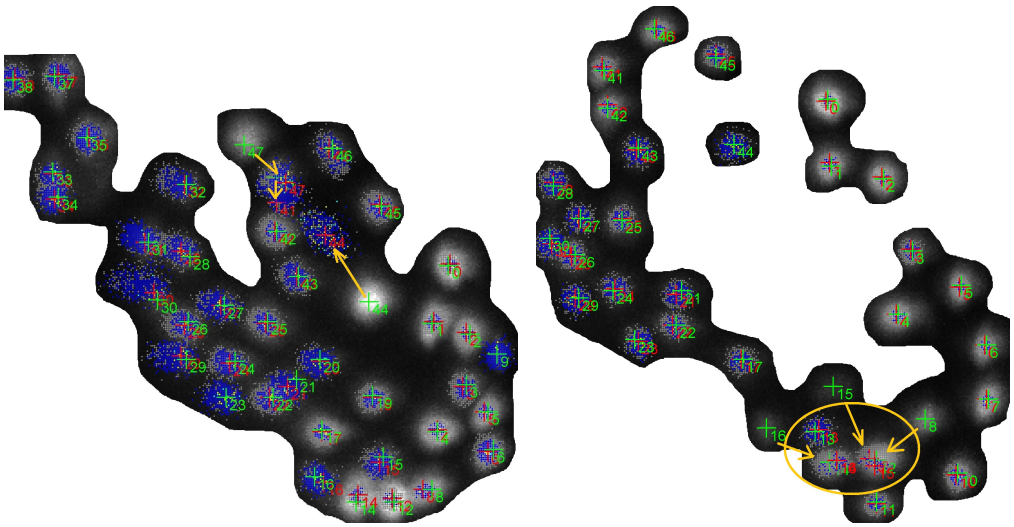


Figure 5.13: Average error in 3D space (left) and failure rates (right) per iteration using all feature functions.



(a) (b)

Figure 5.14: Selected frames of the tracked training sequence - (a) Frame 27 and (b) Frame 81. Except for Cell 44, the errors are caused by drifting to brighter neighbouring cells.

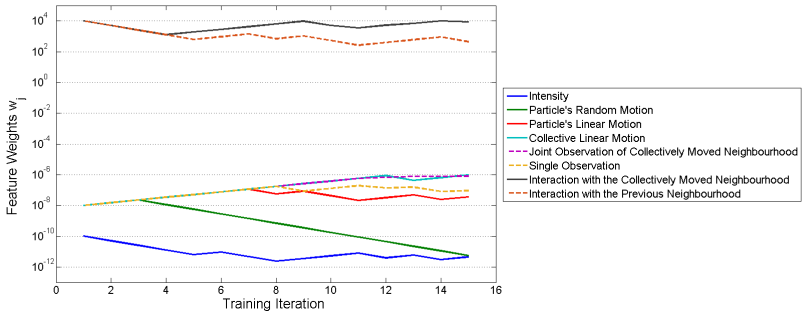


Figure 5.15: Course of the feature weight during the learning process

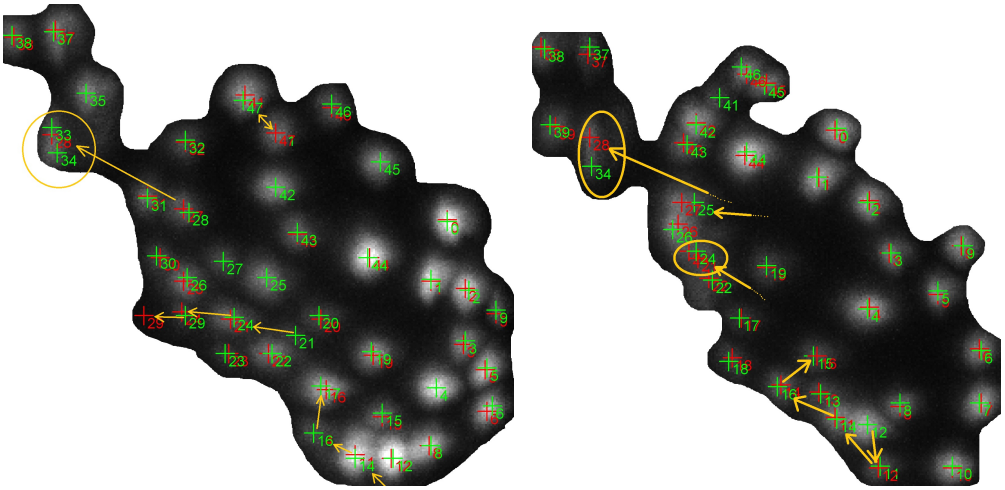
Evaluating the Fiji Built-In Feature Point Tracker The Feature Point Tracker, which is described in [Sbalzarini and Koumoutsakos, 2005], is the baseline object tracker to which the implemented discriminative particle filters are compared to. It is implemented in the MosaicSuite[1] and is shipped with Fiji. This tracker is developed to analyze and track cells in video microscopy and has been successfully applied to track cells in various biological applications. To find an object's trajectory in a video sequence, this tracker first detects object occurrences, which they call feature points, in each frame and then links these feature points into trajectories.

Once the cells have been tracked, the output trajectories need matching to the ground truth trajectories to evaluate the errors and failure rates. Therefore, each ground truth trajectory is matched to the closest output trajectory. The error is then computed on these tracks until one of them disappears or until the sequence ends. There might be multiple output trajectories that are close to one ground truth path, which is problematic for a fair comparison of the approaches. Such mistakes can be caused by running the tracker with poorly selected parameters, which therefore must be changed. However, even when the parameters have properly selected and tested the tracker might detect cells where no ground truth label is found. It might be that the either tracker is wrong or a ground truth label is missing, as these are manually added by a non-expert. To bypass this problem only the first[2] output track is evaluated, the other one is ignored. On the other hand, if a ground truth is present but no output track can be assigned, then a failure is reported (but no error is measured). If an cell disappears in the sequence but occurs again and if the the distance in time between disappearing and recurrence is greater than the explained link range of the Feature Point Tracker, then the ground truth track (which is continuous) is treated as if it where two independent tracks.

[1] `http://mosaic.mpi-cbg.de/?q=downloads/imageJ`
[2] The first track: The track that begins in the frame with the smallest frame number.

	Training Sequence	1st Validation Sequence	2nd Validation Sequence
Average Error (in nm)	4943	3125	8565
Failure Rate	0.2683	0.1168	0.3862

Table 5.4: Results of the Feature Point Tracker



(a) (b)

Figure 5.16: Two frames from the training sequence that has been tracked by the Feature Point Tracker [Sbalzarini and Koumoutsakos, 2005] - frame 27 (left) and frame 42 (right).

The parameters of the Feature Point Tracker are set up as follows: radius $w = 7$, cutoff threshold $T_s = 0$, upper percentile $r = 0.60$, link range $R = 4$, maximal displacement $L = 20$, and the tracked cells are assumed to move with constant velocity. The results with this setup are listed in the Table 5.4. After matching the tracker's output to the ground truth, the tracks can be displayed just as the outputs of the discriminative particle filters, which is done in Figure 5.16. Errors where multiple estimates explain a single appearance rarely occur; it happens for Cell 24 in Figure 5.16b. The feature point detector can detect the cells very accurately as their appearance in the given images is quite clear. Linking these feature points to trajectories is more prone to errors. A common mistake is the jumping to an appearance of another cell and, since there can be only one feature point for each appearance, this may lead to different succeeding errors. The cells may exchange their place, as shown in Figure 5.16a by the Cells 41 and 27. But one track could also terminate another trajectory by taking its place, see for example Cells 17 and Cell 33 in Figure 5.16a. A jumping cell may also push the previous track away to another appearance, this happens e.g. with the Cells 12, 11, 14, and 16 (in this order) in Figure 5.16b.

Average Error	Training Seq.	Validation Seq. 1	Validation Seq. 2	Validation Mean[3]
Simple Model	6417	10529	8635	9341
Collectively Moved Model	3497	2994	10301	7576
All Features	2993	2926	11194	8111
Feature Point Tracker	4943	3125	8565	6537

Failure Rate	Training Seq.	Validation Seq. 1	Validation Seq. 2	Validation Mean[3]
Simple Model	0.3125	0.3478	0.4474	0.4103
Collectively Moved Model	0.1042	0.0909	0.3514	0.2543
All Features	0.0833	0.0909	0.2973	0.2203
Feature Point Tracker	0.2683	0.1168	0.3862	0.2857

Table 5.5: Summary of the trackers' results.

Concluding the Experiments with a Comparison The experiments show that the problem of tracking heart muscle cells with discriminative particle filters can be solved. Properly designed feature functions have huge impact on the results as they describe the characteristics and behaviour of the targeted objects.

Comparing the set of feature functions, it becomes clear that the simple features which have been evaluated in Section 5.1 do not sufficiently explain the heart cells. The experiment with the collectively moved features shows that more expressive features which consider the motion of the neighbouring cells are needed. However, it is not necessary to manually select the features to get an expressive model, but the experiment on the full feature set shows that the automatic parameter can find the most essential feature functions. Table 5.5 summarizes the results of this section's experiments.[3]

The Feature Point Tracker from [Sbalzarini and Koumoutsakos, 2005] is able to accurately detect cells in the given image, since the cells' appearances are quite clear. However, linking the detected feature points into trajectory is more prone to errors and leads to a higher failure rate. The discriminative particle filters, on the other side, may always continue a track and, as they run pseudo-independently, will not displace another cell's tracker.

[3] Mean over all validation objects in both sequence. Denote that the number of objects in both sequences differ.

6 Summary

This thesis examines the discriminative particle filters of [Hess and Fern, 2009] as an approach to track heart muscle cells in microscopy recording. Knowing the location of all heart muscle cells is necessary to reconstruct the heart's complex structure and to investigate the development process of the heart beat. It is hard to distinguish the cells in the given recording only by their appearance in the image, however, tracking of all cells jointly at the same time is infeasible. The discriminative particle filters allow to track each heart muscle cell pseudo-independently and evaluates multiple different features that describe the appearance and the motion of each cell, but also the interaction between different cells. The implemented tracking plugin has been evaluated on the given data and it has been shown that discriminative particle filters are able to locate and distinguish heart cells during the whole cardiac cycle. Future work may focus on producing more stable tracks by, for instance, exploiting more detailed features and by further tuning the learning parameters.

Bibliography

Léon Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012.

Janick Cardinale. Pftracking3d tracker software for imagej. online, 2008. `http://mosaic.mpi-cbg.de/?q=downloads/particle_filter_tracking`.

Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics, 2002.

Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and computing*, 10(3):197–208, 2000.

Neil J Gordon, David J Salmond, and Adrian FM Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proceedings F (Radar and Signal Processing)*, volume 140, pages 107–113. IET, 1993.

Rob Hess and Alan Fern. Discriminatively trained particle filters for complex multi-object tracking. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 240–247. IEEE, 2009.

Jan Huisken, Jim Swoger, Filippo Del Bene, Joachim Wittbrodt, and Ernst HK Stelzer. Optical sectioning deep inside live embryos by selective plane illumination microscopy. *Science*, 305(5686):1007–1009, 2004.

Zia Khan, Tucker Balch, and Frank Dellaert. An mcmc-based particle filter for tracking multiple interacting targets. In *Computer Vision-ECCV 2004*, pages 279–290. Springer, 2004.

Benson Limketkai, Dieter Fox, and Lin Liao. Crf-filters: Discriminative particle filters for sequential state estimation. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3142–3147. IEEE, 2007.

Petros Nemtsas, Erich Wettwer, Torsten Christ, Gilbert Weidinger, and Ursula Ravens.

Adult zebrafish heart as a model for human heart? an electrophysiological study. *Journal of Molecular and Cellular Cardiology*, 48(1):161–171, January 2010.

Christopher Rasmussen and Gregory D Hager. Probabilistic data association methods for tracking complex visual objects. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(6):560–576, 2001.

Yong Rui and Yunqiang Chen. Better proposal distributions: Object tracking using unscented particle filter. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 2, pages II–786. IEEE, 2001.

Ivo F Sbalzarini and Petros Koumoutsakos. Feature point tracking and trajectory analysis for video imaging in cell biology. *Journal of structural biology*, 151(2):182–195, 2005.

Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. *Introduction to statistical relational learning*, pages 93–128, 2006.

Appendix

Derivation: Gradient of the Minimizing Mean Squared Error

$$
\begin{aligned}
\frac{d\mathbb{E}[||\mathbf{x}_t - \mathbf{x}_t^{(GT)}||^2]}{dw_j} &= \frac{d}{dw_j}\left[\frac{\sum_i ||\mathbf{x}_t^{(i)} - \mathbf{x}_t^{(GT)}||^2 \exp\left(\sum_j w_j f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t)\right)}{\sum_k \exp\left(\sum_j w_j f_j(\mathbf{x}_t^{(k)}, \mathbf{y}_t)\right)}\right] \\
&= \frac{\sum_i ||\mathbf{x}_t^{(i)} - \mathbf{x}_t^{(GT)}||^2 f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t) \exp\left(\sum_j w_j f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t)\right)}{\sum_k \exp\left(\sum_j w_j f_j(\mathbf{x}_t^{(k)}, \mathbf{y}_t)\right)} \\
&- \frac{\sum_i ||\mathbf{x}_t^{(i)} - \mathbf{x}_t^{(GT)}||^2 \exp\left(\sum_j w_j f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t)\right)}{\sum_k \exp\left(\sum_j w_j f_j(\mathbf{x}_t^{(k)}, \mathbf{y}_t)\right)} \\
&\cdot \frac{\sum_i f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t) \exp\left(\sum_j w_j f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t)\right)}{\sum_k \exp\left(\sum_j w_j f_j(\mathbf{x}_t^{(k)}, \mathbf{y}_t)\right)} \\
&= \mathbb{E}[||\mathbf{x}_t - \mathbf{x}_t^{(GT)}||^2 \cdot f_j(\mathbf{x}_t, \mathbf{y}_t)] \\
&- \mathbb{E}[||\mathbf{x}_t - \mathbf{x}_t^{(GT)}||^2] \cdot \mathbb{E}[f_j(\mathbf{x}_t, \mathbf{y}_t)]
\end{aligned}
$$

Proof of Claim 1

Proof. In the code, the particle's weight is computed by

$$\begin{aligned}
\pi_t^{(i)} &= \frac{\exp\left(\tilde{\pi}_t^{(i)}\right)}{\sum_k \exp\left(\tilde{\pi}_t^{(k)}\right)} \\
&= \frac{\exp\left(\sum_j w_j \cdot f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t) - C_{max}\right)}{\sum_k \exp\left(\sum_j w_j \cdot f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t) - C_{max}\right)} \\
&= \frac{\exp\left(\sum_j w_j \cdot f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t)\right) \cdot \exp\left(-C_{max}\right)}{\sum_k \exp\left(\sum_j w_j \cdot f_j(\mathbf{x}_t^{(k)}, \mathbf{y}_t)\right) \cdot \exp\left(-C_{max}\right)} \\
&= \frac{1}{\sum_k \exp\left(\sum_j w_j \cdot f_j(\mathbf{x}_t^{(k)}, \mathbf{y}_t)\right)} \exp\left(\sum_j w_j \cdot f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t)\right) \\
&= \frac{1}{Z(\mathbf{y}_t)} \exp\left(\sum_j w_j \cdot f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t)\right).
\end{aligned}$$

This fits to Definition 2, where the particle's weight is determined by

$$\pi_t^{(i)} \propto \exp\left(\sum_j w_j \cdot f_j(\mathbf{x}_t^{(i)}, \mathbf{y}_t)\right).$$

□